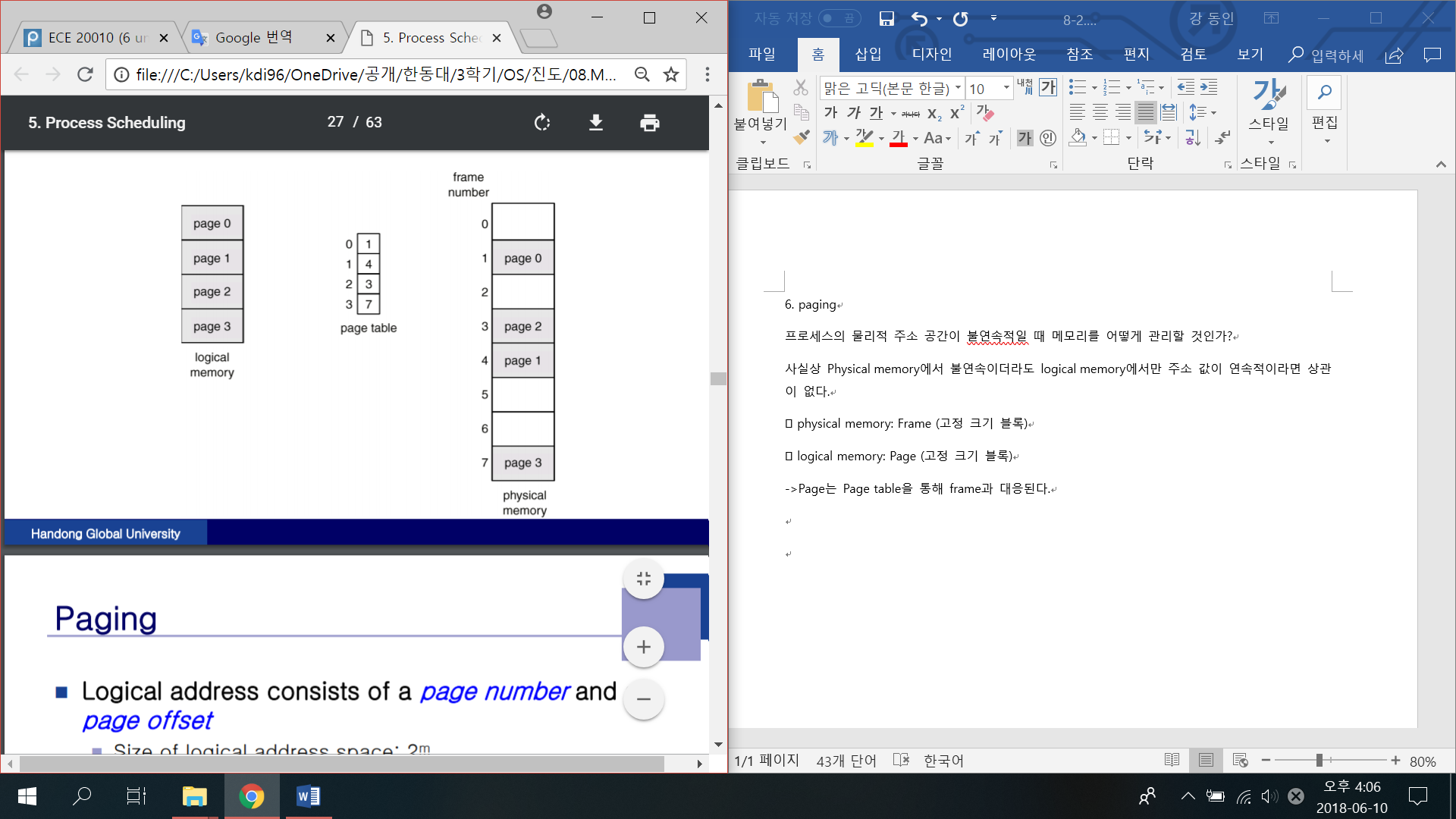
6. paging

프로세스의 물리적 주소 공간이 불연속적일 때 메모리를 어떻게 관리할 것인가?

Physical memory에서 불연속이더라도 logical memory에서만 주소 값이 연속적이라면 상관이 없다.

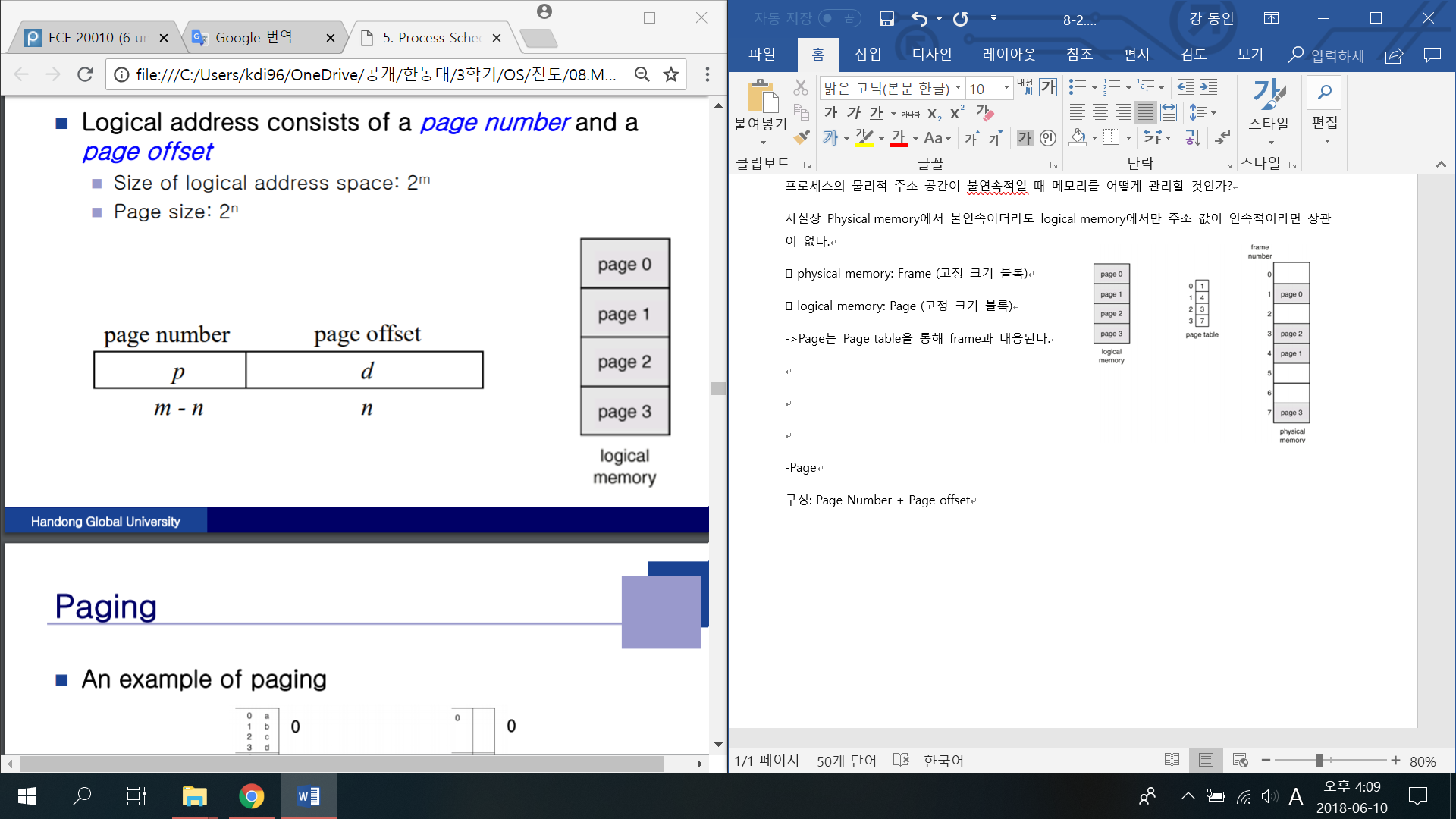
- physical memory: Frame (고정 크기 블록)

- logical memory: Page (고정 크기 블록)

->Page는 Page table을 통해 frame과 대응된다.

- Page

구성: Page Number + Page offset

* 논리적 주소 공간의 크기:
* 페이지 크기(=Page offset):
* 페이지 개수(page number):

설띵)

0 - 논리적 주소 공간의 크기: 256 (= (X))

4 - 페이지 크기: 4 (=)

8 - 페이지 개수: 64 (=)

12 일 때, 187번을 찾는다고 한다면

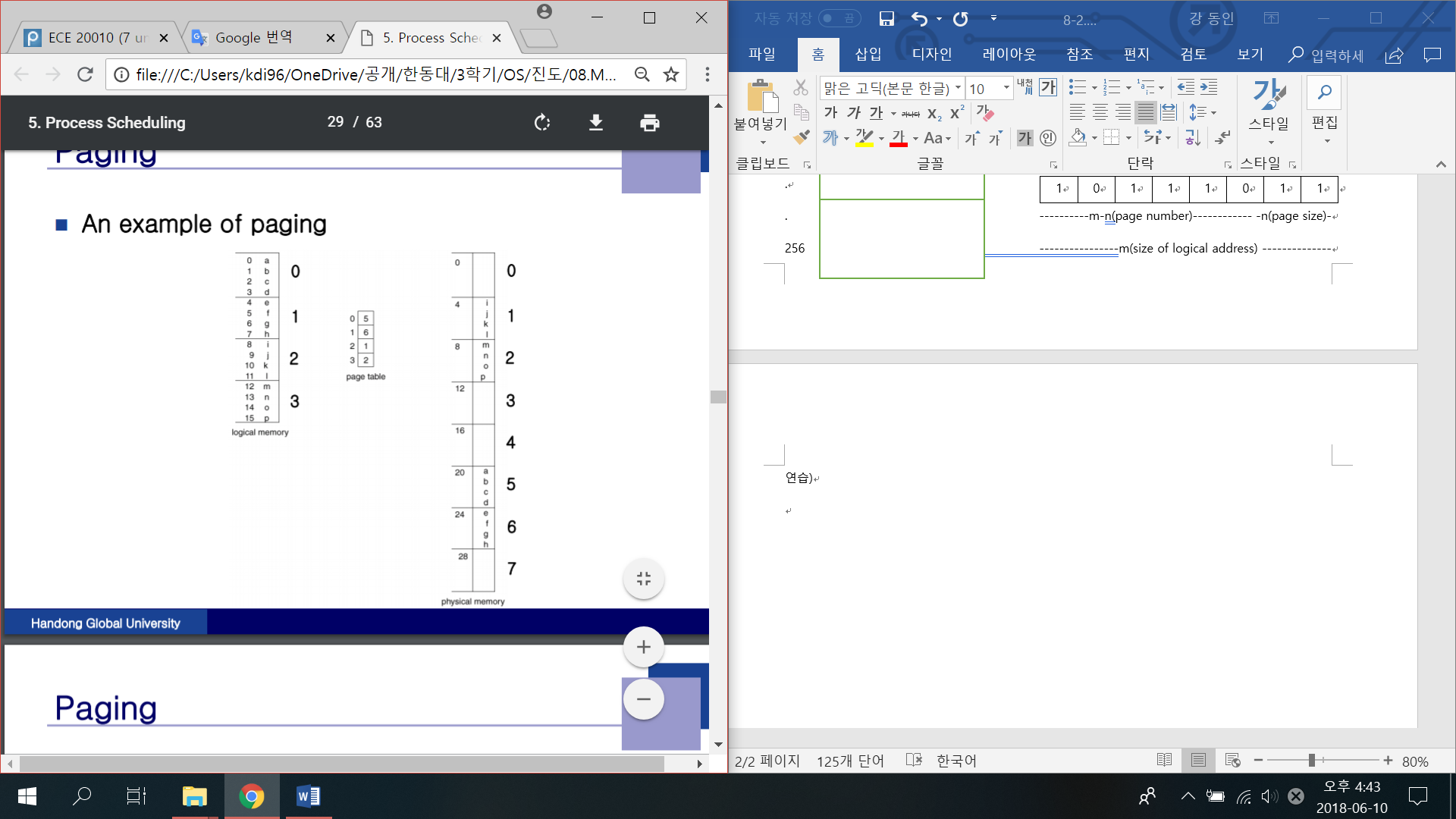
. 187 = 184(=4X46) + 3 (184page, 3offset)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |

.

. ----------m-n(page number)------------ -n(page size)-

256 ----------------m(size of logical address) --------------

연습) k의 physical address를 page table을 사용해 구하여라

Size of logical address: 16(=)

Page number (=)

Page size (=)

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 0 | 1 | 0 |

Logical k =

Page table = 2->1

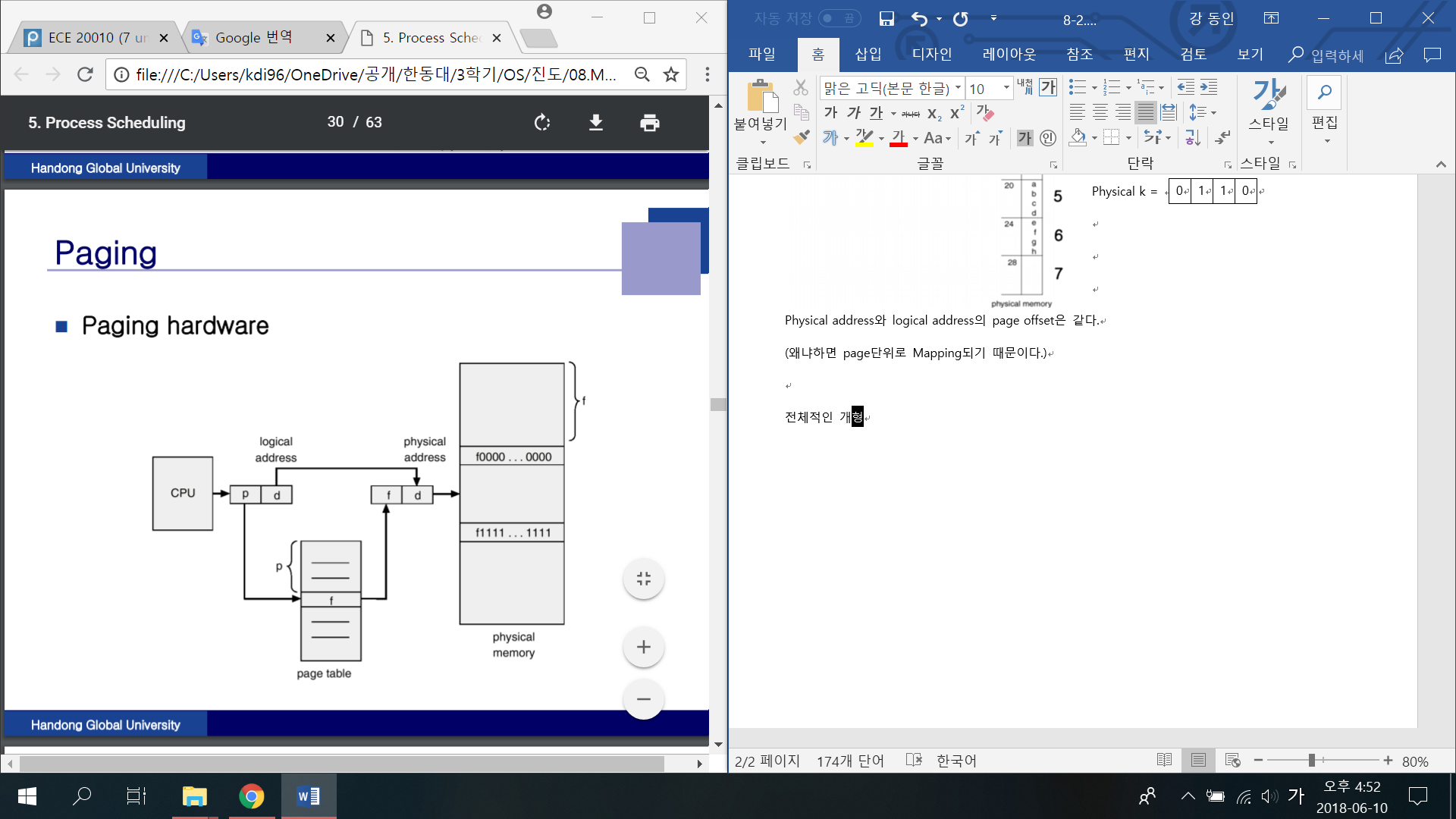
|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 1 | 0 |

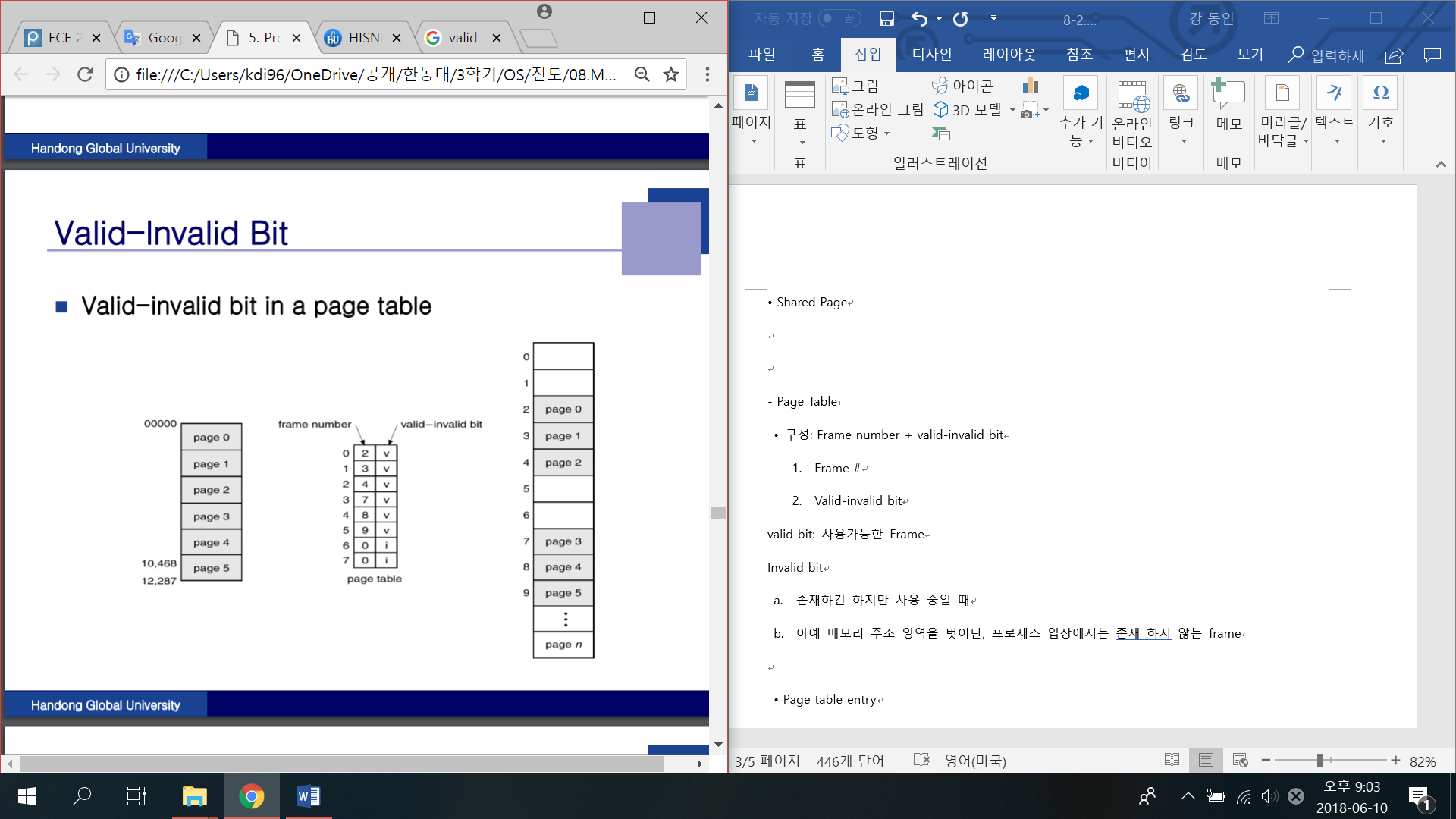
Physical k =

Physical address와 logical address의 page offset은 같다.

(왜냐하면 page단위로 Mapping되기 때문이다.)

전체적인 개형



- Page Table

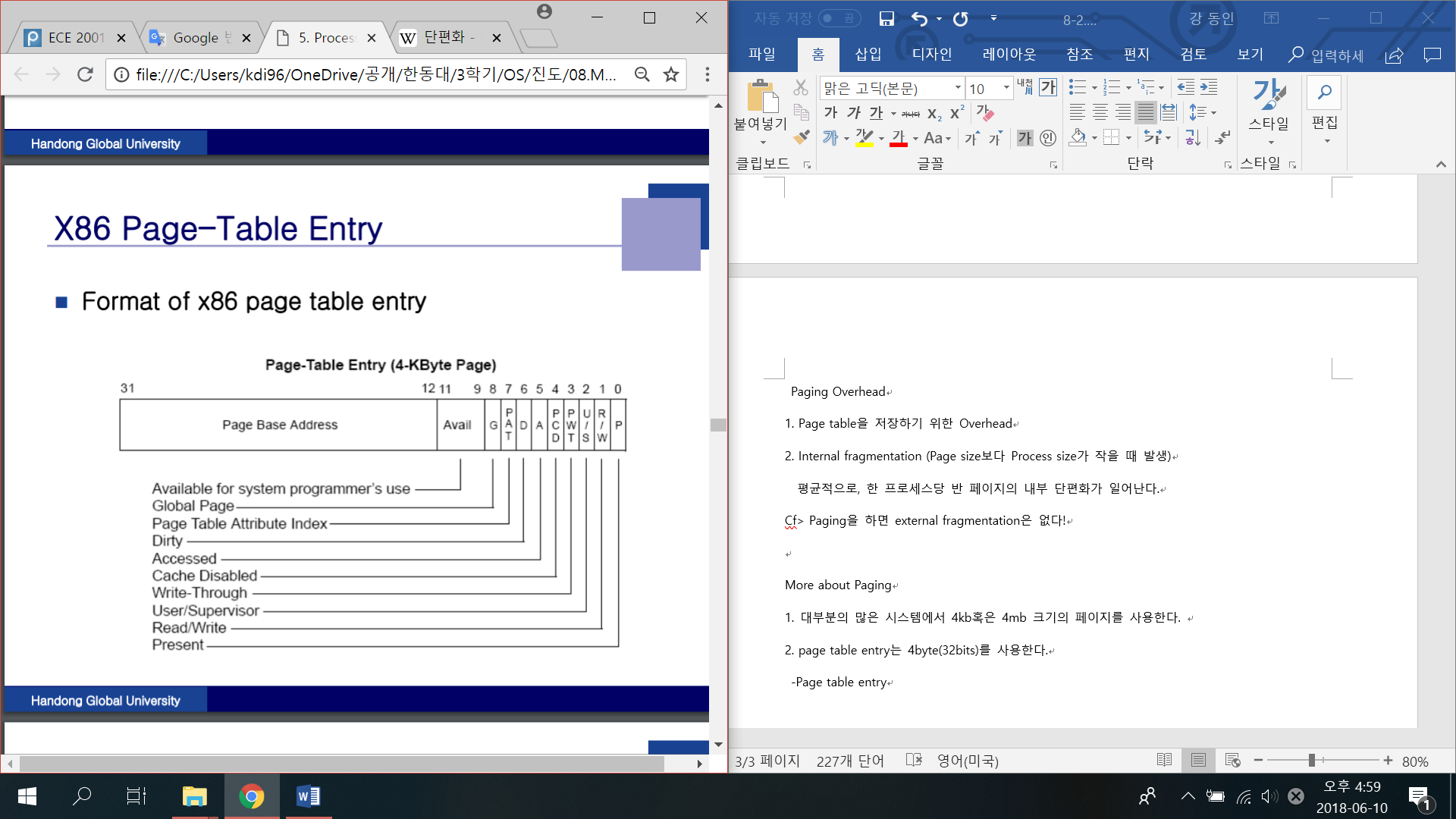
• 구성: Frame number + valid-invalid bit

1. Frame #
2. Valid-invalid bit

valid bit: 사용가능한 Frame

Invalid bit

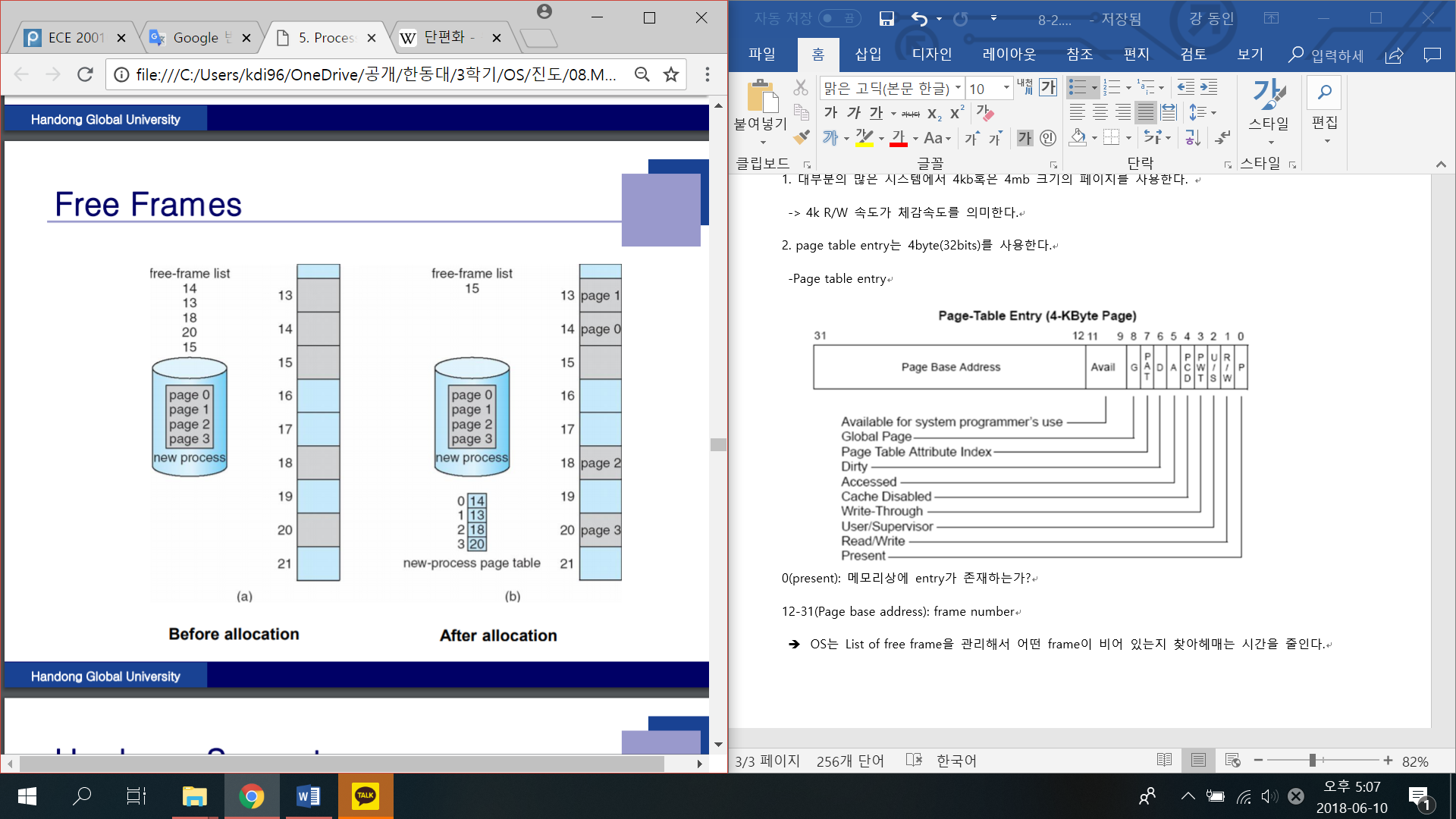
1. 존재하긴 하지만 사용 중일 때
2. 아예 메모리 주소 영역을 벗어난, 프로세스 입장에서는 존재하지 않는 frame

• Page table entry

0(present): 메모리상에 entry가 존재하는가?

12-31(Page base address): frame number

* OS는 List of free frame을 관리해서 어떤 frame이 비어 있는지 찾아 헤매는 시간을 줄인다.

※ Free Frame List

파란색: 할당됨

회색: Free-frame

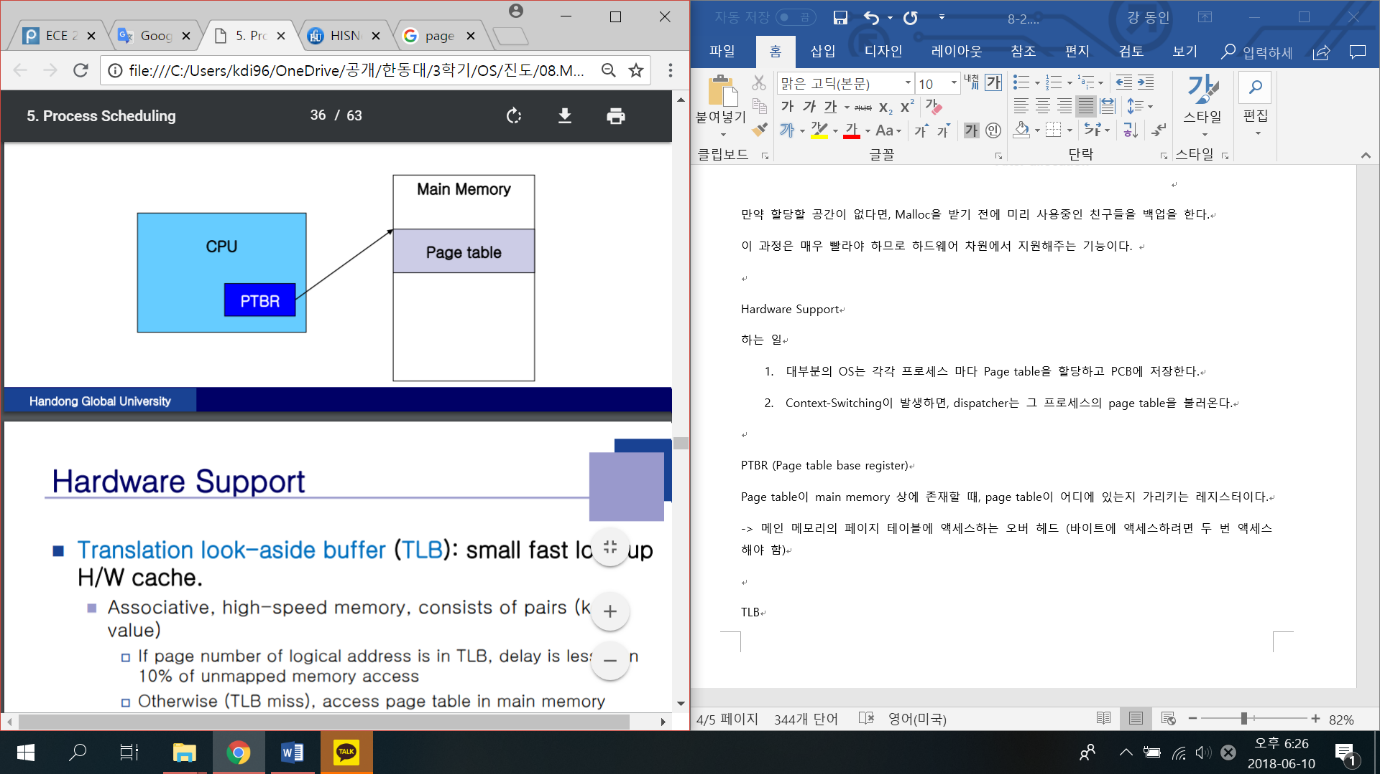
만약 할당할 공간이 없다면, Malloc을 받기 전에 미리 사용중인 친구들을 백업을 한다.

이 과정은 매우 빨라야 하므로 하드웨어 차원에서 지원해주는 기능이다.

• Implement by Hardware

1. 대부분의 OS는 각각 프로세스 마다 Page table을 할당하고 PCB에 저장한다.
2. Context-Switching이 발생하면, dispatcher는 그 프로세스의 page table을 불러온다.
3. PTBR (Page table base register)

Page table이 main memory 상에 존재할 때, page table이 어디에 있는지 가리키는 레지스터이다.

-> 문제점: 메인 메모리의 페이지 테이블에 액세스하고 페이지 테이블을 참조해 physical address를 찾는다. 즉, 2번의 엑세스로 오버헤드가 발생

1. TLB (Translation look-aside buffer)

-> PTBR의 문제점의 해결, 하드웨어 캐시를 빠르게 탐색.

Associative memory 혹은, 고속 메모리를 사용, (페이지 번호, 프레임 번호) 쌍으로 구성되어 있다.

- TLB hit: logical address의 page number가 TLB에 있는 경우

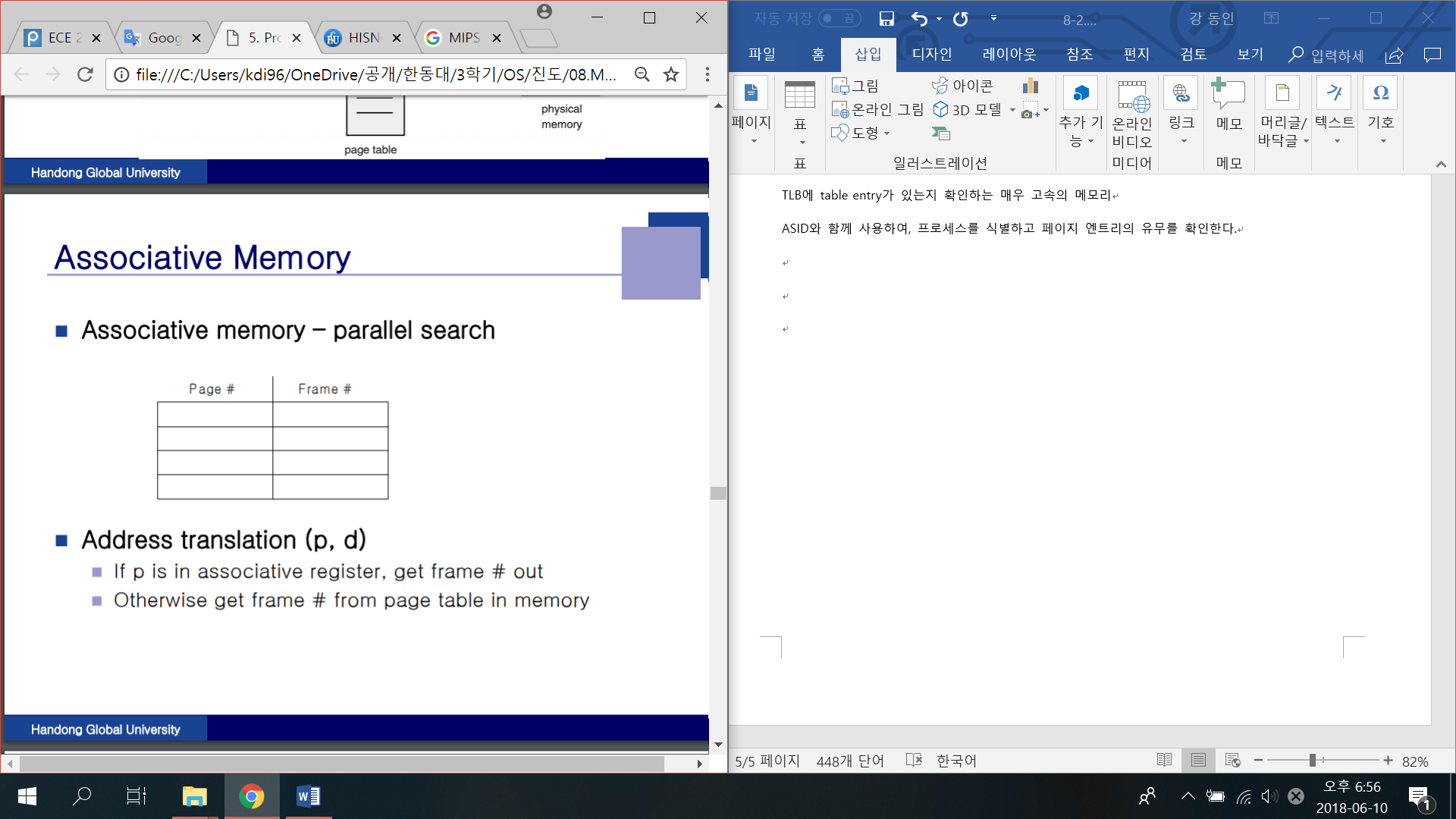
- TLB miss: Main memory에 있는 page table에 접근해야 한다.

※ ASID(Address-Space identifiers)

: 현재 로드 되어있는 Page table이 해당 프로세스의 Page table인지 식별해주는 정보

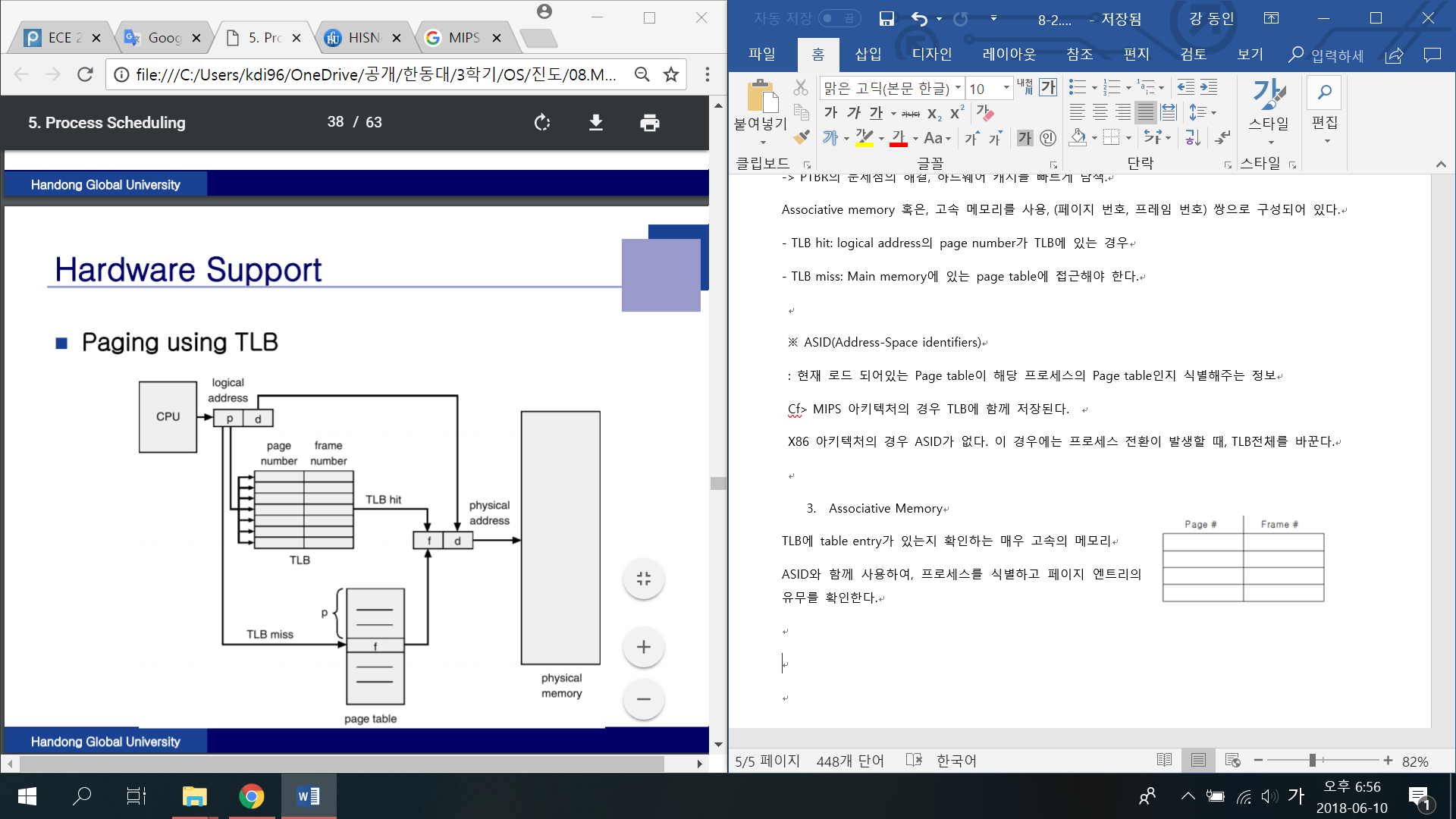
Cf> MIPS 아키텍처의 경우 TLB에 함께 저장된다.

X86 아키텍처의 경우 ASID가 없다. 이 경우에는 프로세스 전환이 발생할 때, TLB전체를 바꾼다.

1. Associative Memory

TLB에 table entry가 있는지 확인하는 매우 고속의 메모리

ASID와 함께 사용하여, 프로세스를 식별하고 페이지 엔트리의 유무를 확인한다.

TLB를 사용한 Paging

1. PTLR: Page Table Length Table, 페이지 테이블의 길이를 저장하는 레지스터

• 구조

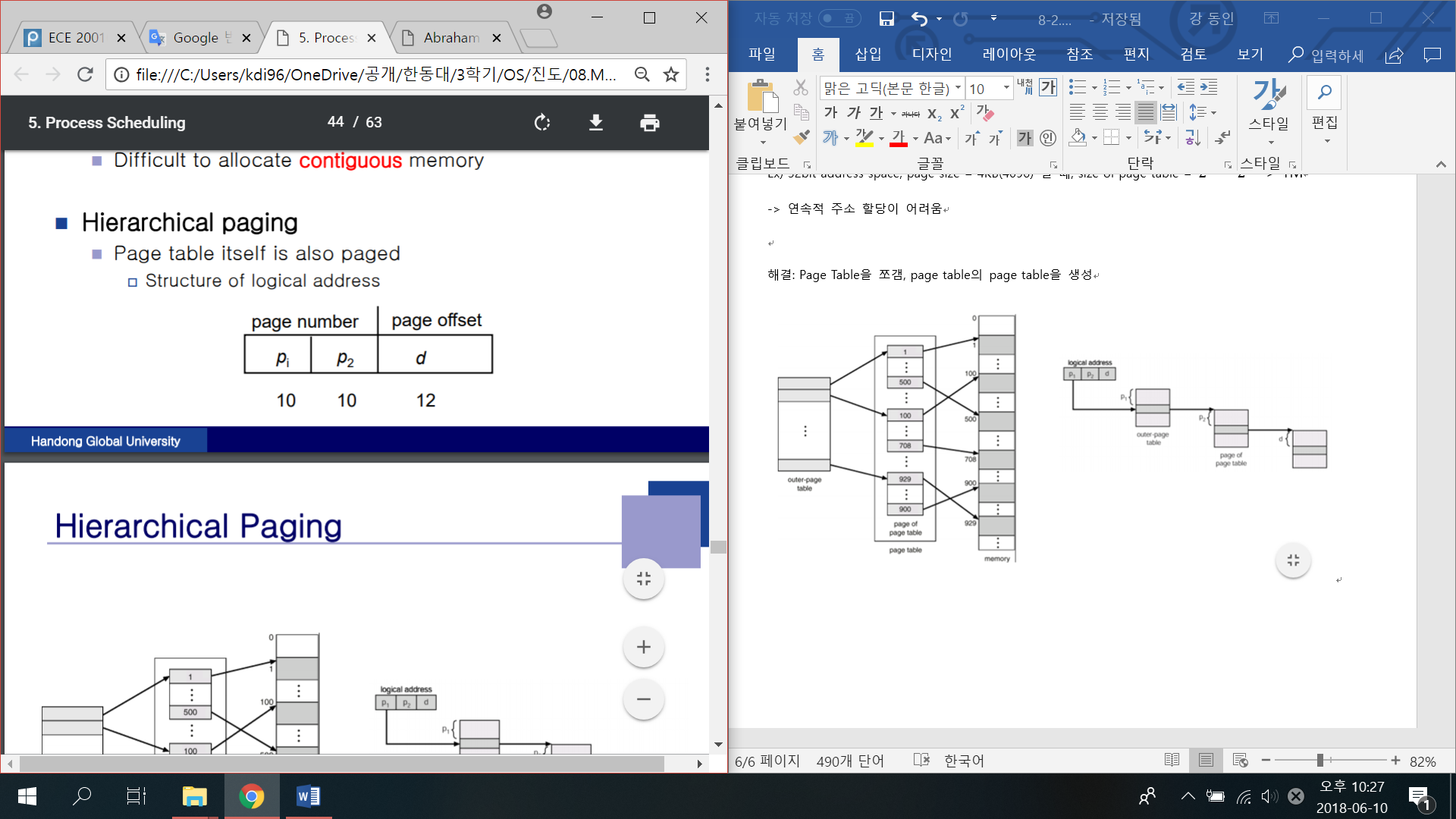
1. Hierarchical Paging

큰 logical address space를 지원할 때 Page table 자체가 Overhead가 된다.

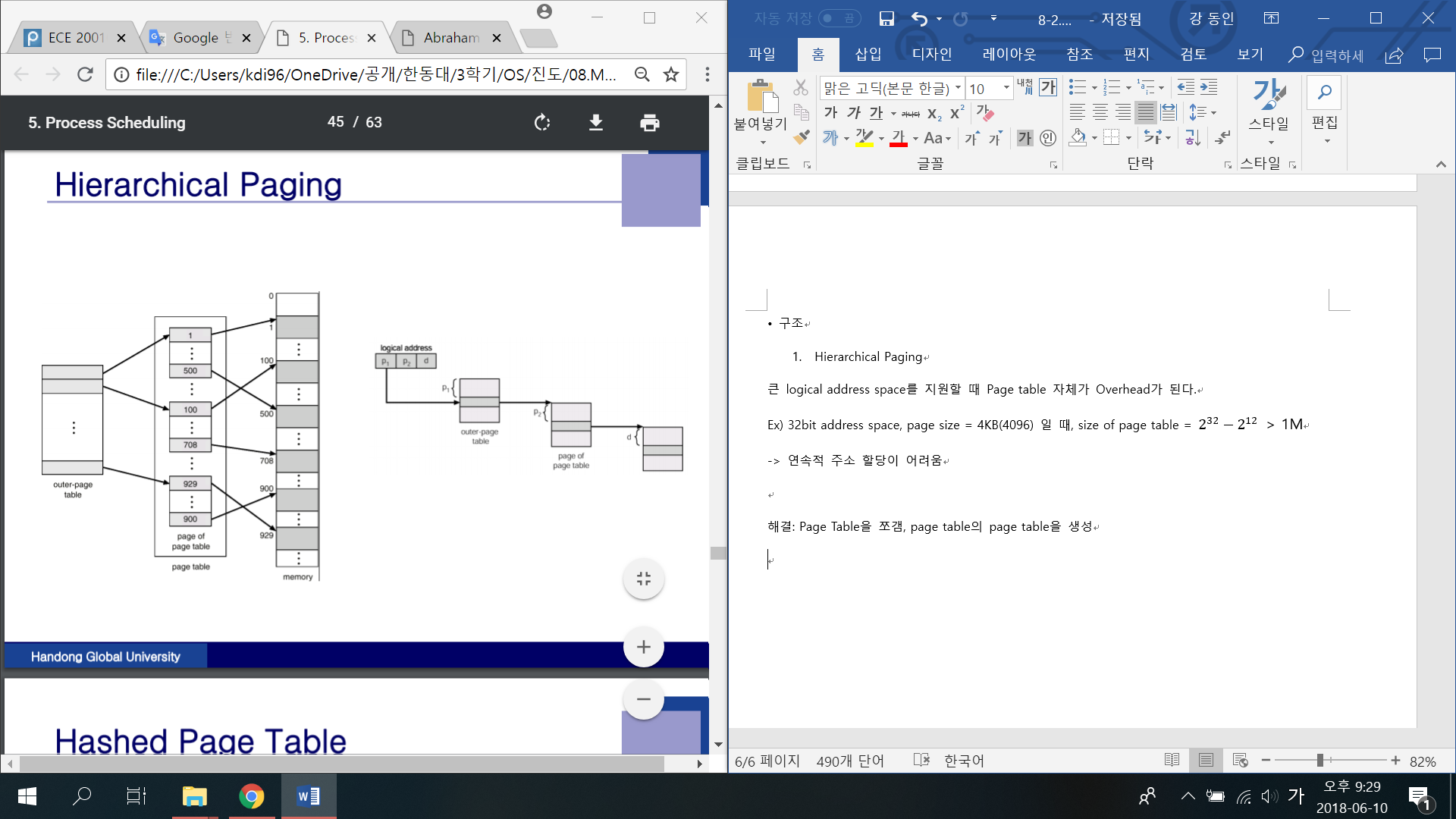
Ex) 32bit address space, page size = 4KB(4096) 일 때, size of page table = > 1M

-> 연속적 주소 할당이 어려움

해결: Page Table을 쪼갬, page table의 page table을 생성 (Tree의 구조)

p1: Outer-page table Index

p2: Page of Page table

변환 방법(forward-mapped page table)

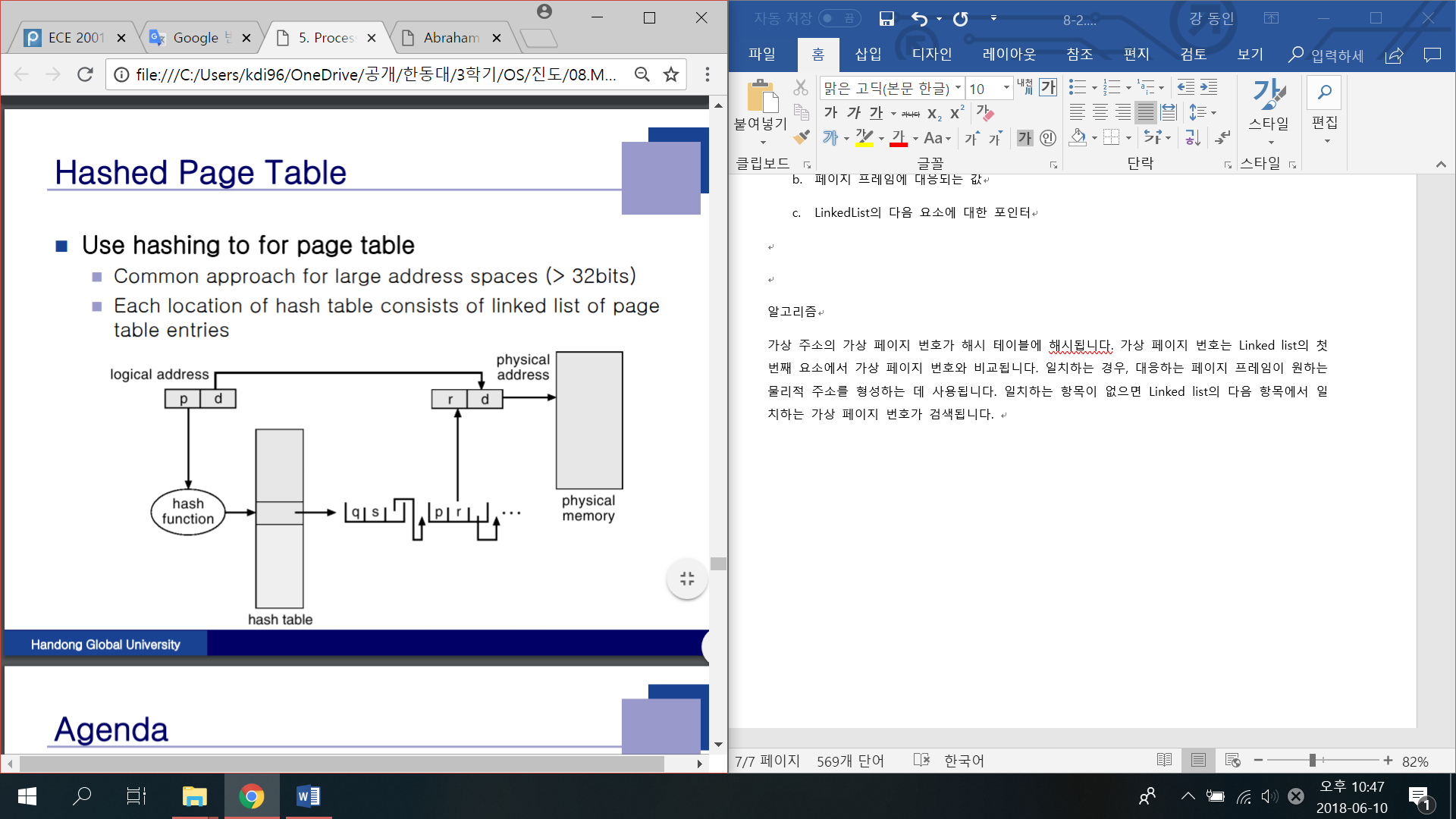
1. Hashed Page Table

큰 주소 공간에서 유용한 Paging 방식이다.

구성요소

1. 가상 페이지 번호
2. 페이지 프레임에 대응되는 값
3. LinkedList의 다음 요소에 대한 포인터

알고리즘

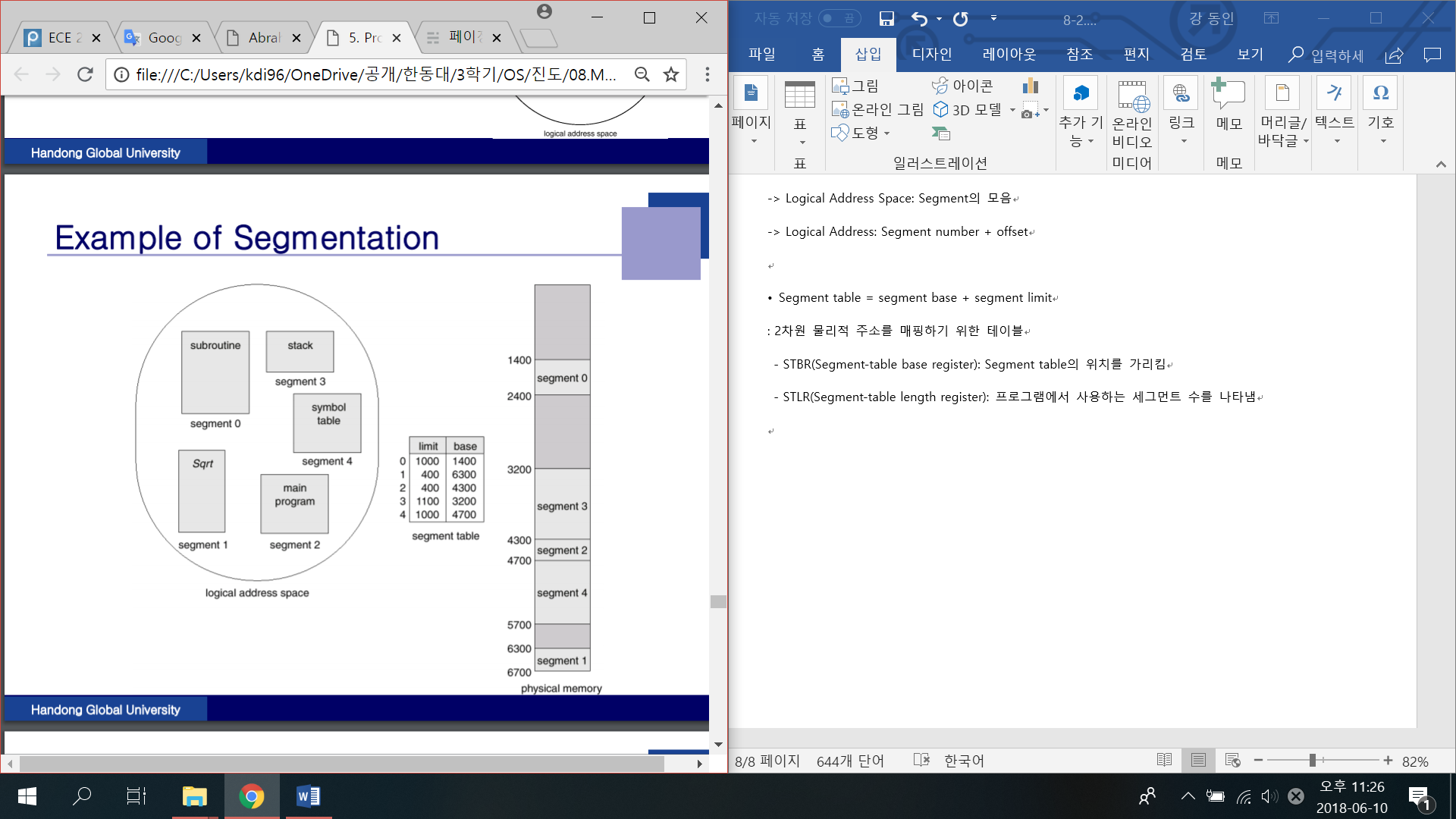
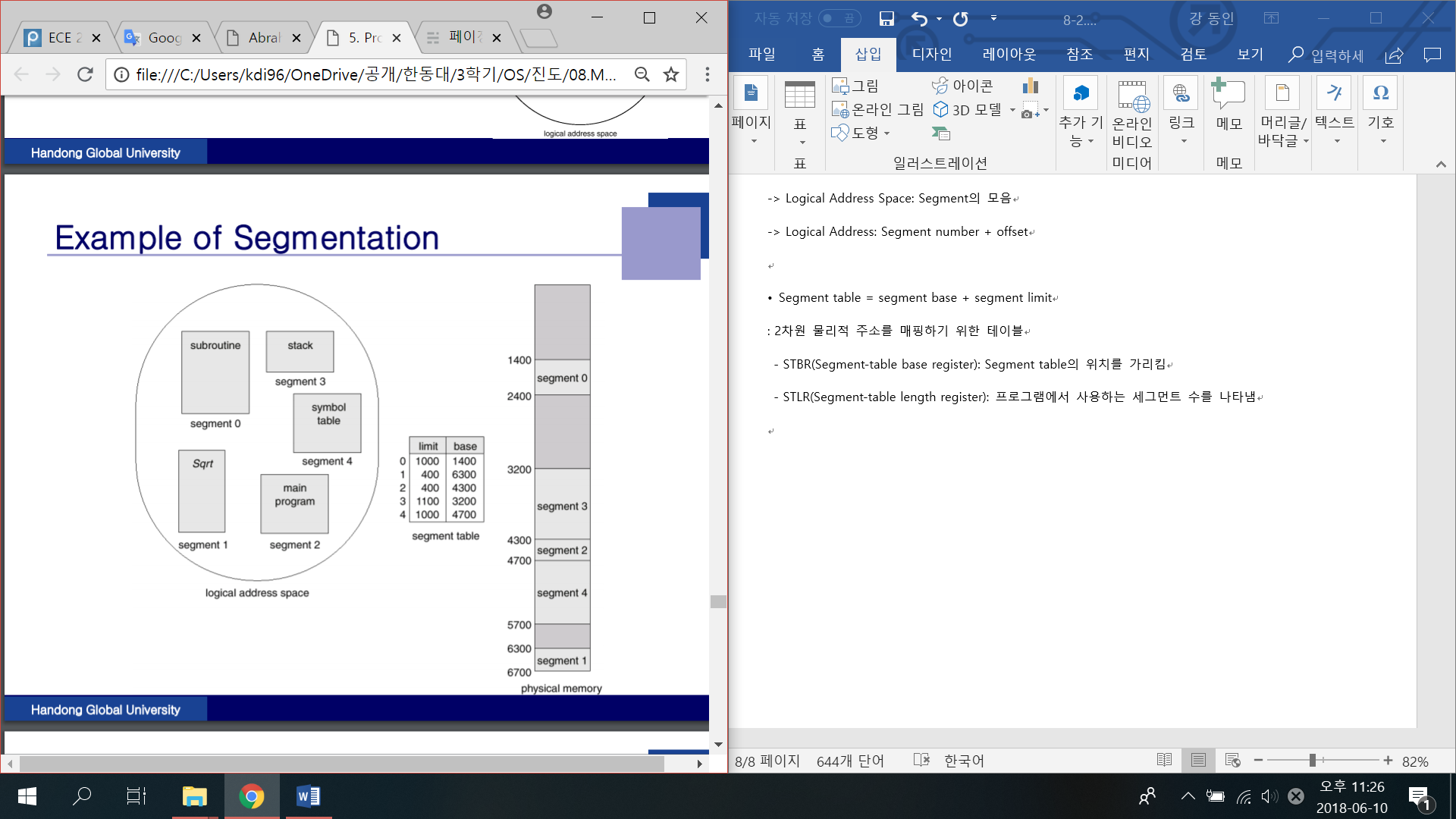
가상 주소의 가상 페이지 번호가 해시 테이블에 해시됩니다. 가상 페이지 번호는 Linked list의 첫 번째 요소에서 가상 페이지 번호와 비교됩니다. 일치하는 경우, 대응하는 페이지 프레임이 원하는 물리적 주소를 형성하는 데 사용됩니다. 일치하는 항목이 없으면 Linked list의 다음 항목에서 일치하는 가상 페이지 번호가 검색됩니다.

7. Segmentation

Logical memory와 Physical memory를 서로 다른 크기로 분할하는 기법

• Segment: 서로 다른 크기의 논리적 단위 ex) Global variable, heap, stack, standard library…

-> Logical Address Space: Segment의 모음

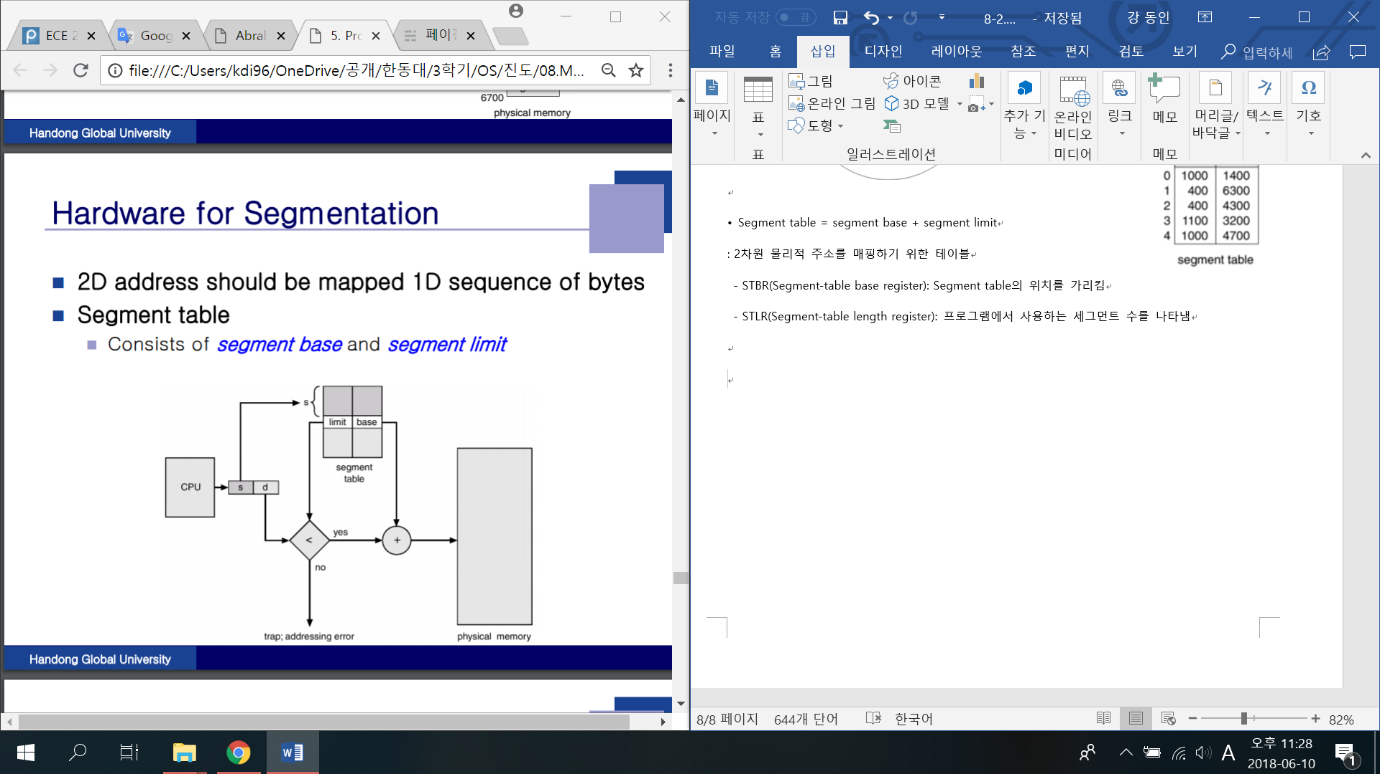
-> Logical Address: Segment number + offset

• Segment table = segment base + segment limit

: 2차원 물리적 주소를 매핑하기 위한 테이블

- STBR(Segment-table base register): Segment table의 위치를 가리킴

- STLR(Segment-table length register): 프로그램에서 사용하는 세그먼트 수를 나타냄

변환 과정

Paging과 마찬가지로

Protection, Sharing이 모두 가능

요즘은 Paging과 Segmentation을 같이 사용하는

Paged Segmentation 기법 사용

Ex) Intel Pentium – Intel x86

**CPU -------> Segmentation unit --------->Paging unit**

Logical address: = ()

- Segment Offset: Maximum size of segment: 4GB (bit)

- Segment number(Selector):

s. Maximum number of segments: 16K (=16384, bit)

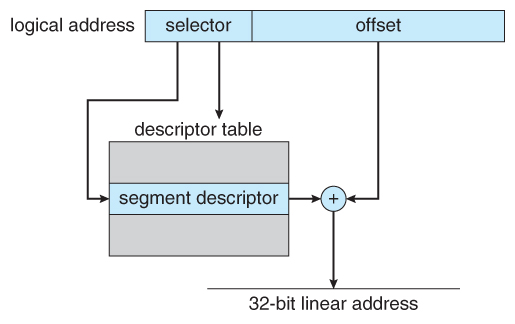
프로세스의 Logical address space를 두 개의 파티션으로 나눈다.

• 1번째 파티션(Private segments(8K, ): LDT (Local descriptor Table)에 보관

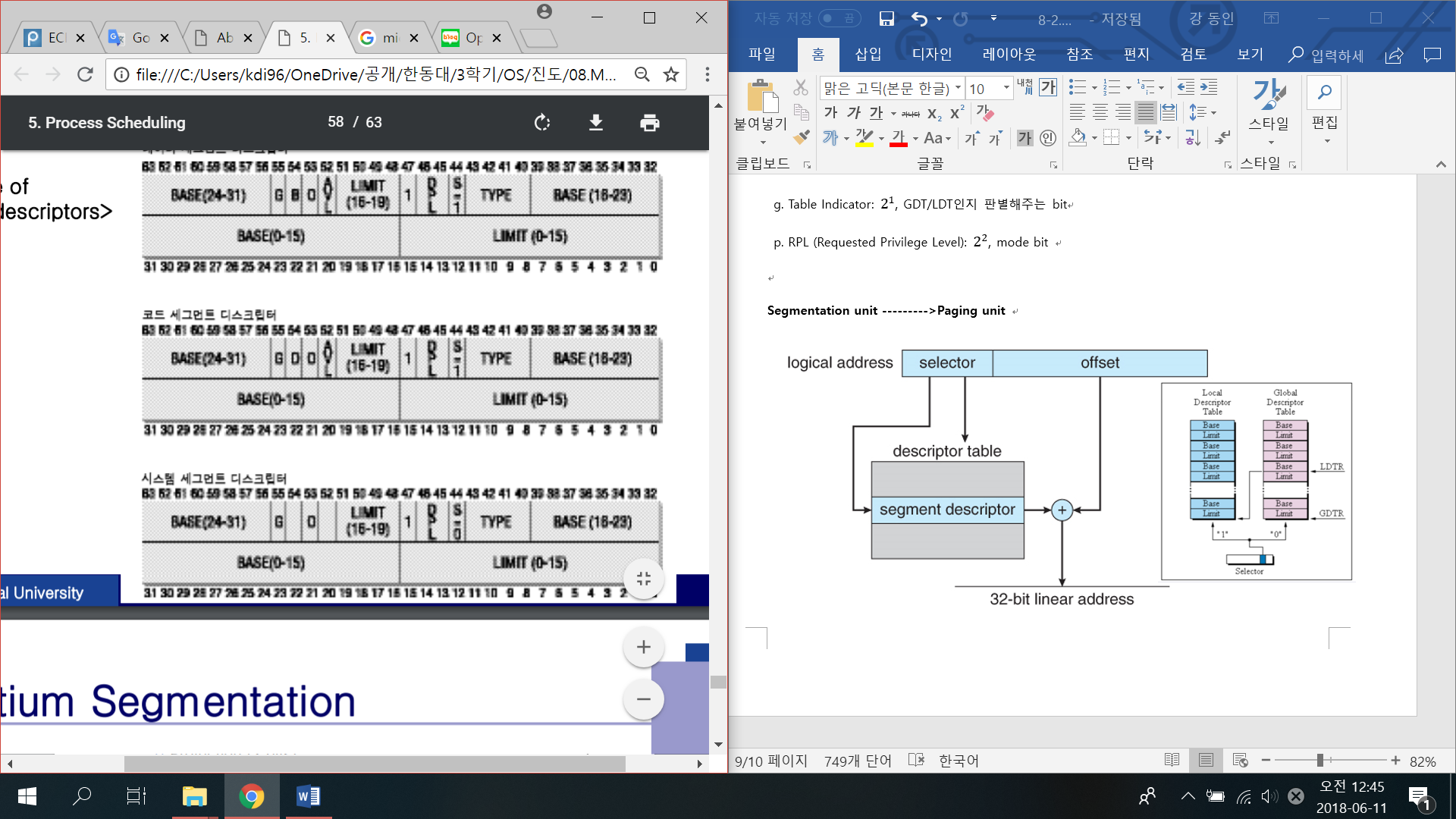
• 2번째 파티션(Shared segments(8K, ): GDT (Global descriptor Table)에 보관

g. Table Indicator: , GDT/LDT인지 판별해주는 bit

p. RPL (Requested Privilege Level): , mode bit

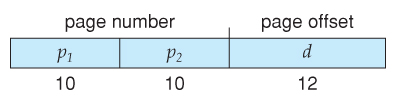
**Segmentation unit --------->Paging unit**

Segment descriptor

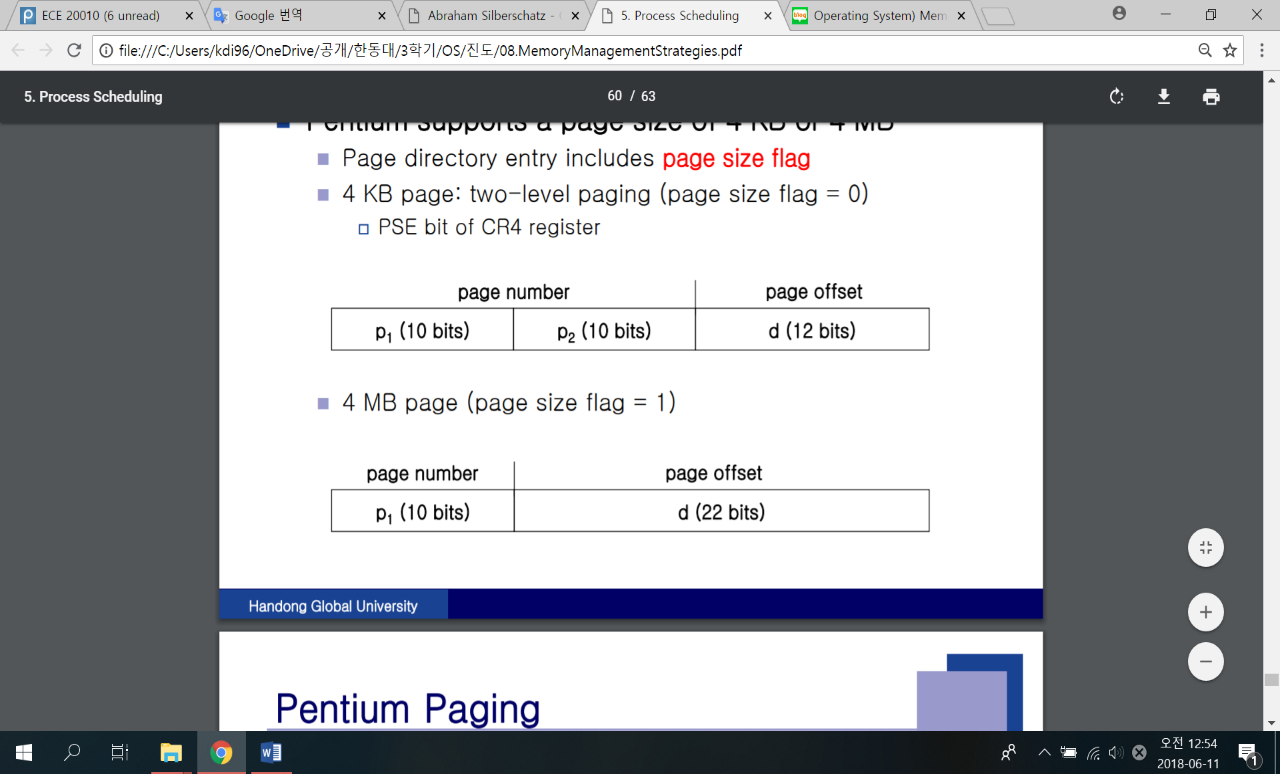
6개의 segment register 가 존재

Microprogram register

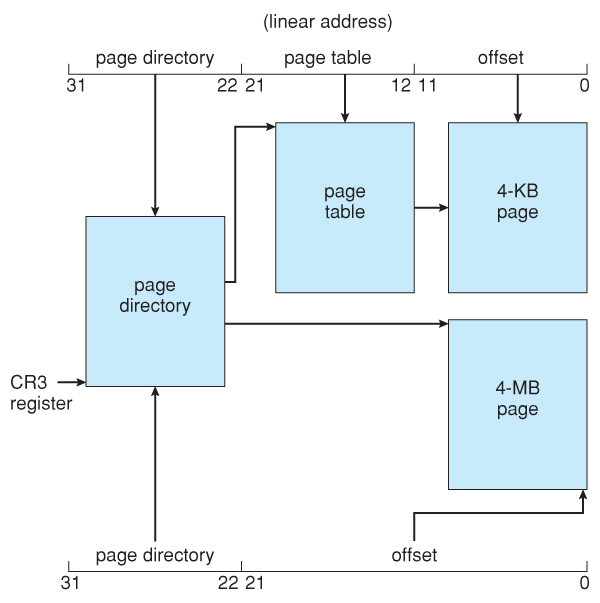
: Cache for LDT/GDT entry

* Pentium Paging

Pentium은 Page size를 2종류를 지원한다.

1. 4KB Page: two-level paging
2. 4MB Page

- Page size flag: Page directory에 포함된 엔트리로 0이면 4KB page, 1이면 4MB page를 지원한다.

총 메커니즘