8. Memory Management (2)

2023.07.06 **박영서, 황재천, 김하림**

Index

01 Multilevel Paging and Performance

02 Memory Protection

03 Inverted Page Table

04 Shared Page

05 Segmentation

06 Segmentation with Paging

01 Multilevel Paging and Performance

- Paging을 사용했을 때의 퍼포먼스 계산
  - TLB hit ratio ? 찾길 원하는 페이지 번호를 가지고 TLB에 접근했을 때, 찾는 경우의 비율
  - effective memory-access time ? 평균 메모리 접근 시간

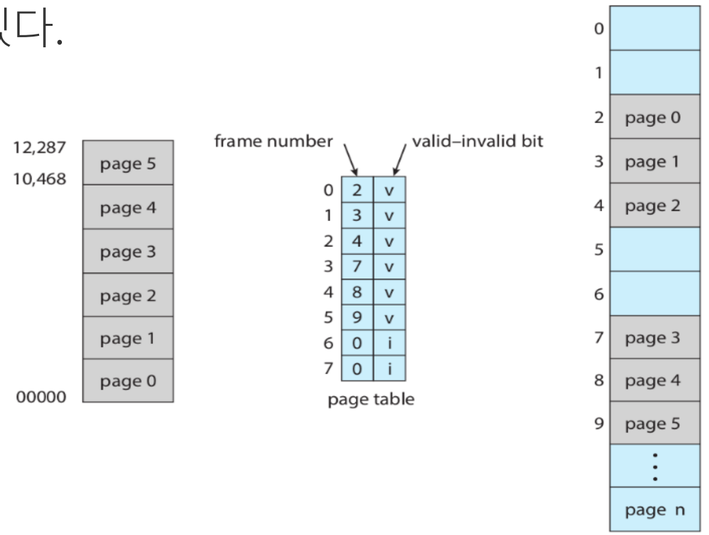
예) hit ratio = 80%, page table에 접근 10ns, 물리 메모리 접근 10ns

0.80 * 10(hit) + 0.20 * 20(miss)

- Address space가 더 커지면 다단계 페이지 테이블이 필요
  - 각 단계의 페이지 테이블이 메모리에 존재하므로 주소 변환에 더 많은 메모리 접근이 필요함
  - TLB를 통해 메모리 접근 시간을 줄일 수 있음
  - 성능 계산은 더 복잡해짐

02 Memory Protection

- Paged Environment에서의 Memory Protection은 페이지 테이블에서 **protection bits**를 두어 이루어진다.
  - 해당 페이지의 접근 권한을 표시하는 비트 (하림 : Valid/Invalid bit 외의 다른 bit에는 뭐가 있나요?)
    - 이 페이지의 접근 권한이 read-write인지, read-only를 정한다.
      - read-only 페이지에 쓰려고 하면 OS에 하드웨어 trap을 발생시킨다.
    - 위를 확장시켜 read-write/read-only/execute-only로 접근 권한을 구분할 수도 있다.
      - 각 접근 유형에 1bit씩을 줘서, 접근 권한의 조합을 가능하게 할 수도 있다.
  - Valid-Invalid bit
    - 이 페이지에 대한 접근이 유효한 것인지 그렇지 않은지를 표시해주는 비트
    - 한 프로세스가 사용할 수 있는 전체 주소공간을 사용하는 경우는 드물다.
      - 페이지 테이블의 엔트리는 프레임 수만큼 만들어져야 한다. (n개)
      - 이 중 쓰이지 않는 부분에의 접근을 막기 위해 사용 (0 ~ 5를 제외한 것들)



03 Inverted Page Table

- 보통은 프로세스들이 각자의 페이지 테이블을 가진다.
  - 프로세스가 가진 logical address를 physical address로 변환해야하기 때문에 자연스럽다.
  - 각 프로세스의 page table이 가지는 수많은 entry가 physical memory에 담겨 비효율적이다.

- Inverted Page Table
  - 반대로 physical memory를 기준으로 page table을 구성한다.
  - system-wide하게 하나의 page table만이 존재한다.
  - 이 table은 physical memory의 frame 수만큼의 entry를 가진다.
    - page frame 하나에 page table의 한 entry를 둔다.
    - 각 page table의 entry는 {pid, page number} 쌍으로 이루어져있다.

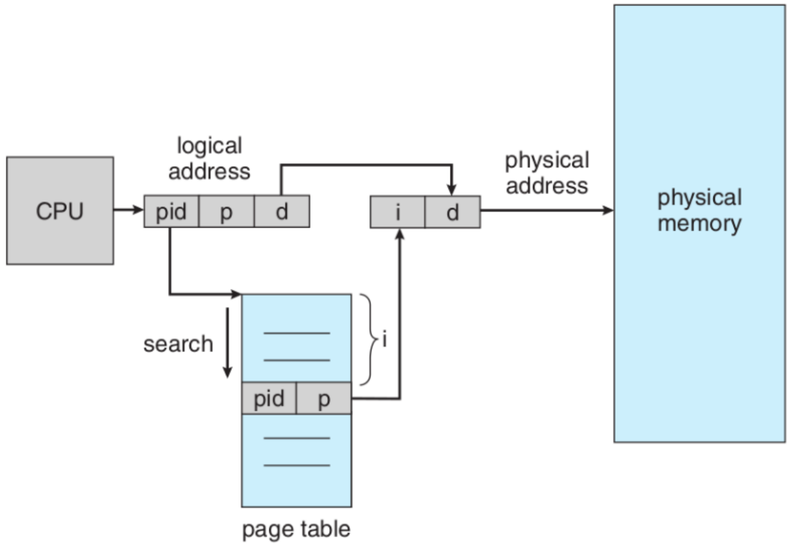


Figure 9.18 Inverted page table.

03 Inverted Page Table

- 주소 변환
  - logical address는 {pid, page number p, offset d}로 이루어진다.
  - table을 모두 뒤져보며 {pid, p} 쌍을 찾는다. 이것의 index i가 physical memory의 frame 번호가 된다.
    - 여기서 p는 logical address이므로 서로 다른 프로세스라도 같을 수 있다.
    - 따라서 pid를 함께 이용해 어떤 프로세스가 찾고자 하는 p인지를 알아야 한다.
- 장단점
  - 하나의 page table만 있으면 되므로 공간적으로 이득이다.
  - table 전체를 뒤져야하므로 시간적인 오버헤드가 있다.
  - Shared memory가 있는 경우 사용할 수 없다.
    (한 physical address에는 한 entry만이 대응하고, 여기에는 pid가 명시되어 있기 때문에)

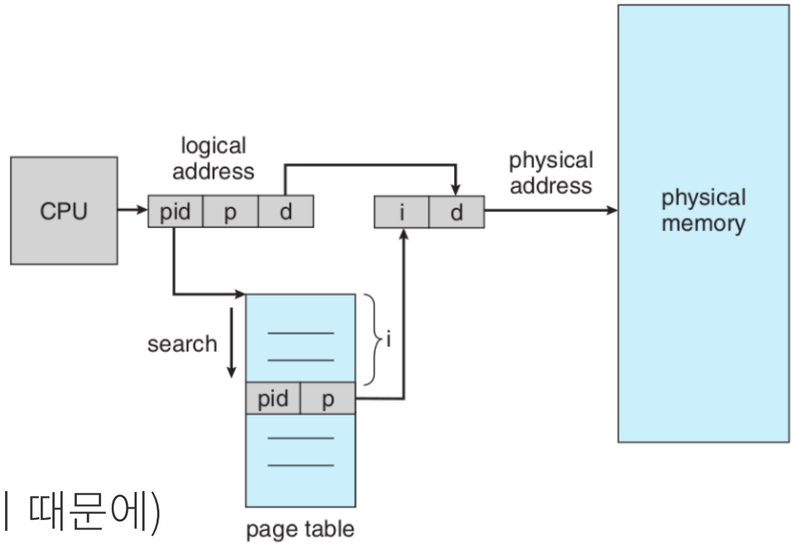
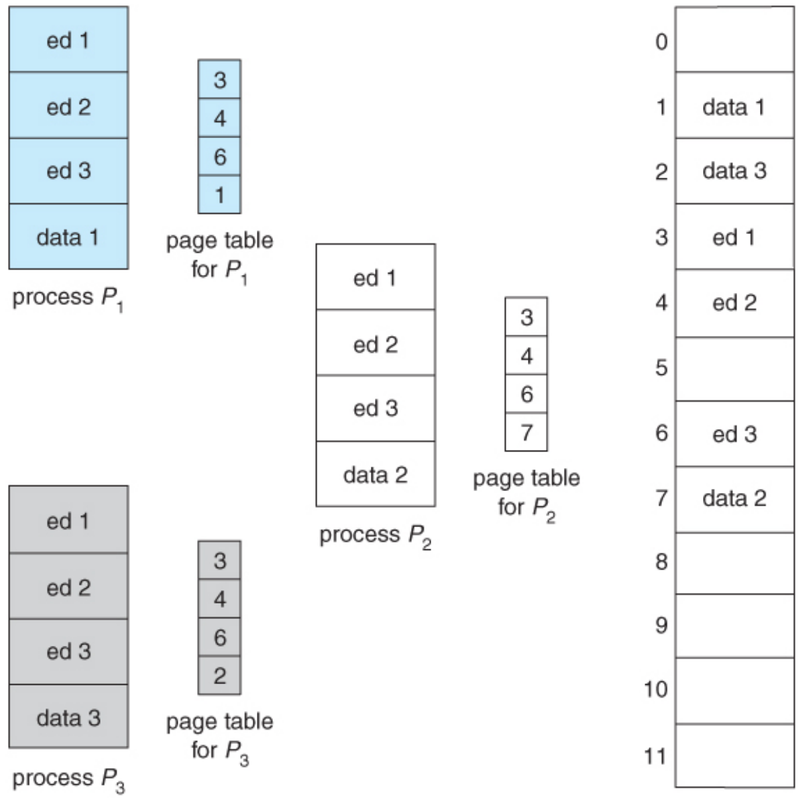


Figure 9.18 Inverted page table.

04 Shared Page

- **Shared Page** 조건(=Re-entrant, pure code)
  1. Shared code(=Re-entrant, pure code)가 read-only여야 한다.
  2. 하나의 code만 메모리에 올린다.
  3. Shared code는 모든 프로세스에서 logical address가 동일해야 한다.
     - ex. text editors, window systems, compilers

- Private code and data(↔ Shared code)
  - 각 프로세스들은 독자적으로 메모리에 올림
  - Private data는 logical address가 달라도 무방



04 Shared Page

- Shared Page가 사용되는 상황

1. Shared Library

   - 여러 프로세스가 동일한 라이브러리를 사용하는 경우

   ex) 여러 프로세스가 동일한 텍스트 에디터를 실행한다면, 텍스트 편집기의 실행 코드는 메모리에서 공유되고
   각 프로세스의 개별 데이터만 별도로 할당됨.

2. Shared Data Structure

   - 여러 프로세스가 동일한 데이터 구조(공유 메모리, 공유 파일 등)를 사용하는 경우

04 Shared Page

- Paging에서는 Page 단위로 메모리를 관리하고 할당하기 때문에 한 페이지 안에 코드 섹션과 데이터 섹션이 함께 들어갈 수 있다.

- Paging에서 어떻게 shared code가 결정되는지에 대해서는 명확한 규칙은 없음

- Shared code 결정은 운영 체제나 링커(linker) 등의 구현에 따라 다를 수 있음.

- Shared code의 결정은 운영 체제의 정책과 알고리즘에 의해 이루어짐
  - ex 1) PTE에 공유 속성을 추가하여 여러 프로세스 간에 동일한 페이지를 공유하도록 할 수 있음
  - ex 2) 링커가 실행 파일을 생성할 때 shared code를 특별한 섹션에 배치하여 공유할 수 있도록 할 수 있음

05 Segmentation

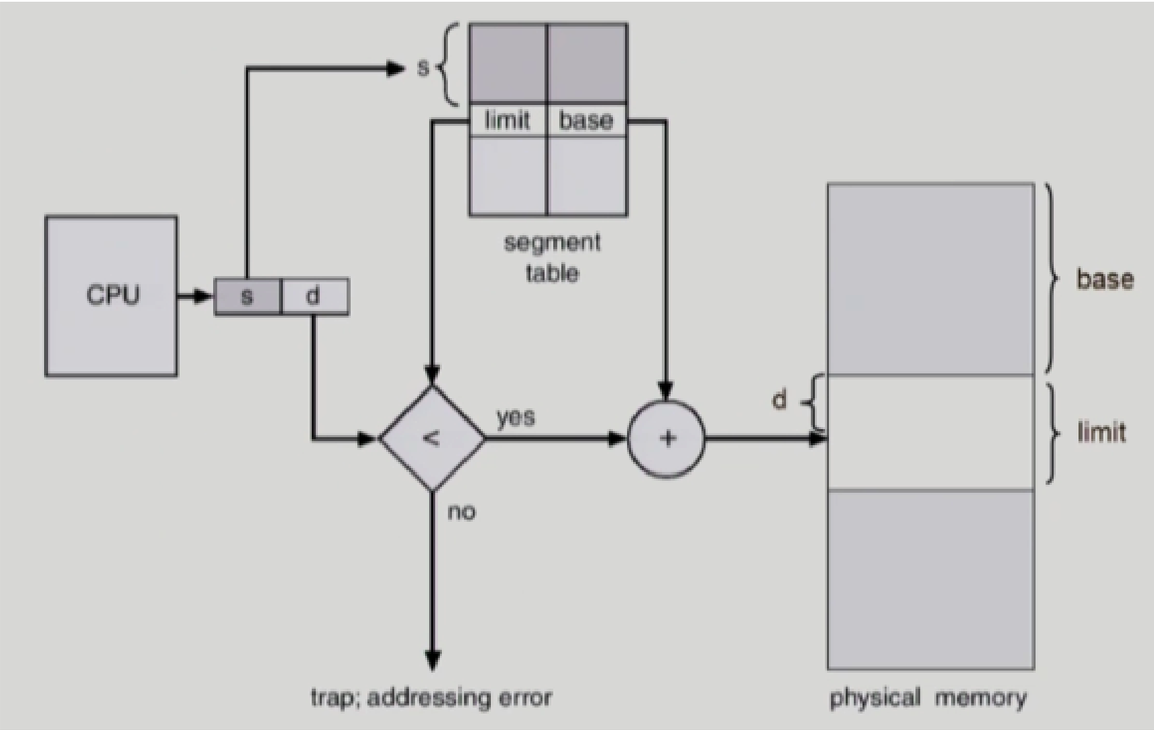
- 프로세스를 구성하는 주소공간을 의미단위로 자르는 기법
- 프로그램은 의미 단위인 여러 개의 segment로 구성
  - 작게는 프로그램을 구성하는 함수 하나하나를 세그먼트로 정의
  - 크게는 프로그램 전체를 하나의 세그먼트로 정의 가능
  - 일반적으로 code, data, stack부분이 하나씩의 세그먼트로 정의됨

05 Segmentation Architecture

- Logical address는 다음의 두 가지로 구성
  - segment-number, offset
- Segment table

  세그먼트별로 서로 다른 물리적인 메모리에 올리기 위해 세그먼트별로 주소변환이 필요함
  - each table entry has : base, limit
- Segment-table base register(STBR)
  - 물리적 메모리에서의 segment table의 위치
- Segment-table length register(STLR)
  - 프로그램이 사용하는 segment의 수
    - segment number ***s*** is legal if ***s*** < STLR

05 Segmentation - Physical address 계산방법



- **Logical address**
  - s : Segment number - 20bit
  - d : displacement(offset) - 12bit

1. s 를 통해 Segment Table Entry 찾음
2. Segment Length(limit)가 d(offset)보다 크면 통과(valid)
3. STE의 base + d를 통해 physical address 찾음

재첩국꺼

05 Segmentation Architecture(Cont - Strengths)

- Protection
  - 의미단위로 쪼개기 때문에 의미단위로 일을 할 때는 매우 좋다!
  - 각 세그먼트 별로 protection bit가 있음
  - Each entry
    - Valid bit = 0 ⇒ illegal segment
    - Read / Write / Execution 권한 bit
- Sharing
  - Shared segment
  - same segment number, same role

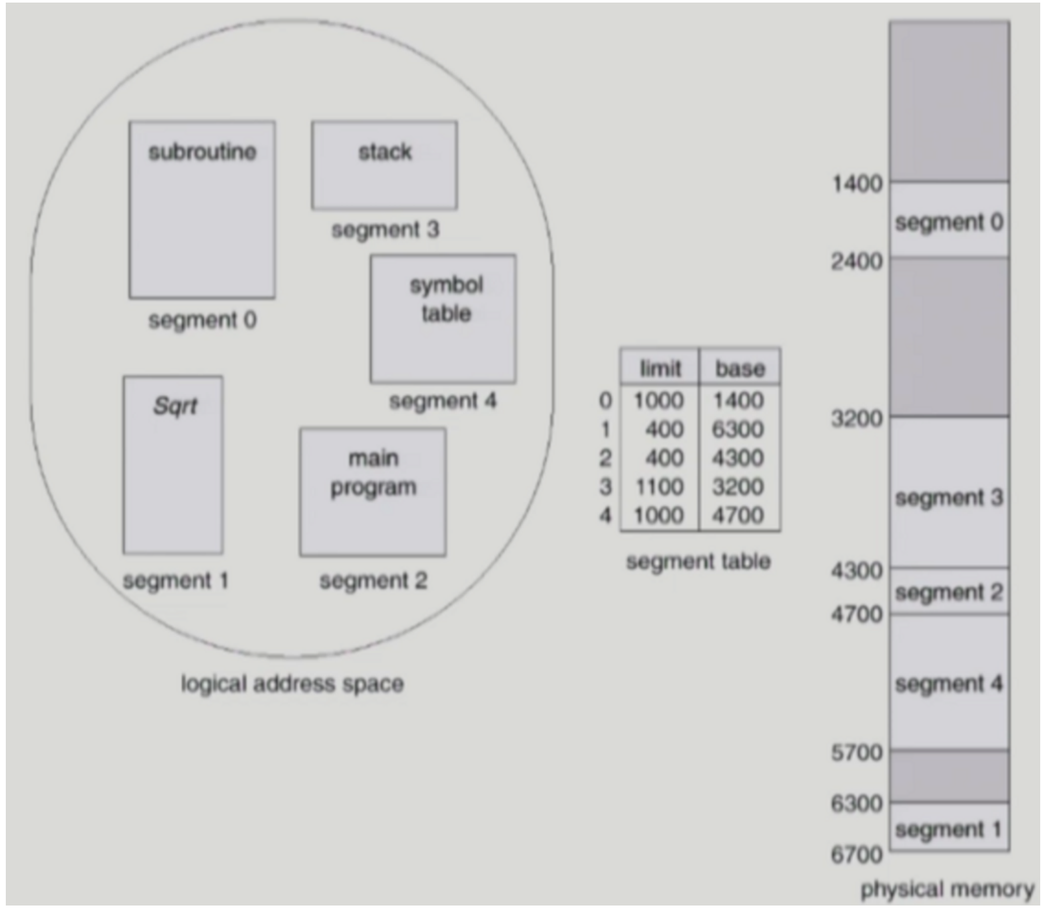
segment는 결국 의미 단위이기 때문에 공유(sharing)와 보안(protection)에 있어 paging보다 효과적

05 Segmentation Architecture(Cont - Weaknesses)

- Allocation
  - first fit / best fit
  - external fragmentation(외부 조각) 발생

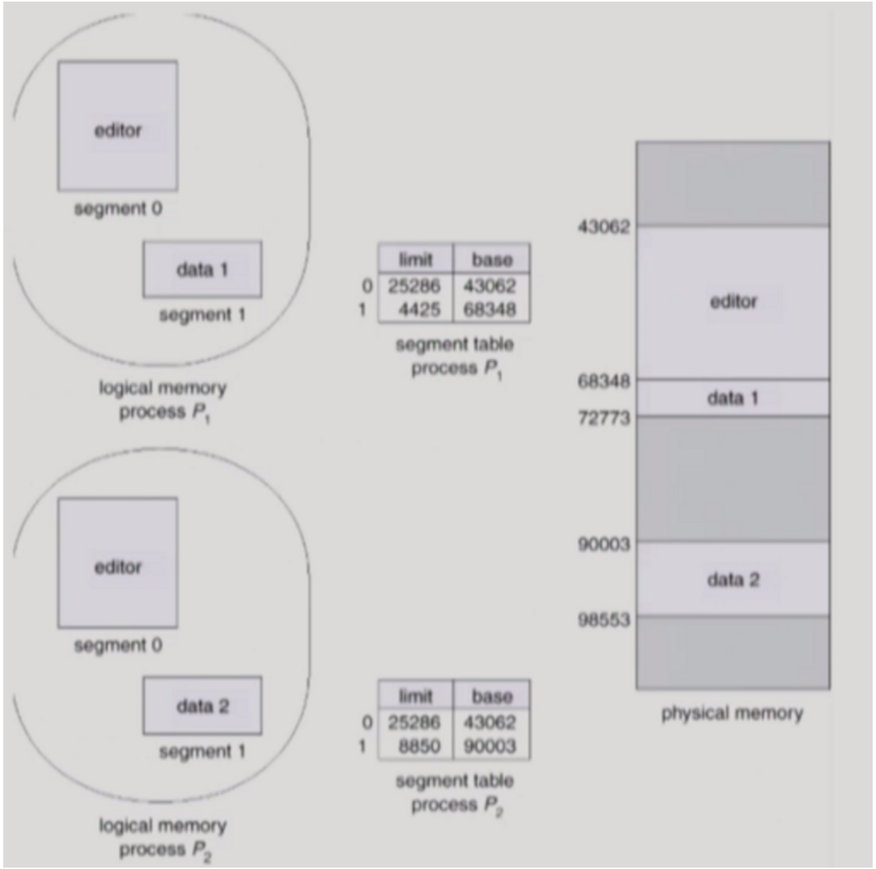
segment의 길이가 동일하지 않아 가변 분할 방식에서와 같은 문제점 발생

05 Example of Segmentation



필요시 재첩국이 함

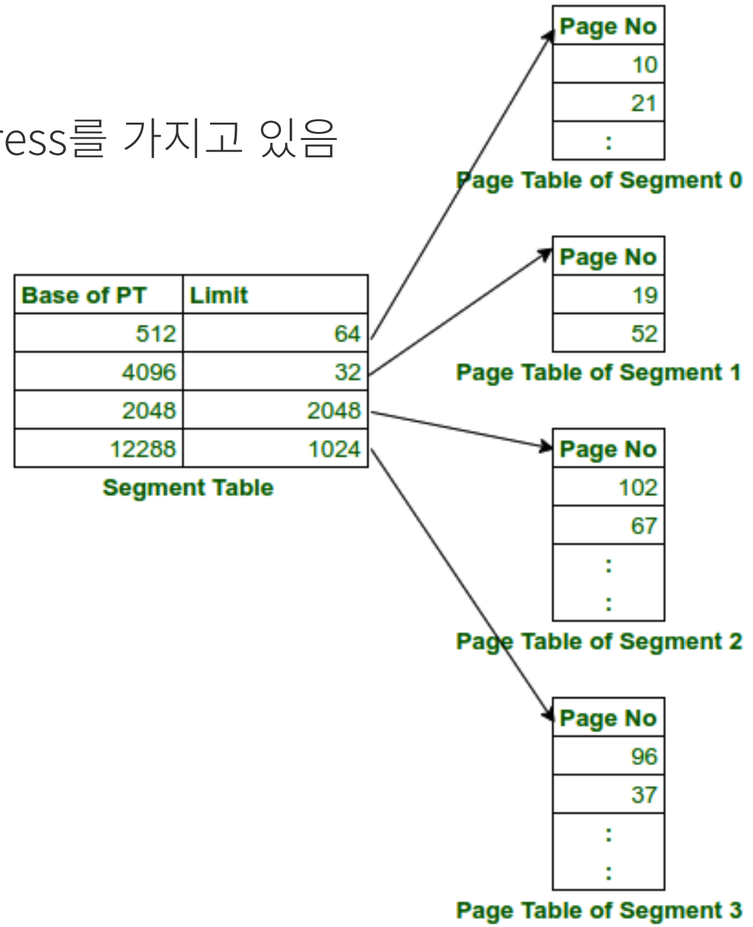
05 Sharing of Segments



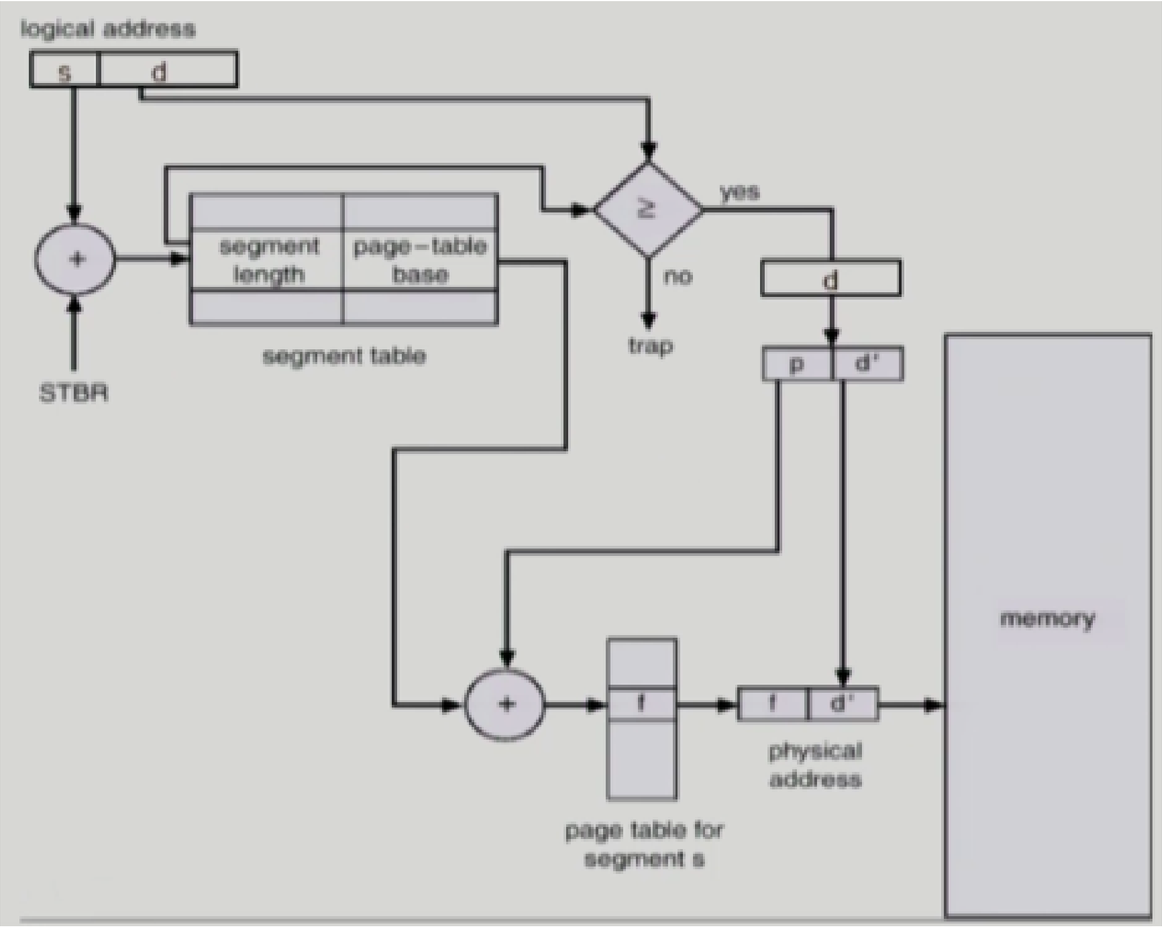
필요시 재첩국이 함2

06 Segmentation with Paging

- **Pure segmentation과의 차이점**
  - Segment-table entry가 segment를 구성하는 Page Table의 base address를 가지고 있음
- Segment offset(12bit)이 segment length 이내일 경우만 valid
- Outer Table이 Segmentation, Inner Table이 Paging 기법 사용



06 Segmentation with Paging - Physical address 계산방법



- **Logical address**
  - s : Segment number
  - d : displacement(offset)
  - p : Page number
  - d' == d (if valid)
  - f : Frame number

1. STBR + s 를 통해 Segment Table Entry 찾음
2. Segment Length가 d(offset)보다 크면 통과(valid)
3. Page Table Base + p를 통해 Page Table Entry 찾음
4. PTE의 f 와 d'을 합치면 physical address