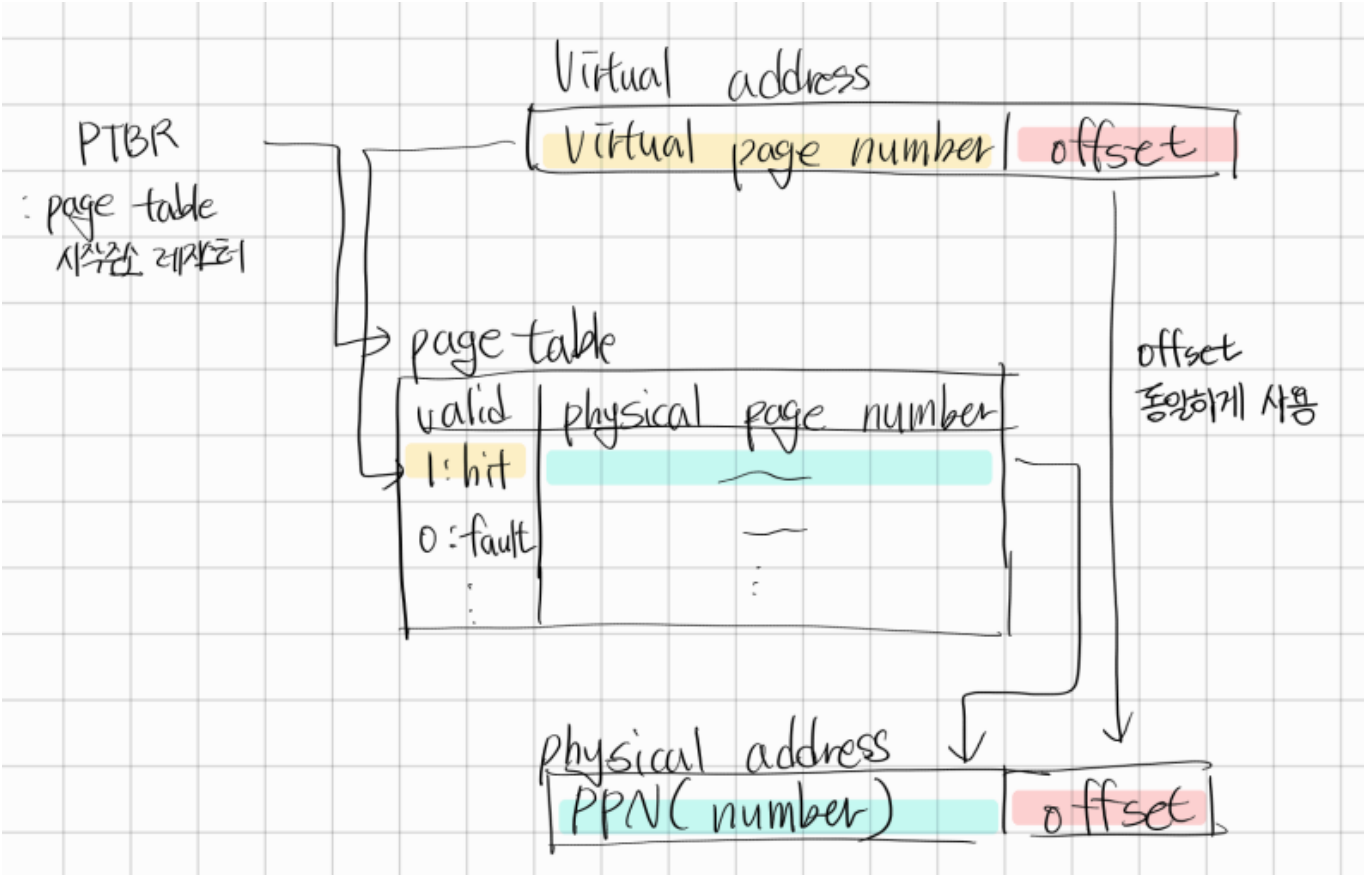
Address Translation With a Page Table

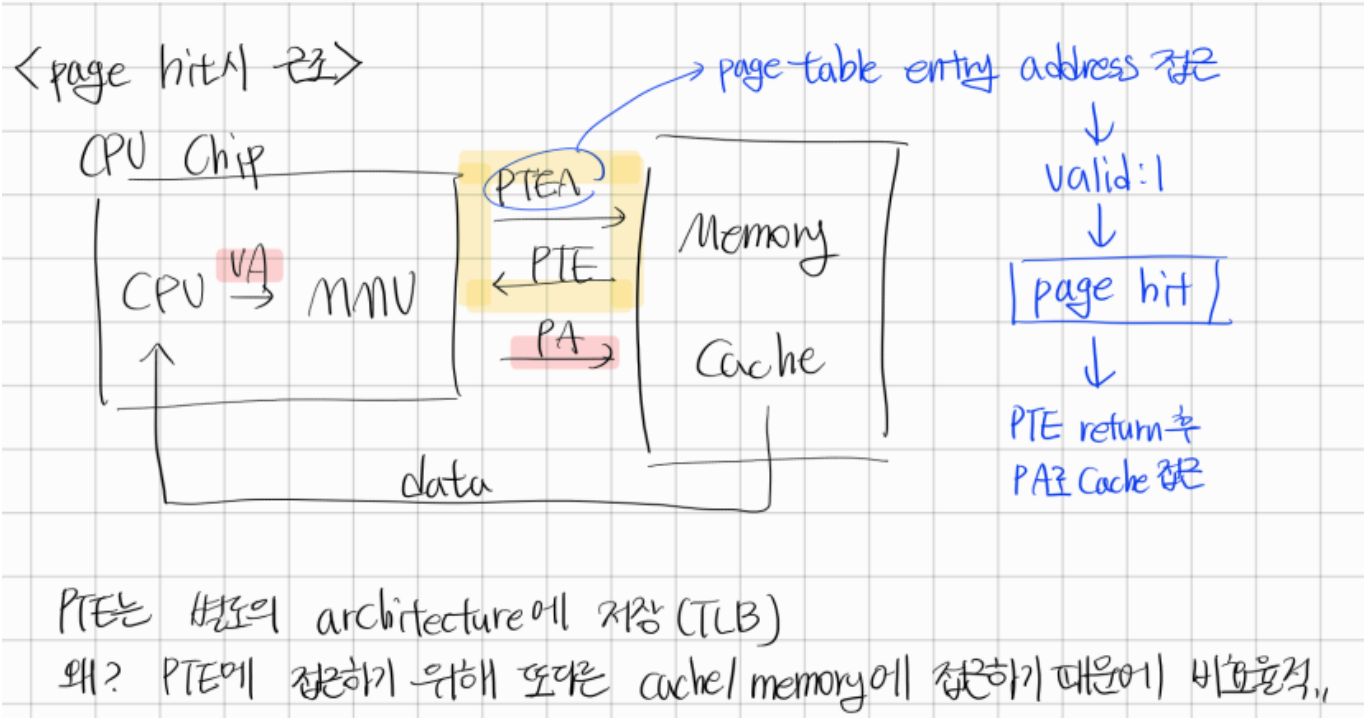


Page Table에는 VPN, PPN만 사용해서 매칭
offset은 그대로 사용한다.
valid bit로 hit/fault 판단한다.

페이지 테이블의 valid는 VPN이 참조한다.
VPN에 해당하는 PPN이 PTE에 같이 저장된다.

Page hit

Page Table이 메모리에 저장되어있는 구조 기준이다.



1. CPU가 MMU에게 VA 제공함
2. MMU가 PTEA로 메모리에 접근하여 PTE를 들고옴
3. PTE에서 VA에 매칭되는 PA 고르기
4. PA로 Cache 접근
5. Cacha에서 CPU에게 데이터 전송

Q1. PTE도 다른 것처럼 캐시에 저장되나?

-> Yes and NO.. PTE는 TLB라는 별도의 architecutre에 저장됨

Q2. PTE가 또다른 캐시에 저장되는거면 느리지않니?

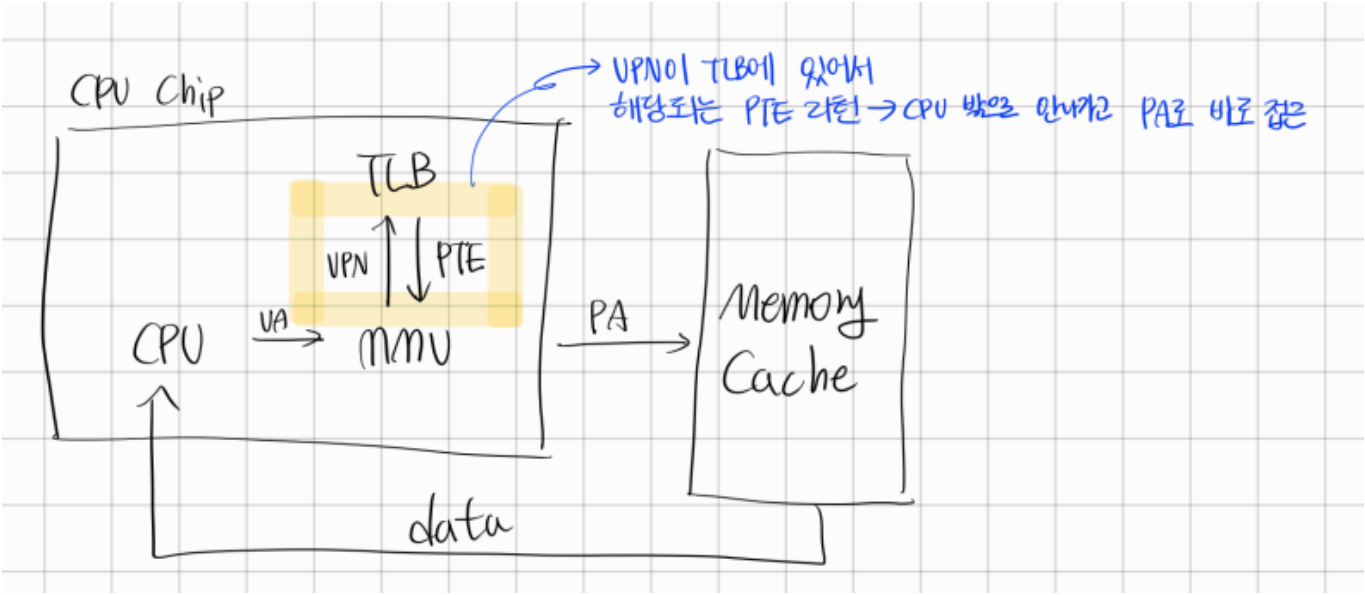
-> 느린거맞음.. 그래서 TLB에 저장

TLB

Translation Lookaside Buffer

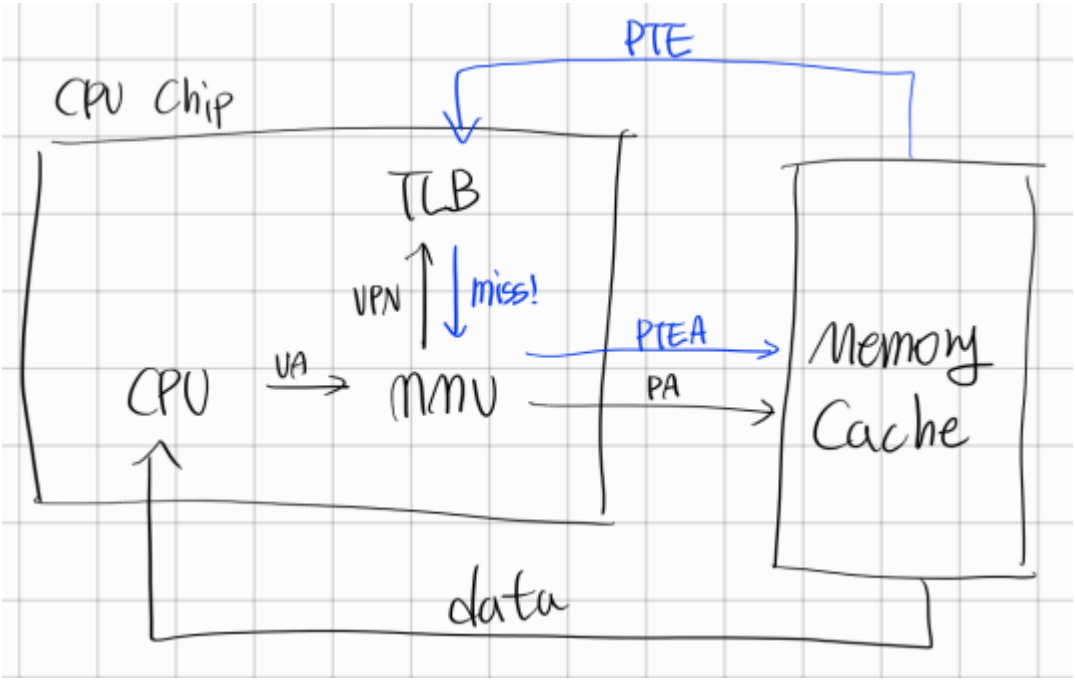
- 작고, 섬세, 매우 빠른 PTE를 저장하는 캐시
- CPU Chip 안에 같이 저장된다.

TLB Hit



1. MMU는 CPU가 준 VA로 TLB에 VPN 있는지 물어봄
2. 있다면, 매칭되는 PTE 리턴
3. PTE에 있는 PPN + PPO = PA로 바로 캐시 접근

TLB Fault



1. MMU는 CPU가 준 VA로 TLB에 VPN 있는지 물어봄
2. 없다면, miss처리 후 MMU가 메모리에 PTEA로 물어봄
3. 메모리가 TLB로 PTE 리턴
4. TLB로부터 PTE받은 MMU는 PA로 캐시 접근

-> TLB Fault시, 메모리 접근 추가적으로 일어남.

Fortunately, TLB misses are rare. Why?

10~20%만 miss 발생 -> 효율적임

Q3. 페이지 테이블 크지 않니? 어떻게 RAM에 저장될수가..

-> 그렇지만, 실제 페이지테이블은 단순 배열이 아닌 트리구조

Page Table 구조

1. Multi_Level
   Level 1 table: each PTE points to a page table
   Level 2 table: each PTE points to a page (실제 PT)

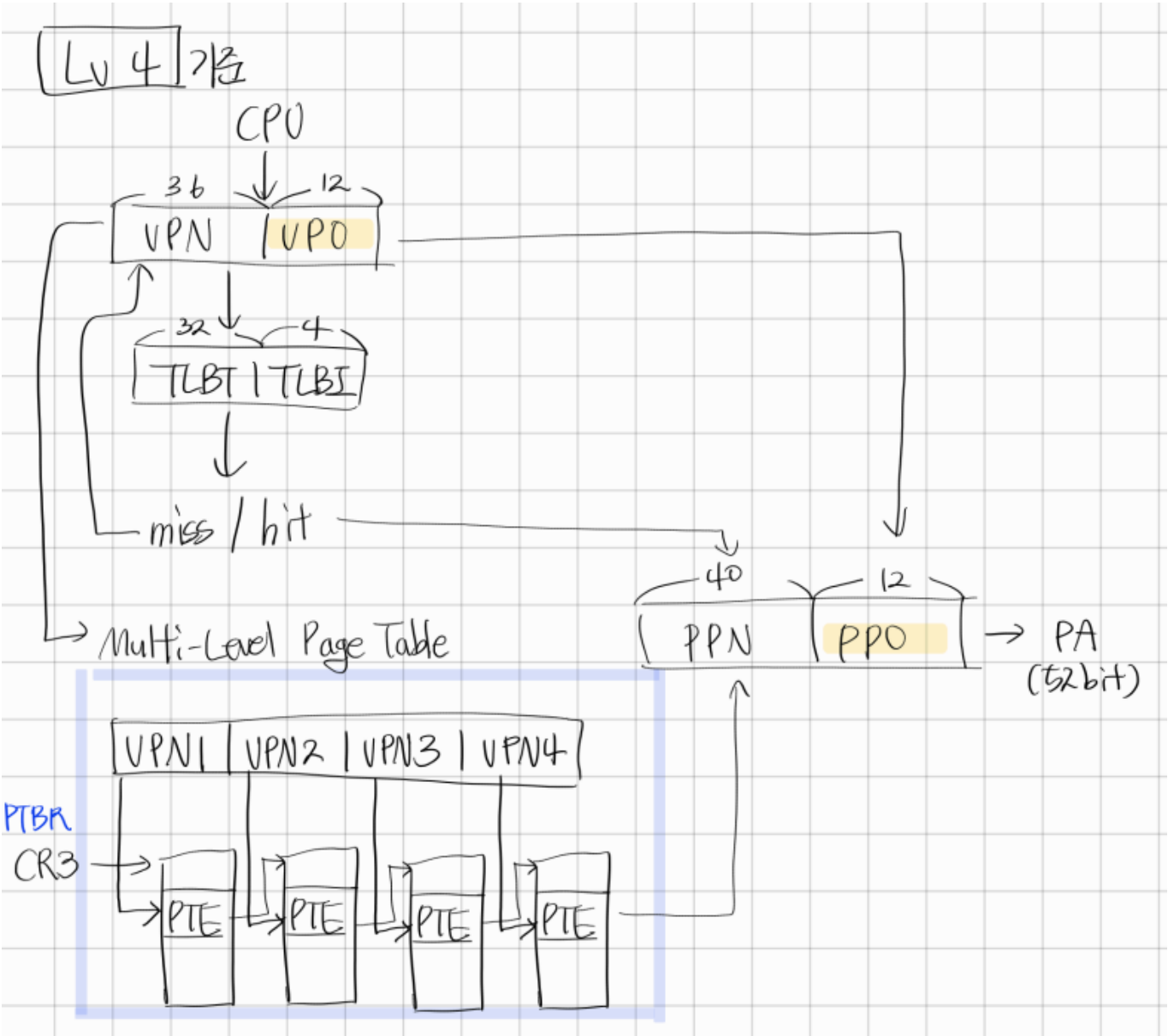
2. Two-Level Page Table Hierarchy (예시)
   PTE의 주소를 저장하고 있는 PTE...

--> k-level page table(주로 4, 5 level)

레벨 거듭하면서 찾고 찾고 찾고 찾아서 VPN으로 PPN도달해 PA로 메모리 접근

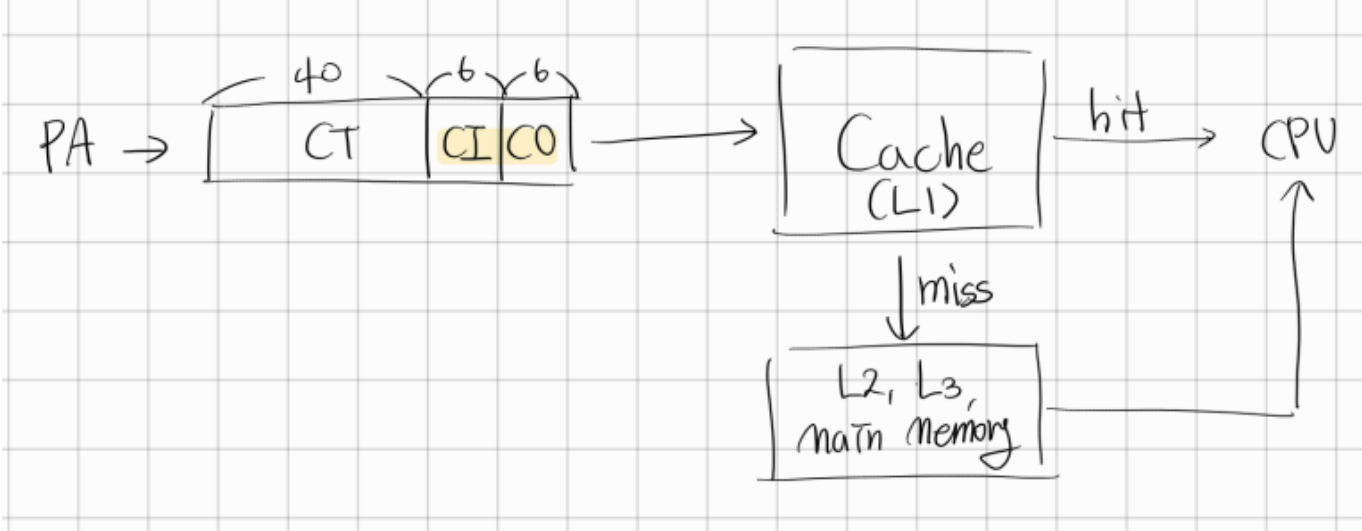
보통 하나의 VPN당 9bit 9 * 4(level) + 12(offset) = 48bit 정도 Lv 5는 57bit...

End-to-End Core i7 Address Translation



LV 4(48bit)기준

1. VPN으로 TLB접근
2. hit -> 바로 PPN으로 PA 알기
3. fault -> Multi-Level Page Table 접근해 PTE 얻기 (현상황: 4level)
4. PTE로 PPN -> PA 알기

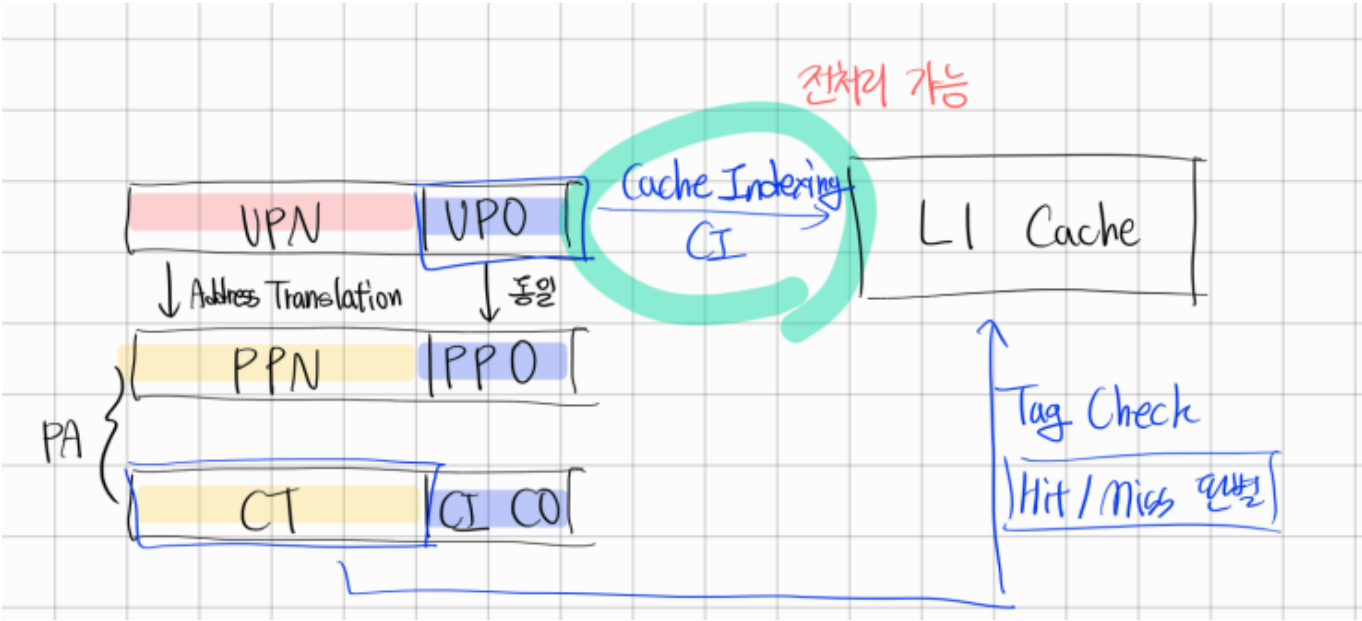


5. PA로 Cache 접근
6. CI/CO로 캐시 인덱싱, CT로 캐시 태그 검사
7. L1 캐시에 PA 존재하면 바로 CPU로 데이터 리턴
8. L2, L3, Main Memory 순으로 PA매치 찾아서 CPU로 리턴

단위
VA: 레벨 별 상이 - lv.4 -> 48bit, lv.5 -> 57bit PA: 52bit 고정

Cute Trick for Speeding Up L1 Access

VPO -> PPO 동일하기 때문에 CI|CO 비트 수만 맞다면 전처리 가능하다는 아이디어.



CT: Cache Tag, PPN이 진짜로 넘어와야 얻을 수 있는 정보이다.
CI|CO: PPN 없어도 알 수 있는 값, 전처리 가능하다면 tag check 전 해당 데이터가 있을 확률이 높은 비트를 뽑

아놓고 CT 오면 빠른 check를 가능하게 해준다.

전처리 가능

offset의 12bit는 불변하기 때문에 실행 전, VPO로 cache 먼저 접근할 수 있음
-> 미리 체크하는 개념, 나중에 tag로 진짜 hit/miss 여부 판단

- VPO == PPO
- CI|CO가 12bit로 VPO/PPO와 길이가 동일하면 전처리 가능하다
    - Cache Size 정할 때 CI|CO 변하지 않도록 조정할 필요 존재하며, 캐시가 무조건 큰게 좋은 것이 아님

Virtually indexed, physically tagged

CI|CO 비트 수 동일해야 전처리 가능하기 때문에, 동일하지 않다면 -> PIPT