Paging – Addressing

Paging

- Paging is a memory-management scheme that permits the physical address space of a process to be non-contiguous
- Paging avoids the considerable problem of fitting memory chunks of varying sizes onto the backing store; most memory-management schemes used before the introduction of paging suffered from this problem.
- The problem arises because, when some code fragments or data residing in main memory need to be swapped out, space must be found on the backing store.

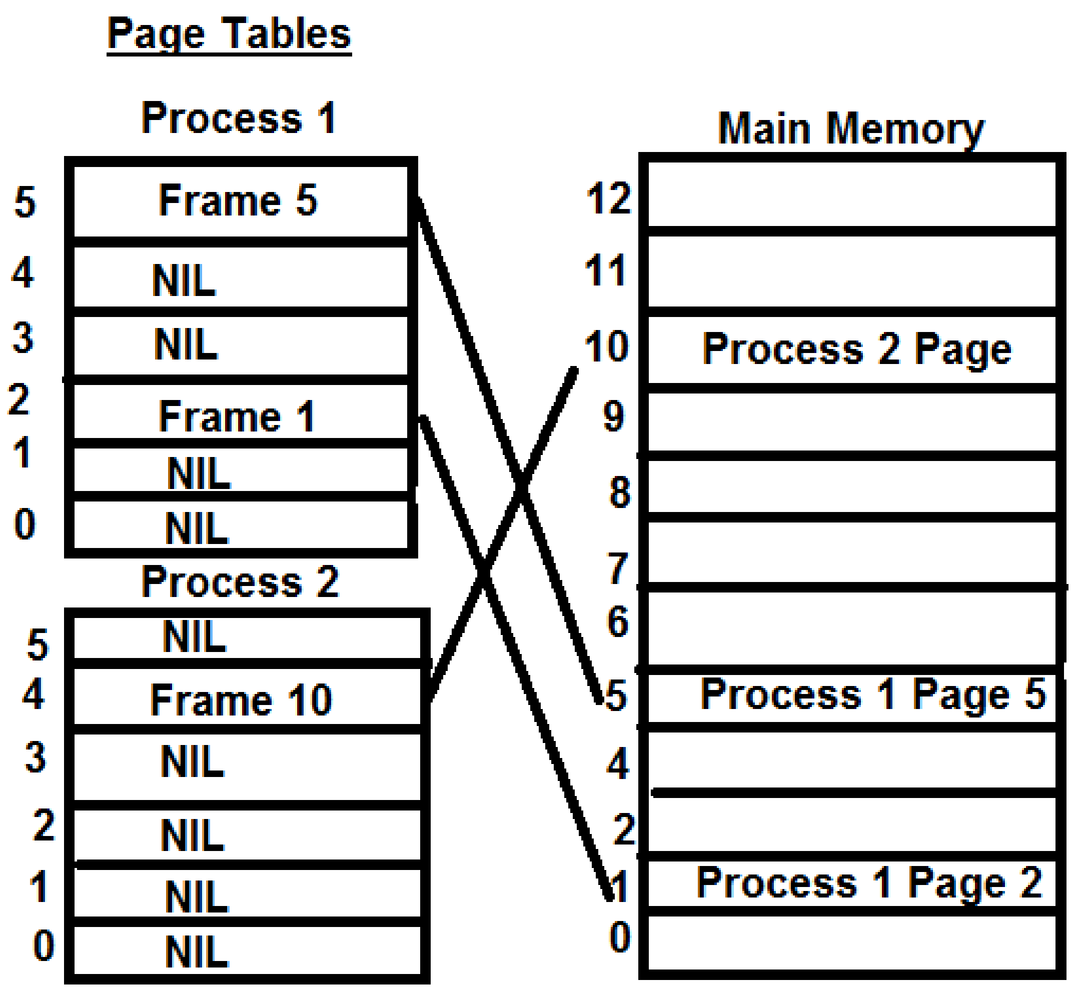
- The backing store also has the fragmentation problems; except that access is much slower.
- so compaction is impossible.
- Since paging keeps every memory partition to be of a fixed size, no possibility of small pieces of partitions that are unusable.
- Because of its advantages over earlier methods, paging in its various forms is ***commonly used*** in most operating systems.

Basic Method

- The basic method for implementing paging involves breaking physical memory into fixed-sized blocks called **frames** and breaking logical memory (**virtual memory**) into blocks of the same size called **pages.**
- When a process is to be executed, its pages are loaded into any available memory frames from the backing store (**disk**).
- The backing store is also divided into fixed-sized blocks that are of the same size as the memory frames.

- Virtual memory is the one that comprise of main memory and a portion of secondary memory.
- No process is completely loaded in main memory.
- Some pages are loaded into main memory and remaining are maintained in the secondary memory
- But logically all the pages are fully loaded.

- Since some pages are not loaded into main memory when a instruction references a address that is not yet loaded in MM it results in a page fault.
- But to determine whether a page is loaded in memory or not and if it is already loaded in memory then in when which memory frame it is loaded a mechanism is needed.
- Page table provides that facility to map a given logical address comprising of page no into a physical address comprising of frame no.



Page Tables
Process 1
5
Frame 5
4
NIL
3
NIL
2
Frame 1
1
NIL
0
NIL
Process 2
5
NIL
4
Frame 10
3
NIL
2
NIL
1
NIL
0
NIL
Main Memory
12
11
10
Process 2 Page
9
8
7
6
5
Process 1 Page 5
4
2
1
Process 1 Page 2
0

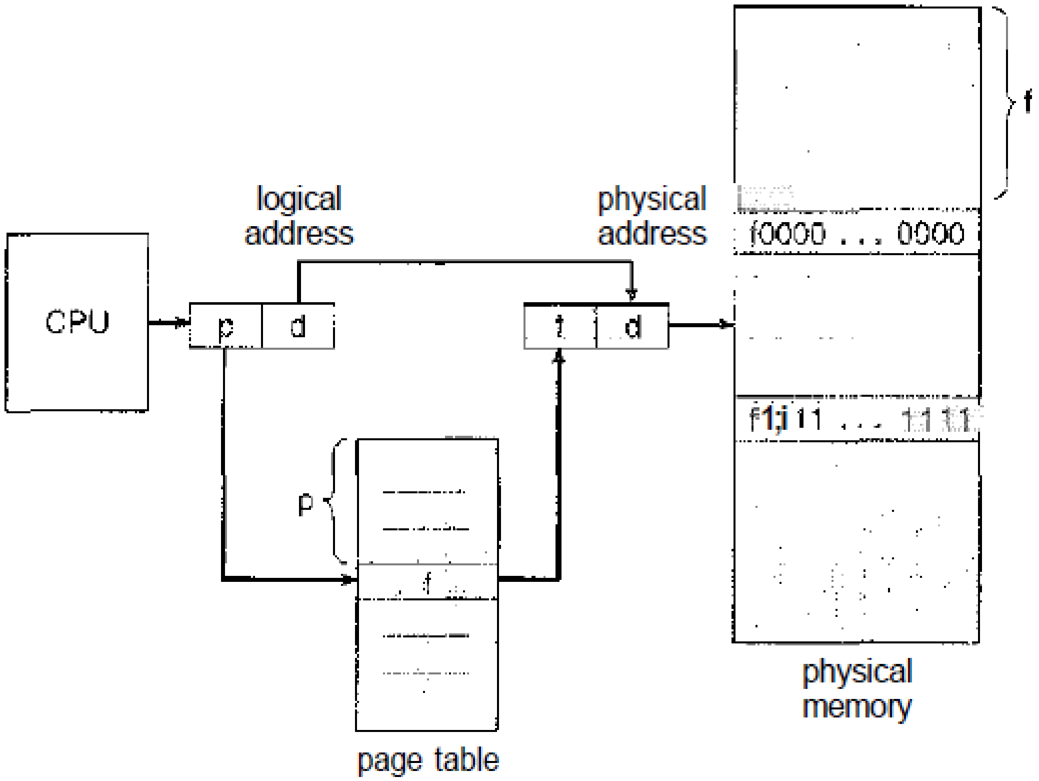
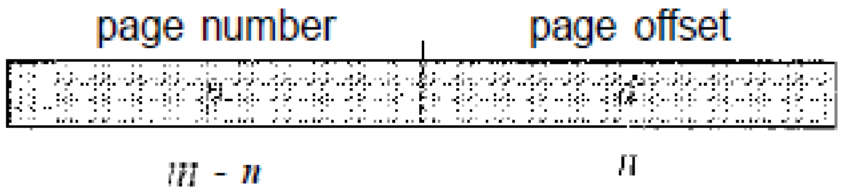


Figure 8.7 Paging hardware.

- Every address generated by the CPU is divided into two parts:
  - **page number (p)**
  - **page offset (d).**
- The page number is used as an index into a **page table.**
- The page table contains the base address (frame no) of each page in physical memory.
- This base address is combined with the page offset to define the physical memory address that is sent to the memory unit.

- The page size (like the frame size) is defined by the hardware.
- The size of a page is typically a power of 2, varying between 512 bytes and 16 MB per page, depending on the computer architecture.
- The selection of a power of 2 as a page size makes the translation of a logical address into a page number and page offset particularly easy.

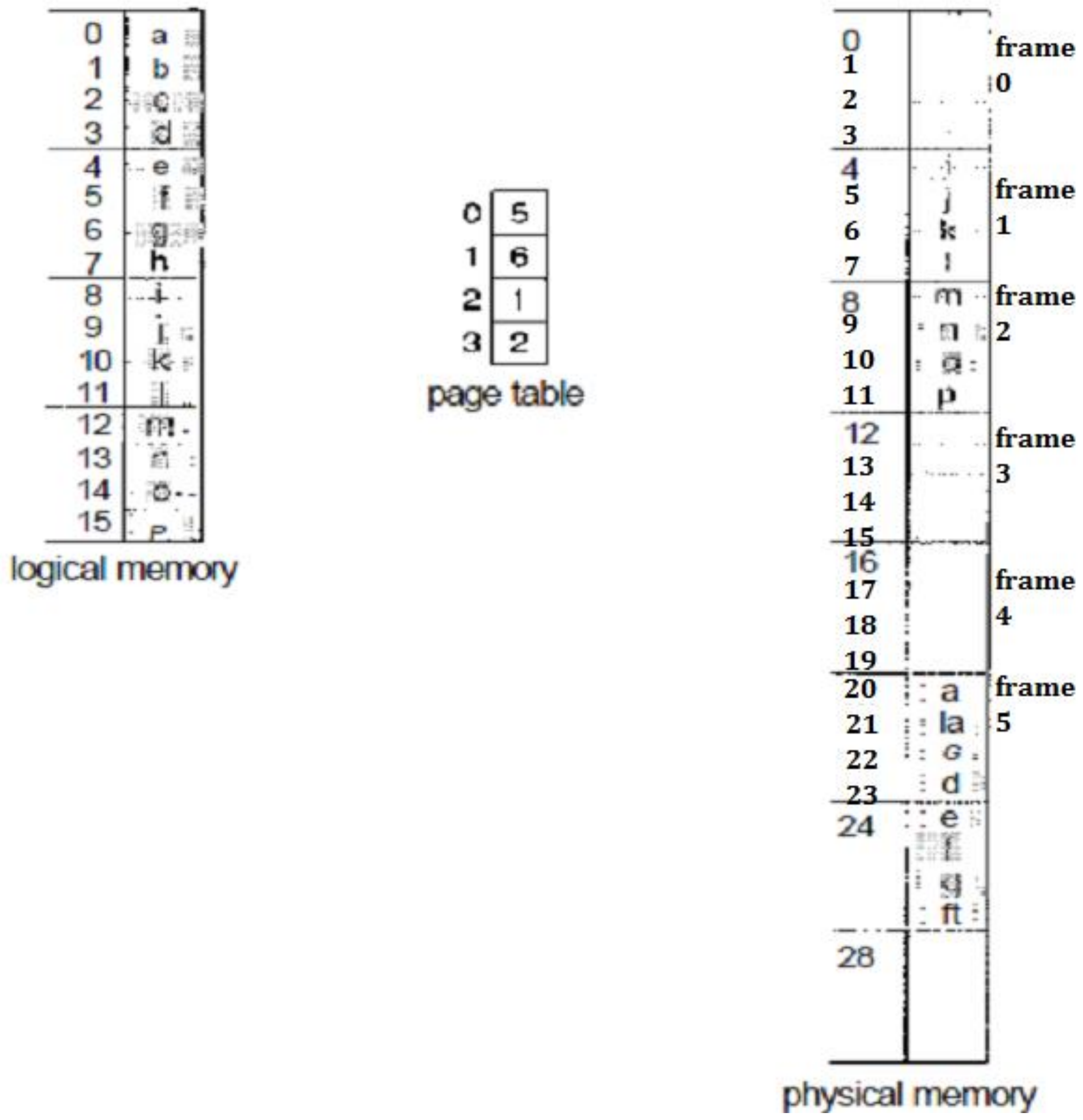
- If the size of logical address space is 2^m and a page size is 2^n addressing units (bytes or words), then the high-order *m – n* bits of a logical address designate the page number, and the *n* low-order bits designate the page offset.



where *p* is an index into the page table and *d* is the displacement within the page.

Example in decimal

- Using a page size of 4 bytes and a physical memory of 32 bytes (8 pages).
- Then logical address 0 is page 0 (0/4) & offset 0 (0%4)
- Indexing into the page table, we find that page 0 is in frame 5.
- Thus, logical address 0 maps to physical address 20 (= (5 x 4) + 0).
- Logical address 3 (page 0 (3/4=0), offset 3 (3 % 4)) maps to physical address 23 ((5x4) + 3).
- Logical address 4 is page 1, offset 0; according to the page table, page 1 is mapped to frame 6. Thus, logical address 4 maps to physical address 24 (= (6x4) + 0(4%4)).


0 a
1 b
2 c
3 d
4 e
5 f
6 g
7 h
8 i
9 j
10 k
11 l
12 m
13 n
14 o
15 p
logical memory
0 5
1 6
2 1
3 2
page table
0
1
2
3
frame 0
4
5 j
6 k
7 l
frame 1
8 m
9 n
10 o
11 p
frame 2
12
13
14
15
frame 3
16
17
18
19
frame 4
20 a
21 la
22 G
23 d
frame 5
24 e
f
g
ft
28
physical memory

Logical address 13 maps to physical address ?

- **13/4 = 3 So page is 3**
- **13%4 = 1 so offset =1**
- **Page 3 is in Frame 2**
- **Physical address = (2 X 4) + 1) = 9**

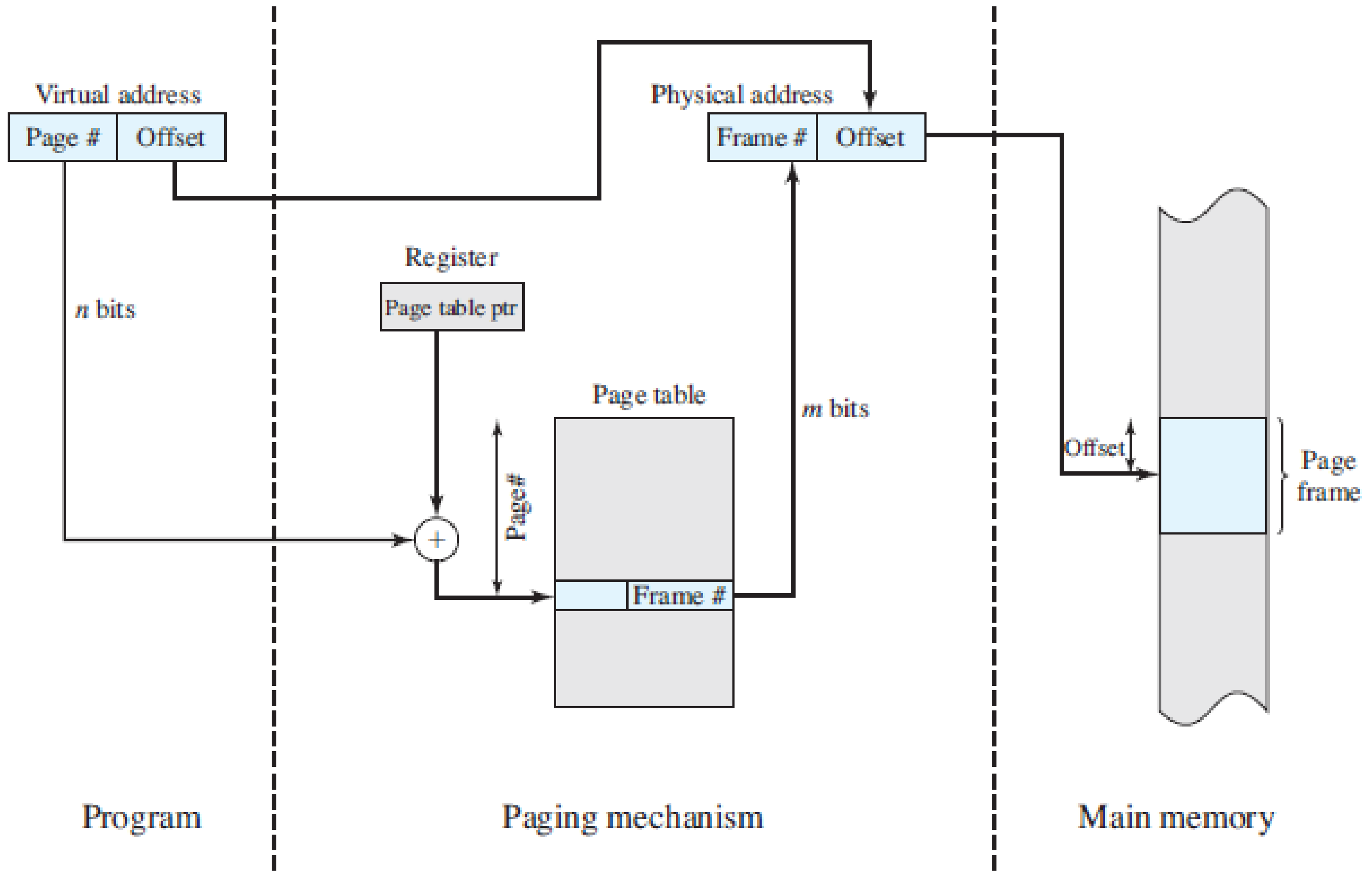


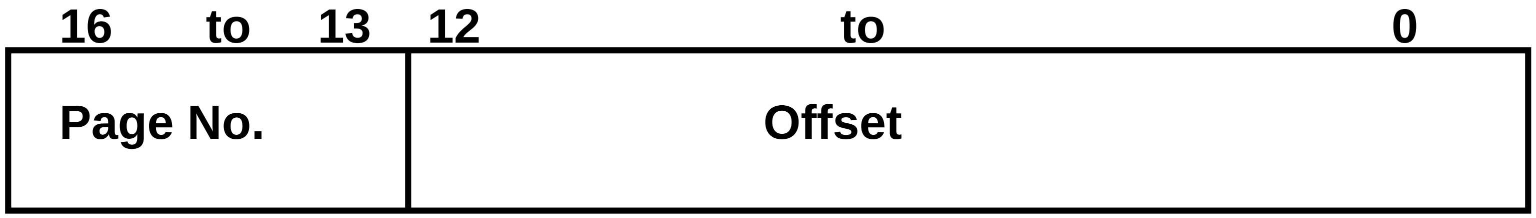
Figure 8.3 **Address Translation in a Paging System**

- Typically, the page number field is longer than the frame number field ($n > m$). Virtual address
- In most systems, there is one page table per process

Example in binary

- Given that the virtual memory size is **64 KB, 32 KB** main memory and page size is **4K**
- Then **16-bit addresses** are needed for virtual memory of 64 KB (64 KB = 2^{16}).
- Address range is 0 to 65535 (2^{16}).
- The relative address **1502**, in binary form, is **0000010111011110**.
- Page size is 4 K. Then there are 16 pages in virtual memory and 8 frames in main memory.
- No of bits to needed identify a page is 4 (2^4 =16)
- (P1 - 0000,P2 - 0001, P3 – 0010, P4 – 0011, P5- 0100, P6- 0101, P7- 0110, P8- 0111, P9- 1000, P10- 1001, P11- 1010, P12- 1011, P13- 1100, P14- 1101, P15- 1110, P16 – 1111)

- A single page comprise of 4096 bytes. So an offset field of 10 bits is needed (2^{12} = 4024).
- Therefore virtual address format is



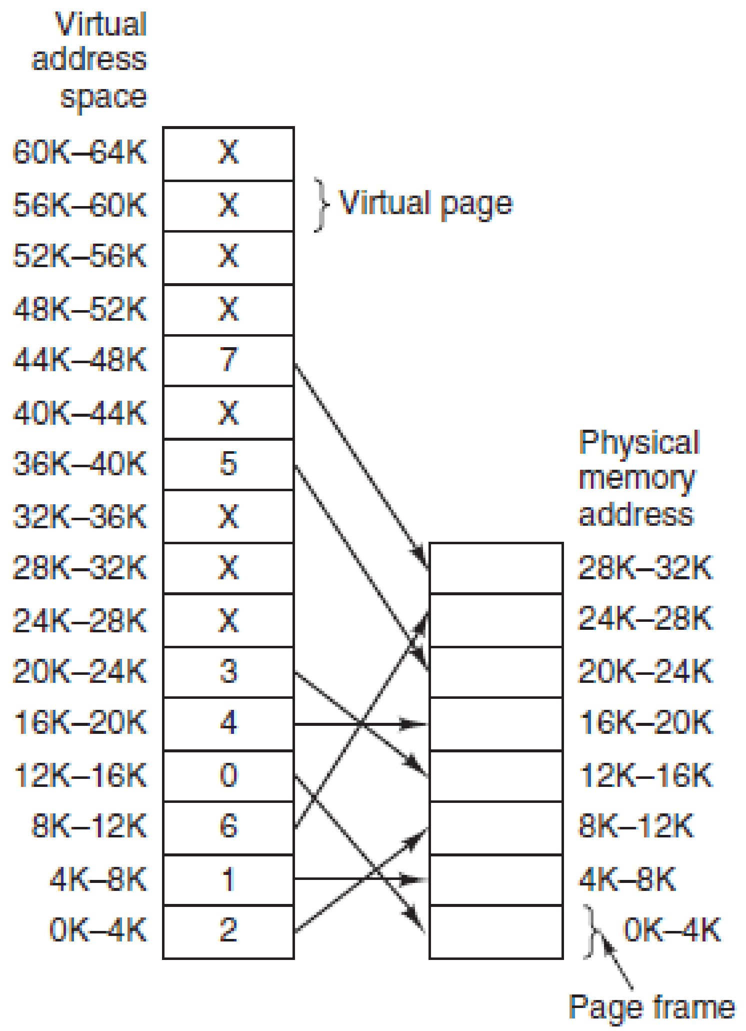
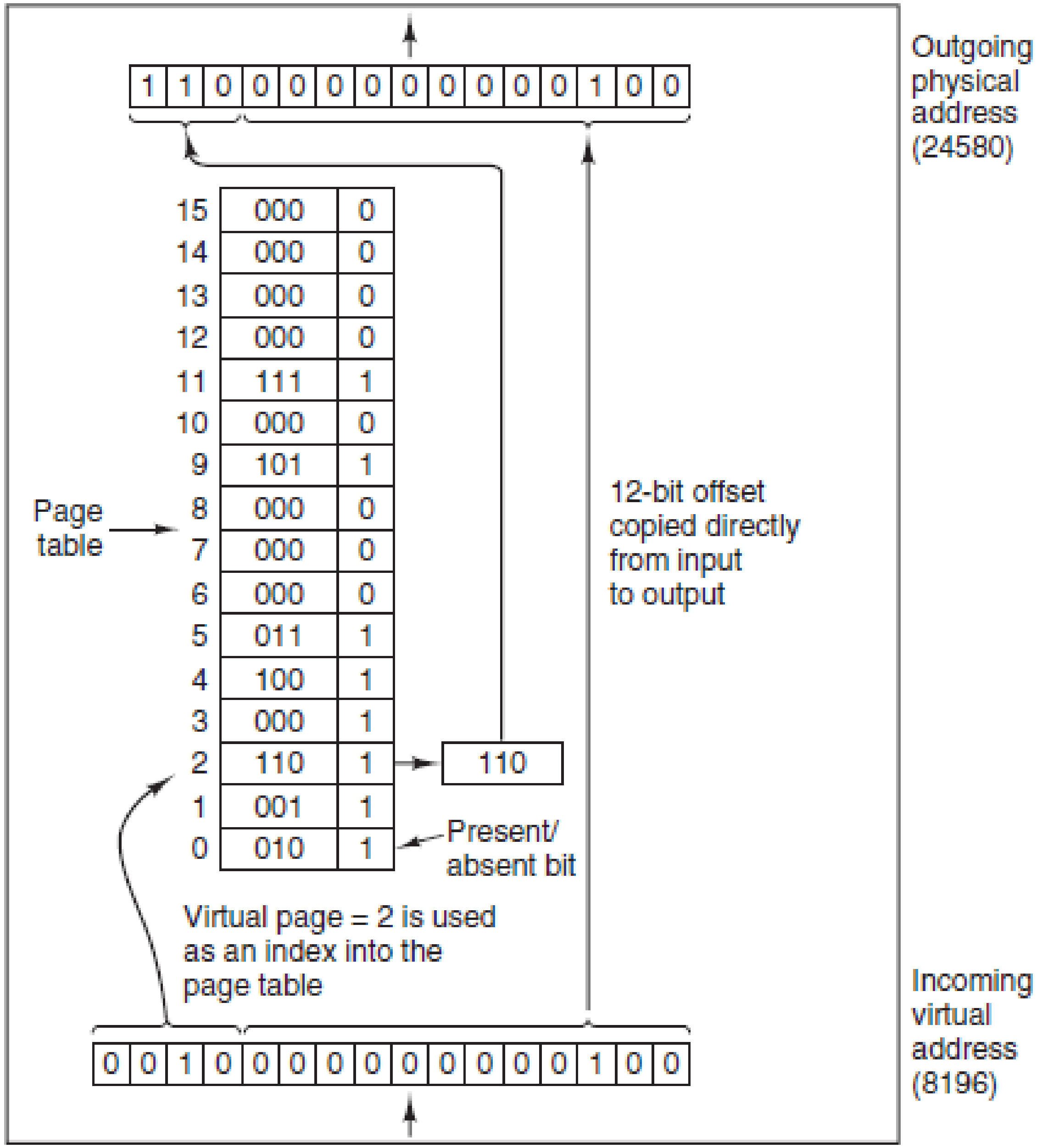


Figure 3-9. The relation between virtual addresses and physical memory addresses is given by the **page table**. Every page begins on a multiple of 4096 and ends 4095 addresses higher, so 4K–8K really means 4096–8191 and 8K to 12K means 8192–12287.

- The range marked 0K–4K means that the virtual or physical addresses in that page are 0 to 4095.
- The range 4K–8K refers to addresses 4096 to 8191, and so on.
- Each page contains exactly **4096 addresses** starting at a multiple of 4096 and ending one shy of a multiple of 4096.
- When the program tries to access address 0 MMU sees that this virtual address falls in page 0 (0 to 4095), which according to its mapping is page frame 2 (8192 to 12287). It thus transforms the address to 8192 and outputs address 8192 onto the bus.

- The instruction MOV REG,8192 is effectively transformed into MOV REG, 24576 because virtual address 8192 (in virtual page 2) is mapped onto 24576 (in physical page frame 6).
- Third example, virtual address 20500 is 20 bytes from the start of virtual page 5 (virtual addresses 20480 to 24575) and maps onto physical address 12288 + 20 = 12308.



Outgoing physical address (24580)
1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0
Page table
15 000 0
14 000 0
13 000 0
12 000 0
11 111 1
10 000 0
9 101 1
8 000 0
7 000 0
6 000 0
5 011 1
4 100 1
3 000 1
2 110 1
1 001 1
0 010 1
110
Present/ absent bit
12-bit offset copied directly from input to output
Virtual page = 2 is used as an index into the page table
Incoming virtual address (8196)
0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0