Chapter 19: Translation Lookaside Buffer

Adam Disney

THE UNIVERSITY OF
TENNESSEE
KNOXVILLE

Crux: How To Speed Up Address Translation

THE CRUX:
HOW TO SPEED UP ADDRESS TRANSLATION

How can we speed up address translation, and generally avoid the extra memory reference that paging seems to require? What hardware support is required? What OS involvement is needed?

Overview

- Let's add a hardware cache specifically for the page tables called a Translation Lookaside Buffer (TLB).
- Sidenote:
  - Caching is a fundamental performance technique used throughout computer systems to make "the common case" fast.
  - Take advantage of locality (temporal and spatial)

Overview

```
1  VPN = (VirtualAddress & VPN_MASK) >> SHIFT
2  (Success, TlbEntry) = TLB_Lookup(VPN)
3  if (Success == True)    // TLB Hit
4      if (CanAccess(TlbEntry.ProtectBits) == True)
5          Offset   = VirtualAddress & OFFSET_MASK
6          PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
7          Register = AccessMemory(PhysAddr)
8      else
9          RaiseException(PROTECTION_FAULT)
10 else                        // TLB Miss
11     PTEAddr = PTBR + (VPN * sizeof(PTE))
12     PTE = AccessMemory(PTEAddr)
13     if (PTE.Valid == False)
14         RaiseException(SEGMENTATION_FAULT)
15     else if (CanAccess(PTE.ProtectBits) == False)
16         RaiseException(PROTECTION_FAULT)
17     else
18         TLB_Insert(VPN, PTE.PFN, PTE.ProtectBits)
19         RetryInstruction()
```

Figure 19.1: **TLB Control Flow Algorithm**

Example: Accessing An Array

- Assumptions
  - Array "a" is at virtual address 100
  - 8-bit virtual address space
  - 16 byte pages thus 4-bit VPN and 4-bit offset
  - Only "a" generates memory accesses (for simplicity)

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

Example: Accessing An Array

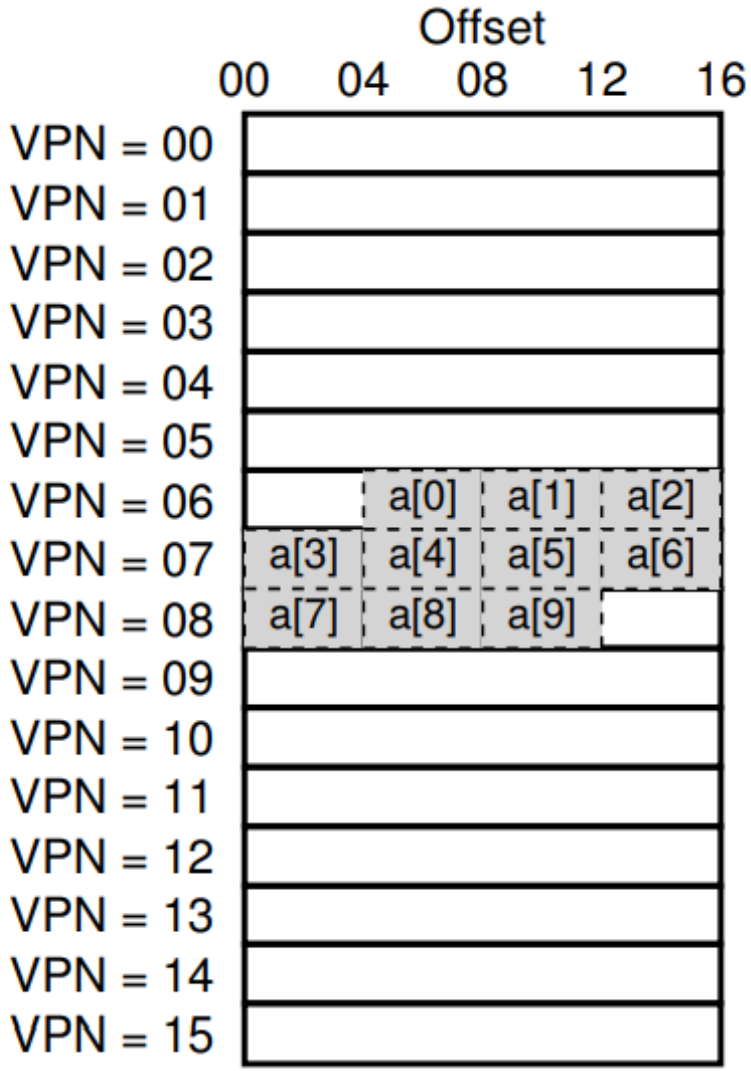

Figure 19.2: **Example: An Array In A Tiny Address Space**

Example: Accessing An Array

- TLB hit rate = 70%
- Sidenote:
  - Caching is a fundamental performance technique used throughout computer systems to make "the common case" fast.
  - Take advantage of locality (temporal and spatial)

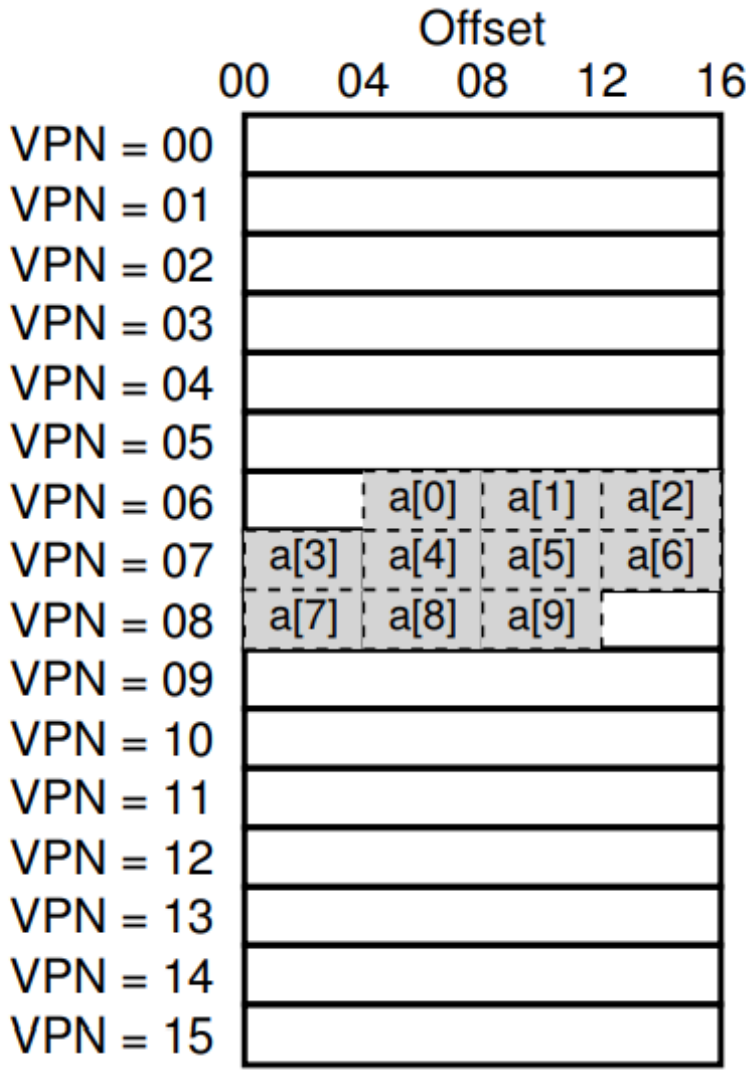

Figure 19.2: **Example: An Array In A Tiny Address Space**

Who Handles The TLB Miss?

- Is it the OS or hardware?
- Hardware Approach
  - Hardware must know exactly where the page tables are located.
  - On a miss, hardware walks the page table to update the TLB.
- OS Approach
  - On a miss, hardware raises privilege level to kernel mode and jumps to a trap handler that will update the TLB through privileged instructions.
  - Must ensure not to generate an infinite loop of TLB misses.
  - More flexible and hardware is simple

Who Handles The TLB Miss?

```
1  VPN = (VirtualAddress & VPN_MASK) >> SHIFT
2  (Success, TlbEntry) = TLB_Lookup(VPN)
3  if (Success == True)   // TLB Hit
4      if (CanAccess(TlbEntry.ProtectBits) == True)
5          Offset   = VirtualAddress & OFFSET_MASK
6          PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
7          Register = AccessMemory(PhysAddr)
8      else
9          RaiseException(PROTECTION_FAULT)
10 else                    // TLB Miss
11     RaiseException(TLB_MISS)
```

Figure 19.3: **TLB Control Flow Algorithm (OS Handled)**

TLB Contents: What's In There?

- Typically, a TLB might have 32, 64, or 128 entries and be fully associative.
- TLB entry might look like this:
  - VPN | PFN | other bits
  - Must have VPN because it could be anywhere in the cache.
  - Other bits are valid, protection, dirty bit, etc.

TLB Issue: Context Switches

- TLB contents is only valid for the current running process. On a context switch, it becomes invalid. We must be careful.
- For example, if we have process P1 running.
  - P1 might have a mapping of VPN 10 -> PFN 100
- Now P1 is context switched out for P2.
  - P2 might have a mapping of VPN 10 -> PFN 170

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

TLB Issue: Context Switches

- How do we solve this issue?
- Obvious simple approach...flush the TLB on context switch.
  - Software approach, perhaps an instruction to flush the TLB.
  - Hardware approach, perhaps flush when the page-table base register changes.
- This works but if we context switch often, we may have a high TLB miss rate.
  - Some systems add hardware support to share the TLB with an address space identifier.
  - Hardware must also know which process is running.

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

TLB Issue: Context Switches

- In the case of sharing pages, maybe our TLB looks like this.

VPN	PFN	valid	prot	ASID
10	101	1	r-x	1
—	—	0	—	—
50	101	1	r-x	2
—	—	0	—	—

Issue: Replacement Policy

- Must consider how we do cache replacement when the cache is full.
- Look at this in more detail in the swapping pages chapters but we might use:
  - Least Recently Used (LRU)
  - Random: Works better in some edge cases like looping over n + 1 pages

A Real TLB Entry (MIPS R4000)

- 32-bit address space with 4KB pages.
  - 20-bit VPN and 12-bit offset
  - TLB is only 19-bit VPN because user addresses are limited to half the address space. Kernel reserves the other half.
  - VPN translates to up to 24-bit PFNs for a max of 64GB of RAM
- Global bit (G) to ignore ASID
- 8-bit ASID (What if running more than 256 processes?)

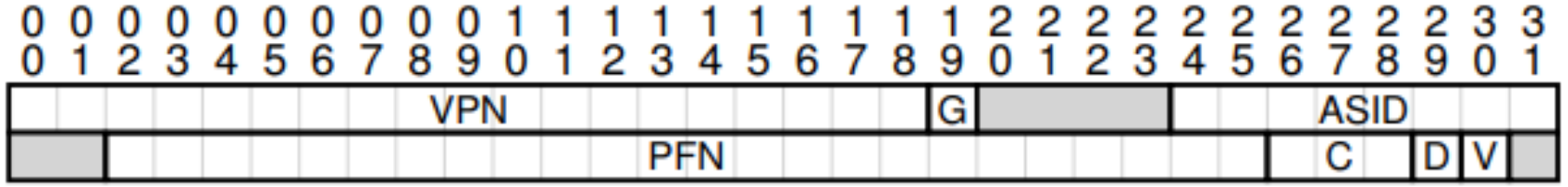

Figure 19.4: **A MIPS TLB Entry**

A Real TLB Entry (MIPS R4000)

- MIPS TLB usually have 32 or 64 entries with a few reserved for the OS.
- A "wired" register can be set by the OS to reserve slots in the TLB.
- MIPS TLB is software managed thus the OS uses specific instructions to manipulate the cache.

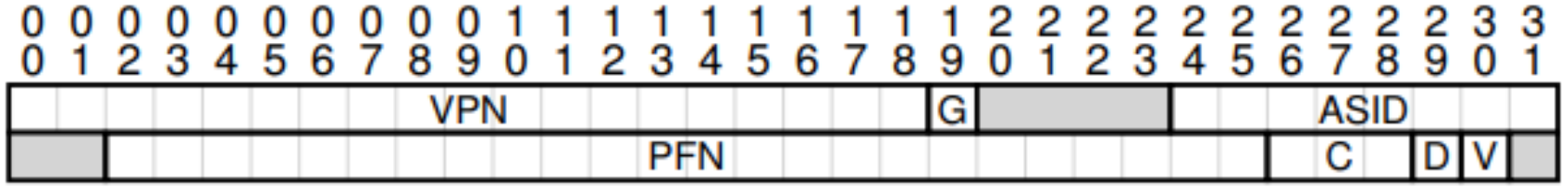

Figure 19.4: **A MIPS TLB Entry**