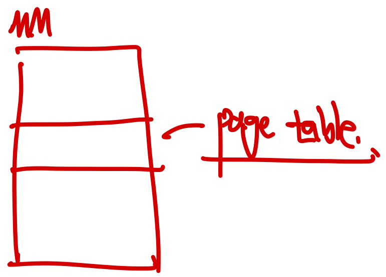
Paging: Faster Translation (TLBs)

Prof. Yongtae Kim

Computer Science and Engineering
Kyungpook National University

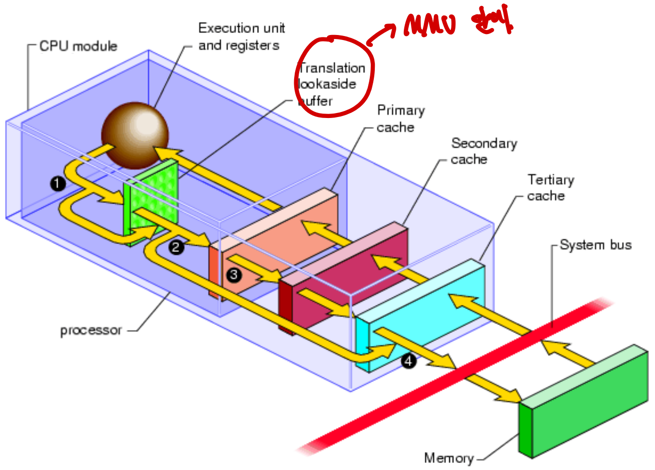


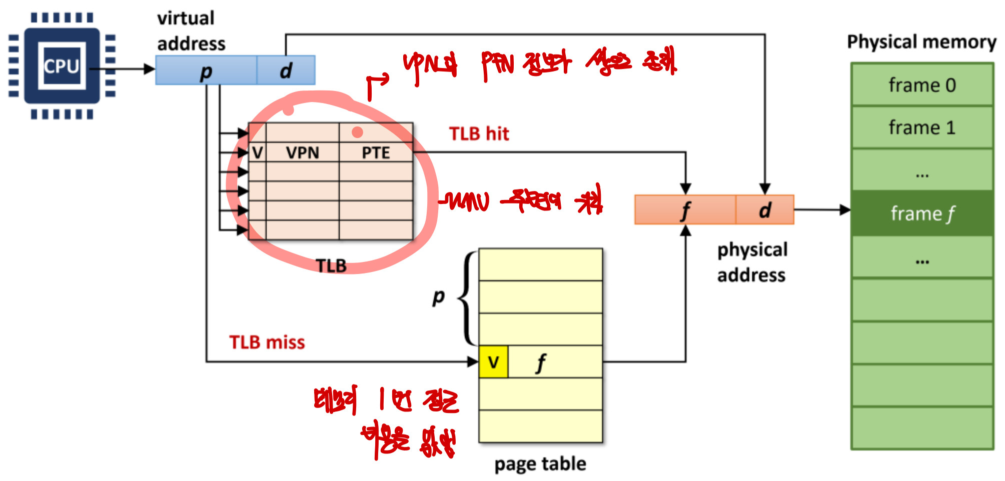
Translation Lookaside Buffer (TLB): Concept

- **Paging logically requires an extra memory lookup for each VA**
  - Going to memory for translation information before every instruction fetch or explicit load or store is prohibitively slow
  - To speed up the address translation, OS needs the hardware's help

    buffer의 응용

    → 지대한 영향
- **Translation lookaside buffer (TLB) expedites address translation**
  - A TLB, part of the chip's MMU, is simply a hardware cache of popular virtual-to-physical address translations; better name would be address-translation cache
  - TLB makes virtual memory possible as it affects the performance tremendously





Courtesy of Prof. Jin-Soo Kim @ SNU

TLB Basic Algorithm

↳ HW

TLB 안에 있다.

- **Translation using hardware-managed TLB with linear page table**
  - TLB hit: 1) memory access
  - TLB miss: 1) memory access for PTE → update TLB
    2) retry → TLB hit → memory access
  - TLB, like all caches, is build on the premise that in common case, translations are found in cache
  - Typically, TLB hit rate ≥ 99%

```
VPN = (VirtualAddress & VPN_MASK) >> SHIFT
(Success, TlbEntry) = TLB_Lookup(VPN)
if (Success == True)    // TLB Hit
    if (CanAccess(TlbEntry.ProtectBits) == True)
        Offset   = VirtualAddress & OFFSET_MASK
        PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
        Register = AccessMemory(PhysAddr)
    else
        RaiseException(PROTECTION_FAULT)
else                   // TLB Miss
    PTEAddr = PTBR + (VPN * sizeof(PTE))
    PTE = AccessMemory(PTEAddr)
    if (PTE.Valid == False)
        RaiseException(SEGMENTATION_FAULT)
    else if (CanAccess(PTE.ProtectBits) == False)
        RaiseException(PROTECTION_FAULT)
    else
        TLB_Insert(VPN, PTE.PFN, PTE.ProtectBits)
        RetryInstruction()
```

→ TLB

→ page table 참조

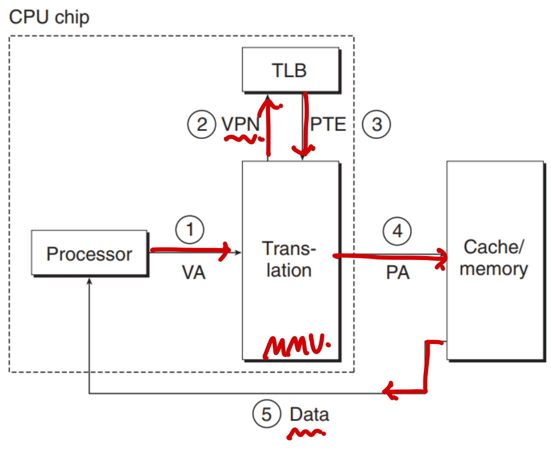
→ TLB에 참조 재 설정

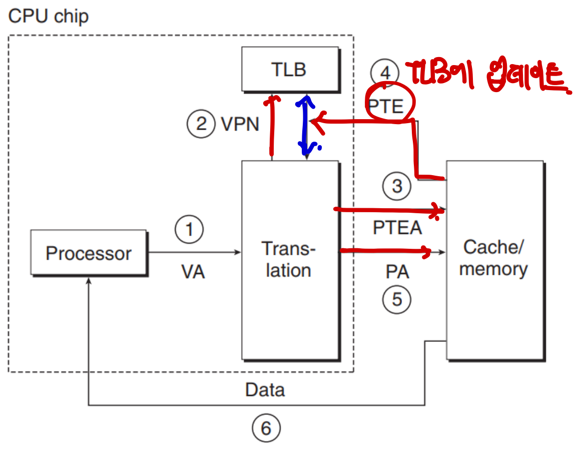
update.

↳ 다시 수행 → 무조건 hit.

TLB hit

TLB miss





Example: Accessing an Array

int a[100]

- **Let's consider an example with assumptions**
  - Array of 10 4-byte integers in memory, staring at $\boxed{\text{VA 100}}$
  - Small 8-bit virtual address space with $\boxed{\text{16-byte}}$ pages (4-bit VPN & 4-bit offset)
  - Only consider memory access for array
  - Memory access behavior:
    1) access **a[0]** → TLB miss → TLB update (VPN 6)
    2) access **a[1]**, **a[2]** → TLB hit
    3) access **a[3]** → TLB miss → TLB update (VPN 7)
    4) access **a[4]**, **a[5]**, **a[6]** → TLB hit
    5) access **a[7]** → TLB miss → TLB update (VPN 8)
    6) access **a[8]**, **a[9]**, **a[10]** → TLB hit
  - TLB hit rate: 7 hit of total 10 access = 70%

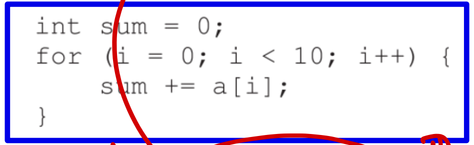
page의 크기가 16B에서 70%가 나온거.
일반적으로 page의 크기 4KB 정도 배열의 크기 1000개 씩 들어가면 개의 100% 4hit.

  - Spatial locality: elements of the array are close to one another in space (page size ↑ → TLB hit rate ↑)
  - Temporal locality: quick re-referencing of memory item in time

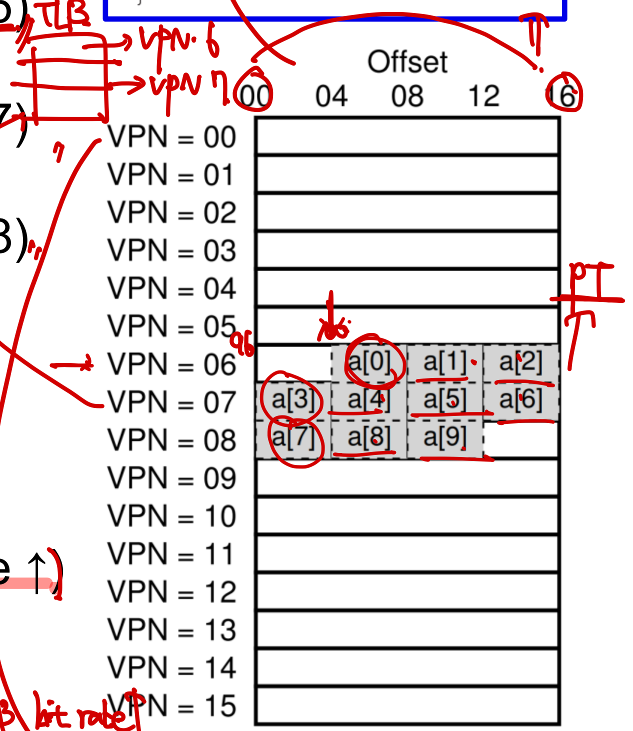
page의 크기 4bit.

$16 \overline{)\,100}$ 6 / 96

page size↑ → TLB hit rate↑



```
int sum = 0;
for (i = 0; i < 10; i++) {
    sum += a[i];
}
```



Who Handles the TLB Miss?

tlb miss? page table → page fault → [illegible]

flexible (x)

- Either hardware (CPU) or software (OS) can handle the TLB miss
- Earlier CISC (x86) has hardware-managed TLB
  - The hardware has to know the location of page table and its format
  - The hardware 1) walks the page table, 2) finds the correct PTE, 3) extracts the desired translation, 4) update TLB with the translation, 5) retry the instruction
- Newer RISC has software-managed TLB
  - On a TLB miss, the hardware simply raises an exception and then: trap.
  - The hardware 1) pauses execution, 2) switch to kernel mode, 3) jumps to trap handler ➔ (OS to process TLB miss) ➔ 4) retries the instruction (will TLB hit)

TLB hit

HW TLB

```
VPN = (VirtualAddress & VPN_MASK) >> SHIFT
(Success, TlbEntry) = TLB_Lookup(VPN)
if (Success == True)    // TLB Hit
    if (CanAccess(TlbEntry.ProtectBits) == True)
        Offset   = VirtualAddress & OFFSET_MASK
        PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
        Register = AccessMemory(PhysAddr)
    else
        RaiseException(PROTECTION_FAULT)
else                    // TLB Miss
    PTEAddr = PTBR + (VPN * sizeof(PTE))
    PTE = AccessMemory(PTEAddr)
    if (PTE.Valid == False)
        RaiseException(SEGMENTATION_FAULT)
    else if (CanAccess(PTE.ProtectBits) == False)
        RaiseException(PROTECTION_FAULT)
    else
        TLB_Insert(VPN, PTE.PFN, PTE.ProtectBits)
        RetryInstruction()
```

위치의 결과.

$\longleftrightarrow$

What hardware do on TLB miss

SW TLB

```
VPN = (VirtualAddress & VPN_MASK) >> SHIFT
(Success, TlbEntry) = TLB_Lookup(VPN)
if (Success == True)    // TLB Hit
    if (CanAccess(TlbEntry.ProtectBits) == True)
        Offset   = VirtualAddress & OFFSET_MASK
        PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
        Register = AccessMemory(PhysAddr)
    else
        RaiseException(PROTECTION_FAULT)
else                    // TLB Miss
    RaiseException(TLB_MISS)
```

→ 무조건 hit.

TLB miss-handling trap

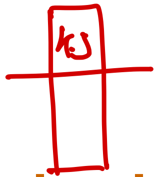
trap

그다음 명령어x

재실행

※ This trap is difference from the other trap (e.g. system call) in that the hardware must resume execution at the instruction that caused the trap, not the next instruction

TLB Contents: What's in There?



스왑아웃 되지 않는 운영체제 커널 영역

- **What if TLB miss when OS is running TLB miss-handling code?**
  - (Original) TLB miss → "TLB miss handling" → TLB miss → TLB miss handing → ...
  - Two simple solutions: → 항상 물리 주소로

    → 메모리 부분으로 페이지가 디스크에 스왑아웃

    1) Keeping TLB miss handler in physical memory (directly access to handler code)

    2) Reserving some entries in TLB for permanently-valid translations and use some of those slots for handler code itself; these wired translations always hit the TLB

    해서 더 빠름

    → 항상 tlb hit.
- **The primary advantage of "software" approach is flexibility**
  - OS can use any data structure to implement page table without hardware change
  - Another advantage is simplicity; hardware doesn't do much on a miss (see codes)

- **Typically, TLB is (fully associative) with 32, 64, or 128 entries**
  - The hardware search the entire TLB in parallel to find the desired translation

    → protection, 유효성 valid)
  - TLB entries would be like the following format:

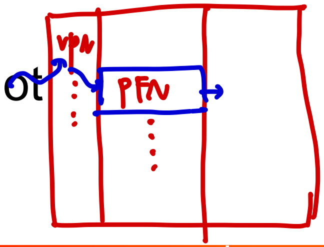
VPN	PFN	other bits

  - Other bits usually contains:

    1) valid bit to indicate whether the entry has a valid translation or not

    2) protection bits to determine how a page can be accessed (**rwx**)

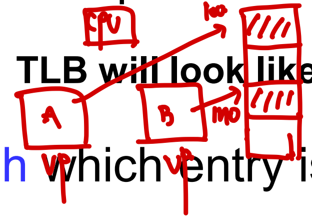
    3) address-space identifier, dirty bit, etc



TLB Issue: Context Switch & Replacement Policy

"→ tlb에서 자주 쓰지 않는 entry는 다시 줄여 가져야 함."

- **With TLBs, some new issues arise when context switching**
  - TLB contains virtual-to-physical translations that are only valid for the currently running process, not meaningful for other processes
  - e.g.) Process A: VPN 10 → PFN 100,
    Process B: VPN 10 → PFN 170
    Then, the hardware cannot distinguish which entry is meant for which process



TLB will look like:

VPN	PFN	valid	prot
10	100	1	rwx
—	—	0	—
10	170	1	rwx
—	—	0	—

10번째가 2개.

캐시 ㅁ

P1

CPU는 1개. 뭐가 A의 정보..?

CPU

- **One solution is to simply flush the TLB on context switches**
  - The flush operation simply sets all valid bits to 0; effectively clearing TLB contents
    = clearing
  - There is cost: the flush incurs high TLB misses after context switches
- **Some hardware can have address space identifier (ASID) in TLB**
  - ASID makes TLB be shared across context switches
  - OS must, on a context switch, set a privileged register to the ASID of the current process

VPN	PFN	valid	prot	ASID
10	100	1	rwx	1
—	—	0	—	—
10	170	1	rwx	2
—	—	0	—	—

tlb는 서로 공유.

↳context switch할 때 무조건 privileged register를 process의 ASID로 tlb에 설정

그럼 CPU가

운영체제가 관리.

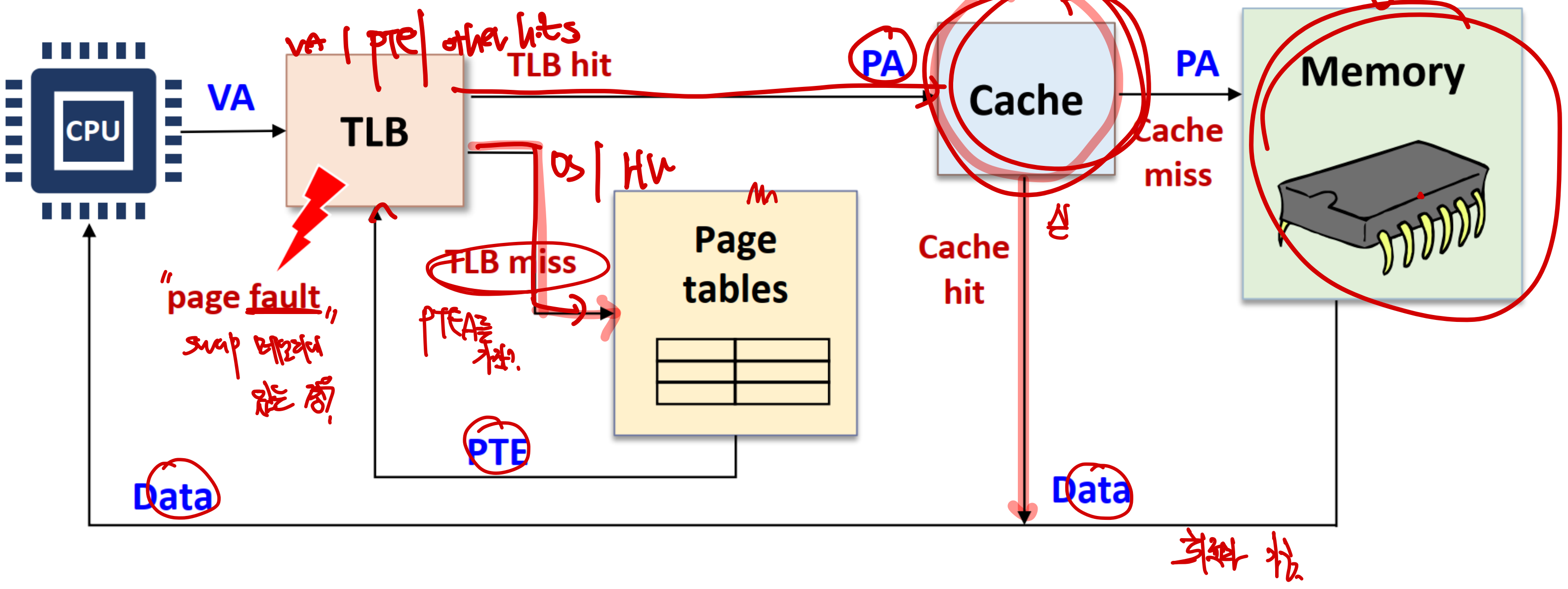
- **Like cache, TLB replacement policy should also be considered**
  - Least-recently-used (LRU) policy (locality) or random policy (simplicity)

(관리하는 방법이 존재!

Summary

- **Translate lookaside buffer (TLB), part of MMU, is a hardware cache to expedite virtual-to-physical address translations**
  - A TLB entry 포함하다. includes both VPN and PFN + other bits (valid, protection, etc)
  - Two kinds of TLBs: hardware-managed TLB and software-managed TLB

실제 캐시. (PA를 통해 접근 하는법.)
caching



Courtesy of Prof. Jin-Soo Kim @ SNU