Chapter 12 – Part 2

Memory Organization

Based on slides by:
Prof. Myung-Eui Lee
Korea University of Technology & Education
Department of Information & Communication

Alon Schclar, Tel-Aviv College, 2009

Virtual Memory (VM)

- Programs and data are **stored** in **auxiliary** memory
- **Portions brought** into **main** mem when **needed**
  - by CPU
- Virtual memory is a **concept** (illusion)
  - Allows programs **as big as auxiliary** memory size
  - Actually relatively **small main** memory
- Each address goes through **address mapping**
  - From ***virtual*** address to ***physical*** address in main mem
  - **handled** automatically **hardware** via a **mapping table**.
  - **Dynamically** – during **execution**

Address space and Memory space

- ***Virtual address*** - address used by **program**
- ***Address space*** - set of **all virtual** addresses
  - Reference **data** and **instructions**
- ***Location*** or ***Physical address*** – **main** memory address
- ***Memory space*** – set of **all locations**
  - **actual main** memory locations **directly addressable** for **processing**
- **Non-virtual** / **virtual** memory systems –
  - the address space **=** / **>** memory spaces
- Example
  - **main-memory** capacity of **M = 32K** words – physical **address** consists **15 bits**
  - **Auxiliary** memory size is **N = 2^{20} = 1024K** words virtual address consists **20 bits** (=32**M* = 32 main memories)

Virtual Memory

- **Multiprogramming**: **programs** and **data** transferred **between auxiliary** and **main** memory
- Example: **Program 1** is being **executed** in the CPU
  - **Code** and **portion** of its **data** moved from **auxiliary to main** mem.
  - Portions need **not** be in **contiguous** locations in memory
    - **empty spaces** may be **available** in scattered locations
- **Programmers** may **use** the **entire address space**
  - Length of **address field** of the **instruction supports all virtual** addresses.
  - In our **example**,
    the **address** field is **20 bits** long
    but **physical** memory **addresses** have **only 15 bits**

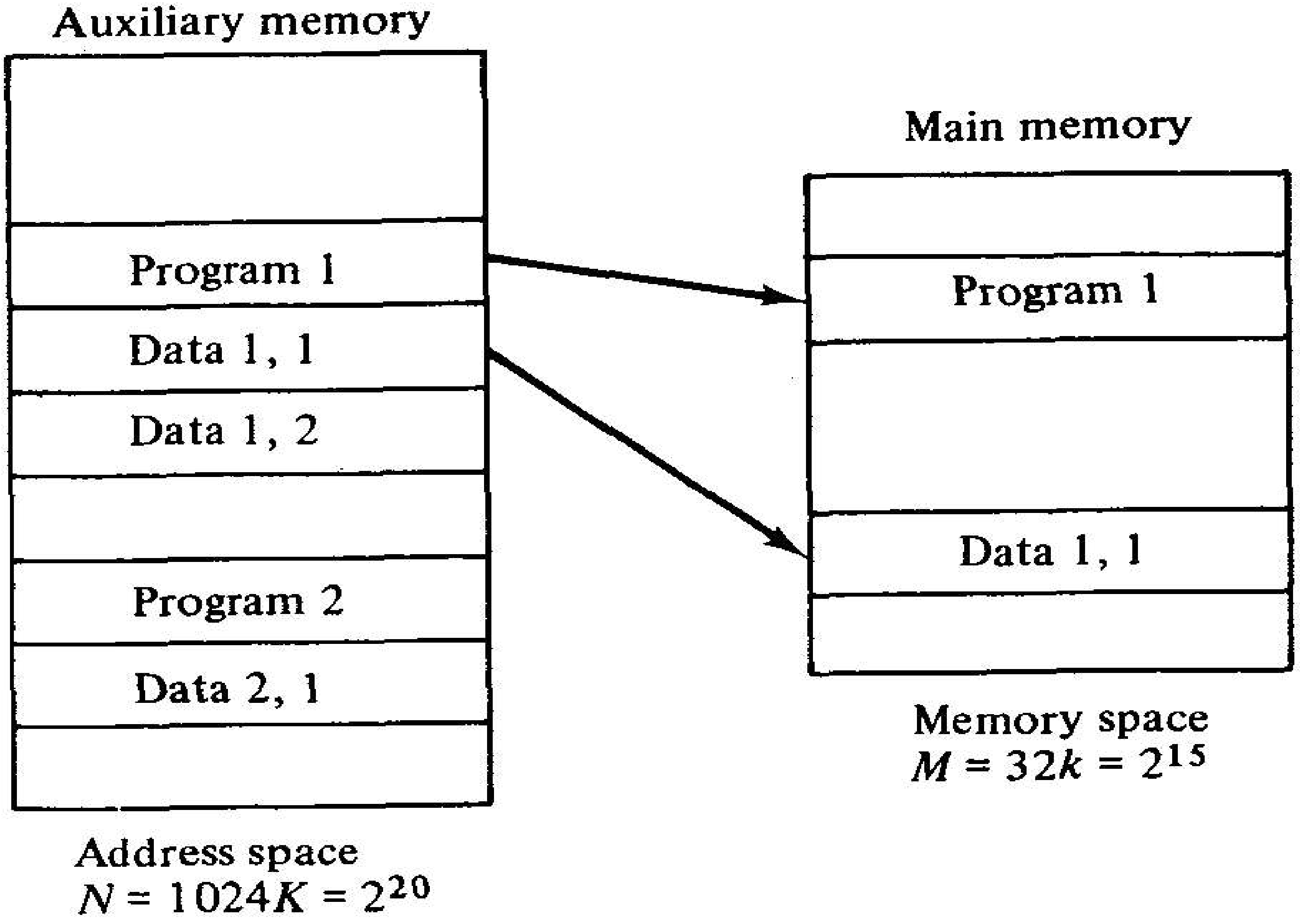


Figure 12-16 Relation between address and memory space in a virtual memory system.

Taken from: **M. Mano/Computer Design and Architecture 3rd Ed.**

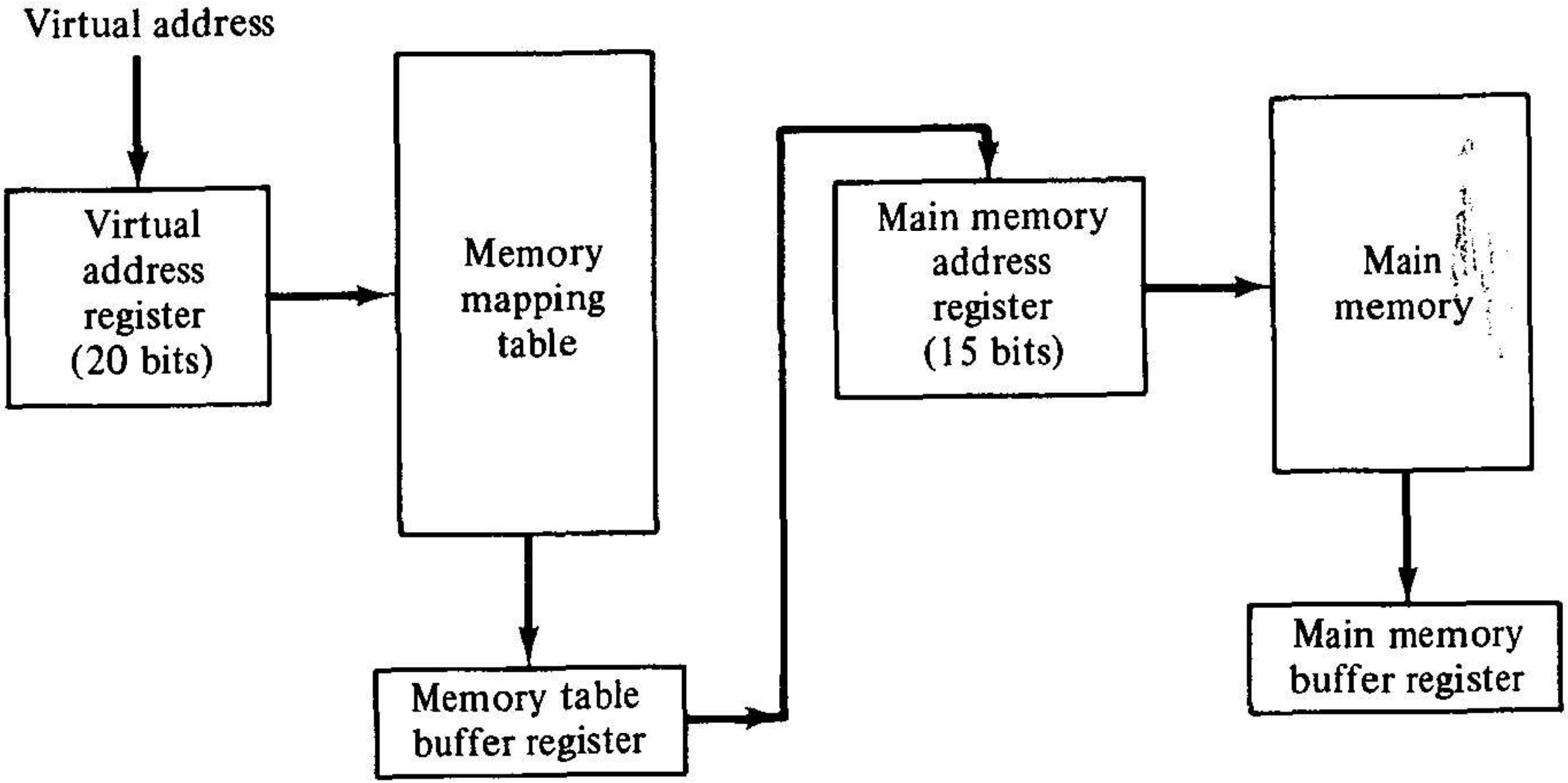
Virtual address mapping

- **CPU references** instructions/data **20-bit address**
- **Information** is **taken** from ***physical*** memory
  - **auxiliary** storage is **extremely slow**
  - **entire records transferred** from aux. to main mem. (efficiency)
- ***Page Table*** - **maps** a **virtual** address of **20 bits** to a **physical** address of **15 bits**
  - **translated immediately** when **referenced** by **CPU**
- Table **storage**
  1. **additional memory unit** –
     one extra memory access time.
  2. **in main memory – RAM**
     two accesses to memory - program runs at **half speed**
  3. **Associative memory**

Address Mapping Using Pages

- **Page = portion** of program/data **moved** between **auxiliary** and **main** memory
- Memory and programs **divided** into **pages** and **blocks** (page frame)
  - **page** refers to **address space**
  - **block** refers to the organization of **memory space**
  - Size range: 64 - 4096 words each.
- <u>Back to example</u>
  - A **page / block** consists of **1K words**
  - **address space** is divided into **1024 *pages***
  - **main memory** is divided into **32 *blocks***

Figure 12-17 Memory table for mapping a virtual address.


Virtual address
Virtual address register (20 bits)
Memory mapping table
Memory table buffer register
Main memory address register (15 bits)
Main memory
Main memory buffer register

Taken from: **M. Mano/Computer Design and Architecture 3rd Ed.**

RAM Example 1

- **Address space** = 2^{13} = 8K words
- **Memory space** = 2^{12} = 4K words
- **Page/block size** = 2^{10} = 1K words
  - **8 pages** and **4 blocks**
  - **Up to 4** pages **in main** memory – at any given time
- **Virtual address** consists of
  - **Page number** address (**3 msb**) and
  - **Line within page** (**10 lsb**)
- Note:
  - **line address** in address and memory space is the **same**
  - **mapping only** required from **page num** to **block num**

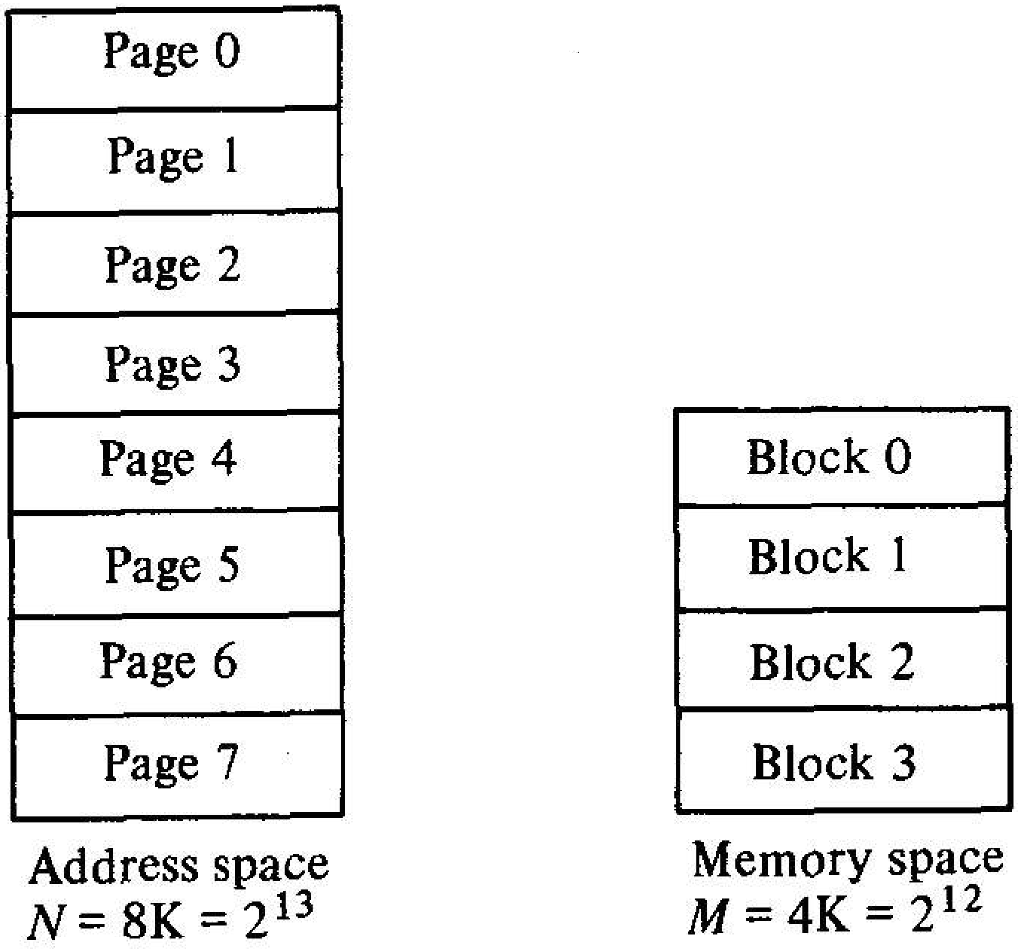


Figure 12-18 Address space and memory space split into groups of 1K words.

Taken from: **M. Mano/Computer Design and Architecture 3rd Ed.**

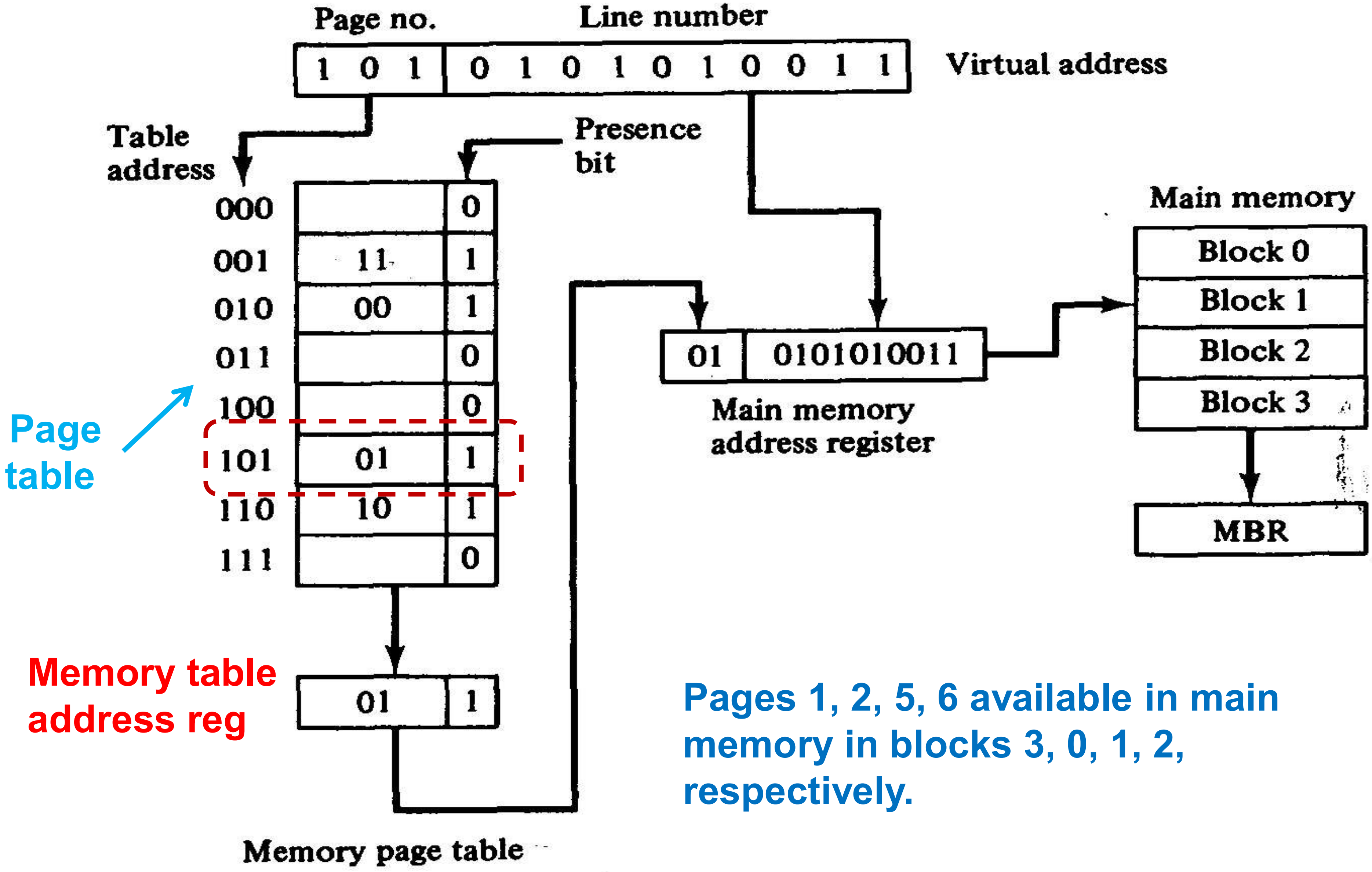
RAM - Example 2

- **Page table** consists **8 words** - one per page
  - **Address (entry#)** in page table **denotes** the **page number**
  - **Content** of entry is **block number** where page is in **main mem**.
  - **Presence bit - indicates** whether page **transferred from auxiliary** memory **into main** memory.
    - ***0*** - page is **not available** in main memory.
- **CPU** uses a **virtual address** of **13 bits**
  - The **3 msb** of the virtual address specify
    - a **page number equivalent** to
    - an **address** (entry#) for the **memory-page table**.
- **table buffer register** ← **page table[page number]**
  - The **content** of **memory page table word** at the **page number** address is **read into the memory table buffer register**.

Example 2 – cont.

- IF **presence bit = 1**
  - **block number** transferred **to 2 msb** of the main memory address register
  - **line number** from the virtual address is transferred **into** the **10 lsb** of *main memory address register*
  - A **read signal** to **main memory** transfers address **content** to the main memory buffer register
    - ready for CPU use
- IF **presence bit = 0**
  - **Content** of referenced word **not in main** memory.
  - **OS called** to **fetch** required **page** from **auxiliary** into main memory before resuming computation – ***Paging***

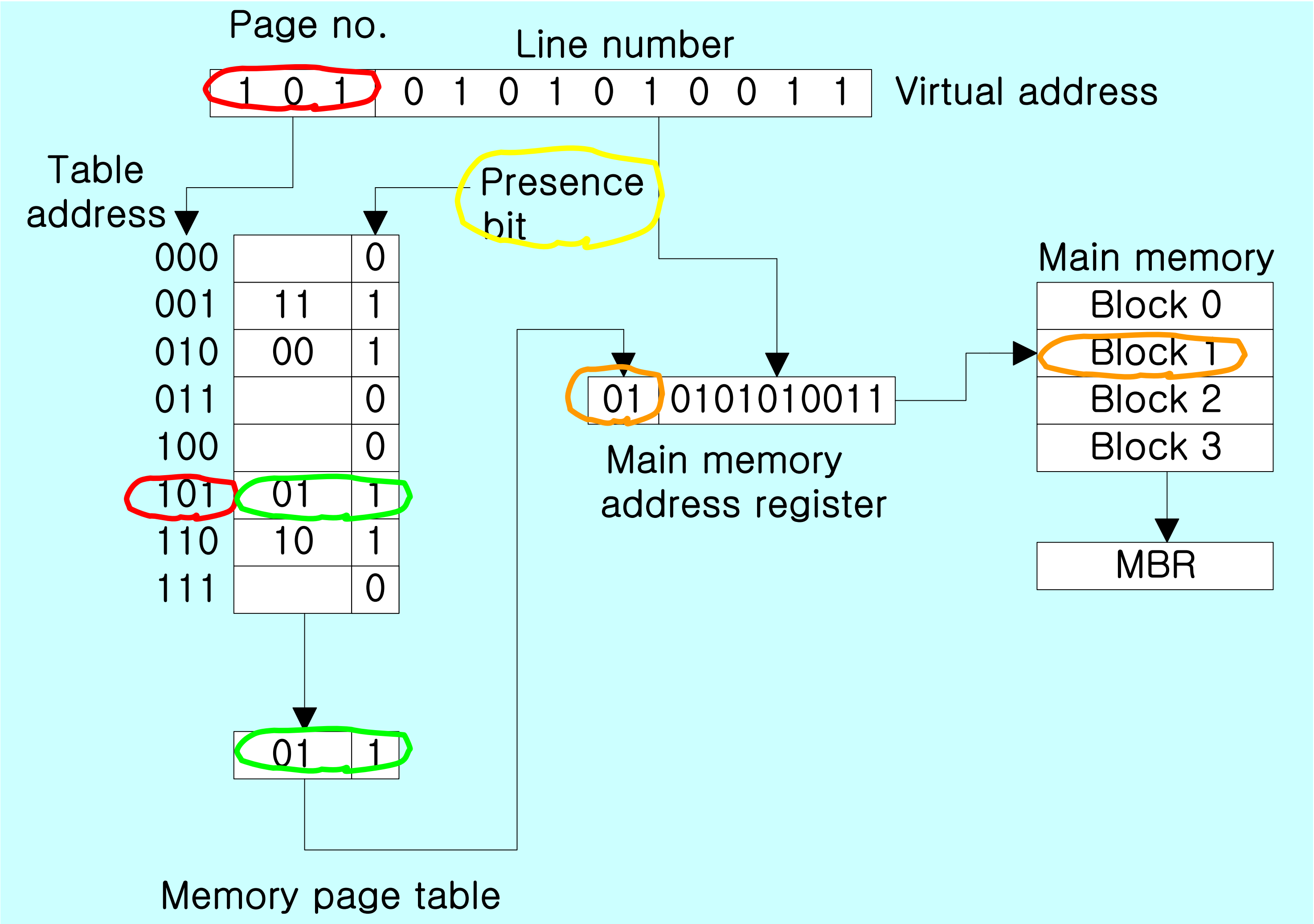
Figure 12-19 Memory table in a paged system.



Pages 1, 2, 5, 6 available in main memory in blocks 3, 0, 1, 2, respectively.

Taken from: **M. Mano/Computer Design and Architecture 3rd Ed.**

- Memory table in a paged system : *Fig. 12-19*



RAM Memory Page Table

- A **RAM** page table is **inefficient** in **storage**
  - In previous example: **8 memory words** needed, 1 per page
  - **At least 4** words are **always empty** –
    - main memory cannot accommodate more than 4 blocks.
  - **In general**: ***n*** pages and ***m*** blocks require
    - Memory-page table of ***n*** entries
    - **At most *m*** blocks will be **marked** with block numbers
    - **At least *n-m*** will be **empty**
- **Example**:
  - **Address space** - 1024K words
  - **Memory space** - 32K words
  - **Page/block** = 1K words
  - **Number of pages** - 1024
  - **Number of blocks** – 32
  - **Memory-page table size** = 1024 words
  - **At most 32** entries may have a **presence bit** equal to **1**
  - **At least 992** entries will be **empty** – **97% waste**

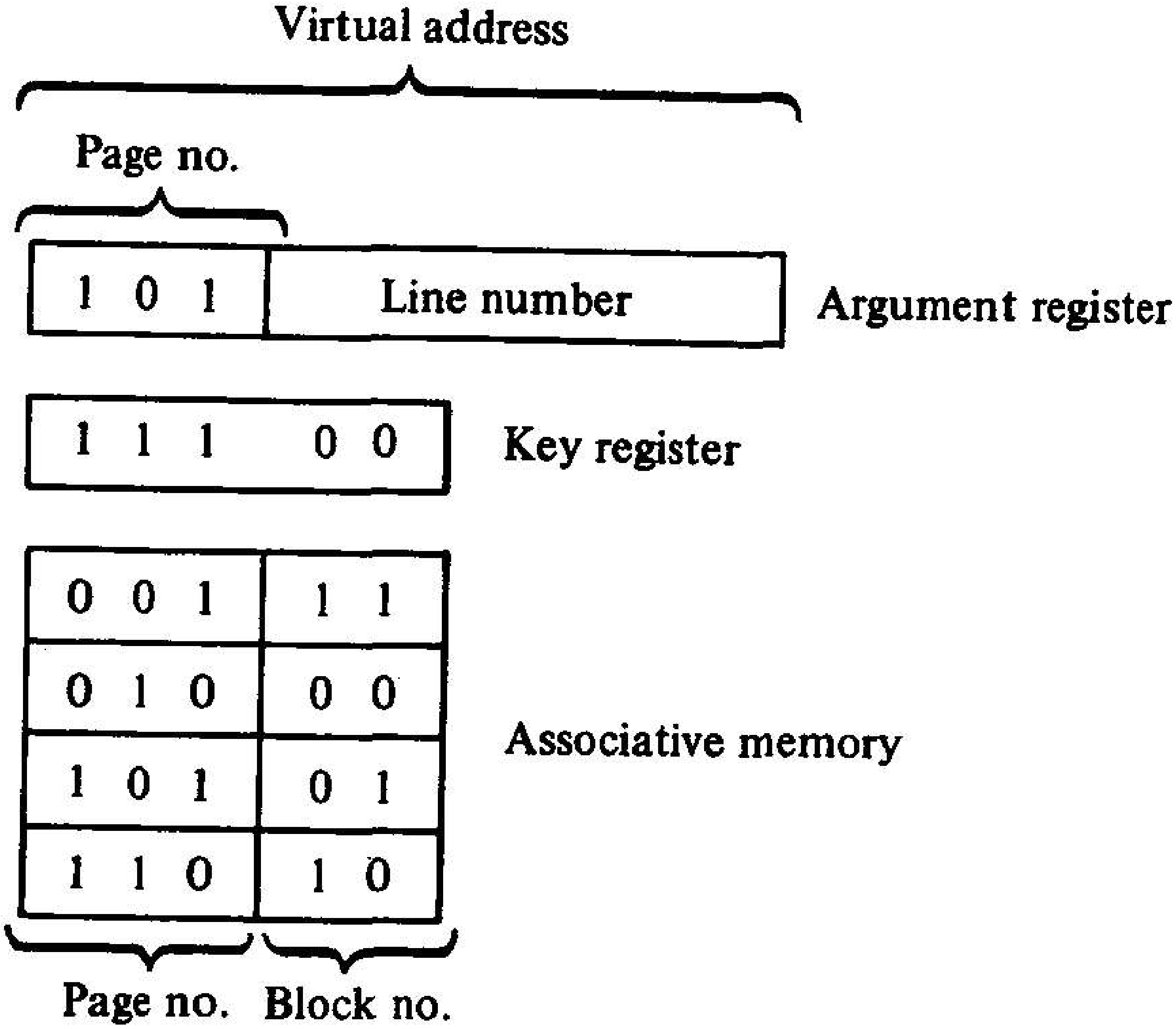
Associative Memory Page Table

- **More efficient**:
  Page table **size = number of main memory blocks**
  - **No wasted** space
- Implemented by **associative memory (AM)** - each word contains
  1. a **page number**
  2. corresponding **block number**
- **Content (search key)** is the **page number**
  - **Page fields** in table are **compared** to **page number** of **virtual address**
  - IF **match**: the **word** is **read** and its **block number** is **extracted**

Associative Memory Page Table – cont.

- <u>**Back to example**</u>
  - RAM replaced by **AM** of **4 words**
  - **Page number** field – **3 msb's**
  - **Block number** – **2 lsb's**
  - **Virtual address** is **placed** in **argument register (AR)**
  - The **page number bits** in the **AR** are **compared** with all **page numbers** in the **page field** of the **AM**
  - **IF page number found**
    - 5-bit **word** is **read** from **memory**
    - The **block number**, is **transferred to main** memory **address register**
  - **IF no match**
    - **OS is called** to **fetch required** page **from auxiliary** memory

Figure 12-20 An associative memory page table.



Page no.	Block no.
0 0 1	1 1
0 1 0	0 0
1 0 1	0 1
1 1 0	1 0

Taken from: **M. Mano/Computer Design and Architecture 3rd Ed.**

Page replacement

- A **VM system** contains **hardware** and **software**
- **Hardware**: **page mapping** mechanism
- **Software**: efficient **utilization** of **memory**
  - **which page** in main memory is **removed** to **make room** for a new page,
  - **when** a **new page** is **transferred** from **auxiliary** memory **to main** memory
  - **where** a **new page** is **placed** in **main** memory

Page fault

- When **program X** starts execution
  - **one or more pages** loaded into **main from aux**
  - **page table set** to indicate their position
- The program **executed from main memory until** it **references** a **page not** in **main** memory

  → ***page fault***
  - **Execution** of program **X** is **suspended**
  - **OS instructed to load** page from auxiliary memory
  - **OS assigns IOP** to the task (paging is an I/O op)
  - **Control** is **transferred** to the **next program** in memory
  - **After** page is **loaded** (**interrupt**) into main memory
  - **Program X** execution **resumes**

Replacement policy

- **Remove** page **least likely** to be **referenced** in the **immediate future**
- **FIFO**
  - Implemented using a **queue**
  - *Advantage* - **easy** to implement.
  - *Disadvantage* - under certain **circumstances** pages are **removed and loaded** from memory **too frequently**
- **LRU**
  - **Associating** a **counter** (aging registers) with **every page** in **main** mem.
  - When **page** is **referenced**, its **counter** is **zeroed**
  - Counters associated with **all pages** presently **in memory** are **incremented** at **fixed intervals** of time
  - **Least recently used pag**e = with the **highest count**

Memory Management

- A **multiprogramming** environment **requires**
  - **moving programs** and data **around** the **memory**
  - **Vary amount** of memory in **use** by a given program,
  - **Prevent** a **program** from **changing other** programs
- **How?** memory management system **MMS**
  - **Hardware** and **software** for managing programs residing in memory
  - **Part** of the **OS** in many computers
- **Focus** on **hardware** unit
  - **Software part** in "***Operating Systems***" course

MMS components

1. A facility for **dynamic storage relocation**
   - **maps logical** memory references into **physical** memory addresses
2. A provision for **sharing common programs** stored in memory **by different users**
   - **Example**: A single copy of a **C complier** used by all programmers
3. **Protection** against unauthorized access between users
   - **Example**: **unauthorized copying** of other users' programs
   - **Prevent abuses** of programs performing confidential activities
   - **Prevent** users from **performing OS** functions that may **interrupt** the **orderly** sequence of operations the computer

Storage allocation - Segments

- **Fixed-sized** pages cause **difficulties** with respect to **program size** and the **logical structure** of programs.
- ***Solution***: **divide** programs and data into **sets** of **logically** related instructions or data elements – ***segments***
  - **variable size**
  - associated with a given **name**
  - may be **generated** by the **programmer** or by the **OS**
  - Each segment has **protection** info associated with it.
- Examples: **procedure**, **array**

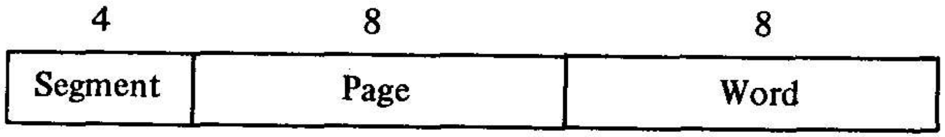
Logical address

- **Generated** by a **segmented** program.
- **Similar concept** as **virtual address** using **pages**
- **May** be **larger/equal/smaller** than **physical** address **memory address**
- **Shared programs** are placed in a **unique segment** in each user's logical address space
    - a **single physical copy** can be **shared**
- **MMS maps logical** addresses into **physical addresses**
    - **similar** to the **virtual** memory mapping concept

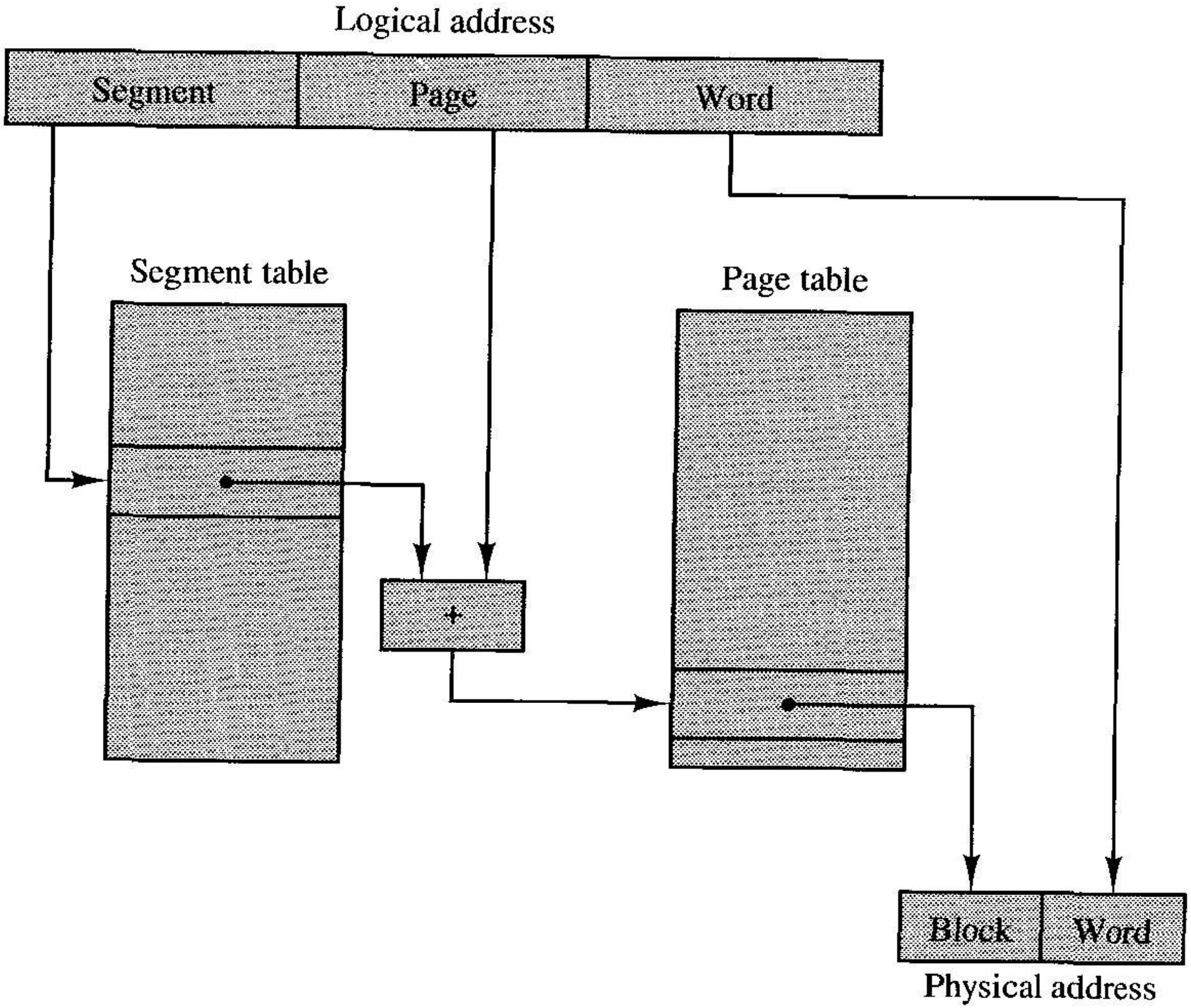
Segmented-Page Mapping

- **Segment length** can **grow/shrink** as **needed** by program
  - One way to specify segment length: **number of equal-size pages**
    - Length depends on the number of pages assigned to it
- The logical address consists of three fields:

1. **Segment number**
   - may be associated with ***1*** to ***2***^{x} pages
2. **Page within segment**
   - ***k*** bits can specify up to ***2***^{k} pages
3. **Word within page**



Taken from: **M. Mano/Computer Design and Architecture 3**rd **Ed.**



(a) Logical to physical address mapping

Taken from: **M. Mano/Computer Design and Architecture 3rd Ed.**

Segmented-Page Mapping – cont.

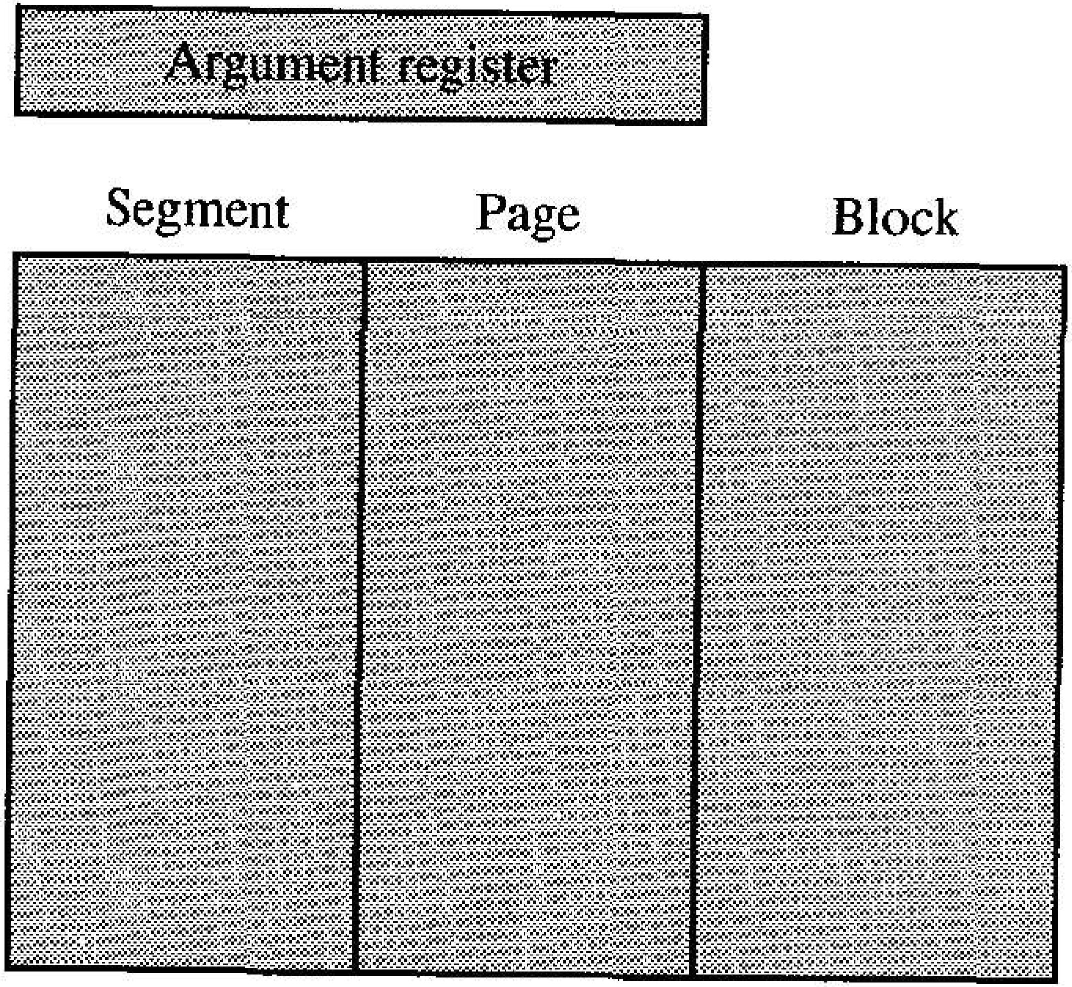
- **Mapping** logical to physical address done by **two tables**
  - **Segment** table
  - **Page** table
- Segment number - address (entry#) in segment table
- Segment table **entry points** to a **page table** **base address**
- Page table **base** is **added** to the **page number**
- **Sum** produces a **pointer address** to **entry** in the **page table**
- **Page table entry** is the **block number** in **physical** mem
- **Concatenation** of
  block field + word field → final physical mapped address

Segmented-Page Mapping – storage

- The two mapping tables may be stored in
  - **two separate small memories** or
  - in **main memory**
- Either case: 3 accesses required per CPU memory reference
  1. one from the **segment table**
  2. one from the **page table**
  3. the third from **main memory**
- **Slows** the system **significantly compared** to a conventional system that requires only **1 reference to memory**

Enhancing segment mapping

- Use **associative memory** to store the **most recently referenced** table entries
  - Sometimes called a ***translation lookaside buffer (TLB)***
  - **Cache principle**
  - Search key is **(segment #, page #)**
  - The **first** time a given block is **referenced** – **block#** with the corresponding **segment#** and **page#** are **stored**
  - **Mapping** process
    - **first** perform **associative search** with **(segment #, page #)**
    - **If succeeds** - mapping **delay** is **only** that of the **associative memory**
    - **Otherwise** –
      - the **slower table** mapping is **used** and
      - the **result stored** in the **associative** memory for **future reference**



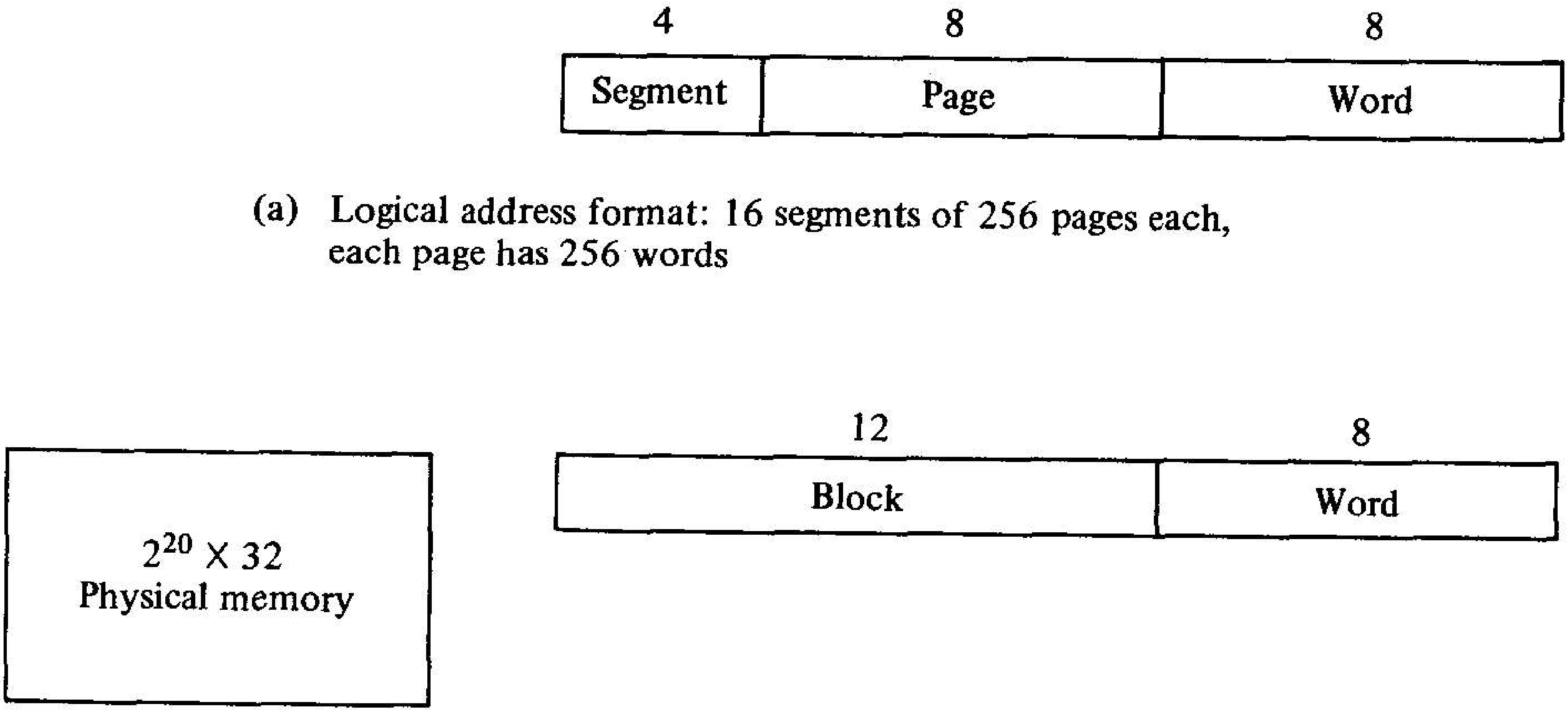
(b) Associative memory translation look-aside buffer (TLB)

Figure 12-21 Mapping in segmented-page memory management unit.

Taken from: **M. Mano/Computer Design and Architecture 3rd Ed.**

Numerical Example

- 20-bit logical address
  - **4-bit segment number** - one of **16** possible **segments**.
  - **8-bit page number** – segment **length**: **1-256 pages** (256-64K words)
  - **8-bit word field – page** size of **256 words**
- **Physical** memory consists of $\boldsymbol{2^{20}}$ words of 32 bits each.
- The 20-bit address is divided into two fields:
  - **12-bit block number – 4096 blocks**
  - **8-bit word number - 256 words each**
- A **page** in a **logical** address has a **corresponding block** in **physical** memory.
- Note: both the logical and physical address have 20 bits.
- In the absence of a MMS, the 20-bit address from the CPU can be used to access physical memory directly.



(a) Logical address format: 16 segments of 256 pages each, each page has 256 words

(b) Physical address format: 4096 blocks of 256 words each, each word has 32 bits

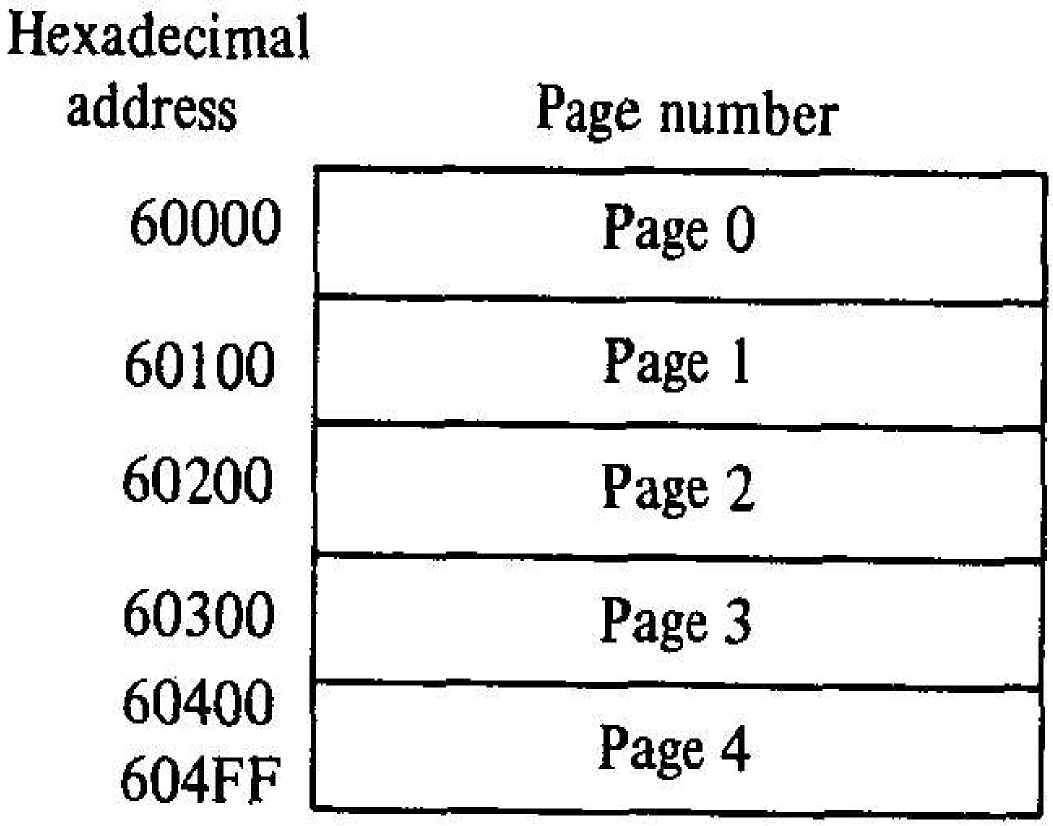
Figure 12-22 An example of logical and physical addresses.

Taken from: **M. Mano/Computer Design and Architecture 3rd Ed.**

Example – cont.

- Consider a program that requires **5 memory pages**.
- **OS assigns** to this program **segment 6** and **pages 0 – 4**
- **Logical address range** hexadecimal **60000** to **604FF**.
- When **program** is **loaded** into **physical** memory, it is **distributed among 5 blocks** in **physical** memory
  - where the operating system finds empty spaces
- What is the physical of logical address **6 02 7E** ?
  - *word **number 7E** of **page 2** in **segment 6***
- **Base** of **segment 6** in the page table is at **address 35**
- Segment 6 **contains 5 pages** at addresses **35 - 39**
- **Page 2** of segment 6 is **at address** 35 + 2 = **37**
- **Physical memory block = 019** is found in the **page table**
- Word **7E in block** 19 gives the 20-bit physical address **0197E**

Figure 12-23 Example of logical and physical memory address assignment.



(a) Logical address assignment

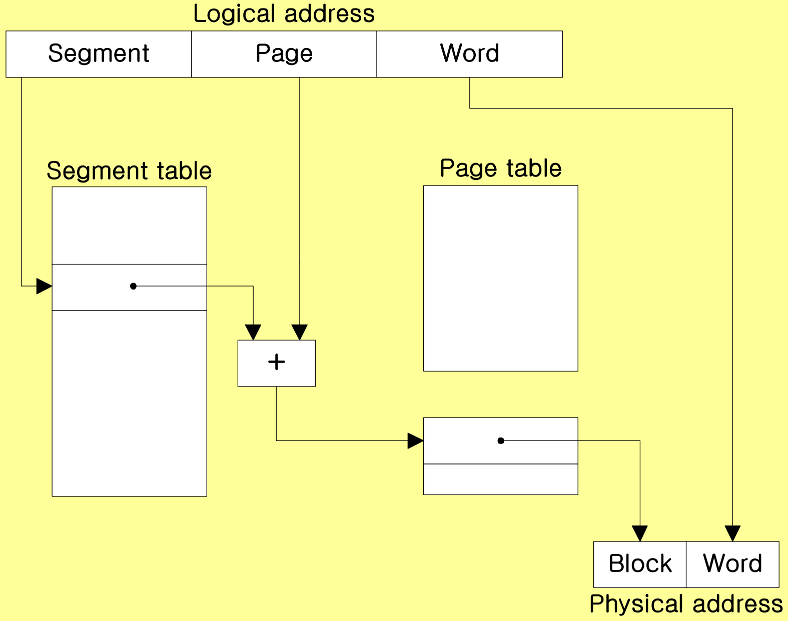
Correspondence between memory block and logical page number

Segment	Page	Block
6	00	012
6	01	000
6	02	019
6	03	053
6	04	A61

In segment 6:
page 0 maps into block 12 and
page 1 maps into block 0.

(b) Segment-page versus memory block assignment

Taken from: **M. Mano/Computer Design and Architecture 3rd Ed.**

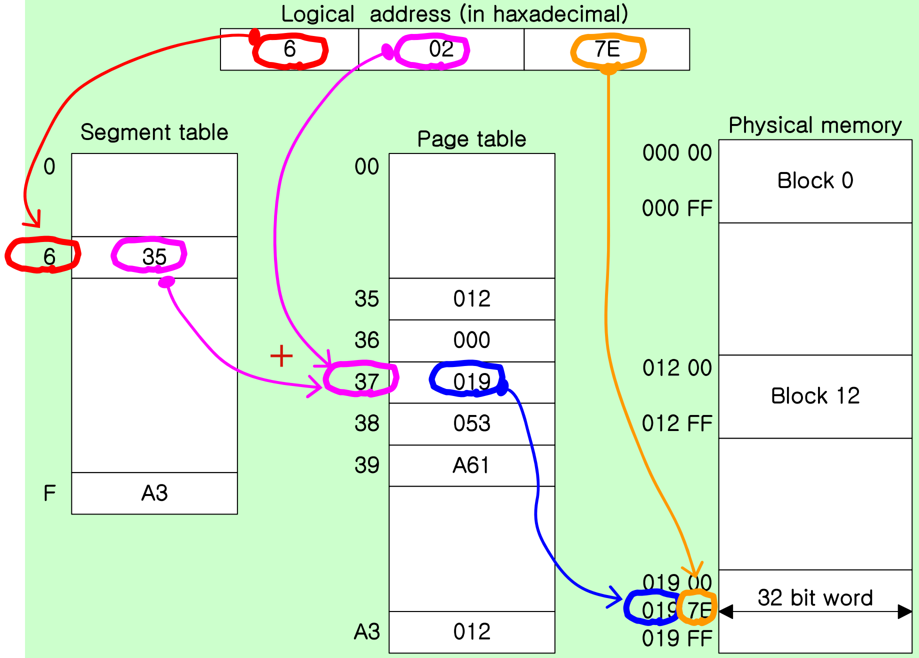


(a) Logical to physical address mapping

Argument register

Segment	Page	Block

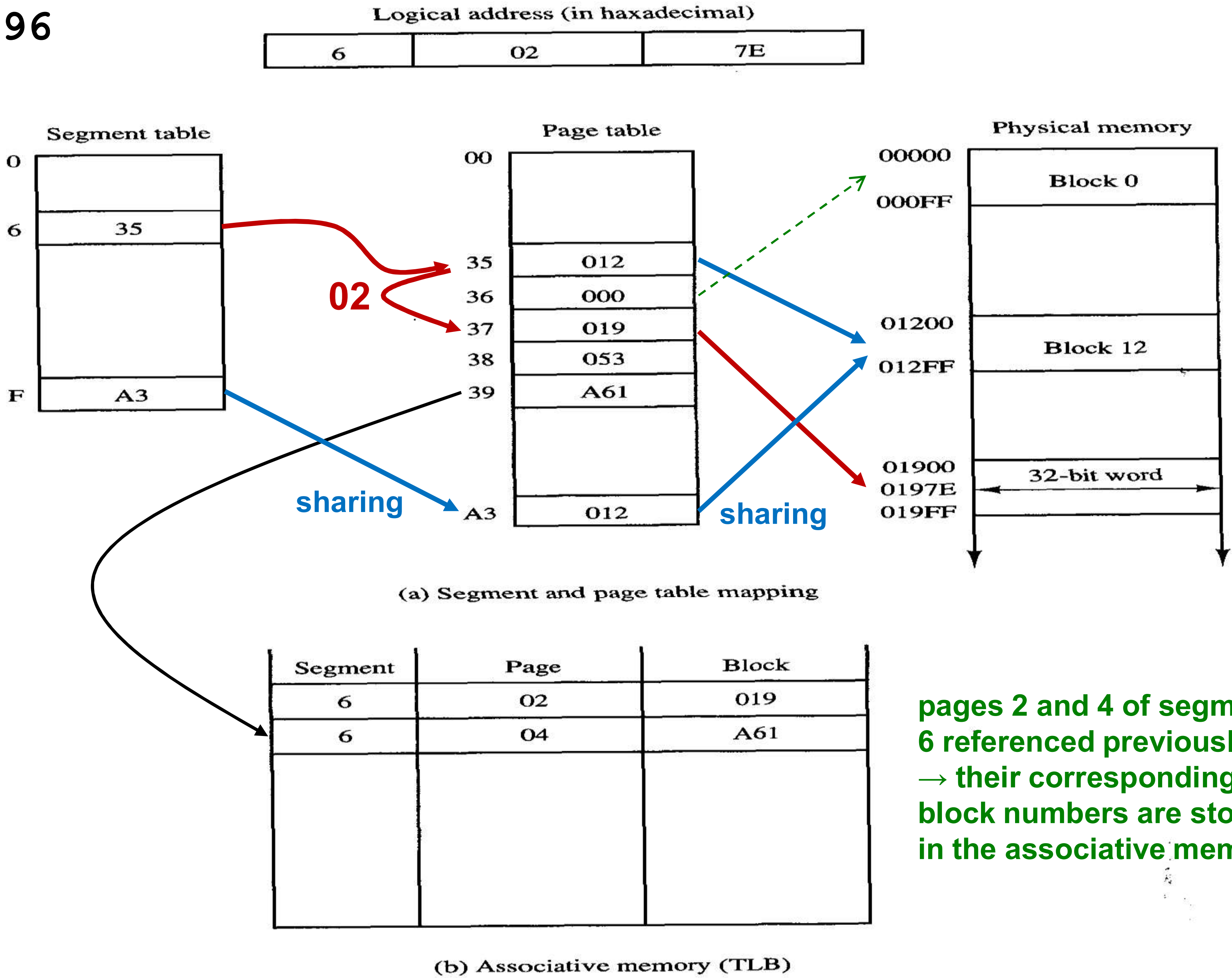
(b) Associative memory translation look-aside buffer (TLB)



(a) Segment and page table mapping

Segment	Page	Block
6	02	019
6	04	A61

(b) Associative memory (TLB)



(a) Segment and page table mapping

Segment	Page	Block
6	02	019
6	04	A61

(b) Associative memory (TLB)

pages 2 and 4 of segment 6 referenced previously → their corresponding block numbers are stored in the associative memory

Figure 12-24 Logical to physical memory mapping example (all numbers are in hexadecimal).

Taken from: **M. Mano/Computer Design and Architecture 3rd Ed.**

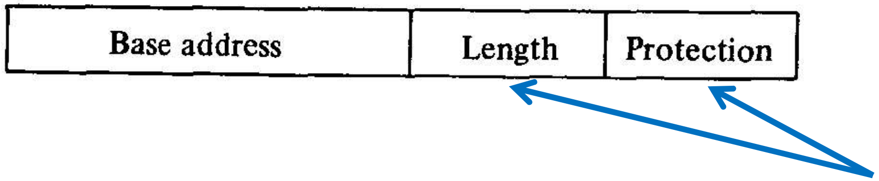
Example – cont.

- MMS can **assign any number** of **pages** to each **segment**
  - According to segment length
- Each **logical page** can be **mapped** into **any block** in **physical** memory
- **Pages can move** to **different blocks** in memory **depending** on memory **space requirements**
  - **Only update required** – **change block number** in the **page table**
- Segments can **grow or shrink w/o affecting** each **other**
- **Different segments** can use the **same block** of memory if it is required to **share** a **program** by **many users**
  - For example, **block#12** in **physical** memory can be **assigned** a **second logical address F0000 - F00FF**.
    - **Segment 15 and page 0 maps to block 12**

Memory Protection

- Can be assigned to **physical** or **logical address**.
- **Physical** address protection:
  - assigning a number of **protection bits** to each block
  - **indicate allowed access** types to the block
  - Protection bits are **updated** when block **moves**
- **Logical** address protection (**Better**)
  - include protection information within the **segment table**
- ***Descriptor*** - content of entry in segment table

Figure 12-25 Format of a typical segment descriptor.



Base address	Length	Protection

for protection purposes

Taken from: **M. Mano/Computer Design and Architecture 3rd Ed.**

Memory Protection – cont.

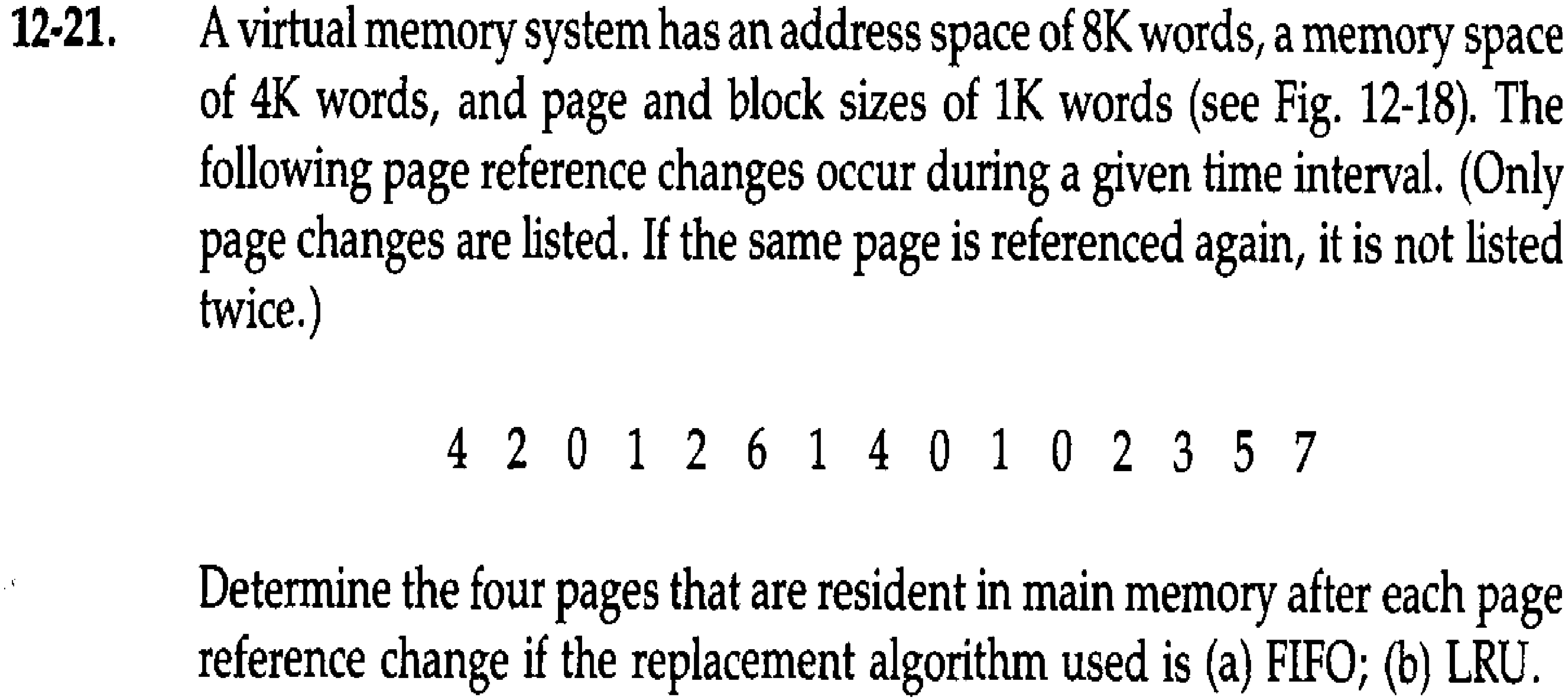
- Base address field
  - the page table base address in a segmented-page organization
  - used in mapping from a logical to the physical address
- Length field - the segment size - maximum number of pages assigned to segment.
  - The length field is compared against the page number in the logical address
  - A **size violation** occurs if page number falls outside the segment length boundary

    → program and its data **cannot access** memory **not assigned** to it by OS
- Protection field
  - **set** into the **descriptor** by the **master control** program of the **OS**
  - specifies the **access rights** available to the particular **segment**
  - In segmented-page organization –
    each page table **entry** may have **own protection**

Access rights

1. Full read and write privileges
   - given to a **program** when it is executing its **own instructions**
2. Read only (write protection)
   - useful for **sharing** system programs such as utility programs
   - Stored in memory area **accessible to all** users
   - **Program cannot** be changed
3. Execute only (program protection)
   - **protects** programs from being **copied**
   - referenced only during instruction **fetch phase not execute phase**
   - **Allows** users to **execute** the segment program instructions but
   - **prevents** them from **reading** the instructions **as data**
4. System only (operating system protection)
   - making **all** system programs **inaccessible** to **unauthorized** users

Exercise 3

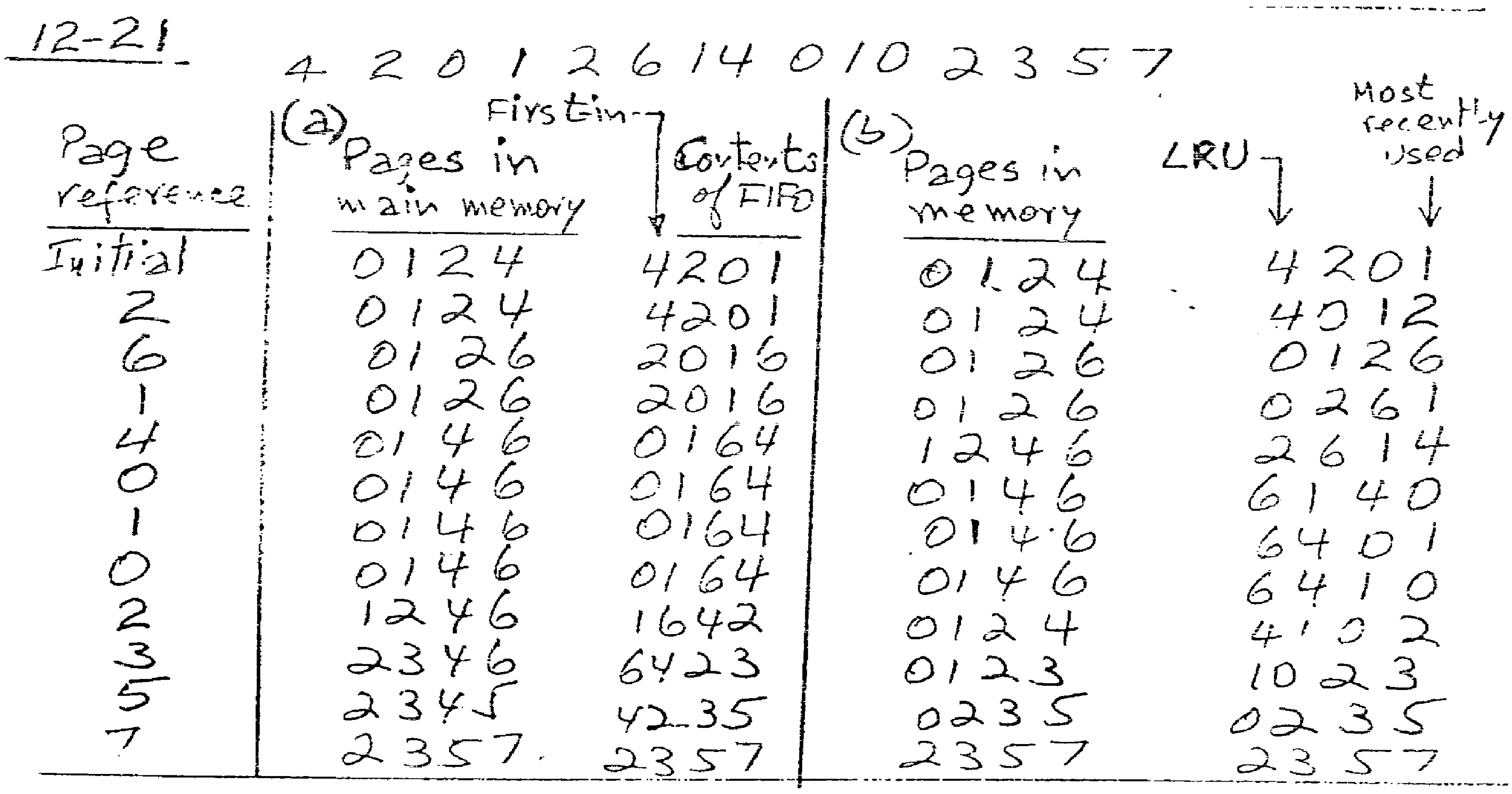


12-21. A virtual memory system has an address space of 8K words, a memory space of 4K words, and page and block sizes of 1K words (see Fig. 12-18). The following page reference changes occur during a given time interval. (Only page changes are listed. If the same page is referenced again, it is not listed twice.)

4 2 0 1 2 6 1 4 0 1 0 2 3 5 7

Determine the four pages that are resident in main memory after each page reference change if the replacement algorithm used is (a) FIFO; (b) LRU.

Solution 3



12-21

4 2 0 1 2 6 1 4 0 1 0 2 3 5 7

Page reference	(a) Pages in main memory	First in Contents of FIFO	(b) Pages in memory	LRU Most recently used
Initial	0124	4201	0124	4201
2	0124	4201	0124	4012
6	0126	2016	0126	0126
1	0126	2016	0126	0261
4	0146	0164	1246	2614
0	0146	0164	0146	6140
1	0146	0164	0146	6401
0	0146	0164	0146	6410
2	1246	1642	0124	4102
3	2346	6423	0123	1023
5	2345	4235	0235	0235
7	2357	2357	2357	2357

Exercise 4

- The logical address space in a computer system consists of 128 segments.
- Each segment can have up to 32 pages of 4K words in each.
- Physical memory consists of 4K blocks of 4K words in each.
- Formulate the logical and physical address formats

Solution 4

12-23

Logical address:

7 bits	5 bits	12 bits	=24 bits
Segment	Page	Word	

Physical address:

12 bits	12 bits
Block	Word