A Typical Memory Hierarchy

Processor

Control

Datapath

On-Chip Level One Cache $L_1$

Managed by Hardware

Managed by OS with hardware assistance

Virtual Memory, Secondary Storage (Disk)

Main Memory (DRAM)

Second Level Cache (SRAM) $L_2$

Second Level Cache

Virtual Memory, Secondary Storage

Tertiary Storage (Tape)

Faster

Larger Capacity

Speed (ns): 1s 10s 100s 10,000,000s (10s ms) 10,000,000,000s (10s sec)

Size (bytes): 100s Ks Ms Gs Ts

Virtual memory Sources:

Textbook Chapters 5.10, 5.11


Virtual Memory

- Motivation
- Paging Versus Segmentation
- Virtual Memory Basic Strategies
- Basic Virtual Memory Management
- Virtual Address Translation: Basic Page Tables
- Speeding-up Translations: Translation Lookaside Buffer (TLB)
  - TLB-Refill Mechanisms
- Global Vs. Per-process Virtual Address Space
- Data/Code Sharing in Virtual Memory Systems
- Address-Space Protection in Virtual Memory Systems
- Page Table Organizations and Page Table Walking
  - Direct Page Tables.
  - Hierarchical Page Tables
  - Inverted/Hashed Page Tables
- The Hardware (MMU) Software (OS) Mismatch In Virtual Memory
- Virtual Memory Architecture Examples
### Virtual Memory

- **Original Motivation:**
  - Illusion of having **more** physical main memory
  - Allows program and data address **relocation** by automating the process of code and data movement between main memory and secondary storage.

- **Additional Current Motivation:**
  - Fast process start-up.
  - **Protection** from illegal memory access.
    - Needed for multi-tasking operating systems.
  - Controlled code and data **sharing** among processes.
Virtual Memory

• Virtual memory controls two levels of the memory hierarchy:
  • Main memory (DRAM).
  • Mass storage (usually magnetic disks).

• Main memory is divided into blocks allocated to different running processes in the system by the OS:
  • Fixed size blocks: Pages (size 4k to 64k bytes). (Most common)
  • Variable size blocks: Segments (largest size $2^{16}$ up to $2^{32}$).
  • Paged segmentation: Large variable/fixed size segments divided into a number of fixed size pages (X86, PowerPC).

• At any given time, for any running process, a portion of its data/code is loaded in main memory while the rest is available only in mass storage.

• A program code/data block needed for process execution and not present in main memory result in a page fault (address fault) and the page has to be loaded into main memory by the OS from disk (demand paging).

• A program can be run in any location in main memory or disk by using a relocation/mapping mechanism controlled by the operating system which maps the address from virtual address space (logical program address) to physical address space (main memory, disk).
Paging Versus Segmentation

Fixed-size blocks (pages)

- Paging

Variable-size blocks (segments)

- Segmentation

<table>
<thead>
<tr>
<th></th>
<th>Page</th>
<th>Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Words per address</td>
<td>One</td>
<td>Two (segment and offset)</td>
</tr>
<tr>
<td>Programmer visible?</td>
<td>Invisible to application programmer</td>
<td>May be visible to application programmer</td>
</tr>
<tr>
<td>Replacing a block</td>
<td>Trivial (all blocks are the same size)</td>
<td>Hard (must find contiguous, variable-size, unused portion of main memory)</td>
</tr>
<tr>
<td>Memory use inefficiency</td>
<td>Internal fragmentation (unused portion of page)</td>
<td>External fragmentation (unused pieces of main memory)</td>
</tr>
<tr>
<td>Efficient disk traffic</td>
<td>Yes (adjust page size to balance access time and transfer time)</td>
<td>Not always (small segments may transfer just a few bytes)</td>
</tr>
</tbody>
</table>
Virtual Address Space Vs. Physical Address Space

Virtual memory stores only the most often used portions of an address space in main memory and retrieves other portions from a disk as needed (demand paging).

The virtual-memory space is divided into pages identified by virtual page numbers (VPNs), shown on the far left, which are mapped to page frames or physical page numbers (PPNs) or page frame numbers (PFNs), shown on the right.

Virtual address to physical address mapping or translation
### Typical Parameter Range For Cache & Virtual Memory

<table>
<thead>
<tr>
<th>Parameter</th>
<th>First-level cache</th>
<th>Virtual memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block (page) size</td>
<td>16–128 bytes</td>
<td>4096–65,536 bytes</td>
</tr>
<tr>
<td>Hit time</td>
<td>1–2 clock cycles</td>
<td>40–100 clock cycles</td>
</tr>
<tr>
<td>Miss penalty (Access time)</td>
<td>8–100 clock cycles</td>
<td>700,000–6,000,000 clock cycles</td>
</tr>
<tr>
<td>(Transfer time)</td>
<td>(6–60 clock cycles)</td>
<td>(500,000–4,000,000 clock cycles)</td>
</tr>
<tr>
<td></td>
<td>(2–40 clock cycles)</td>
<td>(200,000–2,000,000 clock cycles)</td>
</tr>
<tr>
<td>Miss rate</td>
<td>0.5–10%</td>
<td>0.00001–0.001%</td>
</tr>
<tr>
<td>Data memory size</td>
<td>0.016–1MB</td>
<td>16–8192 MB</td>
</tr>
</tbody>
</table>
Virtual Memory Basic Strategies

- **Main memory page placement:** Fully associative placement is used to lower the miss rate.

- **Page replacement:** The least recently used (LRU) page is replaced when a new block is brought into main memory from disk.

- **Write strategy:** Write back is used and only those pages changed in main memory are written to disk (dirty bit scheme is used).

- **Page Identification and address translation:** To locate pages in main memory a page table is utilized to translate from virtual page numbers (VPNs) to physical page numbers (PPNs). The page table is indexed by the virtual page number and contains the physical address of the page.
  - In paging: Offset is concatenated to this physical page address.
  - In segmentation: Offset is added to the physical segment address.

- Utilizing address translation locality, a translation look-aside buffer (TLB) is usually used to cache recent address translations (PTEs) and prevent a second memory access to read the page table.
Virtual to Physical Address Translation

virtual page number (VPN)

physical page numbers (PPN) or page frame numbers (PFN)

Paging is assumed
Virtual → Physical Address Translation

Virtual memory

Contiguous virtual address space of a program

Virtual address:
0
4K
8K
12K

A
B
C
D

Physical address:
0
4K
8K
12K
16K
20K
24K
28K

C
A
B
D

Physical main memory

Disk

Physical location of blocks A, B, C

Page Fault: D in Disk
(not allocated in main memory)

Paging is assumed
Basic Mapping Virtual Addresses to Physical Addresses Using A Page Table

Paging is assumed

Virtual address

Virtual page number

Page offset

VPN

Page Table Entry (PTE)

Main memory

Physical address

Physical Page Number (PPN)

Page table
Virtual to Physical Address Translation:

Page Tables

- Mapping information from virtual page numbers (VPNs) to physical page numbers is organized into a page table which is a collection of page table entries (PTEs).
- At the minimum, a PTE indicates whether its virtual page is in memory, on disk, or unallocated and the PPN if the page is allocated.
- Over time, virtual memory evolved to handle additional functions including data sharing, address-space protection and page level protection, so a typical PTE now contains additional information such as:
  - The page’s location in memory (page frame number, PFN) or location on disk (for example, an offset into a swap file);
  - The ID of the page’s owner (the address-space identifier (ASID), sometimes called Address Space Number (ASN) or access key);
  - The virtual page number (VPN);
  - A valid bit, which indicates whether the PTE contains a valid translation;
  - A reference bit, which indicates whether the page was recently accessed;
  - A modify bit, which indicates whether the page was recently written; and
  - Page-protection bits, such as read-write, read only, kernel vs. user, and so on.
Basic Virtual Memory Management

- Operating system makes decisions regarding which virtual pages should be allocated in real physical memory and where.

- On memory access -- If no valid page table entry (PTE)
  - Page fault to operating system
  - Operating system requests page from disk
  - Operating system chooses page for replacement
    - writes back to disk if modified
  - Operating system updates page table w/ new page table entry.
Virtual Memory Terms

- **Page Table Walking:** The process of searching the page table for the translation PTE.

- **Allocated or Mapped Virtual Page:** The OS has mapping information on its location (in memory or on disk) using its PTE in the page table.

- **Unmapped Virtual Page:** A page that either not yet been allocated or has been deallocated and its mapping information (PTE) has been discarded.

- **Wired down virtual page:** A virtual page for which space is always allocated in physical memory and not allowed to be paged out to disk.

- **Virtual Address Aliasing:** Mapping of two or more virtual pages to the same physical page to allow processes or threads to share memory
  - Provides threads with different “views” of data with different protections

- **Superpages:** A superpage contains a number of contiguous physical memory pages but require a single address translation. A number of virtual memory architectures currently support superpages.

- **Memory Management Unit (MMU):** Hardware mechanisms and structures to aid the operating system in virtual memory management including address generation/translation, sharing, protection. Special OS privileged ISA instructions provide software/OS access to this support. (e.g. TLBs, special protected registers)
Page Table Organizations

Direct Page Table:

- When address spaces were much smaller, a single-level table—called a *direct table* mapped the entire virtual address space and was small enough to be contained in SRAM and maintained entirely in hardware.
- As address spaces grew larger, the table size grew to the point that system designers were forced to move it into main memory.

**Limitations:**

- **Translation requires a main memory access:**
  - Solution: Speedup translation by caching recently used PTEs in a *Translation Lookaside Buffer (TLB).*
- **Large size of direct table:**
  - Example: A 32 bit virtual address with $2^{12} = 4k$ byte pages and 4 byte PTE entries requires a direct page table with $2^{20} = 1M$ PTEs and occupies 4M bytes in memory.
  - **Solution:** Alternative page table organizations:
    - Hierarchical page tables
    - Inverted or hashed page tables
Two memory accesses needed:

- First to page table.
- Second to item.

Page table usually in main memory.
Virtual Address Translation Using A Direct Page Table

Virtual page number (VPN)

Valid

Page table
Physical page or disk address

Allocated in physical memory

Physical memory

PTEs

Disk storage

Page Faults

Paging is assumed
Speeding Up Address Translation:
Translation Lookaside Buffer (TLB)

• Translation Lookaside Buffer (TLB): Utilizing address reference locality, a small on-chip cache used for address translations (PTEs).
  – TLB entries usually 32-128
  – High degree of associativity usually used
  – Separate instruction TLB (I-TLB) and data TLB (D-TLB) are usually used.
  – A unified larger second level TLB is often used to improve TLB performance and reduce the associativity of level 1 TLBs.

• If a virtual address is found in TLB (a TLB hit), the page table in main memory is not accessed.

• TLB-Refill: If a virtual address is not found in TLB, a TLB miss occurs and the system must search (walk) the page table for the appropriate entry and place it into the TLB this is accomplished by the TLB-refill mechanism.

• Types of TLB-refill mechanisms:
  – Hardware-managed TLB: A hardware state machine is used to refill the TLB on a TLB miss by walking the page table. (PowerPC, IA-32)
  – Software-managed TLB: TLB refill handled by the operating system. (MIPS, Alpha, UltraSPARC, HP PA-RISC, …)
Translation Lookaside Buffer (TLB)

Paging is assumed
**Operation of The Alpha 21264 Data TLB (DTLB) During Address Translation**

Virtual address

- Address space number
- Virtual page number
- Page offset
- 8Kbytes pages

**Protection Permissions**

- Valid bit

**Address Space Number (ASN)**
- Identifies process similar to PID (no need to flush TLB on context switch)

**Logical Address**

- (VPN)
- (PPN)

**Physical Address**

- DTLB = 128 entries
- (Low-order 13 bits of address)
- (High-order 31/28 bits of address)

**Diagram Additions**

- 44- or 41-bit physical address

**Notes**

- (no need to flush TLB on context switch)
A Memory Hierarchy With TLB & Two Levels of Cache

TLB: Direct Mapped 256 entries
L1 direct mapped 8KB
L2 direct mapped 4MB
Virtual address 64 bits
Physical address 41 bits
Basic TLB & Cache Operation

TLB Operation

Virtual address

TLB access

No

TLB hit?

Yes

Physical address

Write?

Yes

Normal Cache operation

Try to read data from cache

No

Cache miss stall

Yes

Cache hit?

Deliver data to the CPU

No

Write access bit on?

Yes

Write data into cache, update the tag, and put the data and the address into the write buffer

No

Write protection exception

TLB miss
use page table

TLB Refill

Cache is physically-addressed
CPU Performance with Real TLBs

When a real TLB is used with a TLB miss rate and a TLB miss penalty (time needed to refill the TLB) is used:

\[ CPI = CPI_{\text{execution}} + \text{mem stalls per instruction} + \text{TLB stalls per instruction} \]

Where:

\[ \text{Mem Stalls per instruction} = \text{Mem accesses per instruction} \times \text{mem stalls per access} \]

Similarly:

\[ \text{TLB Stalls per instruction} = \text{Mem accesses per instruction} \times \text{TLB stalls per access} \]

\[ \text{TLB stalls per access} = \text{TLB miss rate} \times \text{TLB miss penalty} \]

Example:

Given: \( CPI_{\text{execution}} = 1.3 \) Mem accesses per instruction = 1.4

Mem stalls per access = 0.5 TLB miss rate = 0.3% TLB miss penalty = 30 cycles

What is the resulting CPU CPI?

Mem Stalls per instruction = 1.4 \times 0.5 = 0.7 \text{ cycles/instruction}

TLB stalls per instruction = 1.4 \times (\text{TLB miss rate} \times \text{TLB miss penalty})

= 1.4 \times 0.003 \times 30 = 0.126 \text{ cycles/instruction}

CPI = 1.3 + 0.7 + 0.126 = 2.126
### Event Combinations of Cache, TLB, Virtual Memory

<table>
<thead>
<tr>
<th>Cache</th>
<th>TLB</th>
<th>Virtual Memory</th>
<th>Possible?</th>
<th>When?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hit</td>
<td>Hit</td>
<td>Hit</td>
<td>TLB/Cache Hit</td>
<td></td>
</tr>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Hit</td>
<td>Possible, no need to check page table</td>
<td></td>
</tr>
<tr>
<td>Hit</td>
<td>Miss</td>
<td>Hit</td>
<td>TLB miss, found in page table</td>
<td></td>
</tr>
<tr>
<td>Miss</td>
<td>Miss</td>
<td>Hit</td>
<td>TLB miss, cache miss</td>
<td></td>
</tr>
<tr>
<td>Miss</td>
<td>Miss</td>
<td>Miss</td>
<td>Page fault</td>
<td></td>
</tr>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Miss</td>
<td>Impossible, cannot be in TLB if not in main memory</td>
<td></td>
</tr>
<tr>
<td>Hit</td>
<td>Hit</td>
<td>Miss</td>
<td>Impossible, cannot be in TLB or cache if not in main memory</td>
<td></td>
</tr>
<tr>
<td>Hit</td>
<td>Miss</td>
<td>Miss</td>
<td>Impossible, cannot be in cache if not in memory</td>
<td></td>
</tr>
</tbody>
</table>
TLB-Refill Mechanisms

(Handle TLB miss or fault)

**Hardware-managed TLB** (ex. PowerPC, Intel IA-32):

- Typical of early memory-management units (MMUs).
- A hardware state machine is used to refill the TLB.
- In the event of a TLB miss, the state machine would walk the page table, locate the mapping, insert it into the TLB, and restart the computation.

- **Advantage: Performance**
  - Disturbs the processor pipeline only slightly. When the state machine handles a TLB miss, the processor stalls faulting instructions only. Compared to taking an interrupt, the contents of the pipeline are unaffected, and the reorder buffer need not be flushed.

- **Disadvantage: Inflexibility of Page table organization design**
  - The page table organization is effectively fixed in the hardware design; the operating system has no flexibility in choosing a design.
TLB-Refill Mechanisms

Software-managed TLB: (ex. MIPS, Alpha, UltraSPARC, HP PA-RISC...)

- Typical of recent memory-management units (MMUs). No hardware TLB-refill state machine to handle TLB misses.
- On a TLB miss, the hardware interrupts the operating system and vectors to a software routine that walks the page table and refills the TLB.
- **Advantage: Flexibility** of Page table organization design
  - The page table can be defined entirely by the operating system, since hardware never directly manages the table.
- **Disadvantage: Performance cost.**
  - The TLB miss handler that walks the page table is an operating system primitive which usually requires 10 to 100 instructions.
  - If the handler code is not in the instruction cache at the time of the TLB miss exception, the time to handle the miss can be much longer than in the hardware walked scheme.
  - In addition, the use of precise exception handling mechanisms adds to the cost by flushing the pipeline, removing a possibly large number of instructions from the reorder buffer. This can add hundreds of cycles to the overhead of walking the page table by software.
Virtual Memory Architectures
Global Vs. Per-process Virtual Address Space

• **Per-process virtual address space:**
  - The effective or logical virtual address generated by a process is extended by an address-space identifier (ASID) included in TLB and page table entries (PTEs) to distinguish between processes or contexts.
  - Each process may have a separate page table to handle address translation.
  - e.g. MIPS, Alpha, PA-RISC, UltraSPARC.

• **Global system-wide virtual address space:**
  - The effective or logical virtual address generated by a process is extended by a segment number forming a global, flat or extended global virtual address. (paged segmentation)
  - Usually a number of segment registers specify the segments assigned to a process.
  - A global page translation table may be used for all processes running on the system.
  - e.g. IA-32 (x86), PowerPC.
Data/Code Sharing in Virtual Memory Systems

• Shared memory allows multiple processes to reference the same physical code and data possibly using different virtual addresses
  
1 Global sharing of data can be accomplished by a global access bit in TLBs, PTEs.
   – For per-process virtual address space using address-space identifiers (ASIDs), the hardware ASID match check is disabled.

2 Sharing at the page level is accomplished by virtual address aliasing, where two or more virtual pages are mapped the same physical page with possibly different protections.
   – Disadvantage: Increases overheads of updating multiple PTEs every time the OS changes a page’s physical location.
   – Used in Unix-based OSs.

3 In systems that support a global virtual address (using paged segmentation), sharing at the segment level can be accomplished by assigning two or more processes the same segment number.
Address-Space Protection in Virtual Memory Systems

• **Per-process virtual address space systems using ASIDs:**
  - The OS places the running process’s address-space identifier (ASID) in a protected register, and every virtual address the process generates is concatenated with the address-space identifier.
  - Each process is **unable** to produce addresses that mimic those of other processes, because to do so it must control the contents of the protected register (OS access only) holding the active ASID.

• **Global virtual address space using using paged segmentation:**
  - A process address space is usually composed of many segments, the OS maintains a set of segment identifiers for each process.
  - The hardware can provide registers to hold the process’s segment identifiers, and if those registers can be modified only by the OS, the segmentation mechanism also provides address-space protection.
Alternative Page Table Organizations: Hierarchical page tables

- **Partition the page table into two or more levels:**
  - Based on the idea that a large data array can be mapped by a smaller array, which can in turn be mapped by an even smaller array.
  - For example, the DEC Alpha supports a four-tiered hierarchical page table composed of Level-0, Level-1, Level-2, and Level-3 tables.

- **Highest level(s) typically locked (wired down) in physical memory**
  - Not all higher level tables have to be resident in physical memory or even have to initially exist.

- **Hierarchical page table Walking (access or search) Methods:**
  - Top-down traversal (e.g. IA-32)
  - Bottom-up traversal (e.g. MIPS, Alpha)
Example:

Two-Level Hierarchical Page tables

- Assume 32-bit virtual addresses, byte addressing, and 4-Kbyte pages, the 4-Gbyte address space is composed of 1,048,576 \(2^{20}\) pages.
- If each of these pages is mapped by a 4-byte PTE, we can organize the \(2^{20}\) PTEs into a 4-Mbyte linear structure composed of 1,024 \(2^{10}\) pages, which can be mapped by a first level or root table with 1,024 PTEs.
- Organized into a linear array, the first level table with 1,024 PTEs occupy 4 Kbytes.
  - Since 4 Kbytes is a fairly small amount of memory, the OS wires down this root-level table in memory while the process is running.
  - Not all the highest page level (level two here also, referred to as the user page table) have to be resident in physical memory or even have to initially exist.
  - As, shown on next page.
Example:
Two-Level Hierarchical Page tables

32-bit 4-GByte virtual addresses, 4-Kbyte pages, 4-byte PTEs

Typically, the root page table is wired down in the physical memory while the process is running. The user page table (lowest level table) is paged in and out of main memory as needed.
Hierarchical Page Table Walking Methods

**Top-down traversal or walking:** (e.g. IA-32)
- Example for the previous two-level tables:

![Diagram of hierarchical page table walking]

**Disadvantage:** The top-down page table walking method requires as many memory references as there are table levels.
Hierarchical Page Tables Walking Methods

**Bottom-up traversal or table walking** (e.g. MIPS, Alpha)

- A bottom-up traversal lowers memory access overhead and typically accesses memory only once to translate a virtual address.

- For the previous two-level tables example:
  - **Step 1:**
    - The top 20 bits (virtual page number) of a TLB faulting virtual address are concatenated with the virtual offset of the user page table (level two).
    - The virtual page number of the faulting address is equal to the PTE index in the user page table. Therefore this virtual address points to the appropriate user PTE.
    - If a load using this address succeeds, the user PTE is placed into the TLB and can translate the faulting virtual address.
    - The user PTE load can, however, cause a TLB miss of its own. Requiring step 2
  - **Step 2:**
    - Perform top-down traversal or walking.
Bottom-up Table Walking Example

- The bottom-up method for accessing the hierarchical page table typically accesses memory only once to translate a virtual address (Step 1).
- It resorts to a top-down traversal if the initial attempt fails (Step 2).
Inverted/Hashed Page Tables (PowerPC, PA-RISC)

- Instead of one PTE entry for every virtual page belonging to a process, the inverted page table has one entry for every page frame in main memory.
  - The index of the PTE in the inverted table is equal to the page frame number (PFN) of the page it maps.
  - Thus, rather than scaling with the size of the virtual space, it scales with the size of physical memory.

- Since the physical page frame number is not usually available, the inverted table uses a hashed index based on the virtual page number (Typically XOR of upper and lower bits of virtual page number)

- Since different virtual page numbers might produce identical hash values, a collision-chain mechanism is used to let these mappings exist in the table simultaneously.
  - In the classical inverted table, the collision chain resides within the table itself.
  - When a collision occurs, the system chooses a different slot in the table and adds the new entry to the end of the chain. It is thus possible to chase a long list of pointers while servicing a single TLB miss.

- Disadvantage: The inverted table only contains entries for virtual pages actively occupying physical memory. An alternate mapping structure is required to maintain information for pages on disk.
Reducing Chain Length in Inverted Page Tables

Increase Size of Inverted Page Table:

- To keep the average chain length short, the range of hash values produced can be increased and thus increasing the size of the hash table.
- However, if the inverted page table’s size were changed, the page frame number (PFN) could no longer be deduced from the PTE’s location within the table.
- It would then be necessary to explicitly include the page frame number in the PTE, thereby increasing the size of every PTE.

Hash Anchor Table (HAT):

- As a trade-off to keep the table small, the designers of early systems increased the number of memory accesses per lookup:
  - They added a level of indirection, the hash anchor table (HAT).
- The hash anchor table is indexed by the hash value and points to the chain head in the inverted table corresponding to each value.
- Doubling the size of the hash anchor table reduces the average collision-chain length by half, without having to change the size of the inverted page table.
- Since the entries in the hash anchor table are smaller than the entries in the inverted table, it is more memory efficient to increase the size of the hash anchor table to reduce the average collision-chain length.
Inverted/Hashed Page Table

- The inverted page table contains one PTE for every page frame in memory, making it densely packed compared to the hierarchical page table.
- It is indexed by a hash of the virtual page number.

With Hash Anchor Table (HAT)
Table Walking Algorithm For Inverted Page Table With Hash Anchor Table (HAT)

Step 1:
• The TLB faulting virtual page number is hashed, indexing the hash anchor table.
• The corresponding anchor-table entry is loaded and points to the chain head for that hash value.

Step 2:
• The indicated PTE is loaded, and its virtual page number is compared with the faulting virtual page number. If the two match, the algorithm terminates.

Step 3a:
• The mapping, composed of the virtual page number and the page frame number (the PTE’s index in the inverted page table), is placed into the TLB.

Step 3b:
• Otherwise, the PTE references the next entry in the chain (step 3b), or indicates that it is the last in the chain. If there is a next entry, it is loaded and compared.
• If the last entry fails to match, the algorithm terminates and causes a page fault.
Table Walking Algorithm For Inverted Page Table

Faulting virtual address

Virtual page number  Page offset

Physical address of hash anchor table entry

Base address of hash anchor table  Hash anchor table index

Hash

Step 1

Hash anchor table entry

Inverted page table index  Flags

Physical address of page table entry

Base address of inverted page table index  Inverted page table index

00 (2 bits)

Step 2

Page table entry

Virtual page number  Index of next PTE  Flags/protection

Next PTE in chain

Loaded into TLB

Virtual page number

Inverted page table index  Flags/protection

Step 3a

Step 3b
The Hardware (MMU)/Software (OS) Mismatch In Virtual Memory

- Most modern processors have hardware to support virtual memory in terms Memory Management Units (MMUs) including TLBs, special registers.
- Unfortunately, there has not been much agreement on the form that this support should take.
- No serious attempts have been made to create a common memory-management support or a standard interface to the OS.
- The result of this lack of agreement is that hardware mechanisms are often completely incompatible in terms of:
  - Hardware support for Global or Per-process Virtual Address Space
  - Protection and data/code sharing mechanisms.
  - Page table organization supported by hardware.
  - TLB-refill & page walking mechanisms
  - Hardware support for Global or Per-process Virtual Address Space.
  - Hardware support for superpages ….
- Thus, designers and porters of system-level software have two somewhat unattractive choices:
  - Write software to fit many different architectures, which can compromise performance and reliability; or
  - Insert layers of software to emulate a particular hardware interface, possibly compromising performance and compatibility
- Operating system developers often use only a small subset of the complete functionality of memory-management units (MMUs) to make OS porting more manageable.
## Virtual Memory Architecture Examples

### Table 1. Comparison of architectural support for virtual memory in six commercial MMUs.

<table>
<thead>
<tr>
<th>Item</th>
<th>MIPS</th>
<th>Alpha</th>
<th>PowerPC</th>
<th>PA-RISC</th>
<th>UltraSPARC</th>
<th>IA-32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address space protection</td>
<td>ASIDs</td>
<td>ASIDs</td>
<td>Segmentation</td>
<td>Multiple ASIDs</td>
<td>ASIDs</td>
<td>Segmentation</td>
</tr>
<tr>
<td>Shared memory</td>
<td>GLOBAL bit in TLB entry</td>
<td>GLOBAL bit in TLB entry</td>
<td>Segmentation</td>
<td>Multiple ASIDs; segmentation</td>
<td>Indirect specification of ASIDs</td>
<td>Segmentation</td>
</tr>
<tr>
<td>Large address spaces</td>
<td>64-bit addressing</td>
<td>64-bit addressing</td>
<td>52-/80-bit segmented addressing</td>
<td>96-bit segmented addressing</td>
<td>64-bit addressing</td>
<td>None</td>
</tr>
<tr>
<td>Fine-grained protection</td>
<td>In TLB entry</td>
<td>In TLB entry</td>
<td>In TLB entry</td>
<td>In TLB entry</td>
<td>In TLB entry; per segment</td>
<td></td>
</tr>
<tr>
<td>Page table support</td>
<td>Software-managed TLB</td>
<td>Software-managed TLB</td>
<td>Hardware-managed TLB; inverted page table</td>
<td>Software-managed TLB</td>
<td>Software-managed TLB</td>
<td>Hardware-managed TLB/hierarchical page table</td>
</tr>
<tr>
<td>Superpages</td>
<td>Variable page size set in TLB entry: 4 Kbyte to 16 Mbyte, by 4</td>
<td>Groupings of 8, 64, 512 pages (set in TLB entry)</td>
<td>Block address translation: 128 Kbytes to 256 Mbytes, by 2</td>
<td>Variable page size set in TLB entry: 4 Kbytes to 64 Mbytes, by 4</td>
<td>Variable page size set in TLB entry: 8, 64, 512 Kbytes, and 4 Mbytes</td>
<td>Segmentation/variable page size set in TLB entry: 4 Kbytes or 4 Mbytes</td>
</tr>
</tbody>
</table>

Source:
MIPS Virtual Memory Architecture

- OS handles TLB misses entirely in software.
- The hardware supports a bottom up hierarchical page table through the TLB context register.
- MIPS uses address-space identifiers (ASIDs) to provide address-space protection.
  - To access a page, the ASID of the currently active process must match the ASID in the page’s TLB entry.
- Periodic cache and TLB flushes are unavoidable, as there are only 64 unique context identifiers in the R2000/R3000 and 256 in the R10000.
  - This is because systems usually have more active processes than this, requiring sharing of address-space identifiers and periodic ASID remapping.
MIPS R10000 Address Translation Mechanism

64-bit virtual address

Region (Sign-extended) 32-bit virtual page number 12-bit page offset

8-bit ASID

Eight-entry instruction TLB, 64-paired-entry, fully associative main TLB

Cache index

32-Kbyte, two-way set-associative instruction or data cache

Tag compare (cache hit/miss)

28-bit page frame number

Tag: page frame number

Cache data
PowerPC 604 Virtual Memory Architecture

- The PowerPC 604, which maps a process’s logical/effective addresses onto a **global flat virtual address space using paged segmentation**.
- Segments are 256-Mbyte contiguous regions of virtual space, and 16 segments make up an application’s 4-Gbyte address space.
  - The top 4 bits of the 32-bit effective address select a segment identifier from a set of 16 hardware segment registers.
- The segment identifier is concatenated with the bottom 28 bits of the effective address to form an extended virtual address that indexes the caches and is mapped by the TLBs and page table.
- The PowerPC defines a **hashed page table** for the OS: a variation on the inverted page table that acts as an eight-way set-associative software cache for PTEs.
  - Similar to the classic inverted table, it requires a backup page table for maintain information for pages on disk
- **Hardware TLB-Refill**: On TLB misses, hardware walks the hashed page table.
- Address-space protection is supported through the segment registers, which can only be modified by the OS.
- The **segment identifiers** are 24 bits wide and can uniquely identify over a million processes.
- If shared memory is implemented through the segment registers, the OS will rarely need to remap segment identifiers.
PowerPC 604
Address Translation Mechanism

32-bit effective address
Seg# | 16-bit segment offset | 12-bit page offset

Segment registers (16 IDs x 24 bits)

52-bit virtual address

40-bit virtual page number

24-bit segment ID | 16-bit segment offset | 12-bit page offset

128-entry, two-way instruction TLB
128-entry, two-way data TLB

Tag compare (cache hit/miss)

20-bit page frame number

Tag: 20-bit page frame number

Cache data

Split L1 cache, (instruction and data) each 16-Kbyte, four-way set associative
PowerPC Hashed Page Table Structure

40-bit virtual page number

Hash function

Index into the hashed page table

$N$ PTE groups in the page table, where $N$ is power of 2 and at least half the number of physical pages

8 page table entries: a PTE group

8-bit PTE

Hashed page table

Physical memory
IA-32 (x86) Virtual Memory Architecture

- X86 another architecture which uses **paged segmentation**.
- The x86’s segmentation mechanism often goes unused by today’s OSs, which instead flush the TLBs on context switch to guarantee protection.
- The per-process hierarchical page tables are hardware defined and hardware-walked.
  - The OS provides to the hardware a physical address for the root page table in one of a set of control registers, CR3.
  - Hardware uses this address to walk the **two-tiered table** in a **top-down fashion** on every TLB miss.
- Unlike the PowerPC, the segmentation mechanism supports variable-sized segments from 1 byte to 4 Gbytes in size, and the global virtual space is the same size as an individual user-level address space (4 Gbytes).
- User-level applications generate 32-bit addresses that are extended by 16-bit segment selectors.
- Hardware uses the 16-bit selector to index one of two software descriptor tables, producing a base address for the segment corresponding to the selector.
- This base address is added to the 32-bit virtual address generated by the application to form a global 32-bit linear address.
- For performance, the hardware caches six of a process’s selectors in a set of on-chip segment registers that are referenced by context.
Intel Pentium II/III Address Translation Mechanism

- 48-bit logical address
  - 16-bit segment selector
  - 32-bit segment offset
  - Local and global descriptor tables
  - 32-bit linear address
    - 20-bit virtual page number
      - 16-entry, four-way instruction TLB
      - 64-entry, four-way data TLB
  - 12-bit page offset

- 32-entry, four-way instruction TLB
- 64-entry, four-way data TLB
- 16-Kbyte, four-way data cache
- 16-Kbyte, four-way instruction cache

- Tag compare (cache hit/miss)
- Page frame number
- Tag: page frame number
- Cache data
Alpha 21164 Address Translation Mechanism

- 64-bit virtual address
  - (Sign-extended)
  - 30-bit virtual page number
  - 13-bit page offset

- 7-bit address space register
  - ASID

- 48-entry, fully associative instruction TLB
  - 64-entry, fully associative data TLB

- 27-bit page frame number
  - Tag: 27-bit page frame number
  - Tag compare (cache hit/miss)

- 8-Kbyte, virtually indexed, physically tagged, direct-mapped data cache

- 8-Kbyte virtually-indexed, virtually-tagged direct-mapped instruction cache

- 30-bit virtual page number
  - 7-bit address space register
  - ASID

- Cache tag
  - ASID
  - 30-bit virtual page number

- Cache data