Chapter 10: Virtual Memory

Lesson 03:

Page tables and address translation process using page tables

Objective

- Understand page tables
- Learn the offset fields in the virtual and physical addresses
- Understand page table entries format
- Learn the use of virtual page number, page frame page number, presence bit, changed indication bit, and access rights bits

Objective

- Understand address translation process using page table
- Learn feasibility of inverted page table

Page table

Page table

- *A* data structure in memory that holds the mapping of virtual to physical addresses
- Enable determination of whether or not the virtual page (in secondary memory) containing the address referenced by the operation is currently mapped onto a physical page frame and present (in physical memory)

Page table

- The page table section does not need to keep track of the full virtual or physical address of a page that is mapped in the main memory
- Both virtual and physical page frames are always aligned on a multiple of their size

Virtual page number

Virtual addresses division into the virtual page identifier and offset

- Virtual page number (VPN) defined by a set of bits identify the virtual page
- Along with VPN, a set of bits describes the offset from the start of the virtual page to the virtual address
- Offset─ displacement defining the relative address with respect to page frame base address

Physical page number

Physical addresses division into the page frame identifier and offset

- Physical page number (PPN) defined by a set of bits identify the physical page frame
- The virtual and physical pages in a given system are generally the same size, so the number of bits ($\log_2$ of the page size) required to hold the offset field of the virtual and physical addresses are the same

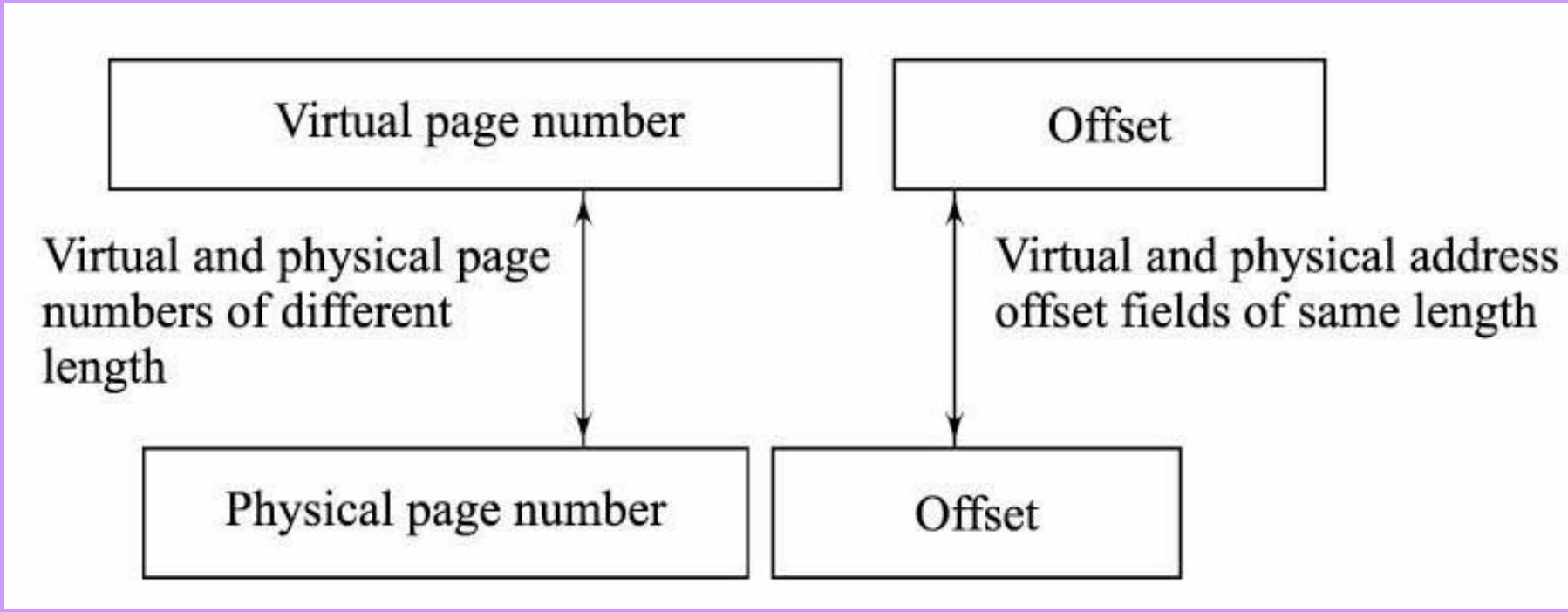
Example of address mapping, 32-bit long virtual and physical addresses and 4-kB pages

Virtual Page Number	*Physical page frame Number*
0xabc89	0x97887
0x13385	0x99910
0x22433	0x00001
0x54483	0x1a8c2

VPN and PPN lengths

- May or may not be identical
- Many systems, particularly 64-bit systems, have longer virtual addresses than physical addresses
- Reason— Current impracticality of building a system with 2^{64} bytes of DRAM memory

Virtual and physical addresses



Address translation

Address translation

- When a virtual address is translated, the operating system looks up the entry corresponding to the VPN in the page table and returns the corresponding PPN
- The offset bits of the virtual address then concatenated onto the PPN to generate the physical address

Example

- A virtual page with 32-bit addresses between 0xffff e800 to 0xffff efff
- Virtual Page size = (0xffff efff – 0xffff e800 + 1) = 2 × 1024 B = 2 kB
- Number of offset bits = $\log_2 (2 \times 1024) = \log_2(2^{11})$
- Number of VPN bits = 32 – 11 = 21

VPN and PPN

- VPN_1 = 0b1111 1111 1111 1111 1110 1 (21 bits)
- Assume that a page frame set of addresses at 0x007000 (24-bits), where the page is loaded (fitted)
- PPN_1 = 0b0000 0000 0111 0 (of 24 – 11 = 13 bits)

Page of VPN1 translation to PPN1

- When the page loads into the page frame from the virtual memory with one virtual page size same as one page frame size

Example 2

- The address of a memory word = 0xffff e90A
- Memory word offset (also called displacement) on the page = 0xffff e90A – 0xffff e800 = 0x10a = 266_d
- When this page loads into main memory on address translation, the offset remains the same
- In the main memory, the address will be 0b0000 0000 0111 0 000 0000 0000 + 0b001 0000 1010 = 0x00710a

64-bit virtual addresses and 43-bit physical addresses in a system

- Find how many bits are required for the VPN and PPN if the pages are 8 kB
- $\log_2$ (8 kB) = 13, so 13 bits are required for the offset field of both the virtual and physical address assume page size same as page frame size

64-bit virtual addresses and 43-bit physical addresses in a system

- 51 bits (64 – 13) are required for the virtual page number
- 30 bits (43 – 13) required for the physical page frame number

Separate page tables for separate programs

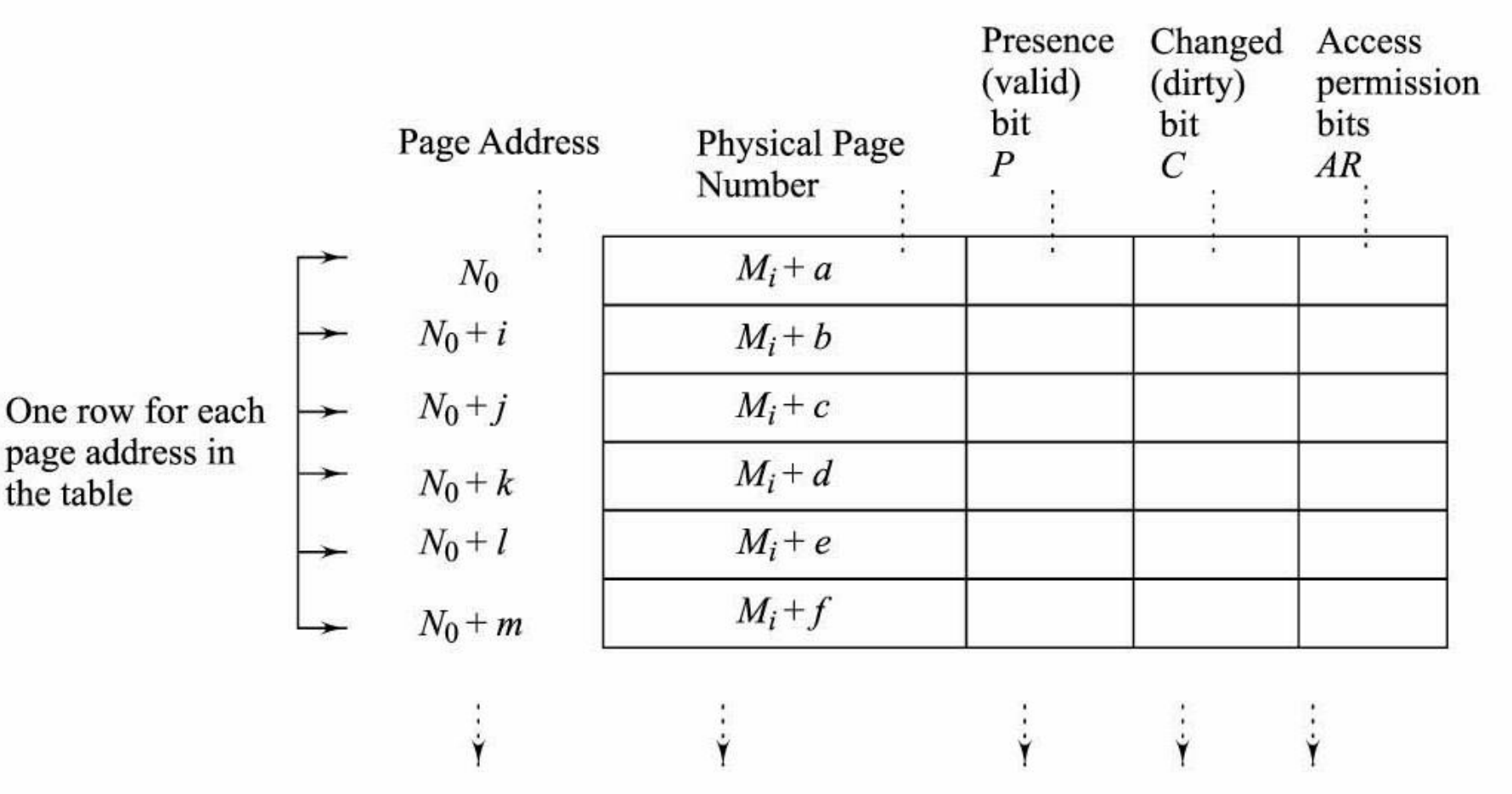
Separate page tables are required for each program on the system

- Because each program has its own virtual-physical address mapping
- The simplest page table implementation is just an array of page table entries, one entry per virtual page, and is known as a single-level page table to distinguish it from the multilevel page tables

Address translation

- To perform an address translation, the operating system uses the virtual page number of the address as an index into the array of page table entries to locate the physical page frame number corresponding to the virtual page

A page table for a system whose virtual address space six pages long



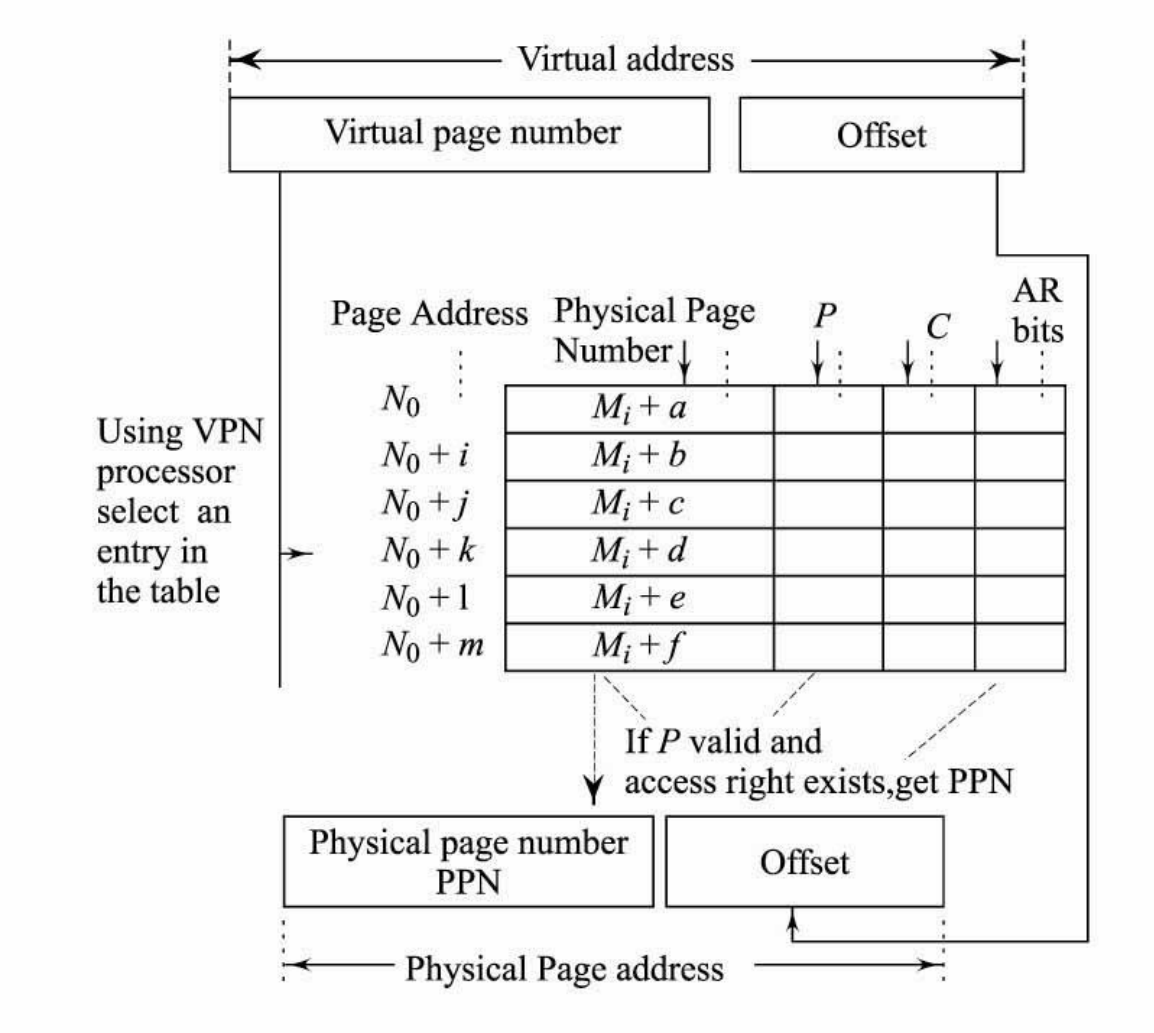
Page table entries

- Generally contain a physical page frame number, a valid bit, P (can also be called presence bit), change bit, C (can also be called dirty bit), and access rights, AR bits (read or write or read-write permission)

Use of a page table to translate a virtual address

- Systems with single-level page tables generally require that the entire page table be kept in physical memory at all times, so that the operating system can access the tab1e for address translations

Address translation using a page table



Enormous storage needs in page tables concept

Enormous storage requirement to hold page tables

- Even with multilevel page tables
- To map the entire address space of a modern processor requirement is enormous
- Amount of storage required for the page table grows with the size of the virtual address space, not the amount of storage on the system

Example

- Assume— on a system with 64-bit addresses and 4-kB pages, 2^{52} page table entries are required
- If each entry requires 7 bytes of storage to hold a 52-bit PPN, the valid bit, and the change (dirty) bit, 7×2^{52} B of storage (about 30,000,000 GB) required to hold the page table for a single program

Example

- Given that hard disks only hold about 75 GB of data, it is clearly impractical to implement such a page table, even if its organization allows much of it not to be mapped onto the main memory at a given time

Inverted page tables concept

Inverted page tables

- A technique that greatly reduces the amount of storage required for page tables in systems with large address spaces
- Consists of a set of entries, one for each physical page frame in the system
- Each entry in the inverted page table contains the VPN of the virtual page mapped into that page frame
- Thus, the amount of storage required for the page table depends on the amount of main memory in the system, not the size of the virtual address space.

Inverted page table Entries

- Each entry in the inverted page table contains the VPN of the virtual page mapped into that page frame
- Thus, the amount of storage required for the page table depends on the amount of main memory in the system, not the size of the virtual address space

Inverted page table entries

- Organized by physical address, not virtual address, they are harder to search than conventional page tables, since the virtual address cannot be used to determine the appropriate entry in the page table
- Typically, data structures such as hash tables are used to reduce the time to search the inverted page table

Mapping of a subset of the virtual address space instead of the entire address space

- The page table contains an additional field that indicates the range of addresses mapped by the page table
- This approach requires page table entries for the entire range of virtual addresses between the lowest and highest addresses used by the program, so its efficiency depends on whether a program allocates data densely or sparsely

Summary

We learnt

- Page tables, offset fields in the virtual and physical addresses, virtual page number, page frame page number, presence bit, changed indication bit, and access rights bits
- Page table entries format
- Address translation process using the page table
- Inverted page table organisation also feasible

End of Lesson 03 on

Page tables and address translation process using page tables