Memory Hierarchy

Virtual Memory, Address Translation

Slides contents from:

Hennessy & Patterson, 5ed. Appendix B and Chapter 2.
David Wentzlaff, ELE 475 – Computer Architecture.
MJT, High Performance Computing, NPTEL.
Process/Program Address Space

Byte Address

0

CODE

DATA

HEAP

STACK

$2^{32} - 1$
Process/Program Address Space

- Compiler assumes a linear address space
  - Byte 0 to Byte $2^{32}-1$
Process/Program Address Space

- Compiler assumes a linear address space
  - Byte 0 to Byte $2^{32}-1$
- **Virtual Address space**
Process/Program Address Space

- Compiler assumes a linear address space
  - Byte 0 to Byte $2^{32}-1$
- **Virtual Address space**
- The entire process data structure may not be present in MM at all times.
Paged Virtual Memory

Virtual Address Space

Virtual Page Number 0

Virtual Page Number 1

Virtual Page Number N-1

$2^{32} - 1$
Paged Virtual Memory

Virtual Address Space

- Virtual Page Number 0
- Virtual Page Number 1
- Virtual Page Number N-1

\[2^{32}-1\]

Page Number 0
Page Number 13

MAIN MEMORY
Paged Virtual Memory

Virtual Address Space

Virtual Page Number 0

Virtual Page Number 1

Virtual Page Number N-1

2^{32}-1
Paged Virtual Memory

Virtual Address Space

Virtual Page Number 0

Virtual Page Number 1

Virtual Page Number N-1

Physical Address Space

Page Number 0

Page Number 13

Page Number 1

Page Number 2

Page Number N-1

Byte

2^{32}-1
The Memory Hierarchy

- **Virtual Addresses**
  - Registers
  - Words
    - (transferred explicitly via load/store)
- **Cache**
- **Main Memory**
- **Physical Addresses**
  - Lines
    - (transferred automatically upon cache miss)
  - Pages
    - (transferred automatically upon page fault)
- **Virtual Memory**
Address Translation Table

Physical Page Numbers

VPN 0
VPN 1
VPN N-1

PPN 0
Address Translation Table

Virtual Address

VPN 0
VPN 1
VPN N-1

Physical Page Numbers

VPN 0
VPN 1
VPN N-1

PPN 0
PPN
PPN
Address Translation Table

Virtual Address

Physical Address

VPN 0
VPN 1
VPN N-1

Physical Page Numbers

VPN 0
VPN 1
VPN N-1

PPN 0
PPN
PPN

VPN
PO
VPN
PO
VPN
PO
Virtual Memory

Hard Disk

Contents of virtual pages

P1 0
P2 3
Pn 2

...
Virtual Memory

Main Memory

0
1
2
3

Hard Disk

Contents of virtual pages

P1
0

P2
3

Pn
2

1
Virtual Memory

Main Memory

Page Tables

Hard Disk

Contents of virtual pages
Address Translation Table

Virtual Address

VPN
PO

Physical Address

PPN
PO

Physical Page Numbers

VPN 0
VPN 1
VPN N

MAIN MEMORY
### Address Translation Table

<table>
<thead>
<tr>
<th>Virtual Address</th>
<th>Physical Page Numbers</th>
<th>Disk Addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPN 0</td>
<td>VPN 1</td>
<td>VPN N</td>
</tr>
<tr>
<td>PPN</td>
<td>Disk Address</td>
<td></td>
</tr>
</tbody>
</table>

- **VPN** (Virtual Page Number)
- **PO** (Page Offset)
- **PPN** (Physical Page Number)
- **MAIN MEMORY**
- **HARD DISK**
- **V** (Virtual Page)
- **MAIN MEMORY**
- **HARD DISK**

- **Physical Address**
- **Physical Page Numbers**
- **Disk Addresses**
Implementation of Address Translation

- Process always uses virtual addresses
Implementation of Address Translation

- Process always uses virtual addresses
- **Memory Management Unit (MMU):** part of CPU; hardware that does address translation
Implementation of Address Translation

- Process always uses virtual addresses
- **Memory Management Unit (MMU):** part of CPU; hardware that does address translation
- The page tables are (at best) present in the MM (OS virtual address space)
  - One main memory reference per address translation!
- 
Implementation of Address Translation

- Process always uses virtual addresses
- **Memory Management Unit (MMU)**: part of CPU; hardware that does address translation
- To translate a virtual memory address, the MMU has to read the relevant page table entry out of memory
Implementation of Address Translation

- Process always uses virtual addresses

**Memory Management Unit (MMU):** part of CPU; hardware that does address translation

- To translate a virtual memory address, the MMU has to read the relevant page table entry out of memory
  - Caches recently used translations in a **Translation Lookaside Buffer** (Page Table Cache)
Caches and Address Translation

CPU \rightarrow MMU \rightarrow Cache

Virtual Address \rightarrow Physical Address
Caches and Address Translation

CPU → MMU → Cache

Virtual Address → Physical Address → Physically Addressed Cache
Caches and Address Translation
Caches and Address Translation

CPU  Virtual Address  MMU  Physical Address  Cache

Physically Addressed Cache

CPU  Virtual Address  Cache Miss  MMU  Physical Address  Main Memory

Virtually Addressed Cache
Which is less preferable?

- Physical addressed cache

- Virtual addressed cache
Which is less preferable?

- **Physical addressed cache**
  - Hit time higher (cache access after translation)
- **Virtual addressed cache**
Which is less preferable?

- **Physical addressed cache**
  - Hit time higher (cache access after translation)

- **Virtual addressed cache**
  - Data/instruction of different processes with same virtual address in cache at the same time ...
Which is less preferable?

- **Physical addressed cache**
  - Hit time higher (cache access after translation)

- **Virtual addressed cache**
  - Data/instruction of different processes with same virtual address in cache at the same time ...
    - Flush cache on context switch, or
    - Include Process id as part of each cache directory entry
Which is less preferable?

- **Physical addressed cache**
  - Hit time higher (cache access after translation)

- **Virtual addressed cache**
  - Data/instruction of different processes with same virtual address in cache at the same time...
    - Flush cache on context switch, or
    - Include Process id as part of each cache directory entry
  - Synonyms
Synonyms (Aliases)

- VA-T1
  - P1
  - t1
- VA-T2
  - P2
  - t2
- Shared L2

- t1: Read x

- X
Synonyms (Aliases)

VA-T1

P1

P2

VA-T2

Shared L2

X

X

t1: Read x
Synonyms (Aliases)

- VA-T1
- VA-T2
- Shared L2

P1 t1
P2 t2

X

- t1: Read x
- t2: Read x
L2 uses virtual addresses

2 copies of one physical page in the cache!
Overlapped Operation

- CPU
- MMU
- Virtual Address
- Indexing using VA
- Cache
- Tag check using PA
Overlapped Operation

Virtually Indexed Physically Tagged Cache (VIPT)

Other options: PIPT, VIVT
Recall – Cache Access

Direct mapped, 32 KB, 32B block, 32b main memory address
Recall – Cache Access

32 b address

Direct mapped, 32 KB, 32B block, 32b main memory address
Recall – Cache Access

Direct mapped, 32 KB, 32B block, 32b main memory address

32 b address

Tag | Index (10b) | 3 | 2
Recall – Cache Access

Direct mapped, 32 KB, 32B block, 32b main memory address
Recall – Cache Access

Direct mapped, 32 KB, 32B block, 32b main memory address

32 b address

Tag  Index (10b)  3  2

32B Block
Recall – Cache Access

Direct mapped, 32 KB, 32B block, 32b main memory address

32 b address

Tag Index (10b) 3 2

W 1 W 2 ... 7 W 8
Recall – Cache Access

32 b address

Tag  Index (10b)  3  2

Direct mapped, 32 KB, 32B block, 32b main memory address

=?
Direct mapped, 32 KB, 32B block, 32b main memory address

Recall – Cache Access

32 b address

Tag

Index (10b) 3 2

W 1 W 2 ... 7 W 8

W 1 W 2 ... 7 W 8

...
Recall – Cache Access

Direct mapped, 32 KB, 32B block, 32b main memory address

32 b address

Tag

Index (10b)

3 2

W 1 W 2 ... 7 W 8

W 1 W 2 ... 7 W 8

...
Recall – Cache Access

Direct mapped, 32 KB, 32B block, 32b main memory address

Tag is not needed until the cache line has been read

32 b address

Tag

Index (10b) 3 2

=?

No Cache Miss

Yes Cache Hit

To Processor
VM Example

64 b VM address
VM Example

<table>
<thead>
<tr>
<th>64 b VM address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual Page No. (50)</td>
</tr>
</tbody>
</table>
VM Example

64 b VM address

Virtual Page No. (50) Page Offset (14)

Index (8) Offset (6)
VM Example

64 b VM address

Virtual Page No. (50) Page Offset (14)

Index (8) Offset (6)

26b Cache Block

=? L1 Hit/Miss
VM Example

- 64 b VM address
- Virtual Page No. (50)
- Page Offset (14)
- Index (8) Offset (6)
- Physical Address (Tag)
- 26b Cache Block
- =?
- L1 Hit/Miss

(26)
Virtual Page No. (50) Page Offset (14)

Index (8) Offset (6)

Physical Address (Tag)

L1 Hit/Miss

Index comes from the Virtual Address (Virtually Indexed)

Tag comes from the Physical Address (Physically tagged)
VM Example

Index comes from the Virtual Address (Virtually Indexed)

Tag comes from the Physical Address (Physically tagged)
Index comes from the Virtual Address (Virtually Indexed)

Tag comes from the Physical Address (Physically tagged)
Index comes from the Virtual Address (Virtually Indexed)

Tag comes from the Physical Address (Physically tagged)
Index comes from the Virtual Address (Virtually Indexed)

Tag comes from the Physical Address (Physically tagged)
VM Example

64 b VM address

Virtual Page No. (50) Page Offset (14)

Index (8) Offset (6)

1. Index comes from the Virtual Address (Virtually Indexed)
2. Tag comes from the Physical Address (Physically tagged)

TLB Tag TLB Index

Physical Address (Tag)

TLB Hit/Page Fault

Physical Address (40)

To L2

L1 Hit/Miss

Cache Block

L2 Cache Block

Index from cache block (26)

26b
VM – Hardware vs. Software

• Operating system creates Virtual to physical address mappings.
• Hardware performs the VA to PA translation at runtime.
Why Virtual Memory?

• Relocation
  – Programmer has contiguous view of memory
  – Physical memory is scattered
Why Virtual Memory?

• **Relocation**
  - Programmer has contiguous view of memory
  - Physical memory is scattered

• **Resource management**
  - Programmer can assume there is enough memory
  - System can manage physical memory across concurrent processes
Why Virtual Memory?

• **Relocation**
  – Programmer has contiguous view of memory
  – Physical memory is scattered

• **Resource management**
  – Programmer can assume there is enough memory
  – System can manage physical memory across concurrent processes

• **Isolation**
  – Bug in one process should not corrupt memory of another
  – Provide each process with separate virtual address space
Why Virtual Memory?

- **Relocation**
  - Programmer has contiguous view of memory
  - Physical memory is scattered

- **Resource management**
  - Programmer can assume there is enough memory
  - System can manage physical memory across concurrent processes

- **Isolation**
  - Bug in one process should not corrupt memory of another
  - Provide each process with separate virtual address space

- **Protection**
  - Enforce rules on what memory a process can or cannot access
Size of the Page Table

- Consider a typical 48-bit virtual address space, 4KB page size and 8 byte long page table entry (PTE)
  - Size of the page table = 512GB!
  - Many processes ==> many page tables
Size of the Page Table

- Consider a typical 48-bit virtual address space, 4KB page size and 8 byte long page table entry (PTE)
  - Size of the page table = 512GB !
  - Many processes ==> many page tables
- Single level page table has high cost
Page Table Implementation

- Single level page table has high cost
- Virtual address space is sparsely allocated
Page Table Implementation

- Single level page table has high cost
- Virtual address space is sparsely allocated
- **Multi-level radix tree**
Toy Example

- Toy Page table implementation using radix tree
- 16b Virtual Address, 4KB page size
Toy Example

- Toy Page table implementation using radix tree
- 16b Virtual Address, 4KB page size
Toy Example

- Toy Page table implementation using radix tree
- 16b Virtual Address, 4KB page size
Toy Example

- Toy Page table implementation using radix tree
- 16b Virtual Address, 4KB page size
Toy Example

- Assume: Page Table is split into 4
- Placed at addresses A, B, C, and D in Physical memory.
Toy Example

- Placed at addresses A, B, C, and D in Physical memory.
Toy Example
Toy Example
Toy Example
2 Level, 4-ary Radix Tree

PT-Base

\[(\text{vpn})\]
64bit-x86 Radix Tree

- Is a 4 Level, 512-ary Radix Tree

-
64bit-x86 Radix Tree

- Is a 4 Level, 512-ary Radix Tree

[Diagram of a 4-level Radix Tree with 512 children at each level]
64bit-x86 Radix Tree

- 48b VA, 4KB page
- 36b VPN
64bit-x86 Radix Tree

- 48b VA, 4KB page
- 36b VPN
64bit-x86 Radix Tree

- 48b VA, 4KB page
- 36b VPN
64bit-x86 Radix Tree

- 48b VA, 4KB page
- 36b VPN
64bit-x86 Radix Tree

- 48b VA, 4KB page
- 36b VPN
64bit-x86 Radix Tree

- 48b VA, 4KB page
- 36b VPN
Page Table Walk

- 48b VA, 4KB page
- 36b VPN
Page Table Entry

- **P** (Present bit): If the address present in memory
- **‘U/S’** (User/Supervisor bit): Is the page accessible in supervisor mode (e.g., by OS) only?
- **‘R/W’** (Read/Write): Is the page read-only?
- **A** (Access bit): Is this page has ever been accessed (load/stored to)?
- **D** (Dirty bit): Is the page has been written to?
- **X/D** (Executable bit): Does the page contains executable?
Role of the OS

- Maintains one page table per process (a.k.a., per virtual address space)
Role of the OS

- Maintains one page table per process (a.k.a., per virtual address space)
- Creates, updates, deletes page table entries (PTEs)
Role of the OS

- Maintains one page table per process (a.k.a., per virtual address space)
- Creates, updates, deletes page table entries (PTEs)
- Page table structure is part of agreement between OS and Hardware
  - page table structure is ISA specific
Page Table Walk

- **Hardware Page Table Walker (PTW)**
  - Input: Root of page table (cr3) and VPN
  - Output: Physical page frame number or fault
Page Table Walk

• **Hardware Page Table Walker (PTW)**
  - Input: Root of page table (cr3) and VPN
  - Output: Physical page frame number or fault
  - A hardware fine-state-automata in each CPU core
    Generates load-like “instructions” to access page table
  - Eg. x86, POWER, and ARM processors
Page Table Walk

• **Hardware Page Table Walker (PTW)**
  - Input: Root of page table (cr3) and VPN
  - Output: Physical page frame number or fault
  - A hardware fine-state-automata in each CPU core
    Generates load-like “instructions” to access page table
  - Eg. x86, POWER, and ARM processors

• **Software page table walker**
  - A OS handler walks the page table
  - Slow. Large address translation overhead
  - Example: SPARC (Sun/Oracle) machines
Fast Translation

- Address translation is on the critical path
- Paging – 4+1 memory accesses!
  - Address Translation Table Accesses + Data
- **Translation Lookaside Buffer (TLB)**: Part of MMU that caches address translations
TLB & PTW

Diagram showing the interconnect and memory hierarchy:
- Core 0: L1, L2
- Core 1: L1, L2
- Core 2: L1, L2
- Core 3: L1, L2
- Interconnect
- L3
- DRAM
TLB & PTW

Core 0
- L1
- L2
- L1 TLB
- L2 TLB
- PTW

Core 1
- L1
- L2

Core 2
- L1
- L2

Core 3
- L1
- L2

Interconnect

L3

DRAM
TLB Hierarchy

• Each core typically has:
  – 32-64-entry L1 TLB (fully associative/4-8 way set-associative)
  – 1500-2500 entry L2 TLB (8-16 way set-associative)
  – One to two PTW

Address Translation Steps

- Load/store instructions carries virtual address
  - \( VA = VPN + PO \)
Address Translation Steps

- Load/store instructions carries virtual address
- Search TLBs for VPN to physical page (page frame number) mapping
Address Translation Steps

- Load/store instructions carries virtual address
- Search TLBs for VPN to physical page (page frame number) mapping
- On hit, Tag match with Tag from cache (Virtually indexed Physically Tagged)
Address Translation Steps

- Load/store instructions carries virtual address
- Search TLBs for VPN to physical page (page frame number) mapping
- On hit, Tag match with Tag from cache (Virtually indexed Physically Tagged)
- On miss, PTW starts walking page table to get PFN
  - Found: Return mapping to TLB
  - Not Found: page fault raised to the OS
Page Fault

- Must be `handled’ by operating system
  - Identify slot in main memory to be used
  - Get page contents from disk
  - Update page page table entry
Abstraction: Virtual vs. Physical Memory

- **Programmer** sees *virtual memory*
  - Can assume the memory is “infinite”
Abstraction: Virtual vs. Physical Memory

• **Programmer** sees *virtual memory*
  - Can assume the memory is “infinite”

• **Reality**: *Physical memory* size is much smaller than what the programmer assumes
Abstraction: Virtual vs. Physical Memory

• The system (system software + hardware, cooperatively) maps virtual memory addresses to physical memory
  - System manages the physical memory space transparently to the programmer
Extra
Size of a Page

- Page is the unit of Memory Management
- Too large vs. Too small.
- Page Offset field need not be translated
- What if the Page Offset field was 12 bits? (Page size = 4KB)
Paged Virtual Memory

- 48 bit Virtual Addresses, 40 bit Physical Addresses. Page size = 16KB. How many entries in a process's Page Translation Table? What is the size of the Page Translation Table?
Virtual vs. Physical Memory

+ Programmer does not need to know the physical size of memory nor manage it
  - A small physical memory can appear as a huge one to the programmer
  - Life is easier for the programmer

-- More complex system software and architecture

A classic example of the programmer/(micro)architecture tradeoff
Translation Lookaside Buffer

- Cache of page table mappings
- 32 – 4096 entries long
  - SA, FA, or DM
- Dirty flag – use during page write back
- Ref – used for LRU

<table>
<thead>
<tr>
<th>VPN (tag)</th>
<th>PPN (data)</th>
<th>Valid</th>
<th>Ref</th>
<th>Dirty</th>
<th>Access Rights</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Page Fault

- Must be `handled’ by operating system
  - Identify slot in main memory to be used
  - Get page contents from disk
  - Update page table entry
- Provide data to the processor
Virtual Memory

- What is the size of the Page Table?
- Where is it stored?
- What factors decide the size of a page?
  - What are its side effects?
- Page size is constant/variable?