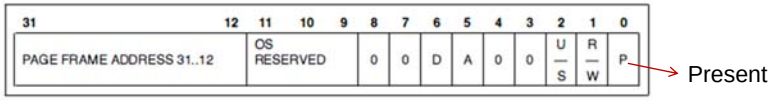
메모리 보호(Memory Protection)

- 메모리 보호를 위해 page table entry에 protection bit와 valid bit 추가
- Protection bits
  - read-write / read-only / executable-only 정의
  - page 단위의 memory protection 제공
- Valid bit (or valid-invalid bit)
  - V=1 (valid): legal page (페이지가 프로세스의 논리 주소공간에 있음)
  - V=0 (invalid): illegal page

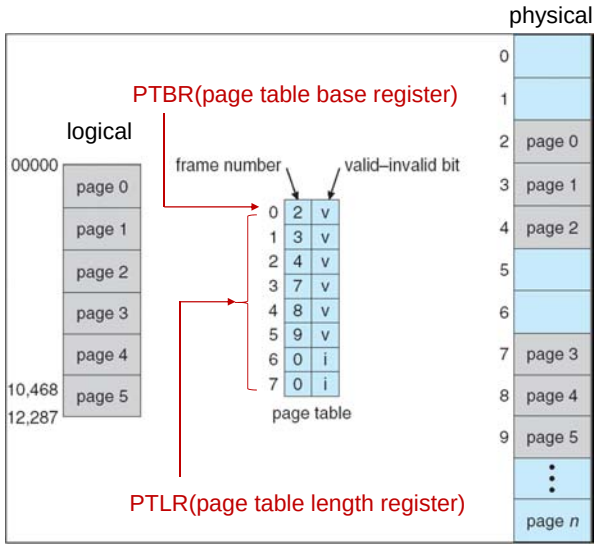
	frame no.	P	V
0		eo	1
1		eo	1
2		rw	1
3		ro	1
4		-	0

page table

i386 page table entry



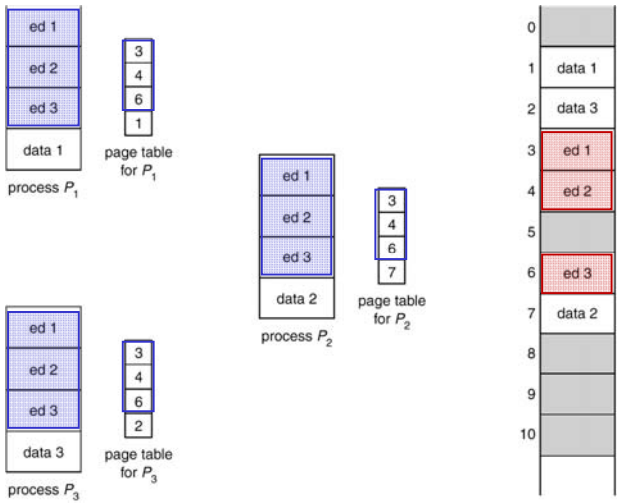
Page Table에서의 Valid/Invalid bit



공유 페이지(Shared Pages)

- paging의 장점 – 코드를 쉽게 공유 가능
- 공유 코드(Shared code)
  - **재진입 코드**(reentrant code)는 공유될 수 있음
    - 재진입 코드는 수행 동안 변하지 않는(non-self-modifying) 읽기전용(read-only) 코드임
  - 두 개 이상의 프로세스가 같은 코드를 동시에 실행할 때에
    - 물리적 메모리에 있는 한 개의 코드 공유
      - 각 프로세스의 페이지 테이블에서 같은 페이지 프레임 공유
    - 프로세스마다 별도의 데이터 저장공간 사용
- 공유 코드는 모든 프로세스의 논리 주소 공간에서 같은 위치에 있어야 함

Shared Pages Example

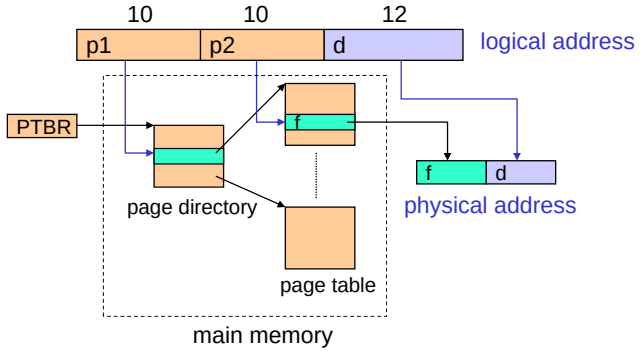


8.6 Page Table 구조

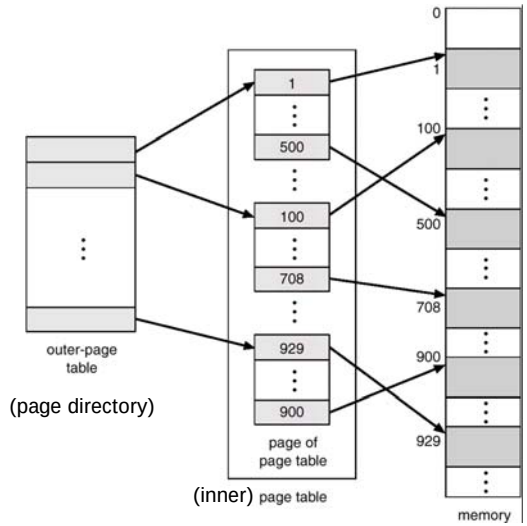
- Page Table의 크기
  - (예) 32비트 논리주소 공간, 4KB 페이지 크기
    - $2^{32} / 2^{12} = 2^{20}$ (1M) page table entries
    - page table entry크기가 32-bit(4B) 이면
      page table 크기는 $2^{20} * 4B = 2^{22}$ = 4MB → 1024 page 크기
  - 최신 컴퓨터시스템은 2^{32}~ 2^{64}정도의 커다란 논리주소 공간을 사용
    - page table의 크기가 상당히 커진다
    - 연속 메모리 공간을 사용하는 page table은 page size보다 훨씬 큼
- 해결책
  - Hierarchical Paging
  - Hashed Page Tables

Hierarchical Page table (Multilevel Paging)

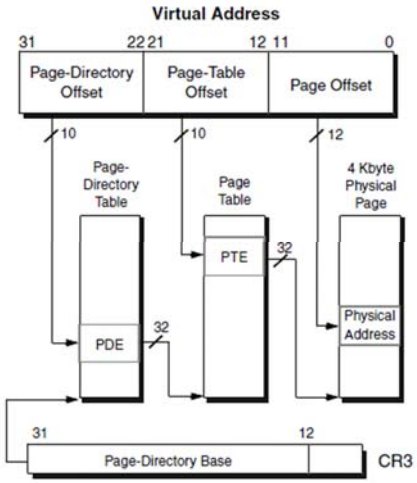
- 다단계 page table
  - page table을 하나의 연속적 메모리를 사용하지 않고
  - 여러 개의 작은 조각(대개 1 page 크기)으로 나누어서 계층적으로 구현



Two-Level Page-Table Scheme



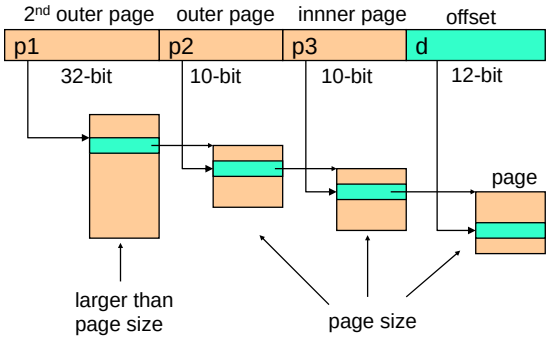
(Example) Two-level paging - 80386



- the page table is paged
  → the page number is divided into
  (1) a 10-bit index into page directory
  (2) a 10-bit index into page table
- page size = 2^{12} = 4KB,
- page table size =
  page directory size =
  2^{10} x 4B/entry = 4KB

Multilevel Paging

- Three-level paging 예



  - 64-bit logical address를 32-bit physical address로 변환하는 데 4번의 메모리 접근이 발생할 수 있음
- 64-bit architecture에서는 hierarchical page table이 부적합함

(Example) x86-64 long mode paging

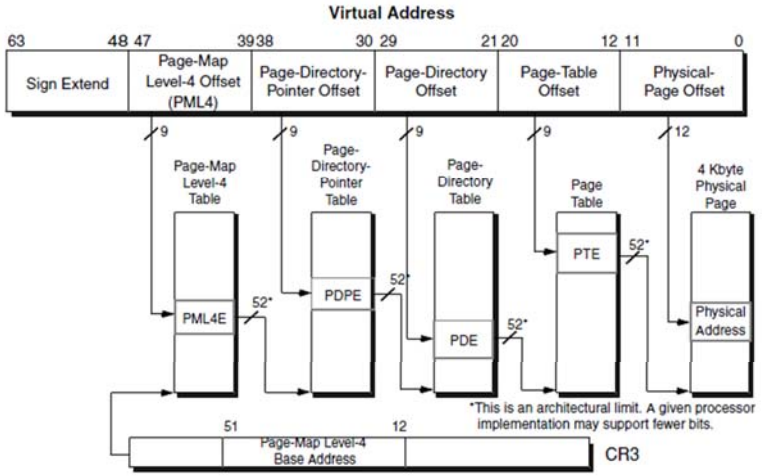
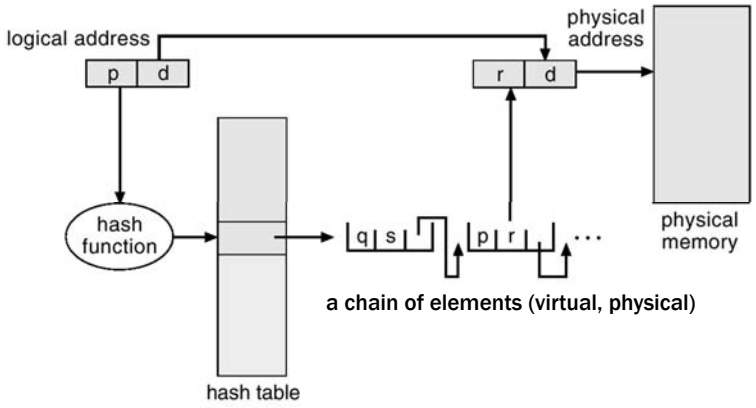


Figure 5-17. 4-Kbyte Page Translation—Long Mode

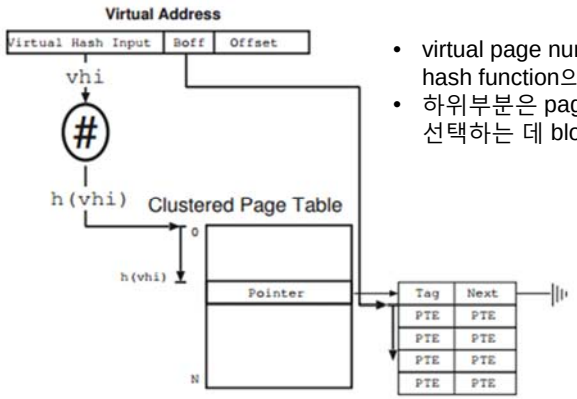
Hashed Page Tables

- 주소 공간이 32비트보다 커지면 hashed page table을 사용
- hashed page table
  - 논리주소의 page number의 hash function 값을 hashed page table의 index로 사용함
  - hash table의 각 entry는 연결리스트를 저장
    - 같은 hash function 값을 갖는 page들의 paging 정보 저장
    - 연결리스트 원소 형식: <page, page frame, next pointer>
- hashed page table을 사용한 주소 변환
  - 논리주소의 page number와 해쉬된 위치의 연결리스트 원소들의 page 번호들과 차례대로 비교함
  - match되면 page를 대응되는 page frame으로 변환함
- clustered page table
  - page table의 각 entry가 한 개의 page가 아닌 여러 page에 대한 주소 변환 정보를 저장함
  - 논리 주소 공간을 sparse하게 사용하는 경우 유용함

Hashed Page Tables(계속)



clustered page table

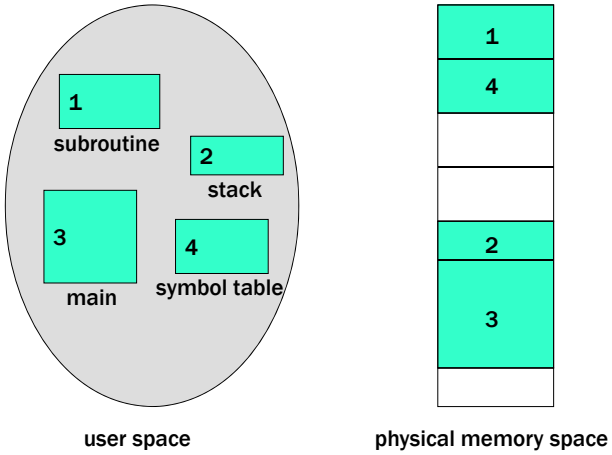


- virtual page number의 상위부분을 hash function의 입력으로 사용
- 하위부분은 page table entry를 선택하는 데 block offset으로 사용

8.4 Segmentation

- 프로그램은 세그먼트의 집합임 → 사용자 관점의 메모리
  - 세그먼트는 프로그램의 논리적 단위
    main program, procedure, function, local variables,
    global variables, common block, stack, symbol table, arrays …
- 세그먼테이션(segmentation)
  - 논리 주소 = <segment number, offset> → 2차원 주소
  - 주소 맵핑을 세그먼트 단위로 수행하는 메모리 관리 방식
    - 2차원 논리 주소를 1차원 물리 주소로 변환

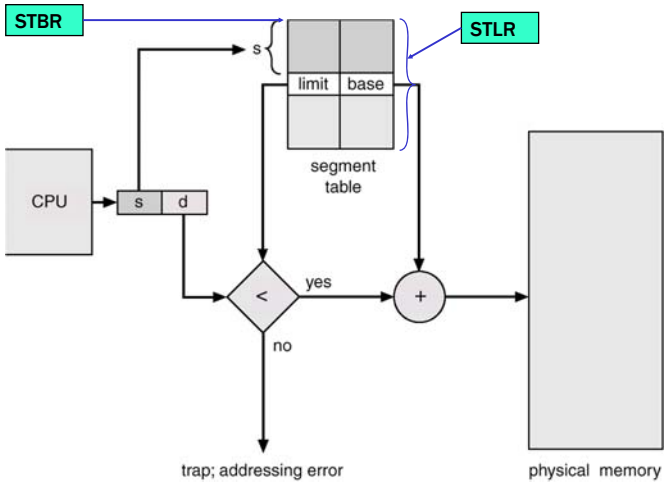
Logical View of Segmentation



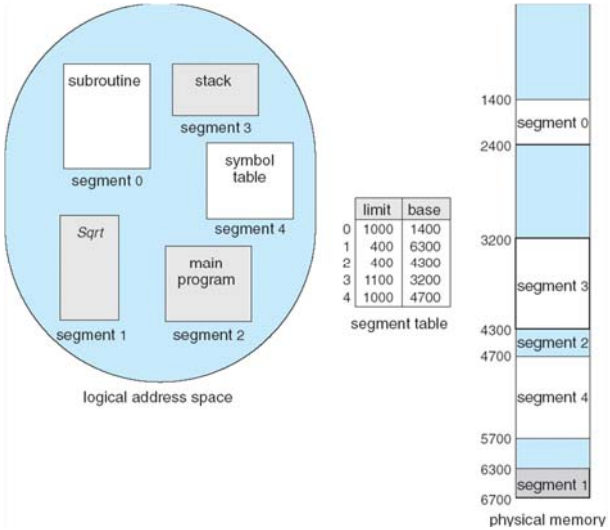
Segmentation 구조

- Segment table
  - 프로그램의 각 세그먼트의 주소 변환 정보 저장
  - table entry 형식
    - **base** – 세그먼트의 starting physical address
    - **limit** – 세그먼트의 길이
  - 세그먼트 길이가 가변이므로 메모리 할당은 동적 할당 방법을 사용
- Segment-table base register (STBR)
  - 현재 프로세스의 segment table의 시작 주소
- Segment-table length register (STLR)
  - 현재 프로세스의 segment 개수
  - segment number *s* 에 대해서 *s* < STLR 이면 s는 합법적

Segmentation 하드웨어와 주소 변환

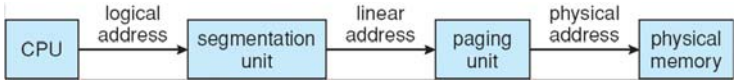


Segmentation 예



8.7 Example - Intel IA-32 (Pentium)

- segmentation과 segmentation with paging을 지원 (in protected addressing mode)
- 주소 변환 과정

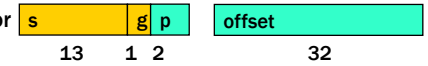


  - logical address : <segment, offset> 2차원 주소
  - linear address : 1차원 주소
    - paging을 사용하지 않으면 physical address와 같음
  - physical address

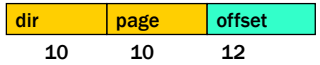
Segmentation과 Paging

- segmentation:
  - two segment tables
    - LDT(local descriptor table)
    - GDT(global descriptor table)

s: segment
g: GDT/LDT
p: privilege level

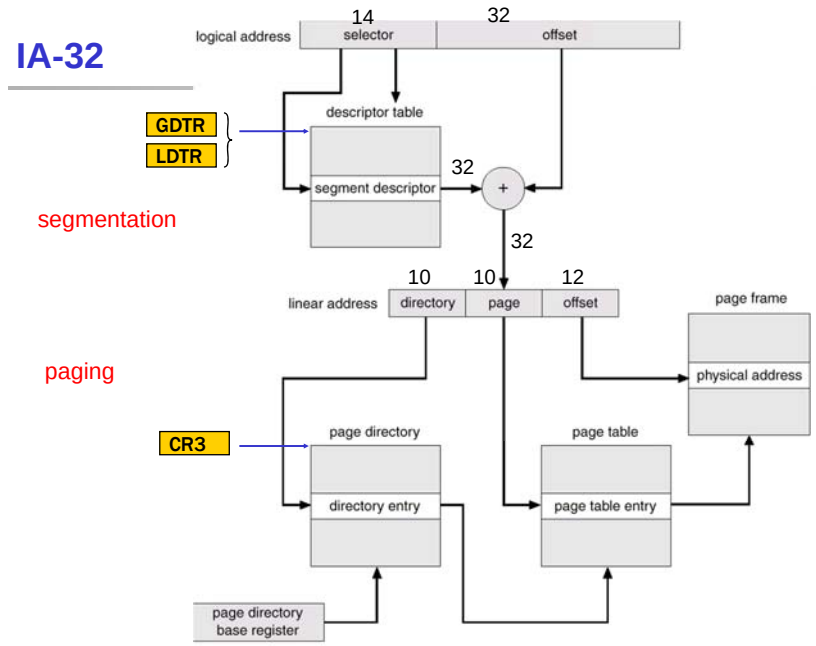


  - translate 46-bit logical address into 32-bit linear address
- paging
  - a two-level paging scheme.



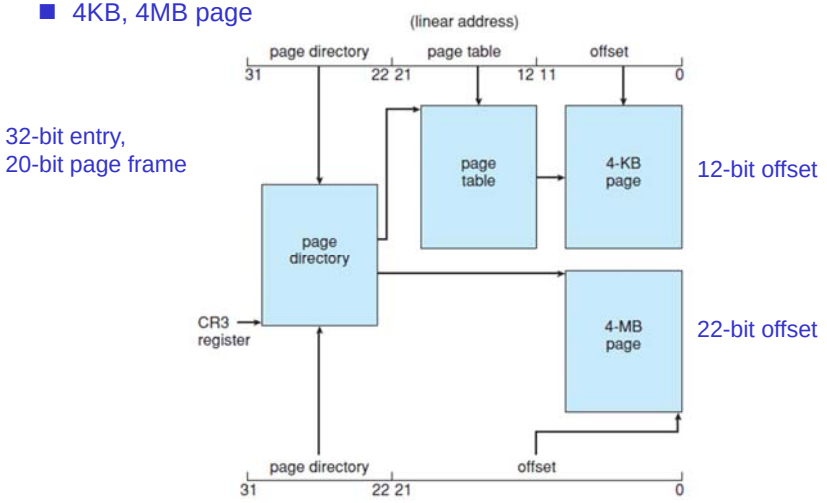
  - translate 32-bit linear address into 32-bit physical address

IA-32



IA-32 paging – multiple page sizes

- 4KB, 4MB page

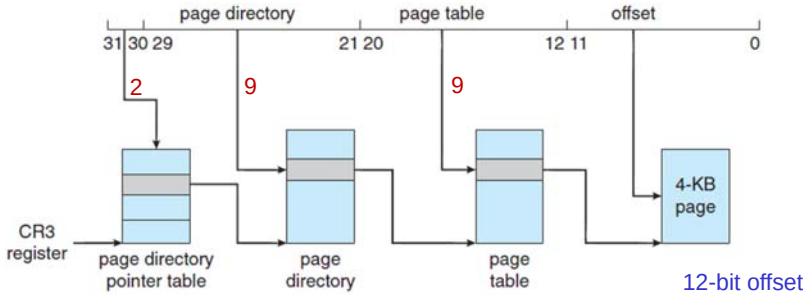


32-bit entry,
20-bit page frame

12-bit offset

22-bit offset

Page Address Extensions (PAE)



12-bit offset

Figure 8.24 Page address extensions.

64-bit entry,
24-bit page frame → 24 + 12 = 36 bit physical address

ARM paging architecture

- Multiple page sizes