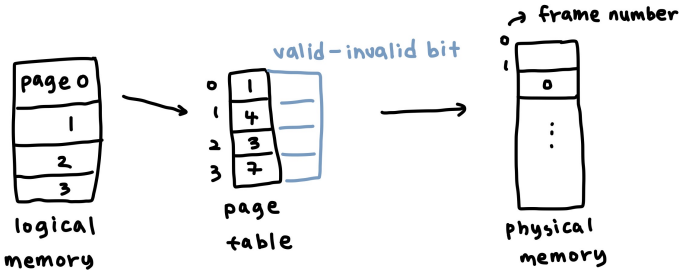
9 메모리 관리2

Paging

- process의 virtual memory를 동일한 사이즈의 page 단위로 나눔
- virtual memory의 내용이 page 단위로 **noncontiguous**하게 저장됨
- 일부는 backing store에, 일부는 physical memory에 저장 → *필요한 부분만* 올려놓음

Basic method

- physical memory를 동일한 크기의 frame으로 나눔 → page가 들어갈 수 있는 크기로 자름
- logical memory를 동일 크기의 page로 나눔 (frame과 같은 크기)
- 모든 가용 frame들을 관리
- page table을 사용하여 logical address를 physical address로 변환
- external fragmentation 발생 안함 → 동일한 크기의 page이기 때문에
- internal fragmentation 발생 가능 → 할당되었지만 page보다 작은 프로세스라 낭비되는 공간

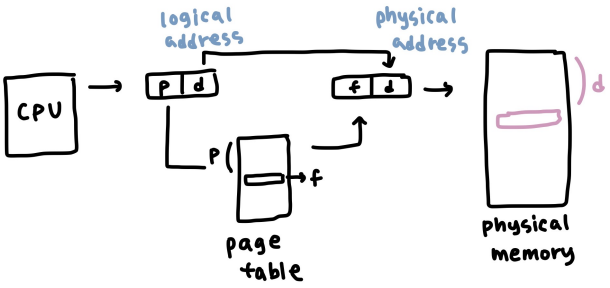


valid - invalid bit

→ 물리적 메모리에 올라가지 않는 page는 page table속 숫자가 의미가 없음

⇒ valid - invalid bit로 물리적 메모리에 올라가 있는지 확인

address translation architecture



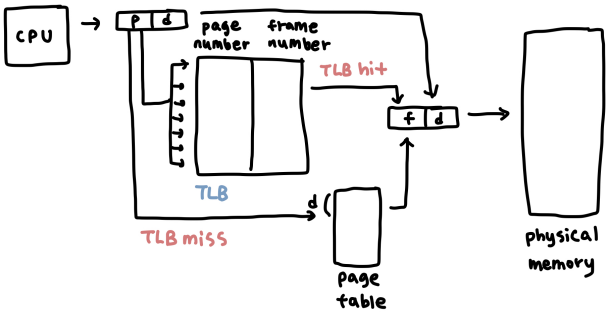
d : **offset** → page내에서 얼마나 떨어져 있는가

Implementation of page table

→ page 개수가 프로세스보다 너무 많아도 안됨

- page table은 main memory에 상주
- *page-table base register(PTBR)*가 page table을 가리킴 → *page table의 시작 위치*
- *page-table length register(PTLR)*가 *테이블 크기*를 보관
- 모든 메모리 접근 연산에는 **2번의 memory access** 필요

  ⇒ page table 접근 1번 + 실제 data / instruction 접근 1번
- 속도 향상을 위해 associative register 혹은 translation look-aside buffer(TLB)라 불리는 고속의 lookup hardware cache 사용 → 일부 정보만을 알고있음



→ TLB에 있는지 전부 다 탐색해야 함 → parallel하게 한꺼번에 찾음

Associative register

associative regiser (TLB) : parallel search가 가능

address translation

- page table 중 *일부가 associative register에 보관*되어 있음
- 만약 해당 page #가 associative register에 있는 경우 곧바로 frame #를 얻음
- 그렇지 않은 경우 main memory에 있는 page table로부터 frame #를 얻음
- TLB는 context switch때 flush (remove old entries)

Effective Access Time

→ 메모리 접근 시간

- associative register lookup time = ϵ
- memory cycle time = 1
- hit ratio = α → associative register에서 찾아지는 비율

Effective Access Time (EAT)

$$EAT = (1+\epsilon)\alpha + (2+\epsilon)(1-\alpha) = 2 + \epsilon - \alpha$$

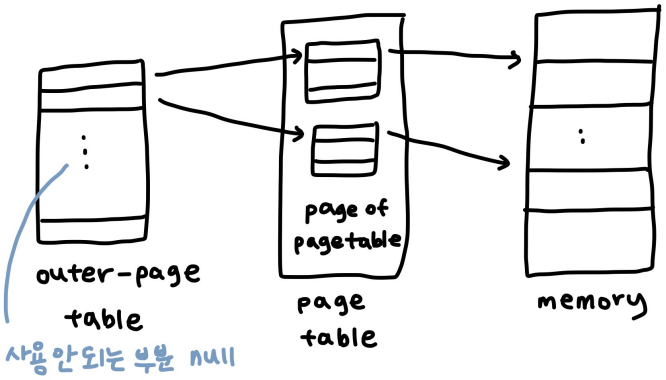
→ hit 시간 + miss 시간

Two-level page table

현대의 컴퓨터는 address space가 매우 큰 프로그램을 지원 → *page table 공간이 심하게 낭비*

⇒ page table 자체를 page로 구성

→ 사용되지 않는 주소공간에 대한 outer page table의 엔트리 값은 NULL (대응하는 inner page table이 없음)

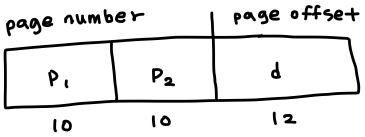


→ 주소 변환을 위해 *메모리를 2번* 거침 + 데이터 접근으로 *메모리 1번 더*

~ 시간은 오래 걸리지만 공간상의 이득이 생김

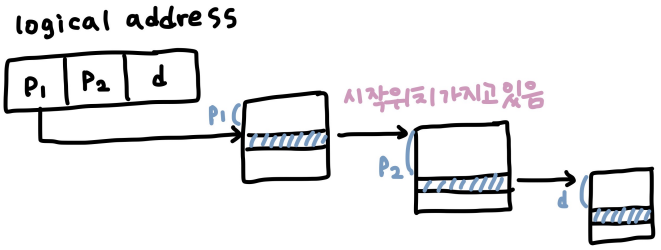
→ Example

- logical address (on 32-bit machine with 4K page size)의 구성
  - 20 bit의 page number
  - 12 bit의 page offset
- page table 자체가 page로 구성되기 때문에 page number는 다음과 같이 나눔 (각 page table entry가 4B)
  - 10 bit의 page number
  - 10 bit의 page offset



→ P_1은 outer page table의 index이고 P_2는 outer page table의 page에서의 변위

Address-Translation Scheme



d : page offset → page 안에서 얼마나 떨어져 있는가

Multiple paging and performance

- address space가 커지면 다단계 페이지 테이블 필요
- 각 단계의 페이지 테이블이 메모리에 존재하므로 *logical address의 physical address 변환에 더 많은 메모리 접근 필요*
- TLB를 통해 *메모리 접근 시간을 줄일 수 있음*
- 4단계 페이지 테이블을 사용하는 경우
  - 메모리 접근 시간이 100ns, TLB 접근 시간이 20ms이고
  - TLB hit ratio 가 98%인 경우
  - effective memory access time = 0.98 * 120 + 0.02 * 520 = 128 nanoseconds
  - 결과적으로 주소 변환을 위해 28ns만 소요

Memory Protection

page table의 각 entry마다 아래의 bit를 둔다

- **protection bit**

  page에 대한 *접근 권한*(read/write/read-only)
- **valid-invalid bit**
  - "valid"는 해당 주소의 frame에 그 프로세스를 구성하는 *유효한 내용이 있음*을 뜻함 (접근 허용)
  - "invalid"는 해당 주소의 frame에 *유효한 내용이 없음*을 뜻함 (접근 불허)

    → 1) 프로세스가 그 주소 부분을 사용하지 않는 경우

→ 2) 해당 페이지가 메모리에 올라와 있지 않고 swap area에 있는 경우