COMP2007: Operating Systems & Concurrency
Week 8 – 3:00pm Monday – 13 November 2023

valid for 65 minutes from 2:55pm
generated 2023-10-17 03:13

Figure: Attendance Monitoring

Operating Systems and Concurrency

Memory Management 4

COMP2007

Geert De Maere
(Dan Marsden)
{Geert.DeMaere,Dan.Marsden}@Nottingham.ac.uk

University Of Nottingham
United Kingdom

2023

Goals for Today

Overview

- **Address translation** implementation (revisited)
- Principles behind **virtual memory**
- Complex/large **page tables**

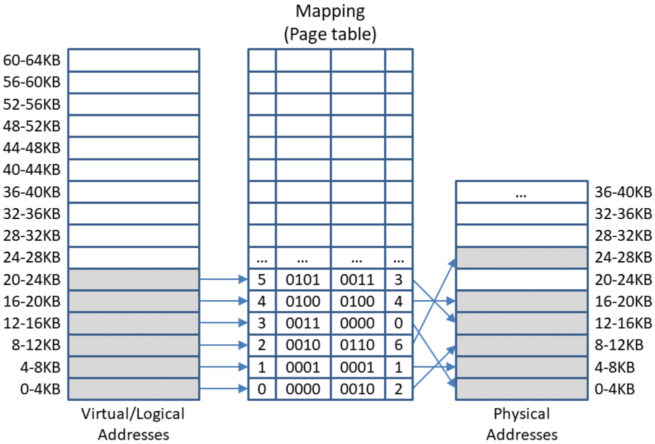
Recall

Last Lecture

- The **principles of paging** are:
  - **Logical address space** is divided into equal sized **pages**
  - **Physical address space** is divided into equal sized **frames**
  - A **page table** contains multiple “**relocation registers**” to map the pages on to frames
- The **benefits** of paging include:
  - Reduced **internal fragmentation**
  - No **external fragmentation**

Paging

Relocation: Address Translation



...	...	...	...
5	0101	0011	3
4	0100	0100	4
3	0011	0000	0
2	0010	0110	6
1	0001	0001	1
0	0000	0010	2

Recall

Memory as a Linear Array

- Memory is a **linear array** of **bytes**
- N **address lines** are used to specify 2^N addresses, e.g., 2^{16} for a 16 bit machine
- If each **memory cell** is a **byte**, we can address up to 64KB.
- If the memory is split into 16 blocks (=2^4), then block size = $2^{16}/2^4 = 2^{12}$ = 4KB

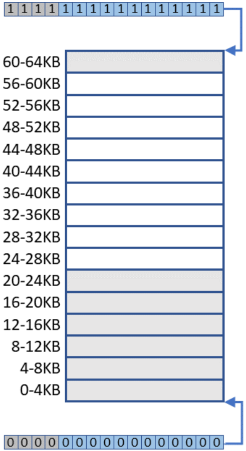


Figure: Linear address space (16-bit)

Paging

Address Translation: Implementation

- A **logical address** is relative to the start of the **program**
- The **left most** *n* **bits** represent the **page number**
  - e.g. 4 bits for the page number ⇒ 16 pages
- The **right most** *m* **bits** represent the **offset within the page**
  - e.g. 12 bits for the offset ⇒ 4KB pages

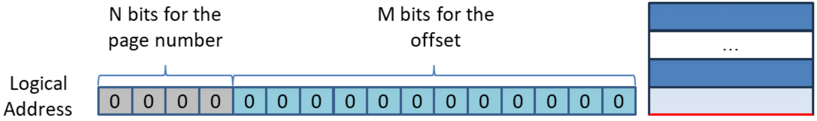


Figure: Logical Address

Paging

Address Translation: Implementation

- A **logical address** is relative to the start of the **program**
- The **left most** *n* **bits** represent the **page number**
  - e.g. 4 bits for the page number ⇒ 16 pages
- The **right most** *m* **bits** represent the **offset within the page**
  - e.g. 12 bits for the offset ⇒ 4KB pages

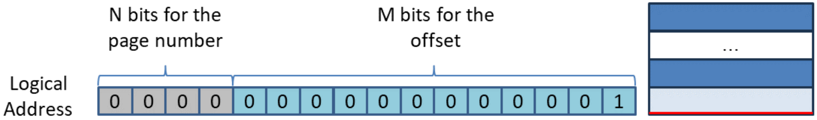


Figure: Logical Address

Paging

Address Translation: Implementation

- A **logical address** is relative to the start of the **program**
- The **left most** *n* **bits** represent the **page number**
  - e.g. 4 bits for the page number ⇒ 16 pages
- The **right most** *m* **bits** represent the **offset within the page**
  - e.g. 12 bits for the offset ⇒ 4KB pages

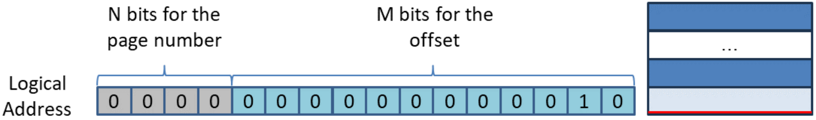


Figure: Logical Address

Paging

Address Translation: Implementation

- A **logical address** is relative to the start of the **program**
- The **left most** *n* **bits** represent the **page number**
  - e.g. 4 bits for the page number ⇒ 16 pages
- The **right most** *m* **bits** represent the **offset within the page**
  - e.g. 12 bits for the offset ⇒ 4KB pages

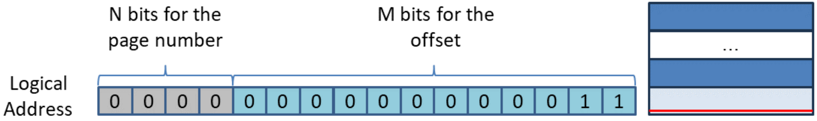


Figure: Logical Address

Paging

Address Translation: Implementation

- A **logical address** is relative to the start of the **program**
- The **left most *n* bits** represent the **page number**
  - e.g. 4 bits for the page number ⇒ 16 pages
- The **right most *m* bits** represent the **offset within the page**
  - e.g. 12 bits for the offset ⇒ 4KB pages

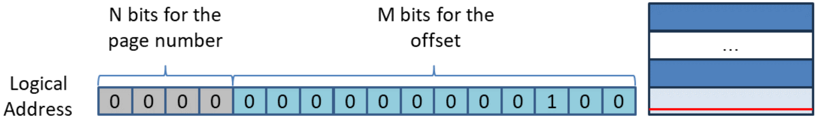


Figure: Logical Address

Paging

Address Translation: Implementation

- A **logical address** is relative to the start of the **program**
- The **left most** *n* **bits** represent the **page number**
    - e.g. 4 bits for the page number ⇒ 16 pages
- The **right most** *m* **bits** represent the **offset within the page**
    - e.g. 12 bits for the offset ⇒ 4KB pages

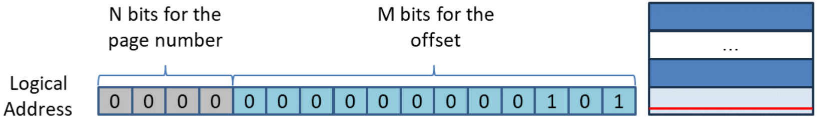


Figure: Logical Address

Paging

Address Translation: Implementation

- A **logical address** is relative to the start of the **program**
- The **left most *n* bits** represent the **page number**
  - e.g. 4 bits for the page number ⇒ 16 pages
- The **right most *m* bits** represent the **offset within the page**
  - e.g. 12 bits for the offset ⇒ 4KB pages

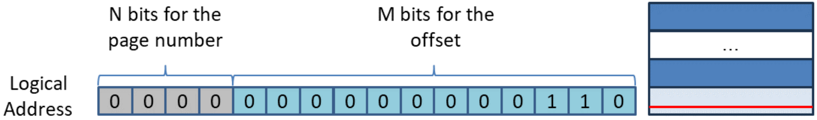
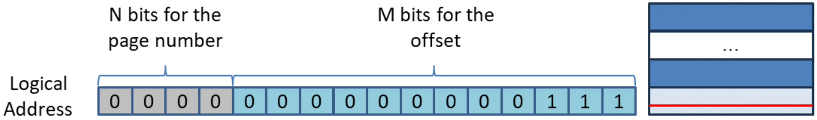


Figure: Logical Address

Paging

Address Translation: Implementation

- A **logical address** is relative to the start of the **program**
- The **left most** *n* **bits** represent the **page number**
    - e.g. 4 bits for the page number ⇒ 16 pages
- The **right most** *m* **bits** represent the **offset within the page**
    - e.g. 12 bits for the offset ⇒ 4KB pages



0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Figure: Logical Address

Paging

Address Translation: Implementation

- A **logical address** is relative to the start of the **program**
- The **left most** *n* **bits** represent the **page number**
  - e.g. 4 bits for the page number ⇒ 16 pages
- The **right most** *m* **bits** represent the **offset within the page**
  - e.g. 12 bits for the offset ⇒ 4KB pages

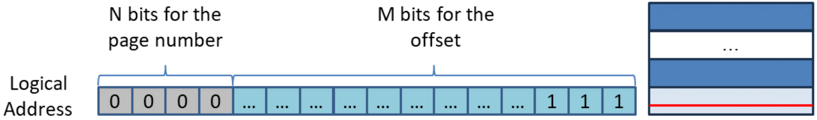


Figure: Logical Address

Paging

Address Translation: Implementation

- A **logical address** is relative to the start of the **program**
- The **left most** *n* **bits** represent the **page number**
  - e.g. 4 bits for the page number ⇒ 16 pages
- The **right most** *m* **bits** represent the **offset within the page**
  - e.g. 12 bits for the offset ⇒ 4KB pages

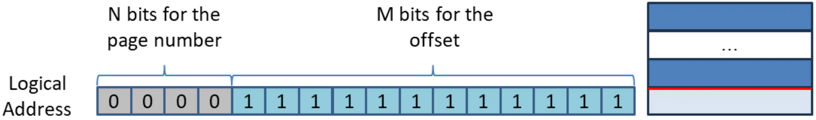


Figure: Logical Address

Paging

Address Translation: Implementation

- A **logical address** is relative to the start of the **program**
- The **left most *n* bits** represent the **page number**
  - e.g. 4 bits for the page number ⇒ 16 pages
- The **right most *m* bits** represent the **offset within the page**
  - e.g. 12 bits for the offset ⇒ 4KB pages

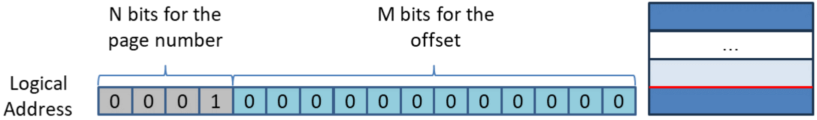
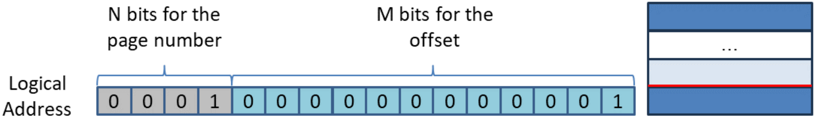


Figure: Logical Address

Paging

Address Translation: Implementation

- A **logical address** is relative to the start of the **program**
- The **left most** *n* **bits** represent the **page number**
    - e.g. 4 bits for the page number ⇒ 16 pages
- The **right most** *m* **bits** represent the **offset within the page**
    - e.g. 12 bits for the offset ⇒ 4KB pages



0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1

Figure: Logical Address

Paging

Address Translation: Implementation

- A **logical address** is relative to the start of the **program**
- The **left most *n* bits** represent the **page number**
  - e.g. 4 bits for the page number ⇒ 16 pages
- The **right most *m* bits** represent the **offset within the page**
  - e.g. 12 bits for the offset ⇒ 4KB pages

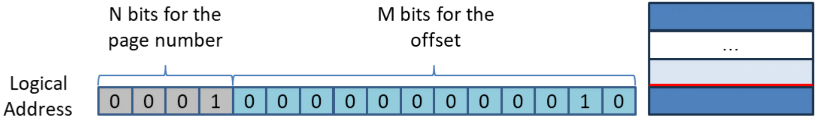


Figure: Logical Address

Paging

Address Translation: Implementation

- A **logical address** is relative to the start of the **program**
- The **left most *n* bits** represent the **page number**
    - e.g. 4 bits for the page number ⇒ 16 pages
- The **right most *m* bits** represent the **offset within the page**
    - e.g. 12 bits for the offset ⇒ 4KB pages

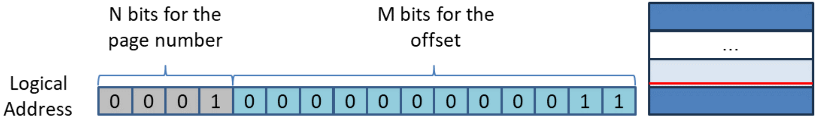


Figure: Logical Address

Paging

Address Translation: Implementation

- A **logical address** is relative to the start of the **program**
- The **left most** *n* **bits** represent the **page number**
  - e.g. 4 bits for the page number ⇒ 16 pages
- The **right most** *m* **bits** represent the **offset within the page**
  - e.g. 12 bits for the offset ⇒ 4KB pages

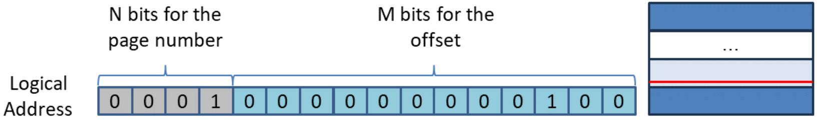


Figure: Logical Address

Paging

Address Translation: Implementation

- A **logical address** is relative to the start of the **program**
- The **left most *n* bits** represent the **page number**
    - e.g. 4 bits for the page number ⇒ 16 pages
- The **right most *m* bits** represent the **offset within the page**
    - e.g. 12 bits for the offset ⇒ 4KB pages

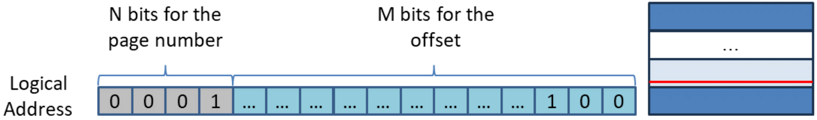
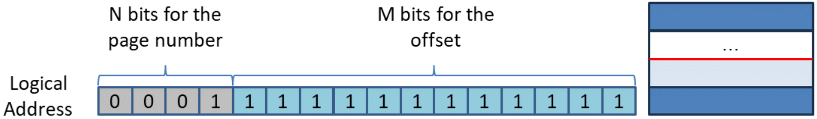


Figure: Logical Address

Paging

Address Translation: Implementation

- A **logical address** is relative to the start of the **program**
- The **left most** *n* **bits** represent the **page number**
    - e.g. 4 bits for the page number ⇒ 16 pages
- The **right most** *m* **bits** represent the **offset within the page**
    - e.g. 12 bits for the offset ⇒ 4KB pages



N bits for the page number

M bits for the offset

Logical Address

0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1

...

Figure: Logical Address

Paging

Address Translation: Implementation

- A **logical address** is relative to the start of the **program**
- The **left most** *n* **bits** represent the **page number**
    - e.g. 4 bits for the page number ⇒ 16 pages
- The **right most** *m* **bits** represent the **offset within the page**
    - e.g. 12 bits for the offset ⇒ 4KB pages

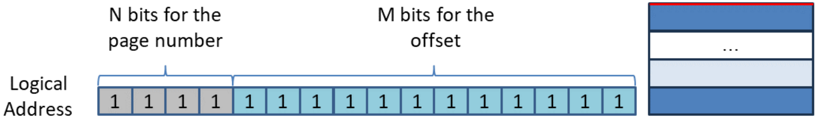


Figure: Logical Address

Paging

Address Translation: Implementation

- A **physical** is relative to the start of the **memory**
- The **left most** *k* **bits** represent the **frame number**
    - e.g. 3 bits for the frame number ⇒ 8 frames
- The **right most** *m* **bits** represent the **offset within the frame**
    - e.g. 12 bits for the offset ⇒ 4KB frames

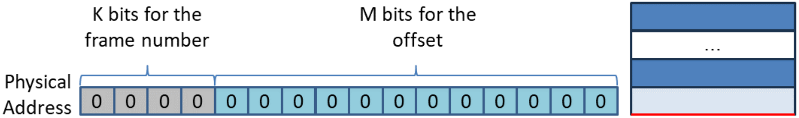


Figure: Physical Address

Paging

Address Translation: Implementation

- A **physical** is relative to the start of the **memory**
- The **left most** *k* **bits** represent the **frame number**
  - e.g. 3 bits for the frame number ⇒ 8 frames
- The **right most** *m* **bits** represent the **offset within the frame**
  - e.g. 12 bits for the offset ⇒ 4KB frames

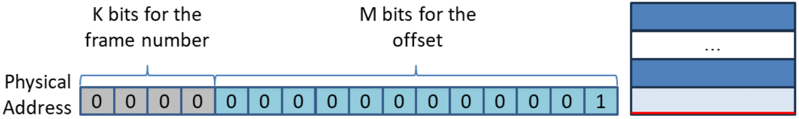


Figure: Physical Address

Paging

Address Translation: Implementation

- A **physical** is relative to the start of the **memory**
- The **left most** *k* **bits** represent the **frame number**
  - e.g. 3 bits for the frame number ⇒ 8 frames
- The **right most** *m* **bits** represent the **offset within the frame**
  - e.g. 12 bits for the offset ⇒ 4KB frames

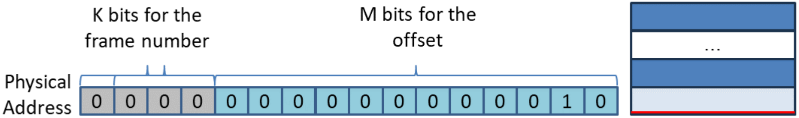


Figure: Physical Address

Paging

Address Translation: Implementation

- A **physical** is relative to the start of the **memory**
- The **left most** *k* **bits** represent the **frame number**
  - e.g. 3 bits for the frame number ⇒ 8 frames
- The **right most** *m* **bits** represent the **offset within the frame**
  - e.g. 12 bits for the offset ⇒ 4KB frames

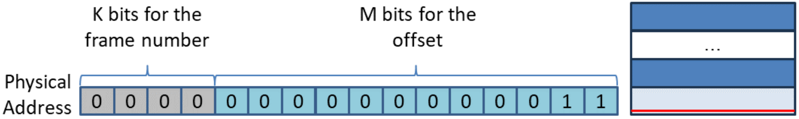


Figure: Physical Address

Paging

Address Translation: Implementation

- A **physical** is relative to the start of the **memory**
- The **left most** *k* **bits** represent the **frame number**
  - e.g. 3 bits for the frame number ⇒ 8 frames
- The **right most** *m* **bits** represent the **offset within the frame**
  - e.g. 12 bits for the offset ⇒ 4KB frames

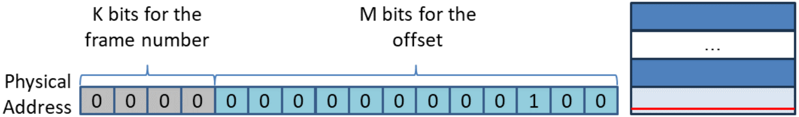


Figure: Physical Address

Paging

Address Translation: Implementation

- A **physical** is relative to the start of the **memory**
- The **left most** *k* **bits** represent the **frame number**
    - e.g. 3 bits for the frame number ⇒ 8 frames
- The **right most** *m* **bits** represent the **offset within the frame**
    - e.g. 12 bits for the offset ⇒ 4KB frames

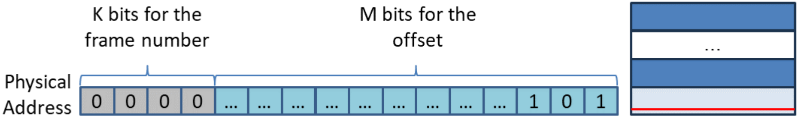


Figure: Physical Address

Paging

Address Translation: Implementation

- A **physical** is relative to the start of the **memory**
- The **left most** *k* **bits** represent the **frame number**
    - e.g. 3 bits for the frame number ⇒ 8 frames
- The **right most** *m* **bits** represent the **offset within the frame**
    - e.g. 12 bits for the offset ⇒ 4KB frames

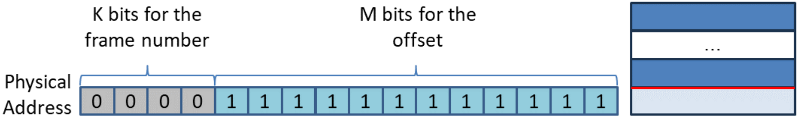


Figure: Physical Address

Paging

Address Translation: Implementation

- A **physical** is relative to the start of the **memory**
- The **left most** *k* **bits** represent the **frame number**
  - e.g. 3 bits for the frame number ⇒ 8 frames
- The **right most** *m* **bits** represent the **offset within the frame**
  - e.g. 12 bits for the offset ⇒ 4KB frames

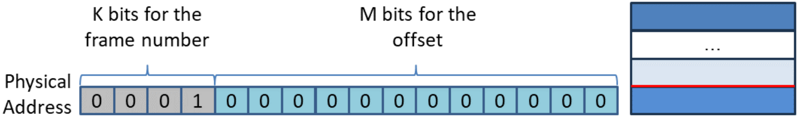
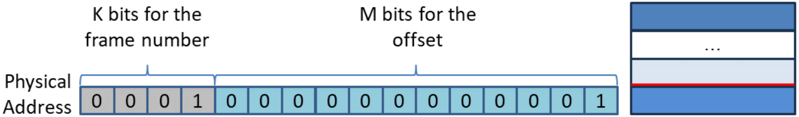


Figure: Physical Address

Paging

Address Translation: Implementation

- A **physical** is relative to the start of the **memory**
- The **left most** *k* **bits** represent the **frame number**
  - e.g. 3 bits for the frame number ⇒ 8 frames
- The **right most** *m* **bits** represent the **offset within the frame**
  - e.g. 12 bits for the offset ⇒ 4KB frames



K bits for the frame number

M bits for the offset

Physical Address

0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1

Figure: Physical Address

Paging

Address Translation: Implementation

- A **physical** is relative to the start of the **memory**
- The **left most** *k* **bits** represent the **frame number**
  - e.g. 3 bits for the frame number ⇒ 8 frames
- The **right most** *m* **bits** represent the **offset within the frame**
  - e.g. 12 bits for the offset ⇒ 4KB frames

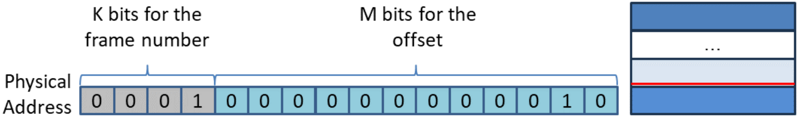


Figure: Physical Address

Paging

Address Translation: Implementation

- A **physical** is relative to the start of the **memory**
- The **left most** *k* **bits** represent the **frame number**
  - e.g. 3 bits for the frame number ⇒ 8 frames
- The **right most** *m* **bits** represent the **offset within the frame**
  - e.g. 12 bits for the offset ⇒ 4KB frames

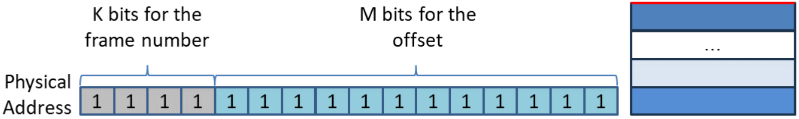


Figure: Physical Address

Paging

Address Translation: Implementation

- The **offset** within the page and frame **remains the same** (they are the same size)
- The page number to frame number mapping is held in the **page table**

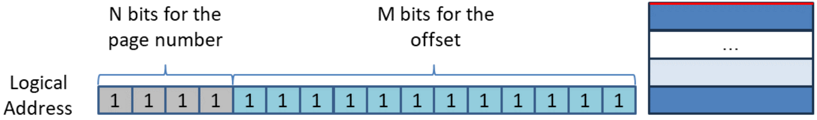


Figure: Logical Address

- The **offset** within the page and frame **remains the same** (they are the same size)
- The page number to frame number mapping is held in the **page table**

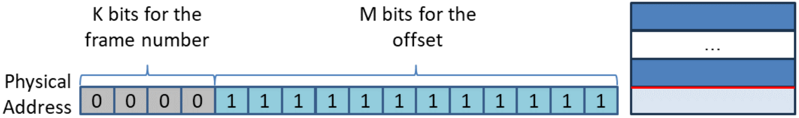
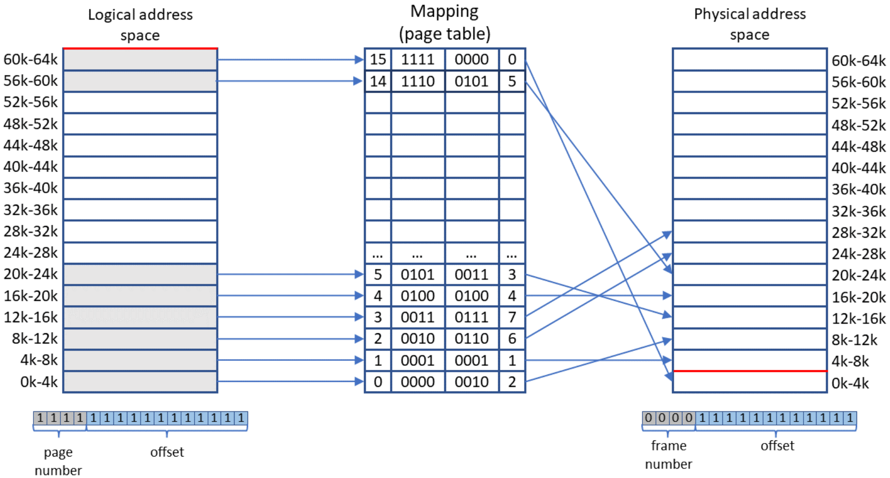


Figure: Logical Address

Paging

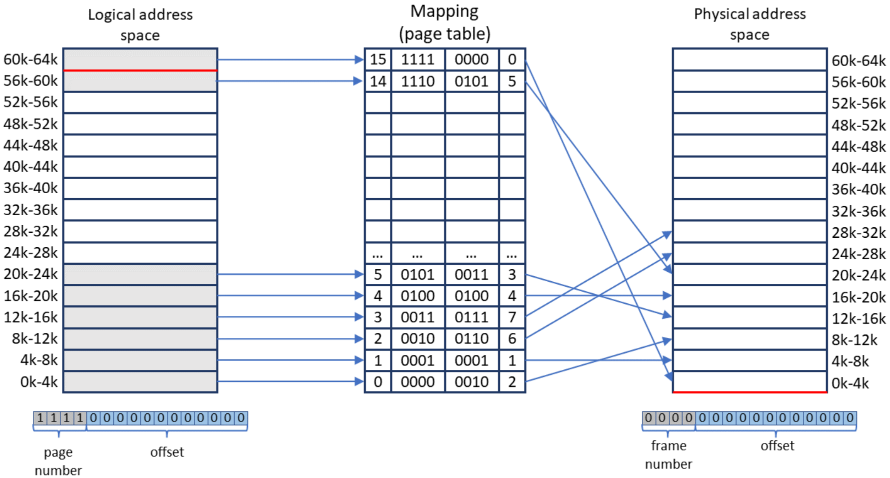
Address Translation: Implementation



15	1111	0000	0
14	1110	0101	5
...	...	...	...
5	0101	0011	3
4	0100	0100	4
3	0011	0111	7
2	0010	0110	6
1	0001	0001	1
0	0000	0010	2

Paging

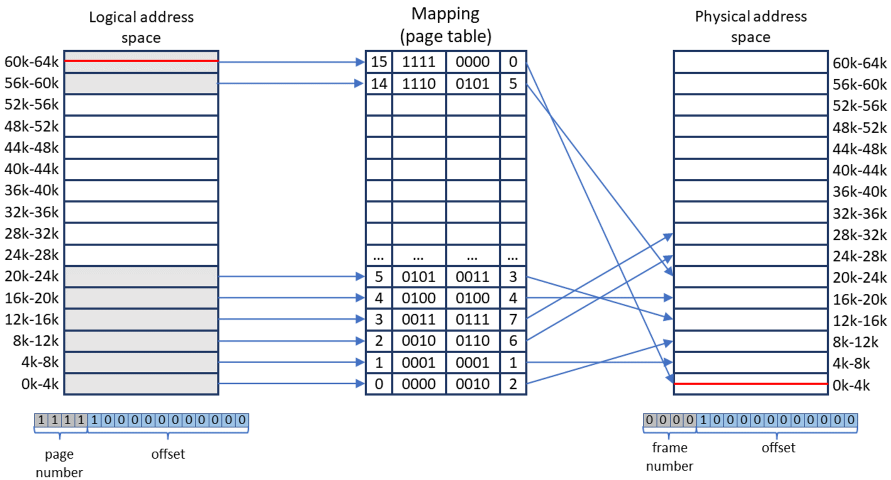
Address Translation: Implementation



15	1111	0000	0
14	1110	0101	5
...	...	...	...
5	0101	0011	3
4	0100	0100	4
3	0011	0111	7
2	0010	0110	6
1	0001	0001	1
0	0000	0010	2

Paging

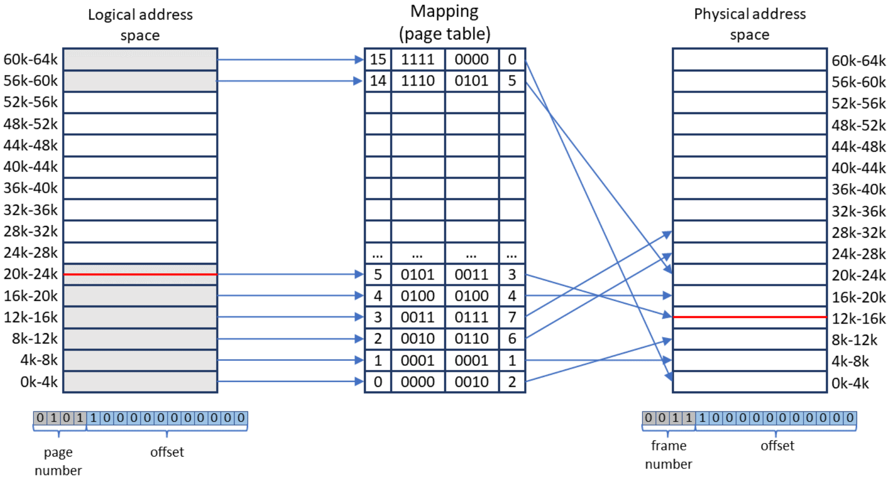
Address Translation: Implementation



15	1111	0000	0
14	1110	0101	5
...	...	...	...
5	0101	0011	3
4	0100	0100	4
3	0011	0111	7
2	0010	0110	6
1	0001	0001	1
0	0000	0010	2

Paging

Address Translation: Implementation



15	1111	0000	0
14	1110	0101	5
...	...	...	...
5	0101	0011	3
4	0100	0100	4
3	0011	0111	7
2	0010	0110	6
1	0001	0001	1
0	0000	0010	2

Paging

Relocation: Address Translation

- Steps in **address translation**:
  1. **Extract the page number** from logical address
  2. Use page number as an index to **retrieve the frame number** in the page table
  3. **Add the "logical offset within the page"** to the start of the physical frame
- **Hardware implementation** of address translation
  1. The CPU's **memory management unit** (MMU) intercepts logical addresses
  2. MMU uses a page table as above
  3. The resulting **physical address** is put on the **memory bus**

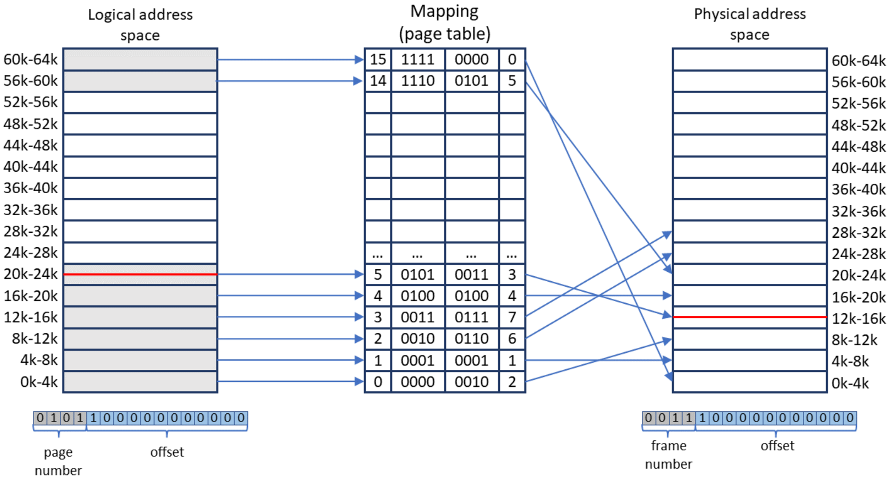
Virtual Memory

Principle of Locality

- Are there any other **benefits of paging**?
    - Principle of **locality**: **code** and **data references** are usually **clustered**
    - Code execution and data manipulation is usually restricted to a small **subset of pages** at a given point in time
- **Not all pages** have to be **loaded** in memory at the same time ⇒ **virtual memory**
    - Loading all program/data pages into memory is **wasteful**
    - Desired blocks could be **loaded on demand**

Virtual Memory

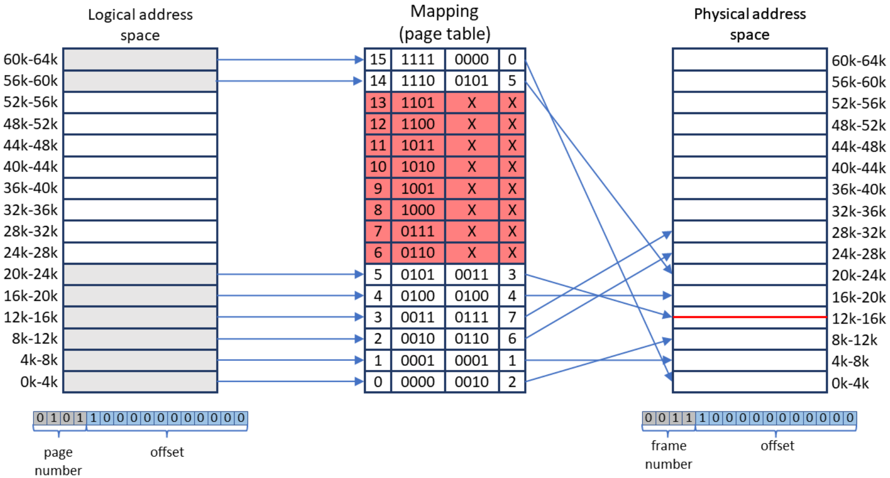
An Example



15	1111	0000	0
14	1110	0101	5
...	...	...	...
5	0101	0011	3
4	0100	0100	4
3	0011	0111	7
2	0010	0110	6
1	0001	0001	1
0	0000	0010	2

Virtual Memory

An Example

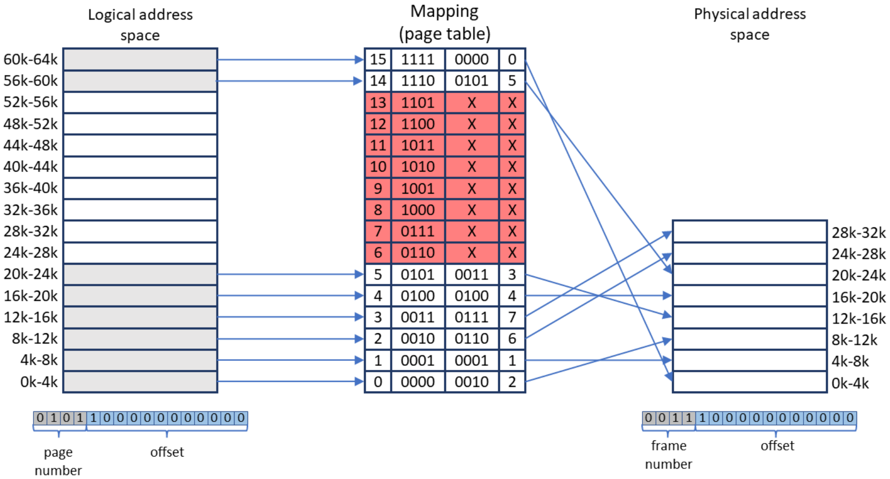


Mapping (page table)

15	1111	0000	0
14	1110	0101	5
13	1101	X	X
12	1100	X	X
11	1011	X	X
10	1010	X	X
9	1001	X	X
8	1000	X	X
7	0111	X	X
6	0110	X	X
5	0101	0011	3
4	0100	0100	4
3	0011	0111	7
2	0010	0110	6
1	0001	0001	1
0	0000	0010	2

Virtual Memory

An Example



15	1111	0000	0
14	1110	0101	5
13	1101	X	X
12	1100	X	X
11	1011	X	X
10	1010	X	X
9	1001	X	X
8	1000	X	X
7	0111	X	X
6	0110	X	X
5	0101	0011	3
4	0100	0100	4
3	0011	0111	7
2	0010	0110	6
1	0001	0001	1
0	0000	0010	2

Virtual Memory

Page Faults

- The **resident set** refers to the pages that are loaded in main memory
- A **page fault** is generated if the processor accesses a page that is **not in memory**
  - A page fault results in an **interrupt** (process enters **blocked state**)
  - An **I/O operation** is started to bring the missing page into main memory
  - A **context switch** (may) take place
  - An **interrupt signals** that the I/O operation is complete (process enters **ready state**)

Demand Paging

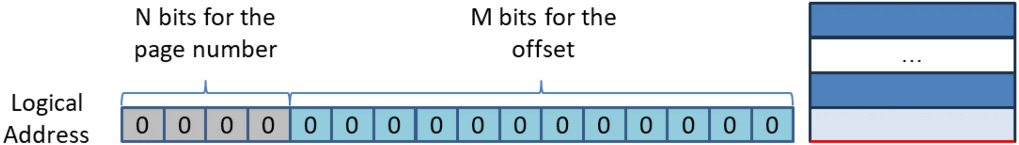
Processing Page Faults

1. Trap operating system
   - Save registers/process state
   - Analyse interrupt (i.e., identify page fault)
   - Validate page reference, determine frame location
   - Issue disk I/O: queueing, seek, latency, transfer
2. Context switch (optional)
3. Interrupt for I/O completion
   - Store process state/registers
   - Analyse interrupt from disk
   - Update page table (page now in memory)
   - Wait for original process to be scheduled
4. Context switch to original process

Virtual Memory

The Benefits

1. Virtual memory **improves CPU utilisation**
   - Individual **processes take up less memory** (only partially loaded)
   - **More processes** in memory
   - More **efficient use of memory** (less internal fragmentation, no external fragmentation)
2. The **logical address space** can be **larger than physical address space**
   - 64 bit machine ⇒ 2^{64} logical addresses (theoretically)



N bits for the page number

M bits for the offset

Logical Address

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

...

Figure: Logical Address

Virtual Memory

The Benefits

1. Virtual memory **improves CPU utilisation**
   - Individual **processes take up less memory** (only partially loaded)
   - **More processes** in memory
   - More **efficient use of memory** (less internal fragmentation, no external fragmentation)
2. The **logical address space** can be **larger than physical address space**
   - 64 bit machine ⇒ 2^{64} logical addresses (theoretically)

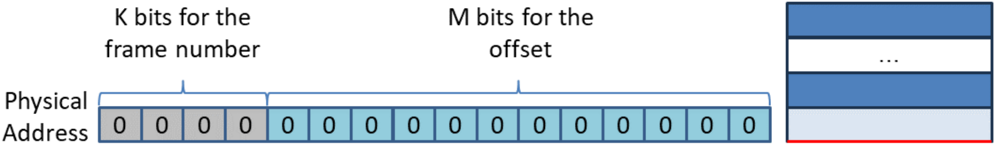
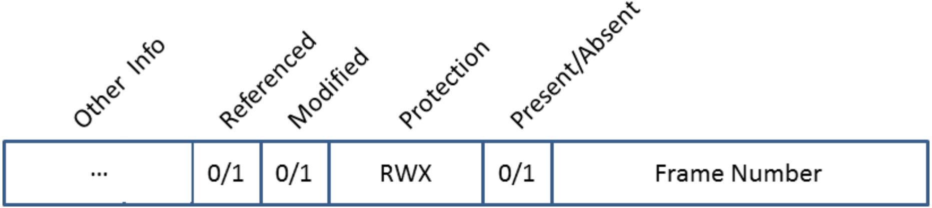


Figure: Logical Address

Virtual Memory

Page Tables Revisited: Contents of a Page Entry

- The “**present/absent bit**” is set if the **page/frame is in memory**
- The “**modified bit**” is set if the **page/frame has been modified**
    - Only modified pages have to be written back to disk when evicted
- The “**referenced bit**” that is set if the page is in use
- **Protection and sharing bits**: read, write, execute or combinations thereof



Other Info	Referenced	Modified	Protection	Present/Absent	
...	0/1	0/1	RWX	0/1	Frame Number

Figure: Page table entry

Page Tables Revisited: Page Table Size

- For a **16 bit machine**, the total address space is 2^{16}
  - Assuming that 10 bits are used for the offset (2^{10})
  - 6 bits can be used to number the pages
  - I.e., $2^6 = 64$ pages can be maintained
- For a **32 bit machine**, total address space is 2^{32}
  - Assuming pages of 2^{12} bytes (4KB)
  - 20 bits can be used to number the pages
  - I.e. 2^{20} pages (approx. 1 million) can be maintained
  - 4MB at 4 bytes per page table entry!
- For a **64 bit machine** . . .

Virtual Memory

Page Tables Revisited: Dealing with Large Page Tables

1. **Size**: how do we deal with **the increasing size of page tables**, i.e., where do we store them?
    - Their size prevents them from being **stored in registers**
    - They have to be stored in (virtual) **main memory**:
        - **Multi-level** page tables
        - **Inverted page tables** (for large virtual address spaces)
2. **Speed**: address translation happens at every memory reference, it has to be fast!
    - How can we maintain **acceptable speeds**?
    - Accessing main memory results in **memory stalls**

Virtual Memory

Page Tables Revisited: Multi-level Page Tables

- **Solution**:
  - Page the page table!
  - **Tree-like** structures to hold page tables
- The **page number** is divided into:
  - An **index to a page table** of second level
  - A **page within a second level** page table
- The page table is **not kept entirely in memory**

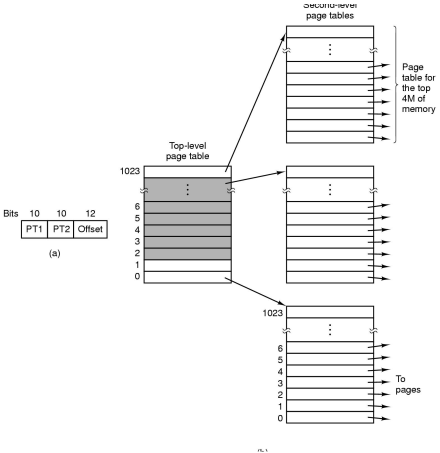


Figure: Multi-level page tables (from Tanenbaum)

Virtual Memory

Page Tables Revisited: Multi-level Page Tables

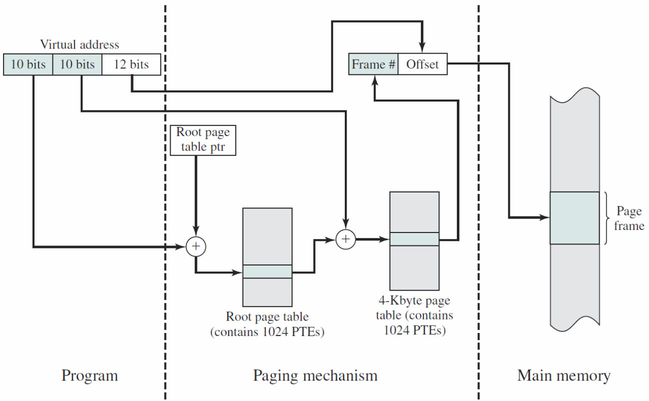


Figure: Multi-level Address Translation (from Stallings)

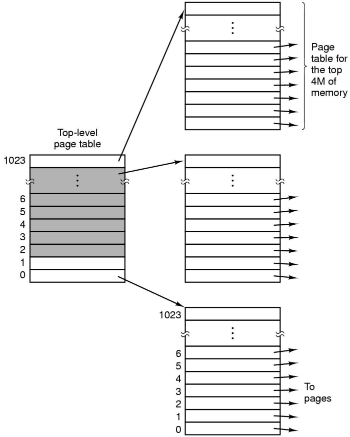


Figure: Multi-level page tables (from Tanenbaum)

Virtual Memory

Page Tables Revisited: Access Speed

- **Memory organisation** of multi-level page tables:
  - The **root page table** is always maintained in memory
  - Page tables themselves are maintained in **virtual memory** due to their size
- Assume that a **fetch** from main memory takes T nano seconds
  - With a **single page table level**, access is $2 \times T$
  - With **two page table levels**, access is $3 \times T$
  - ...

Test Your Understanding

Address Translation: Exercises

- Given a 4KB page/frame size, and a 16-bit address space, calculate:
    - Number **M** of bits for offset within a page
    - Number **N** of bits for representing pages
- What is the physical address for 0, 8192, 20500 using this page table?

Pages		Frames	
0	0000	0010	2
1	0001	0001	1
2	0010	0110	6
3	0011	0000	0
4	0100	0100	4
5	0101	0011	3
6	0110	X	X
7	0111	X	X
8	1000	X	X
9	1001	0101	5
10	1010	X	X
11	1011	0111	7
12	1100	X	X

Table: Page table

Summary

Take-Home Message

- **Paging** splits logical and physical address spaces into small **pages/frames** to reduce internal and external fragmentation
- **Virtual memory** exploits the principle of **locality** and allows for processes to be **loaded only partially** into memory, **large logical address spaces** require “different” approaches
- Reading: Tanenbaum Section 3.3