Computer Structure

Virtual Memory

Lecturer:
Aharon Kupershtok

Prepared by Lihu Rappoport
Virtual Memory

- Provide **isolation**: each process sees its own memory space
  - Many processes can run on a single machine
  - Prevents a process from accessing the memory of other processes

- Provides the illusion of a **large memory for each process**
  - Different machines have different amount of physical memory
    - Allows programs to run regardless of actual physical memory size
    - Sum of memory spaces of all process may be larger than physical memory

- Provides illusion of **contiguous memory**
  - The amount of memory consumed by each process is dynamic
  - Allows adding memory and keep it contiguous
Virtual Memory: Basic Idea

- **Basic terminology**
  - Virtual Address Space: address space used by the programmer
  - Physical Address: actual physical memory address space

- **Divide memory (virtual and physical) into fixed size blocks**
  - Pages in Virtual space, Frames in Physical space
  - Page size = Frame size = some power of 2

- **Virtual pages are mapped either to**
  - A physical frame in memory
  - Or a sector in the disk

- **All addresses in programs use Virtual Memory Address Space**
  - Hardware translates virtual to physical addresses on-the-fly
  - Uses a Page Table for the translation
Page Tables

• The OS manages a page table for each process
  – The OS maps the process virtual pages to physical pages or to disk
  – Only the OS writes into the page tables
  – The page tables reside in the physical memory (DRAM)
Virtual to Physical Address translation

Virtual Address

Virtual Page Number

Page offset

V D AC Phy. page #

Access Control
- Memory type
  (WB, WT, UC, WP ...)
- User / Supervisor

Physical Page Number

Page offset

Valid bit

Dirty bit

Page table base reg

PTE – Page Table Entry

Page size: $2^{12}$ byte = 4K byte
The page table of each process resides in main memory
- When a process is running, the start address of its page table is pointed by a special register in the CPU: the *page table base register*
- The *page table base register* holds the physical adders of entry 0

The physical address of the PTE of virtual page #VPN
- PTE address = Page table base reg + VPN × PTE_size
**Address Mapping Algorithm**

If \( V = 1 \) then

- page is in main memory at frame address stored in table
  - \( \Rightarrow \) Fetch data

else *(page fault)*

- need to fetch page from disk
  - \( \Rightarrow \) causes a trap, usually accompanied by a *context switch*:
    - current process suspended while page is fetched from disk

**Access Control (R = Read-only,  R/W = read/write,  X = execute only)**

- If kind of access not compatible with specified access rights then *protection_violation_fault*
  - \( \Rightarrow \) causes trap to hardware, or software fault handler

- **Missing item fetched from secondary memory only on the occurrence of a fault** \( \Rightarrow \) *demand load policy*
Page Faults

• Page fault
  – Data is not in memory ⇒ retrieve it from disk
  – The CPU detects the situation, but cannot remedy the situation
    ⇒ the CPU traps to the OS to remedy the situation
  – The OS
    ➢ If there is no free space in physical memory
      ▶ Picks a physical page to discard (based on some replacement policy)
      ▶ Marks the page as not present in the page table
      ▶ Copies the page to the disk swap area
    ➢ Loads the new page from disk into the selected physical page
    ➢ Updates the page table entry of the new page
    ➢ Resume to the program so HW retries and succeeds

• A page fault usually causes a context switch
  – Current process suspended while page is fetched from disk
Optimal Page Size

- **Minimize wasted storage**
  - Small page minimizes internal fragmentation
  - Small page increase size of page table

- **Minimize transfer time – use large pages (multiple disk sectors)**
  - Amortize access cost and prefetch useful data
  - But, might transfer unnecessary info and discard useful data early

- **General trend toward larger pages because**
  - Big cheap RAM
  - Increasing memory / disk performance gap
  - Larger address spaces
Translation Look aside Buffer (TLB)

- Page table resides in memory
  ⇒ each translation requires an extra memory access

- The TLB caches recently used PTEs
  - Typically 128 to 256 entries, 4 to 8 way associative

- TLB Indexing

```
<table>
<thead>
<tr>
<th>Virtual page number</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td>Set</td>
</tr>
</tbody>
</table>
```

- On A TLB miss
  - Page Miss Handler (HW PMH) gets PTE from memory

- OS is responsible to maintain coherency between page table and TLB
  - Each time OS writes to a PTE it must invalidate the PTE from the TLB (if exists)
The TLB Caches PTEs

The TLB is a cache for recent address translations:

- **Virtual page number**
- **TLB**
  - Valid
  - Tag
  - Physical Page
  - Physical Memory
- **Page Table**
  - Valid
  - Physical Page
  - Disk
  - Physical Page or Disk Address
TLB Access

Virtual page number  Offset

Tag      Set

Way 0  Way 1  Way 2  Way 3

Set#

Way 0  Way 1  Way 2  Way 3

Way MUX

PTE

Hit/Miss
Processor Caches

- L2 and L3 are unified, as the memory – hold data and instructions
- In case of STLB miss, PMH accesses the data cache for page walk
Virtual Memory And Cache

- TLB access is serial with cache access
- Page table entries are cached in L1 D$, L2$ and L3$ as data
Overlapped TLB & Cache Access

Virtual Memory view of a Physical Address

Cache view of a Physical Address

#Set is not contained within the Page Offset

- The #Set is not known until the physical page number is known
- Cache can be accessed only after address translation done
Overlapped TLB & Cache Access (cont)

Virtual Memory view of a Physical Address

Cache view of a Physical Address

In the above example  #Set is contained within the Page Offset

- The #Set is known immediately
- Cache can be accessed in parallel with address translation
- Once translation is done, match upper bits with tags

Limitation: Cache $\leq (\text{page size} \times \text{associativity})$
Overlapped TLB & Cache Access (cont)

- Virtual page number
- Page offset
- Tag
- Set
- Way MUX
- TLB
- Set#
- Physical page number
- Cache
- Way MUX
- Set#
- Hit/Miss
- Data
Overlapped TLB & Cache Access (cont)

- Assume a cache is 32K Byte, 2 way set-associative, 64 byte/line
  - \((2^{15}/ 2 \text{ ways}) / (2^6 \text{ bytes/line}) = 2^{15-1-6} = 2^8 = 256 \text{ sets}\)

- In order to still allow overlap between set access and TLB access

- Cache offset+set+tag must cover the entire Physical Address
  - Since bits PA[13:12] are not part of the set, they are covered in the tag
  - The tag is comprised of PA[38:12]
Overlapped TLB & Cache Access (cont)

- Example:

  - OS may map two virtual pages to the same physical page
    - Each one of these virtual pages may have different VA[13:12] bits
      - The same physical address is mapped into 2 different sets in the cache
      - Data in one copy may be different than the other
    - Solution: allow only one virtual alias in the cache at any given time
      - On a cache miss, before writing the missed entry to the cache, search for virtual aliases already in the cache and evict them first
      - No special work is necessary during a cache hit

- An external Snoop supplies only the physical address
  - With virtual indexing, all the possible sets must be snooped
Evicting a Page to Disk

- **OS updates the Page Table**
  - OS marks the swapped-out page as *not present* (valid=0) in its PTE
  - OS writes the page disk location into the PTE
  - OS invalidate the page from the TLBs
    - It is OS responsibility to maintain coherency between page table and TLBs for any change in the page table

- **OS code copies the page to the disk**
  - Read each block within the page
    - Snoop-invalidate each byte in the CPU caches (L1, L2, …), and if hits
      - If it is modified read its line from the cache to get updated data
      - Invalidate the line
  - Write the block to the disk controller Memory-Mapped I/O area
  - This means that when a page is swapped-out of memory
    - All data in the caches which belongs to that page is invalidated
    - The page in the disk is up-to-date
Context Switch

- Each process has its own address space
  - Each process has its own page table
  - When the OS allocates to a process pages in physical memory, it updates the process’s page table
  - A process cannot access physical memory allocated to another process
    - Unless the OS deliberately allocates the same physical page to two processes

- On a context switch
  - Save the current architectural state to a dedicated memory location
    - Architectural registers
    - Register that holds the page table base address in memory
  - Flush the TLB
  - Load the new architectural state from memory
    - Architectural registers
    - Register that holds the page table base address in memory
Page Aliasing

- OS may map two different virtual pages to the same phy. page

- DLLs in Linux/Windows
  - DLL pages are used by all processes, marked as read only
    ⇒ one process cannot change the code of another process
  - Same DLL may be loaded by a process twice
    - Two virtual pages of the same process may map to the same phy. page

- Large malloc
  - OS maps all allocated pages to the same all-zero read only physical page
  - In case of a write to one of these virtual pages
    - Access violation page fault
    - OS identifies the page mapped to the special 4K page
    - Allocates a “real” physical page (copy-on-write)

- Shared Memory
  - OS maps virtual pages of two processes to the same physical memory
Virtually-Addressed Cache

- Cache uses virtual addresses (tags are virtual)
- Address translation only done on a cache miss
  - TLB is not in the path to cache hit

- Two virtual pages mapped to the same physical page
  - Must not reside in the cache together
  - On a cache miss, use a reverse TLB to verify that no other cache line already in the cache mapped to the missed physical address

- Flush cache on context switch
  - Alternatively: add process ID as part of the tag
Paging in x86
x86 Paging – Virtual Memory

• A page can be
  – Loaded
  – On disk

• A loaded page can be
  – Dirty
  – Clean

• When a page is not loaded (P bit clear) ⇒ Page fault occurs
  – It may require throwing a loaded page to insert the new one
    ➢ OS prioritize throwing by LRU and dirty/clean/avail bits
    ➢ Dirty page should be written to Disk; Clean need not
  – New page is either loaded from disk or “initialized”
  – CPU sets the page “access” flag when accessed, “dirty” when written
Page Tables and Directories in 32bit Mode

- 32 bit mode supports both 4KB and 4MB pages
  - CR4.PSE = 1 (Page Size Extensions) enables using large page size

- CR3 points to the current Page Directory
  - Loaded per process

- Page directory
  - Holds 1024 page-directory entries (PDEs), each is 32 bits
  - Each PDE contains a PS (page size) flag
    - 0 – entry points to a page table whose entries point to 4KB pages
    - 1 – entry points directly to a 4MB page

- Page table
  - Holds up to 1024 page-table entries (PTEs), each is 32 bit
  - Each PTE points to a 4KB page in physical memory
32bit Mode: 4KB Page Mapping

- 2-level hierarchical mapping
  - Page directory and page tables
  - Pages / page tables are 4KB aligned
- CR3 points to the current Page Directory
- Upper 10 Linear addr. bits point to a PDE
  - PDE (Page Directory Entry) provides a 20 bit, 4KB aligned base physical address of a page table
- Next 10 Linear addr. bits point to a PTE within the given Page Table
  - PTE (Page Table Entry) provides a 20 bit, 4KB aligned, base physical address of a 4KB page
- Lowest 12 Linear Addr. bits provide offset within the selected 4KB page
  - Point to the data bytes
32bit Mode: 4MB Page Mapping

- PDE directly maps up to 1024 4MB pages
- Upper 10 Linear addr. bits point to a PDE
  - PS in the PDE = 1 ⇒ PDE provides a 10 bit, 4MB aligned, base physical address of a 4MB page
- Lowest 22 Linear addr. bits provide offset within selected 4MB page
  - Point to the data bytes
- Mixing 4KByte and 4MByte Pages
  - Separate TLBs for 4MB pages and 4KB pages
    - Often used code (e.g., kernels) is placed in a 4MB page ⇒ frees up 4KB TLB entries
    - Reduces TLB misses and improve overall system performance
PTE (4K-Bbyte Page) Format

- 20 bit pointer to a 4K page
- Virtual memory
  - Present
  - Accessed
  - Dirty
  - Global
- Protection
  - Writable (R#/W)
  - User / Supervisor #
    - 2 levels/type only
- Caching
  - Page WT
  - Page Cache Disabled
  - PAT – PT Attribute Index
- 3 bits available for OS usage

Present
Writable
User / Supervisor
Write-Through
Cache Disable
Accessed
Dirty
PT Attribute Index
Global
Available for OS Use

Page Frame Address 31:12
PDE (4K-Bbyte Page Table) Format

- 20 bit pointer to a 4K page table
- Virtual memory
  - Present
  - Accessed
  - Page size = 0
  - Global
- Protection
  - Writable (R#/W)
  - User / Supervisor #
  - 2 levels/type only
- Caching
  - Page WT
  - Page Cache Disabled
- 4 bits available for OS usage

Present
Writable
User / Supervisor
Write-Through
Cache Disable
Accessed
Available for OS Use
Page Size (0 – 4KB, 1 – 4MB)
Global
Available for OS Use

Page Frame Address 31:12

Available
Global
PM
U
W
D
C
L
A
V
G

31 12 11 9 8 7 6 5 4 3 2 1 0
PDE (4M-Bbyte Page) Format

- 10 bit pointer to a 4M page

Virtual memory
- Present
- Accessed
- Dirty
- Page size = 1
- Global

Protection
- Writable (R#/W)
- User / Supervisor #
  - 2 levels/type only

Caching
- Page WT
- Page Cache Disabled
- PAT – PT Attribute Index

3 bits available for OS usage
Page Table – Virtual Mem. Attributes

- **Present (P) flag**
  - When set: page is in physical memory
  - When clear, page not in memory $\Rightarrow$ processor generates a page-fault
    - Bits $[31:1]$ are available for OS use
  - The processor does not set/clear this flag $\Rightarrow$ OS maintains its state
  - In case of a page-fault, the OS performs the following operations
    1. Copy the page from disk into physical memory
    2. Load page address into PTE/PDE; set present=1, dirty=0, accessed=0
    3. Invalidate victim page from TLB
    4. Return from page-fault handler and restart the program

- **Page size (PS) flag, in PDEs only**
  - 0: The PDE points to a page table of 4KBytes pages
  - 1: The PDE points to a 4MB page
Page Table – Virtual Mem. Attributes

- **Accessed (A) flag and Dirty (D) flag**
  - Typically cleared by the OS when a page/PT initially loaded into physical mem
  - Processor sets the A-flag the first time a page/PT is accessed (read/write)
  - Processor sets the D-flag the first time a page is accessed for a write
    - The D-flag is not used in PDEs which point to page tables
  - Both A and D flag are sticky
    - Once set, the processor does not implicitly clear it – only software can clear it
  - Used by OS to manage transfer of pages/PTs tables into and out of physical memory

- **Global (G) flag**
  - Indicates a global page when set (+page global enable is set: CR4.PGE =1)
  - PTE/PDE in TLB not invalidated when CR3 is loaded / task switch
    - Prevents frequently used pages (e.g. OS) from being flushed from TLB
  - Only software can set or clear this flag
  - Ignored for PDEs that point to page tables (global att. of a page is set in PTEs)
Page Table – Caching Attributes

• Page-level write-through (PWT) flag
  – Controls the write-through or write-back caching policy of the page / PT
    ➢ 0: write-back caching, 1: write-through caching
  – Ignored if the CD (cache disable) flag in CR0 is set

• Page-level cache disable (PCD) flag
  – Controls the caching of individual pages/PT
  – 1: caching of the associated page/PT is prevented
    ➢ Used for pages that contain memory mapped I/O ports or that do not provide a performance benefit when cached
  – 0: the page/PT can be cached
  – Ignored (assumes as set) if the CD (cache disable) flag in CR0 is set

• Page table attribute index (PAT) flag
  – Used along with the PCD and PWT flags to select an entry in the PAT, which in turn selects the memory type for the page
Page Table – Protection Attributes

• **Read/write (R/W) flag**
  - Specifies the read-write privileges for a page or group of pages
    (in the case of a PDE that points to a page table)
  - 0: the page is read only
  - 1: the page can be read and written into

• **User/supervisor (U/S) flag**
  - Specifies the user-supervisor privileges for a page or group of pages
    (in case of a PDE that points to a page table)
  - 0: supervisor privilege level
  - 1: user privilege level
Page Directory Base Address

- The physical address of the page directory is stored in CR3 register
  - Also called the page directory base register (PDBR)

- PDBR is typically loaded with a new as part of a task switch

- OS must ensure that
  - The page directory indicated by the PDBR image in a task's TSS (task state segment) is present in physical memory before the task is dispatched
  - The page directory must also remain in memory as long as the task is active
Paging in 64 bit Mode

- Physical address size grows to \( M > 32 \) bits
  - Grow PTE from 4 bytes to 8 bytes to support \( > 32 \) bits Physical address
  - To keep each table 4KB \( \Rightarrow \) each table holds 512 entries (instead of 1K)
    \( \Rightarrow \) each table is indexed by 9 linear-address bits (instead of 10)
    \( \Rightarrow \) Large pages become \( 512 \times 4KB = 2MB \) (instead of 4MB)

- Linear address size becomes 48 bits
  - Two new levels page tables are added
    - *Page Directory Pointer Table* (PDP) – indexed by LA[38:30]
    - *Page map level 4 table* (PML4) – indexed by LA[47:39]
  - Each entry in PML4, PDP, DIR provides base address of next level table
    - \( \text{maxphyaddr} \) – 12 bits, 4KB aligned (for \( \text{maxphyaddr} = 40 \Rightarrow 28 \) bits)
4KB Page Mapping in 64 bit Mode

- 512 entry PML4 Table
- 512 entry Page Directory Pointer Table
- PML4 entry
  - M-12 (4KB aligned)
  - CR3 (PDPTR)
- 512 entry PDP Table
- PDP entry
  - M-12
- 512 entry Page Directory
- 512 entry PDE
  - M-12
- 512 entry Page Table
  - M-12
- 4KByte Page
  - data
2MB Page Mapping in 64 bit Mode

- **sign ext.**
- **PML4**
- **PDP**
- **DIR**
- **OFFSET**

- 512 entry PML4 Table
- 512 entry Page Directory Pointer Table
- 512 entry Page Directory
- 512 entry PDE
- 2MByte Page

- CR3 (PDPTR)
- M-12
- M-21
- 40 (4KB aligned)

- 2MB Page Mapping in 64 bit Mode
1GB Page Mapping in 64 bit Mode

Diagram showing the mapping of a 1GB page in 64-bit mode, with various directories and pointers involved.

- **sign ext.**
- **PML4**
- **PDP**
- **OFFSET**

Diagram includes:
- 512 entry PML4 Table
- PML4 entry
- 512 entry Page Directory Pointer Table
- PDP entry
- M-12
- M-30
- 1GB Page
- data

**CR3 (PDPTR)**

- **40 (4KB aligned)**
VM Example

8 bits instead of 9 for example purposes only.
## PTE/PDE/PDP/PML4 Entry Format

![Table showing PTE/PDE/PDP/PML4 Entry Format]

### Table

<table>
<thead>
<tr>
<th>CR3</th>
<th>Reserved</th>
<th>Address of PML4 table</th>
<th>Ignored</th>
<th>Address of page-directory-pointer table</th>
<th>Ignored</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>XD</td>
<td>Ignored</td>
<td>Rsvd.</td>
<td>Address of 1GB page frame</td>
<td>Reserved</td>
</tr>
<tr>
<td></td>
<td>Ignored</td>
<td>Rsvd.</td>
<td></td>
<td>Address of page directory</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ignored</td>
<td>Rsvd.</td>
<td></td>
<td>Address of 2MB page frame</td>
<td>Reserved</td>
</tr>
<tr>
<td></td>
<td>Ignored</td>
<td>Rsvd.</td>
<td></td>
<td>Address of page table</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ignored</td>
<td>Rsvd.</td>
<td></td>
<td>Address of 4KB page frame</td>
<td></td>
</tr>
</tbody>
</table>

### Diagram

- **PML4E**: present
- **PDE**: 1GB page
- **PDE**: 2MB page
- **PDE**: page directory
- **PTE**: 4KB page
- **PDPT**: page directory
- **PDPT**: 1GB page
- **PTE**: present
- **Reserved**
- **Ignored**
- **Address of PML4 table**
- **Address of page-directory-pointer table**
- **Address of 1GB page frame**
- **Address of page directory**
- **Address of 2MB page frame**
- **Address of page table**
- **Address of 4KB page frame**

### Columns

- **XD**: Valid/Invalid
- **Rsvd**: Reserved
- **G1**: Global/Local
- **DA**: Present/Absent
- **AV**: Accessed/Not Accessed
- **UP**: User/Privileged
- **UR**: Read/Not Read
- **PT**: Present/Transparent
- **P**: Present
Execute-Disable Bit

- Supported only with PAE enabled / 64 bit mode
  - Bit[63] in PML4 entry, PDP entry, PDE, PTE

- If the execute disable bit of a memory page is set
  - The page can be used only as data
  - An attempt to execute code from a memory page with the execute-disable bit set causes a page-fault exception
  - Setting the execute-disable bit at any level of the paging structure, protects all pages pointed from this entry
TLBs

- The most recently used PDEs and PTEs are cached in TLBs
  - Separate TLB for data and instruction caches
  - Separate TLBs for 4KB, 2/4MB and 1GB page sizes
  - TLB sizes in 4th Generation Intel® Core™ Processors:

<table>
<thead>
<tr>
<th></th>
<th>4KB pages</th>
<th>2MB/4MB Pages</th>
<th>1GB Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Instruction TLBs</strong></td>
<td>128 entries, 4 ways</td>
<td>8 entries</td>
<td>none</td>
</tr>
<tr>
<td><strong>Data TLBs</strong></td>
<td>64 entries, 4 ways</td>
<td>32 entries, 4 ways</td>
<td>4 entries, 4 ways</td>
</tr>
</tbody>
</table>

- In case of a hit in multiple TLBs
  - The largest page that hits is used
TLBs

- OS running at privilege level 0 can invalidate TLB entries
  - INVLPG instruction invalidates a specific PTE in the TLB
    - This instruction ignores the setting of the G flag
  - Whenever a PDE/PTE is changed (also when present flag is set to 0)
    - OS must invalidate the corresponding TLB entry
  - All (non-global) TLBs are automatically invalidated when CR3 is loaded

- The global (G) flag prevents frequently used pages from being automatically invalidated in on a task switch
  - The entry remains in the TLB indefinitely
  - Only INVLPG can invalidate a global page entry
PMH – Page Miss Handler

• In case of an iTLB or a dTLB miss
  – Request PTE from PMH
  – PMH first tries the STLB (2\textsuperscript{nd} level TLB); in case of an STLB hit
    ➢ STLB returns the PTE, and saves the page walk
    ➢ Starting at 4\textsuperscript{th} Gen. Core™, STLB also caches PDEs for 2M/4M pages

• In case of an STLB miss the PMH performs a page walk
  – It traverses the page table tree, starting at the root

• The PMH includes caches to shorten the page walk time

<table>
<thead>
<tr>
<th>cache</th>
<th>Accessed with VA bits</th>
<th>If hits, returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDE cache</td>
<td>[47:21]</td>
<td>PDE</td>
</tr>
<tr>
<td>PDP cache</td>
<td>[47:30]</td>
<td>PDP entry</td>
</tr>
<tr>
<td>PML4 cache</td>
<td>[47:39]</td>
<td>PML4 entry</td>
</tr>
</tbody>
</table>

• PML4 Cache is accessed with VA[47:39], and in case of a hit
  – Returns PML4 entry, saves accessing PML4 table
  – Still need to access the PDP table and the PDE table
PMH – Page Miss Handler

• PDP Cache is accessed with VA[47:30]
  – Includes both the bits used for accessing PML4 table and the PDP table
  – PDP Cache hit ⇒ returns directly the PDP entry (still need PDE table access)
    ➢ Saving both the PML4 table access and the PDP table access
      ▶ Each of R/W flag and U/S flag are logical AND of their values in PML4 and PDP
      ▶ XD flag is logical OR of PML4 and PDP values
      ▶ The values of the PCD and PWT flags are taken for the PDP entry

• PDE Cache is accessed with VA[47:21]
  – Includes the bits used for accessing PML4 table, PDP table, and PDE table
  – PDP Cache hit ⇒ returns directly the PDE entry
    ➢ Saving the accesses to PML4 table, PDP table, and PDE table

• All 3 caches are accessed in parallel
  – Use hit from the lowest table that hits (which saves the most)

• When the PMH accesses a page table during the page walk
  – It issues a “load” to the page table entry address, which as other loads
    ➢ It first goes to the D$, then to the L2$, then the L3$, and then to memory
PMH – Page Miss Handler

No

PML4$ hit

Yes

PML4

No

PDP$ hit

Yes

PDP

No

PDE$ hit

Yes

PDE

inject a load to get PML4

L1$ hit

No

L2$ hit

No

L3$ hit

No

Memory

inject a load to get PDP

L1$ hit

No

L2$ hit

No

L3$ hit

No

Memory

inject a load to get PDE

L1$ hit

No

L2$ hit

No

L3$ hit

No

Memory

inject a load to get PTE

L1$ hit

No

L2$ hit

No

L3$ hit

No

Memory

Virtual Address

Yes

TLB hit

No

STLB hit

Yes

PTE
PMH – Page Miss Handler

- **Flags values**
  - R/W flag: logical AND of R/W flag in all levels
    - Can only write to a page if ALL levels allow it
  - U/S flag: logical AND of U/S flag in all levels
    - Set as Supervisor only if ALL levels indicate Supervisor
  - XD flag: logical OR of XD flag in all levels
    - Execution is disabled if it is disabled in any of the levels
  - The values of the PCD and PWT flags are taken for the PDP entry
Cache and Translation Structures

Core

- L1 Inst. cache
- Inst. TLB
- Data. TLB
- L1 data cache
- L2
- L3
- Memory

On-die

- Load entry

Platform

- Page Walk Logic
- PML4 entry: VA[47:39]
- PDP entry: VA[47:30]
- PDE entry: VA[47:21]
- VA[47:21]: PDE cache
- VA[47:12]: STLB

Translation Structures:

- Instruction bytes
- Translation
- PTE

Core Components:

- L1 Inst. cache
- Inst. TLB
- Data. TLB
- L1 data cache
- L2
- L3
- Memory

Translation Path:

- Instruction bytes
- Translation
- PTE
- Load entry

Memory Components:

- PMH
- Page Walk Logic
- PML4 cache
- PDP cache
- PDE cache
- STLB

Translation Levels:

- PML4 entry: VA[47:39]
- PDP entry: VA[47:30]
- PDE entry: VA[47:21]
- VA[47:21]: PDE cache
- VA[47:12]: STLB
Backup
Why virtual memory?

- **Generality**
  - ability to run programs larger than size of physical memory

- **Storage management**
  - allocation/deallocation of variable sized blocks is costly and leads to (external) fragmentation

- **Protection**
  - regions of the address space can be R/O, Ex, ...

- **Flexibility**
  - portions of a program can be placed anywhere, without relocation

- **Storage efficiency**
  - retain only most important portions of the program in memory

- **Concurrent I/O**
  - execute other processes while loading/dumping page

- **Expandability**
  - can leave room in virtual address space for objects to grow.

- **Performance**
Address Translation with a TLB

- Virtual address = virtual page number + page offset
- Physical address = physical page number + byte offset
- TLB hit
- Cache hit
PAE – Physical Address Extension

- When PAE (physical address extension) flag in CR4 is set
  - Physical addresses is extended to $M$ bits
    - Linear address remains 32 bit
  - Each page table entry becomes 64 bits to hold the extra phy. address bits
    - Page directory and page tables remain 4KB in size
      - number of entries in each table is halved to 512
      - indexed by 9 instead of 10 bits

- A new 4 entry **Page Directory Pointer Table is added**
  - Indexed by bits [31:30] of the linear address
  - Each entry points to a page directory
  - CR3 contains the page-directory-pointer-table base address
    - Provides the m.s.bits of the physical address of the first byte of the page-directory pointer table
    - forcing the table to be located on a 32-byte boundary
**4KB Page Mapping with PAE**

- **Linear address divided to**

  - Dir Ptr (2 bits) – points to a Page-directory-pointer-table entry
    - Provides base physical address of a page directory table
    - Base is M–12 bits, 4KB aligned

  - Dir (9 bits) – points to a PDE in the Page Directory
    - PS in the PDE = 0 ⇒ PDE provides a base physical address of a page table: M–12 bits, 4KB aligned

  - Table (9 bits) – points to a PTE in the Page Table
    - The PTE provides a base physical address of a 4KB page M–12 bits, 4KB aligned

  - Offset (12 bits) – offset within the selected 4KB page

---

**Linear Address Space (4K Page)**

- 31 30 29 | 21 20 12 11 | 0
- Dir ptr | DIR | TABLE | OFFSET
- 2 9 9 12
- 4KByte Page

- 4 entry Page Directory Pointer Table
- 512 entry Page Table
- PTE
- M-12
- M-12
- M-12
- CR3 (PDPTR)
2MB Page Mapping with PAE

- Linear address divided to
  - Dir Ptr (2 bits) – points to a Page-directory-pointer-table entry
    - The selected entry provides the base physical address of a page directory
    - The base has M–12 bits, 4KB aligned
  - Dir (9 bits) – points to a PDE in the Page Directory
    - PS in the PDE = 1 ⇒ PDE provides a base physical address of a 2MB page
    - The base is M–21 bit, 2MB aligned
  - Offset (21 bits) – offset within the selected 2MB page
**PTE/PDE/PDP Entry Format with PAE**

- Page table entries increased from 32 bits to 64 bits
- The base physical address field in each entry is extended to 24 bits

<table>
<thead>
<tr>
<th></th>
<th>Ignored</th>
<th>Address of page-directory-pointer table</th>
<th>Ignored</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>Address of page directory</td>
<td>Ignored</td>
<td>Rsvd.</td>
</tr>
<tr>
<td>XD</td>
<td>Reserved</td>
<td>Address of 2MB page frame</td>
<td>Reserved</td>
</tr>
<tr>
<td>XD</td>
<td>Reserved</td>
<td>Address of page table</td>
<td>Ignored</td>
</tr>
<tr>
<td>XD</td>
<td>Reserved</td>
<td>Address of 4KB page frame</td>
<td>Ignored</td>
</tr>
</tbody>
</table>

Extended Phy. Addr.  Same as in 32 bit mode