Topic 9
Memory Management

Objectives:

- understand various memory partition schemes: fixed partitioning, dynamic partitioning, paging, and segmentation
- understand virtual memory schemes based on paging only, segmentation only, and combination of paging and segmentation
- understand different types of page tables: one level, two-level, and inverted page tables
- understand cache technologies used in memory management: TLB and cache for main memory.
- understand various page replacement algorithms

Reading

- Stalling Chapter 7 and Chapter 8.
- Emphasis: virtual memory based on paging, which is mainly covered in Ch.8.1, including:

  - one level page table

  - two level page tables
  - inverted page table

  - TLB
  - caching of main memory
  - page replacement algorithms

Memory Management

- Subdividing memory to accommodate multiple processes
- Memory needs to be allocated to ensure a reasonable supply of ready processes to consume available processor time

Memory Management Requirements

- Relocation
  - Programmer does not know where the program will be placed in memory when it is executed
  - While the program is executing, it may be swapped to disk and returned to main memory at a different location (relocated)
  - Memory references in the code must be translated to actual physical memory address

Memory Management Requirements

- Protection
  - Processes should not be able to reference memory locations in another process without permission
  - Impossible to check absolute addresses at compile time
  - Must be checked at run-time
  - Memory protection requirement must be satisfied by the processor (hardware) rather than by the operating system (software)
    - Operating system cannot anticipate all of the memory references a program will make

Memory Management Requirements

- Sharing
  - Allow several processes to access the same portion of memory
  - E.g., better to allow each process access to the same copy of the program rather than have their own separate copy

Memory Management Requirements

- Logical Organization
  - Programs are written in modules
  - Modules can be written and compiled independently
  - Different degrees of protection given to modules (read-only, execute-only)
  - Share modules among processes

Memory Management Requirements

- Physical Organization
  - Memory available for a program plus its data may be insufficient
    - Overlaying allows various modules to be assigned the same region of memory
  - Programmer does not know how much space will be available

Fixed Partitioning

- Equal-size partitions
- Unequal-size partitions
- Any process whose size is less than or equal to the partition size can be loaded into an available partition
- If all partitions are full, the operating system can swap a process out of a partition
- A program may not fit in a partition. The programmer must design the program with overlays

Fixed Partitioning

- Main memory use is inefficient. Any program, no matter how small, occupies an entire partition. This is called internal fragmentation.

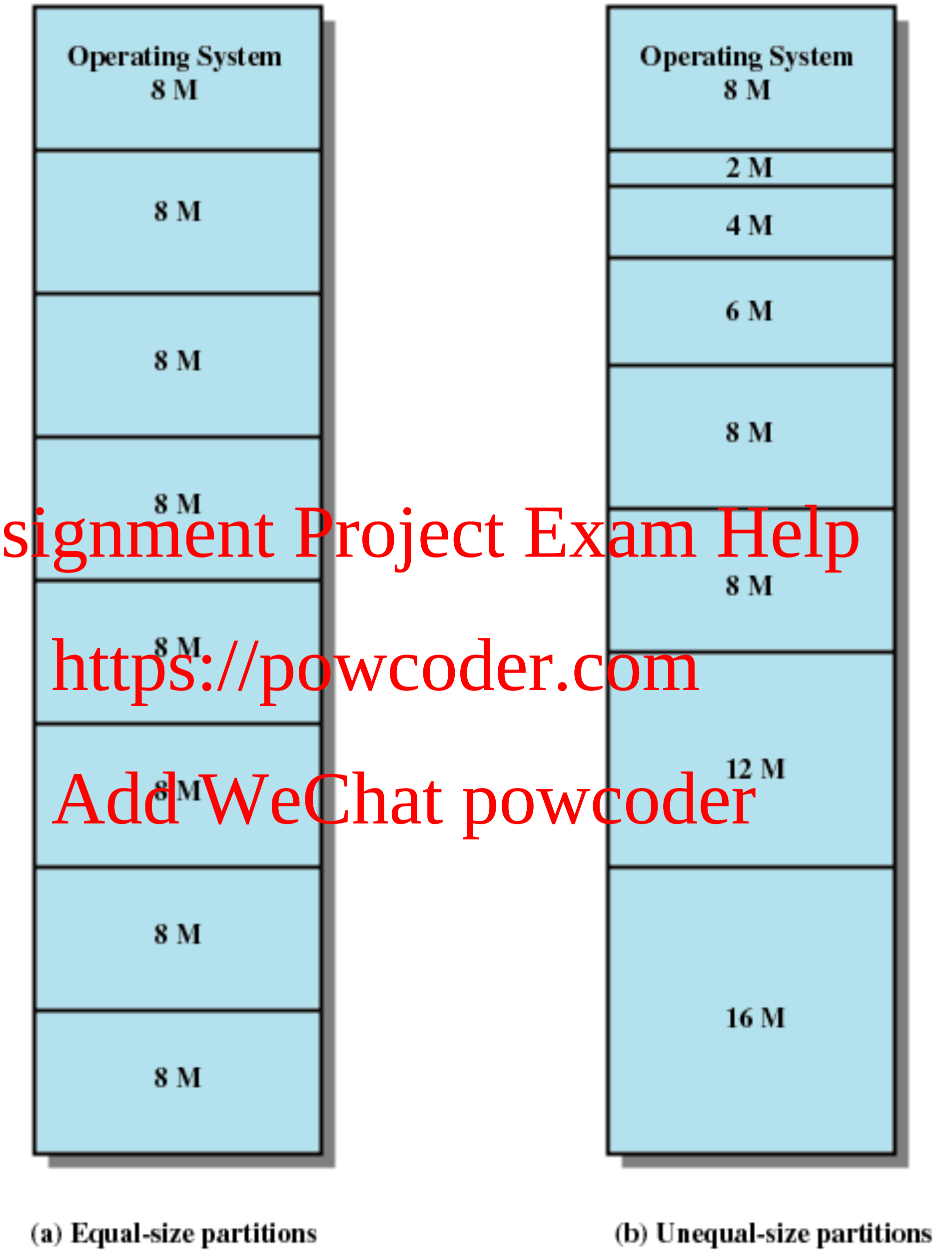


Figure 7.2 Example of Fixed Partitioning of a 64-Mbyte Memory

Dynamic Partitioning

- Partitions are of variable length and number
- Process is allocated exactly as much memory as required
- Eventually get holes in the memory. This is called external fragmentation
- Must use compaction to shift processes so they are contiguous and all free memory is in one block

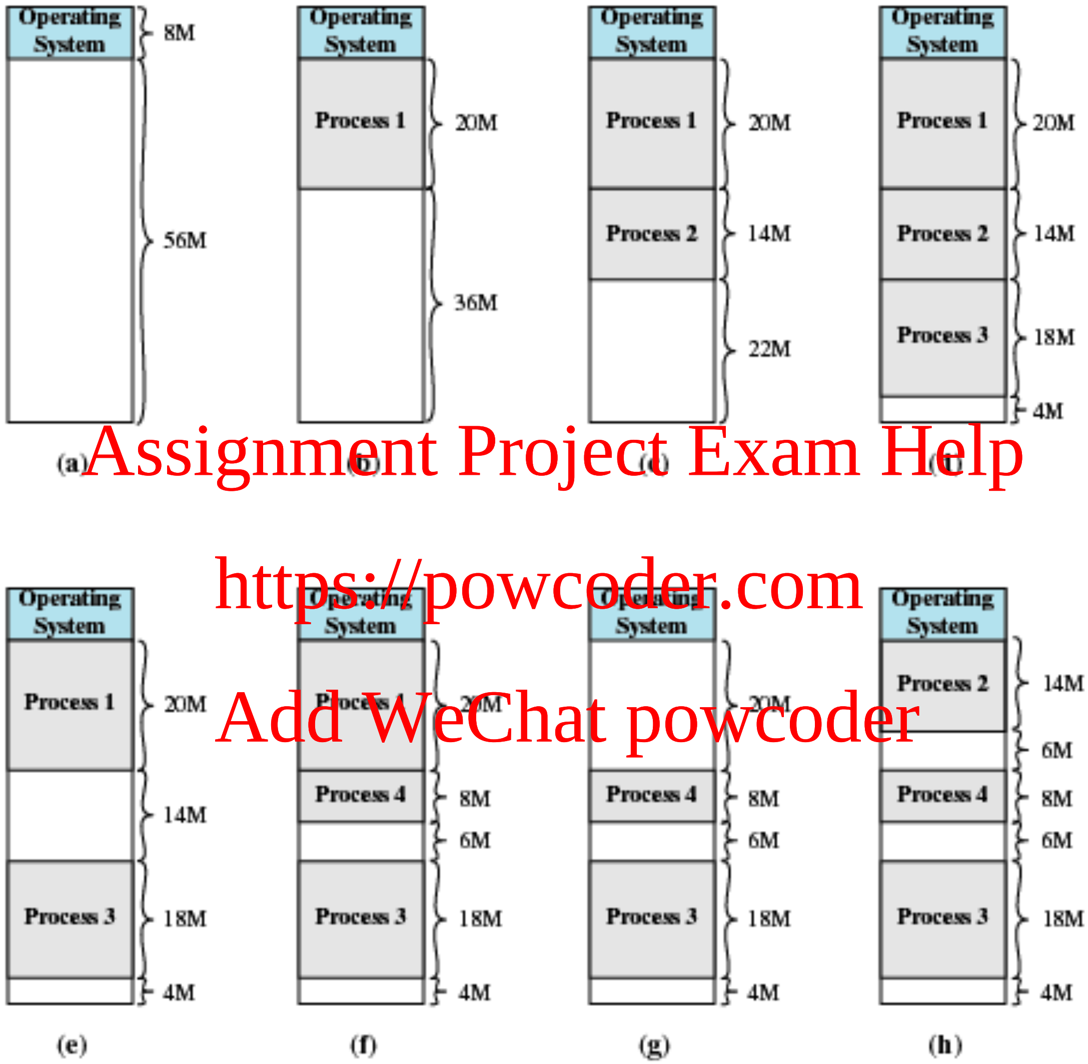


Figure 7.4 The Effect of Dynamic Partitioning

Relocation

- When program loaded into memory the actual (absolute) memory locations are determined
- A process may occupy different partitions at different times, which means different absolute memory locations during execution (from swapping)
- Compaction will also cause a program to occupy a different partition which means different absolute memory locations

Addresses

- Logical
  - Reference to a memory location independent of the current assignment of data to memory
  - Translation must be made to the physical address
- Relative
  - Address expressed as a location relative to some known point
- Physical
  - The absolute address or actual location in main memory

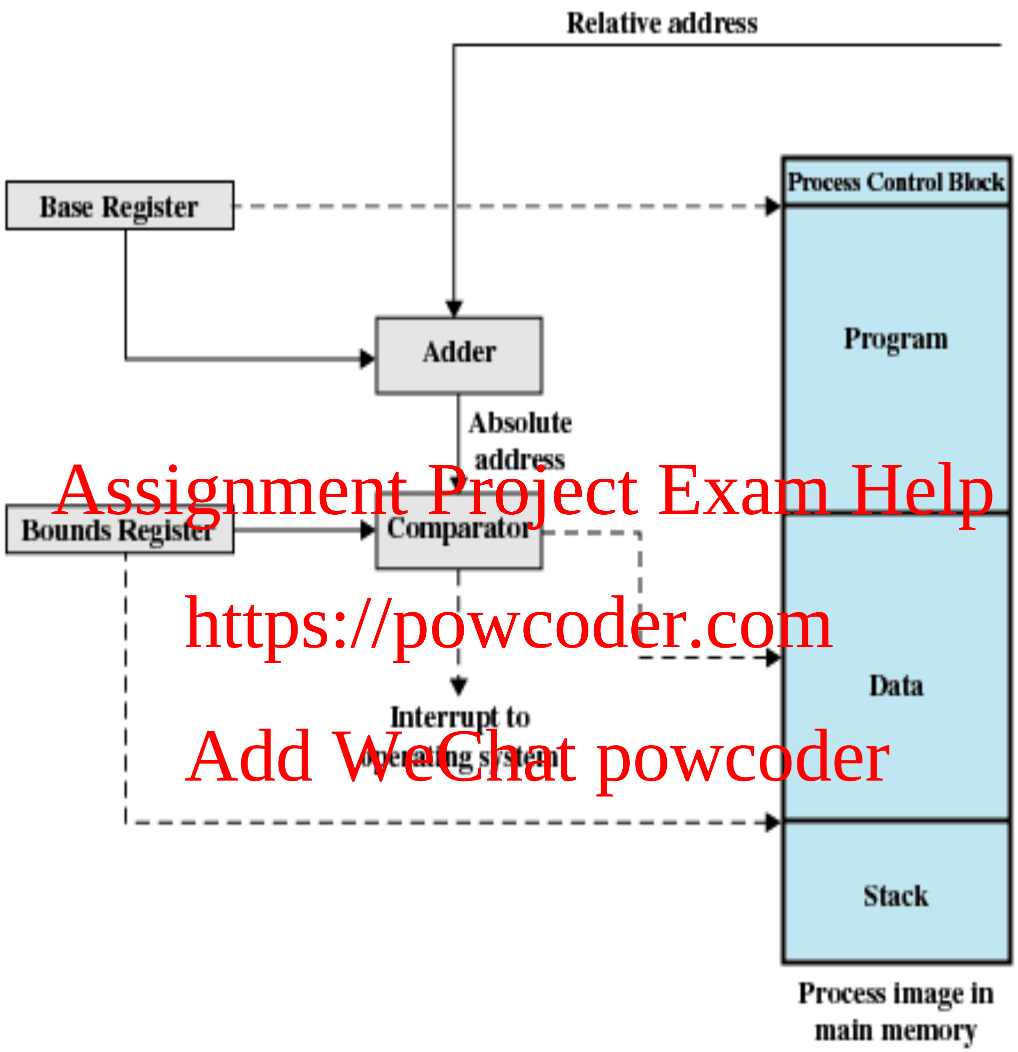


Figure 7.8 Hardware Support for Relocation

Registers Used during Execution

- Base register
  - Starting address for the process
- Bounds register
  - Ending location of the process
- These values are set when the process is loaded or when the process is swapped in

Registers Used during Execution

- The value of the base register is added to a relative address to produce an absolute address
- The resulting address is compared with the value in the bounds register
- If the address is not within bounds, an interrupt is generated to the operating system

Paging

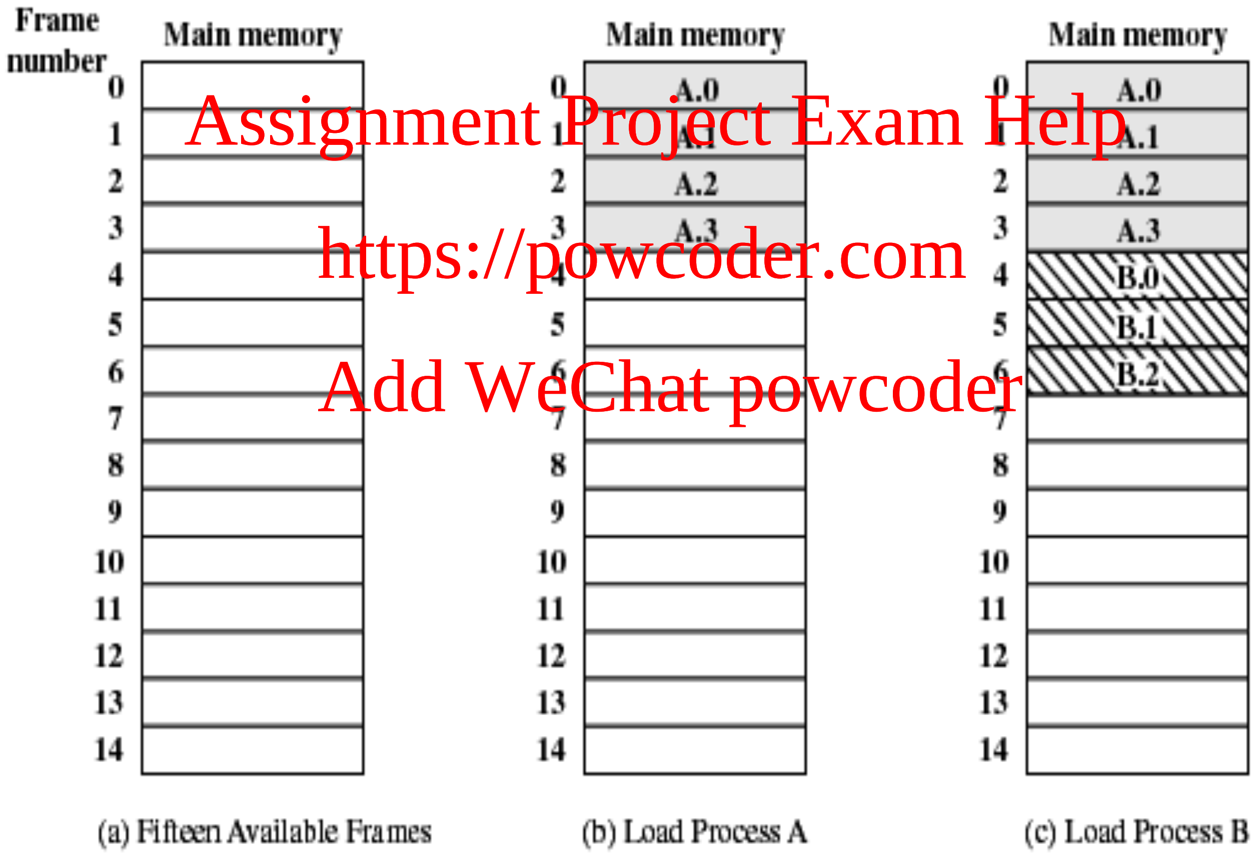
- Partition memory into small equal fixed-size chunks and divide each process into the same size chunks

- The chunks of a process are called <u>pages</u> and chunks of memory are called <u>frames</u>

- Operating system maintains a <u>page table</u> for each process

  - Contains the frame location for each page in the process
  - Memory address consist of a page number and offset within the page

Assignment of Process Pages to Free Frames



Assignment of Process Pages to Free Frames

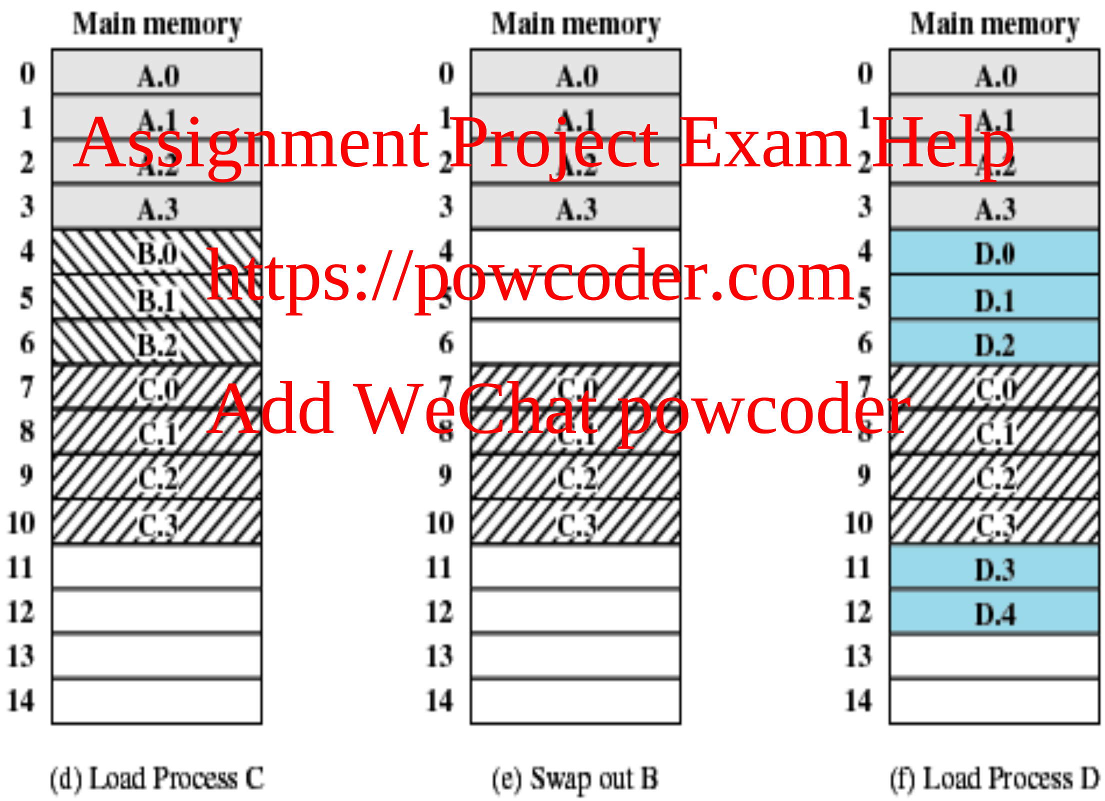


Figure 7.9 Assignment of Process Pages to Free Frames

Page Tables for Example

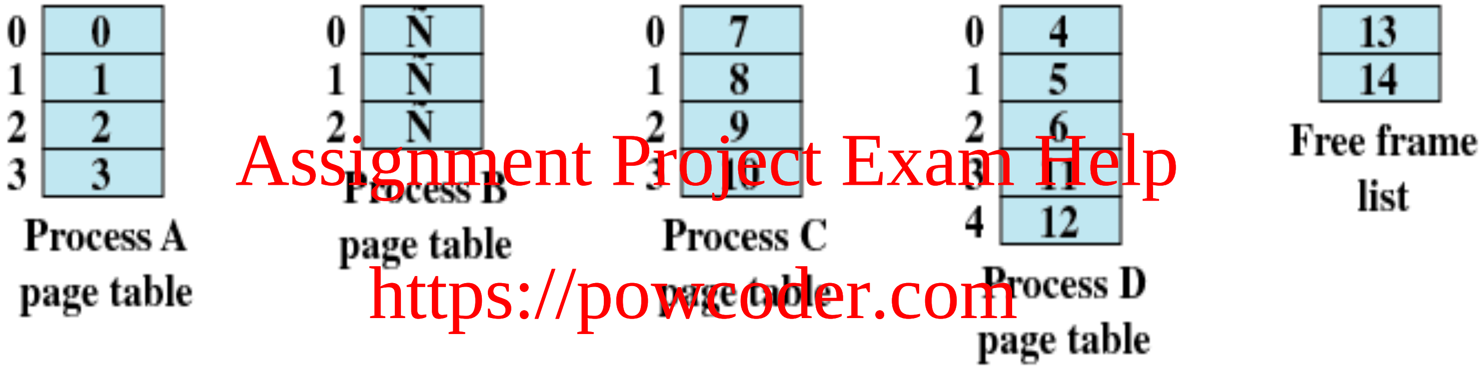


Figure 7.10 Data Structures for the Example of Figure 7.9 at Time Epoch (f)

Segmentation

- All segments of all programs do not have to be of the same length

- There is a maximum segment length

- Addressing consist of two parts - a segment number and an offset

- Since segments are not equal, segmentation is similar to dynamic partitioning

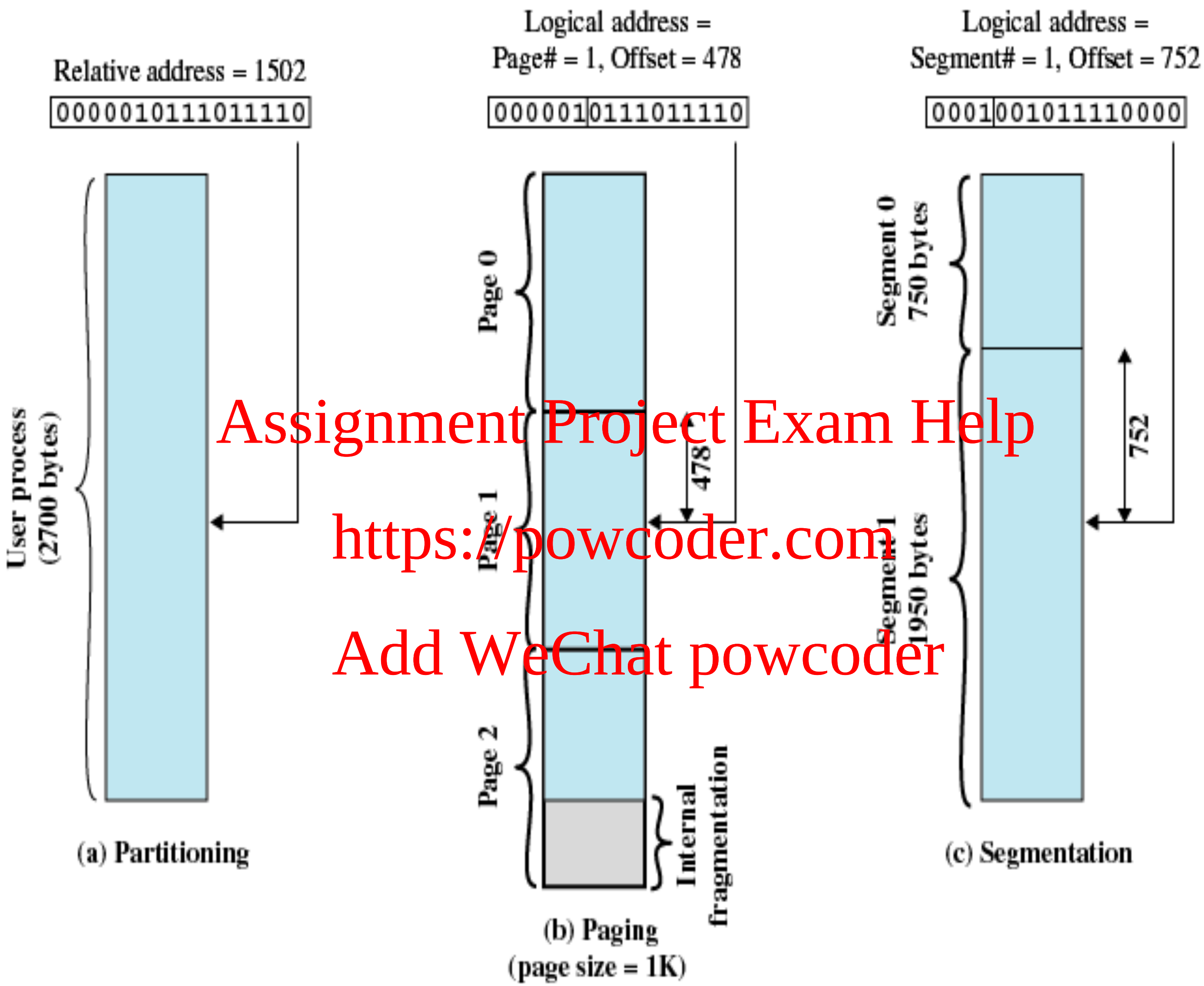
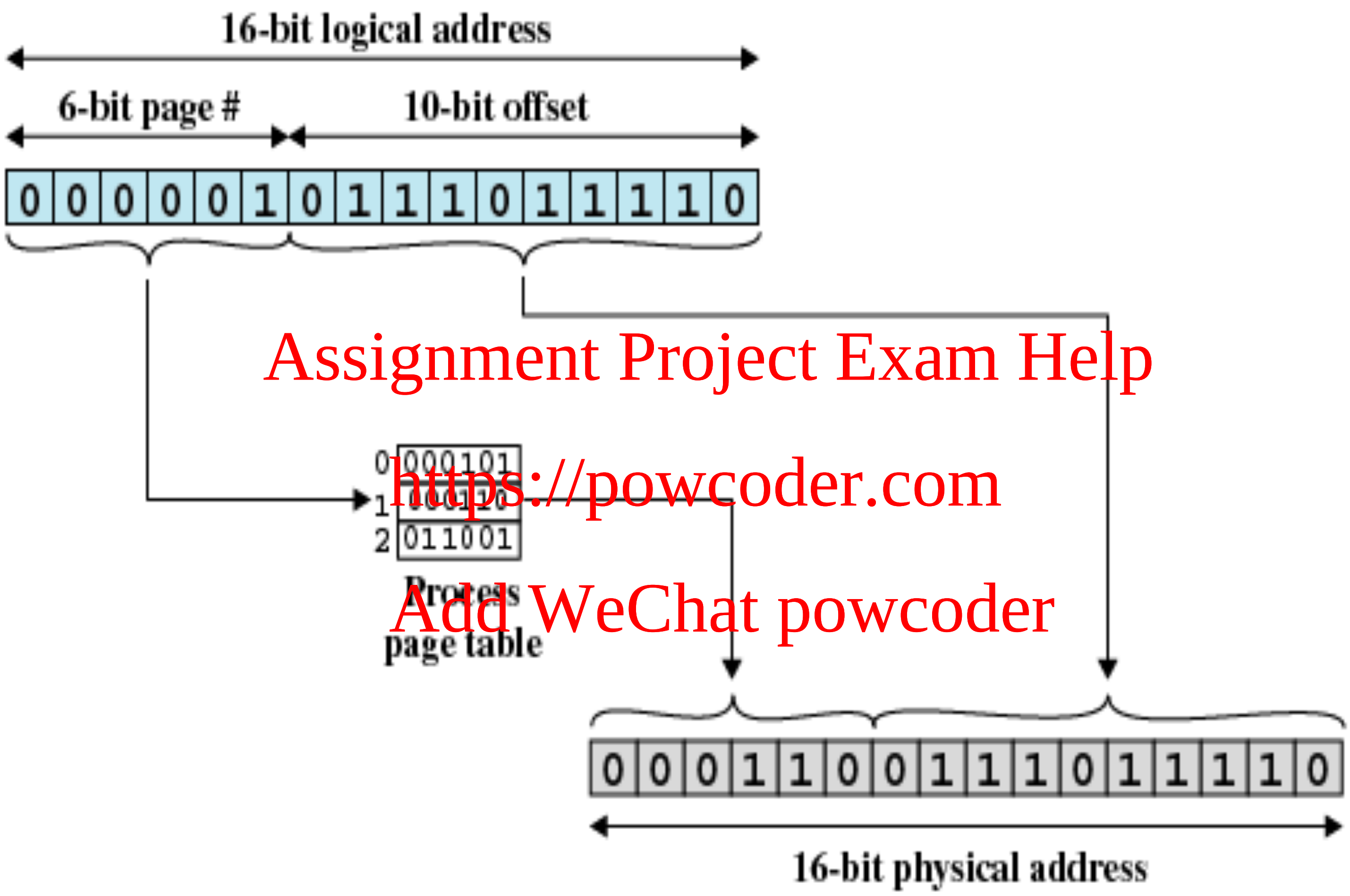
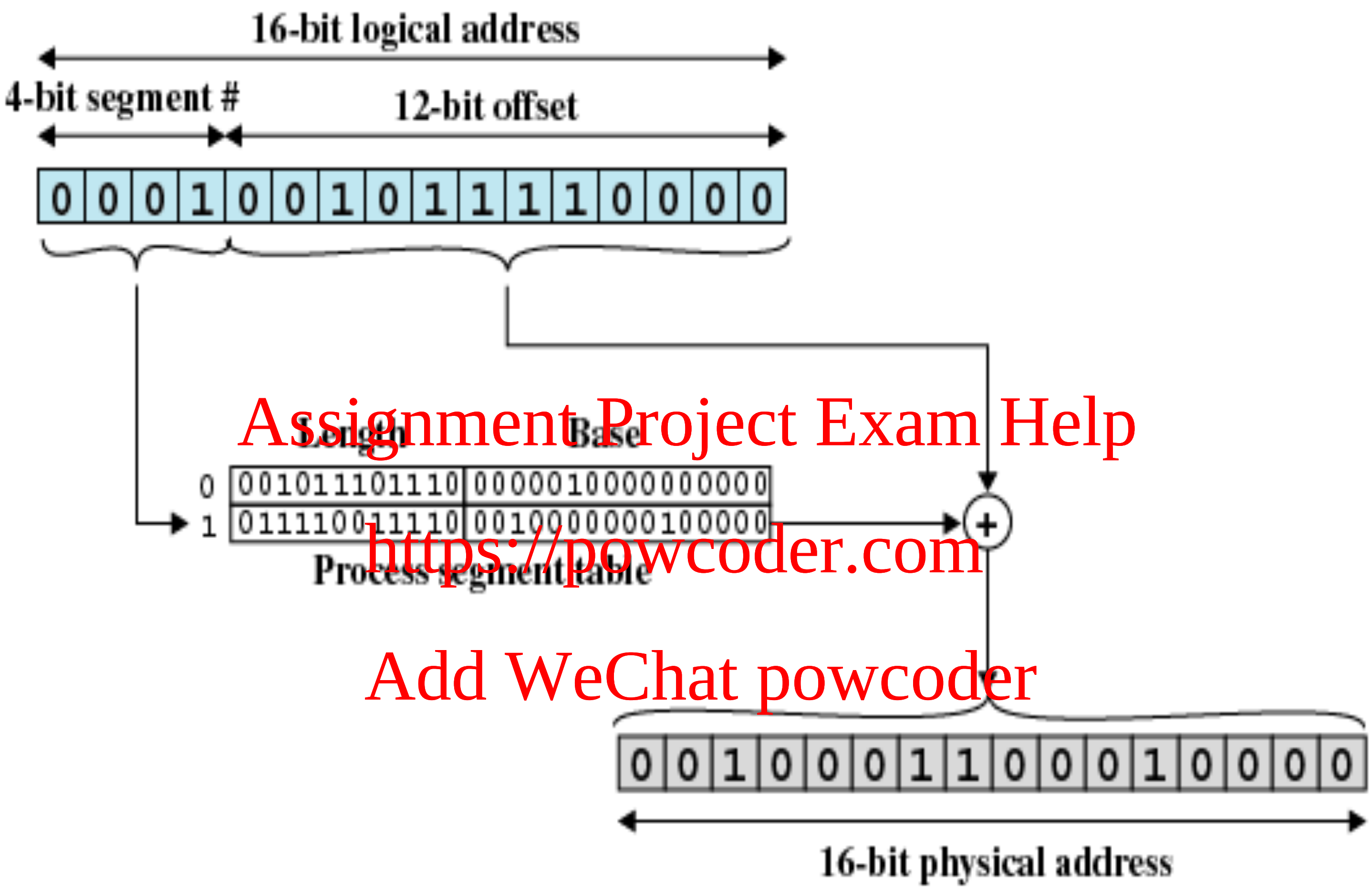


Figure 7.11 Logical Addresses


16-bit logical address
6-bit page #
10-bit offset
0 0 0 0 0 1 0 1 1 1 0 1 1 1 1 0
0 000101
1 000110
2 011001
Process page table
0 0 0 1 1 0 0 1 1 1 0 1 1 1 1 0
16-bit physical address

(a) Paging



(b) Segmentation

Figure 7.12 Examples of Logical-to-Physical Address Translation

Virtual Memory Scheme

- A process may be broken up into pieces that do not need to be located contiguously in main memory
- <u>All pieces of a process do not need to be loaded in main memory during execution</u>
- Memory references are dynamically translated into physical addresses at run time
  - A process may be swapped in and out of main memory such that it occupies different regions

Virtual Memory Scheme

- Operating system brings into main memory a few pieces of the program
- Resident set - portion of process that is in main memory
- An interrupt is generated when an address is needed that is not in main memory
- Operating system places the process in a blocking state

Virtual Memory Scheme

- Piece of process that contains the logical address is brought into main memory
  - Operating system issues a disk I/O Read request
  - Another process is dispatched to run while the disk I/O takes place
  - An interrupt is issued when disk I/O complete which causes the operating system to place the affected process in the Ready state

Advantages of Virtual Memory

- More processes may be maintained in main memory
  - Only load in some of the pieces of each process
  - With so many processes in main memory, it is very likely a process will be in the Ready state at any particular time
- A process may be larger than all of main memory

Types of Memory

- Real memory
  - Main memory
- Virtual memory
  - Memory on disk
  - Allows for effective multiprogramming and relieves the user of tight constraints of main memory

Thrashing

- Swapping out a piece of a process just before that piece is needed
- The processor spends most of its time swapping pieces rather than executing user instructions

Principle of Locality

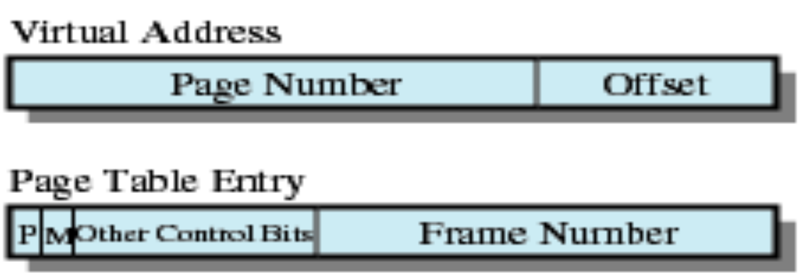
- Program and data references within a process tend to cluster
- Only a few pieces of a process will be needed over a short period of time
- Possible to make intelligent guesses about which pieces will be needed in the future
- This suggests that virtual memory may work efficiently

Support Needed for Virtual Memory

- Hardware must support paging and/or segmentation
- Operating system must be able to manage the movement of pages and/or segments between secondary memory and main memory

Virtual Memory Based on Paging Only

- Each process has its own page table
- Each page table entry contains the frame number of the corresponding page in main memory
- A bit is needed to indicate whether the page is in main memory or not



(a) Paging only

Modify Bit in Page Table

- Modify bit is needed to indicate if the page has been altered since it was last loaded into main memory
- If no change has been made, the page does not have to be written to the disk when it needs to be swapped out

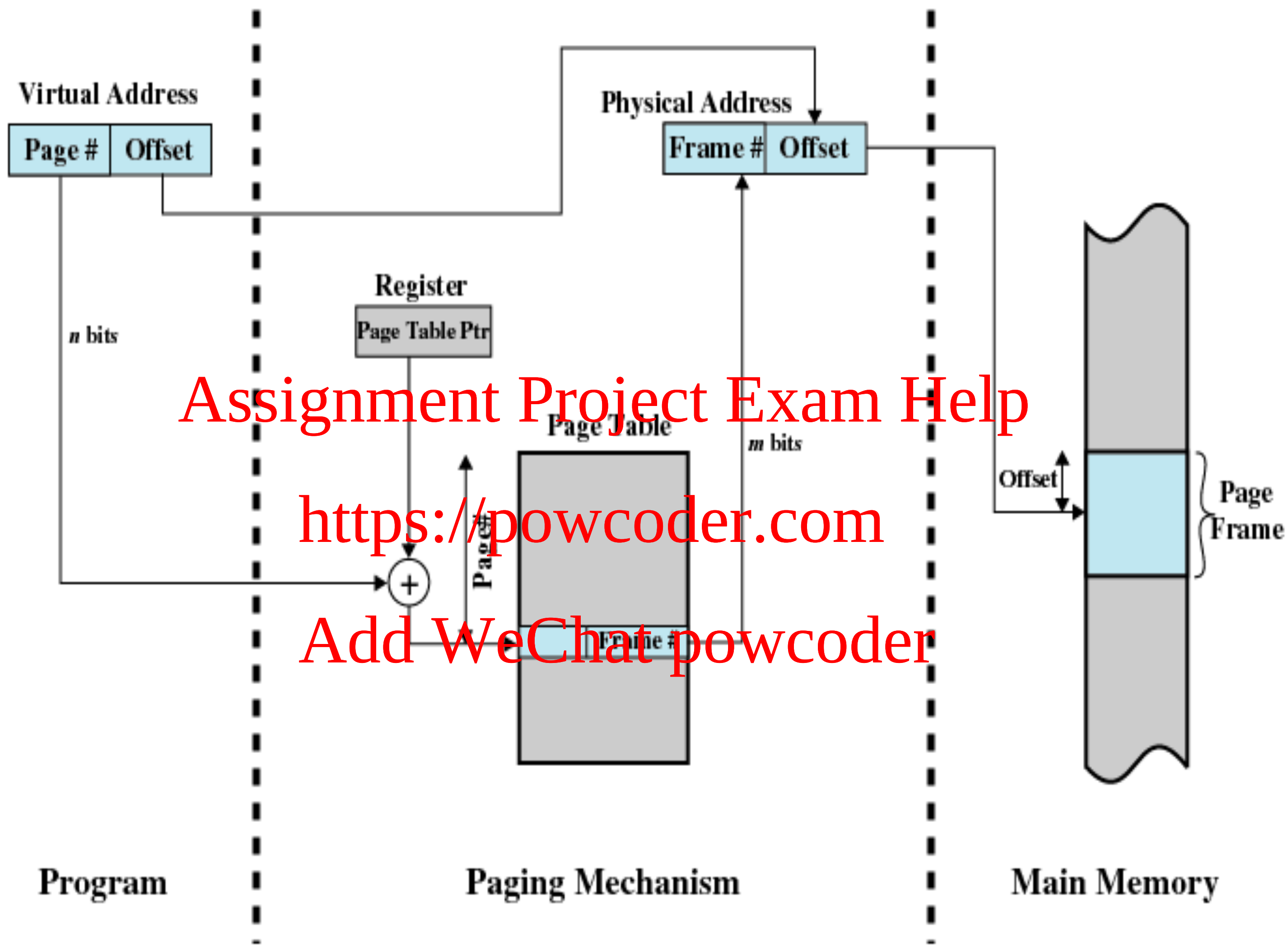


Figure 8.3 Address Translation in a Paging System

Page Tables

- The entire page table may take up too much main memory
- Page tables are also stored in virtual memory
- When a process is running, part of its page table is in main memory

Two-Level Scheme for 32-bit Address

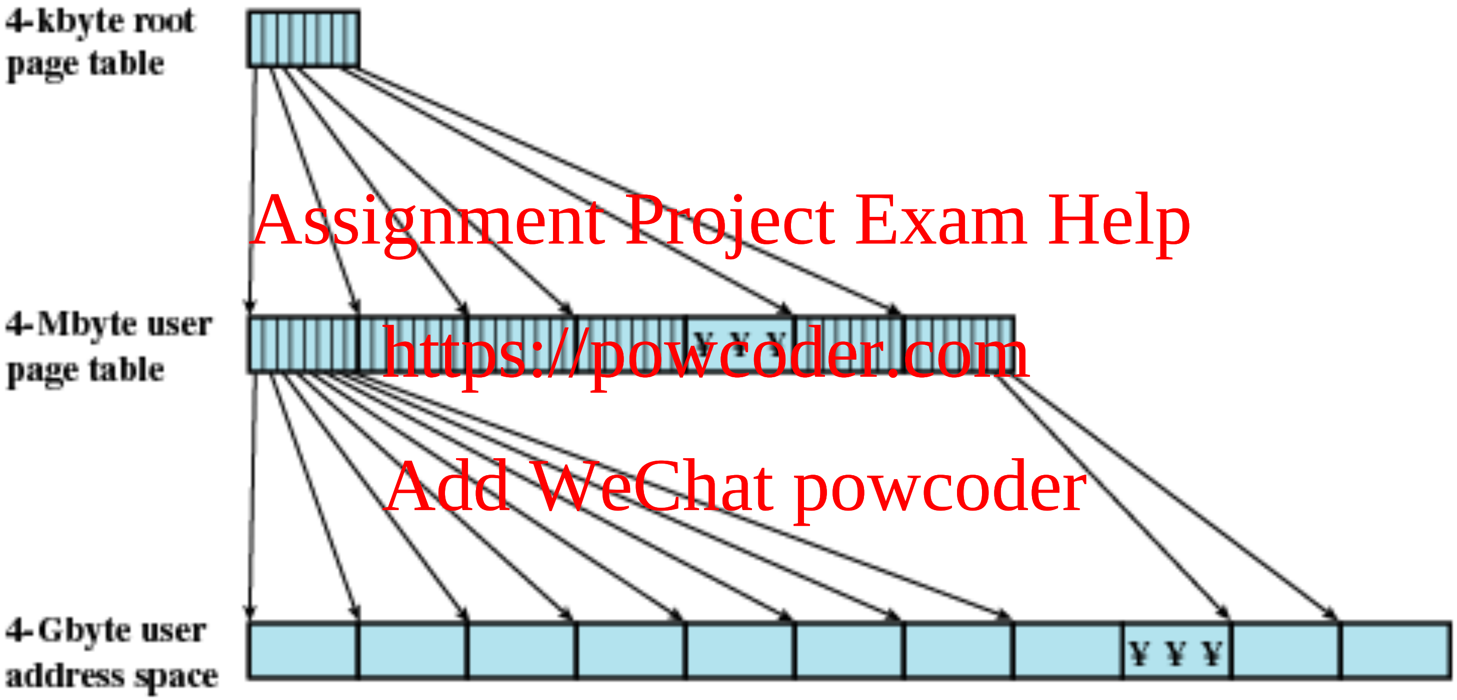


Figure 8.4 A Two-Level Hierarchical Page Table

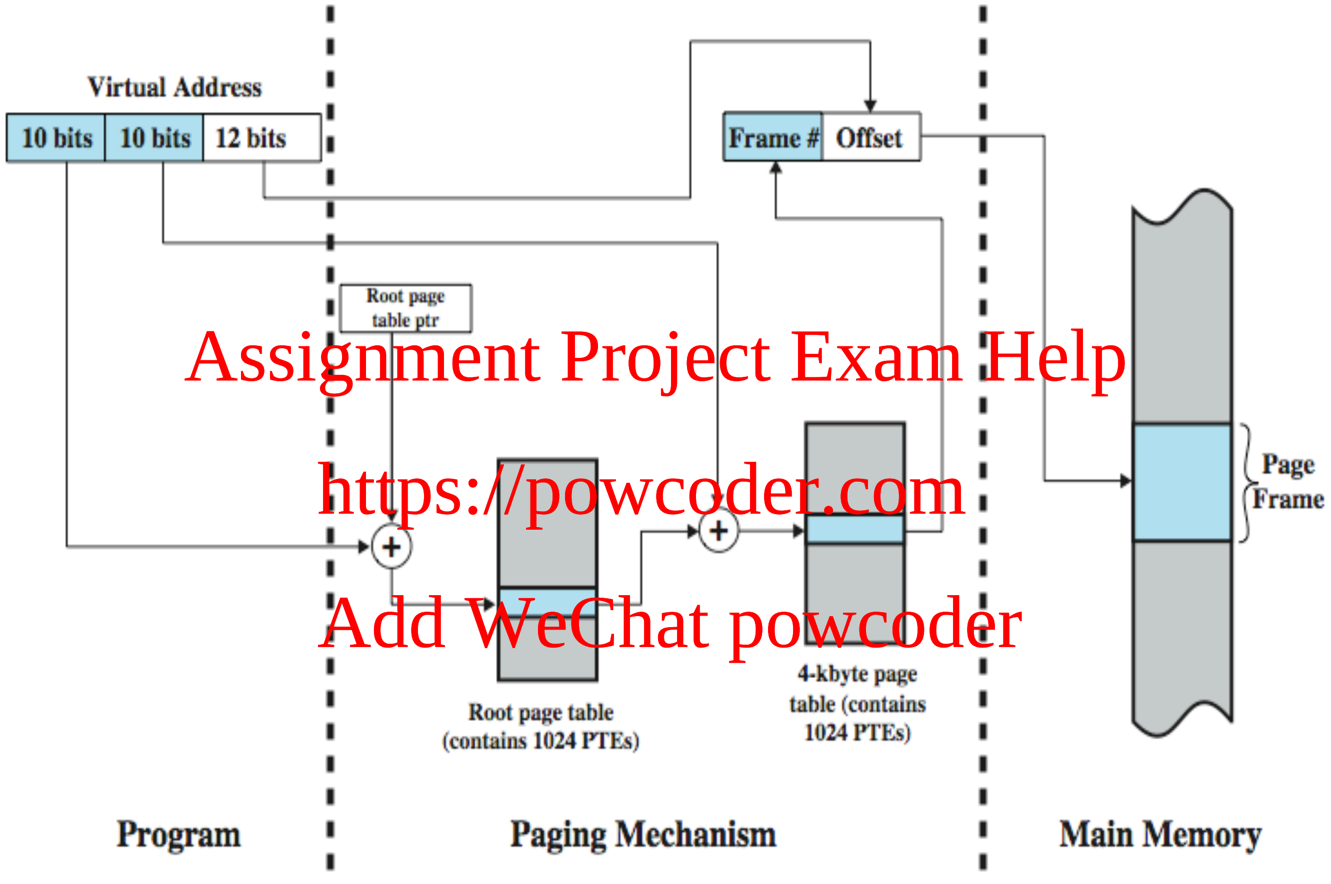


Figure 8.5 Address Translation in a Two-Level Paging System

Inverted Page Table

- Used on PowerPC, UltraSPARC, and IA-64 architecture
- Page number portion of a virtual address is mapped into a hash value
- Hash value points to inverted page table
- Fixed proportion of real memory is required for the tables regardless of the number of processes

Inverted Page Table

- Page number
- Process identifier
- Control bits
- Chain pointer

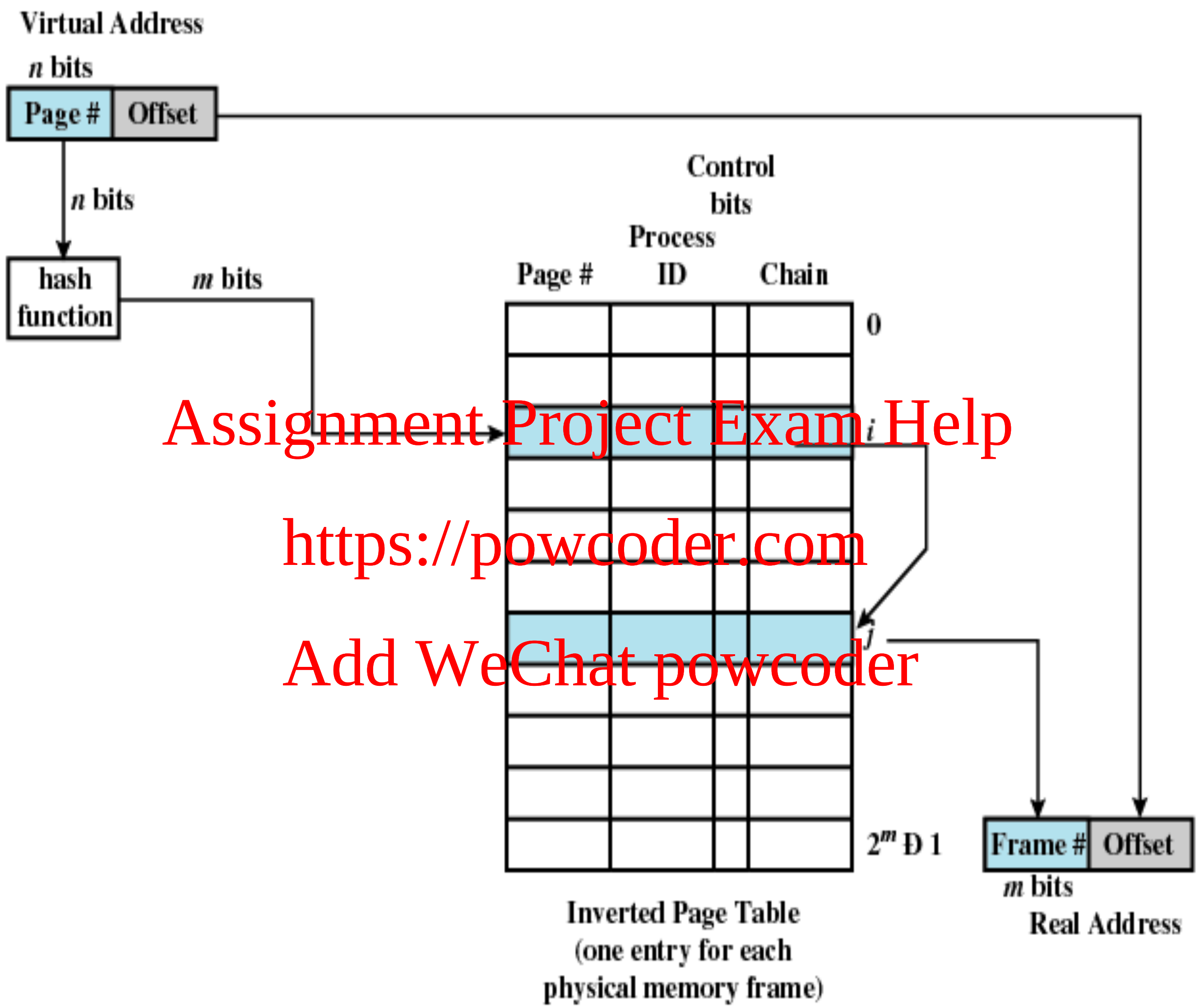


Figure 8.6 Inverted Page Table Structure

Translation Lookaside Buffer

- Each virtual memory reference can cause at least two physical memory accesses
  - One to fetch the page table entry
  - One to fetch the data
- To overcome this problem a high-speed cache is set up for page table entries
  - Called a Translation Lookaside Buffer (TLB)

Translation Lookaside Buffer

- Contains page table entries that have been most recently used
- Given a virtual address, processor examines the TLB
- If page table entry is present (TLB hit), the frame number is retrieved and the real address is formed

Translation Lookaside Buffer

- If page table entry is not found in the TLB (TLB miss), the page number is used to index the process page table

- First checks if page is already in main memory

  - If not in main memory a page fault is issued

- The TLB is updated to include the new page entry

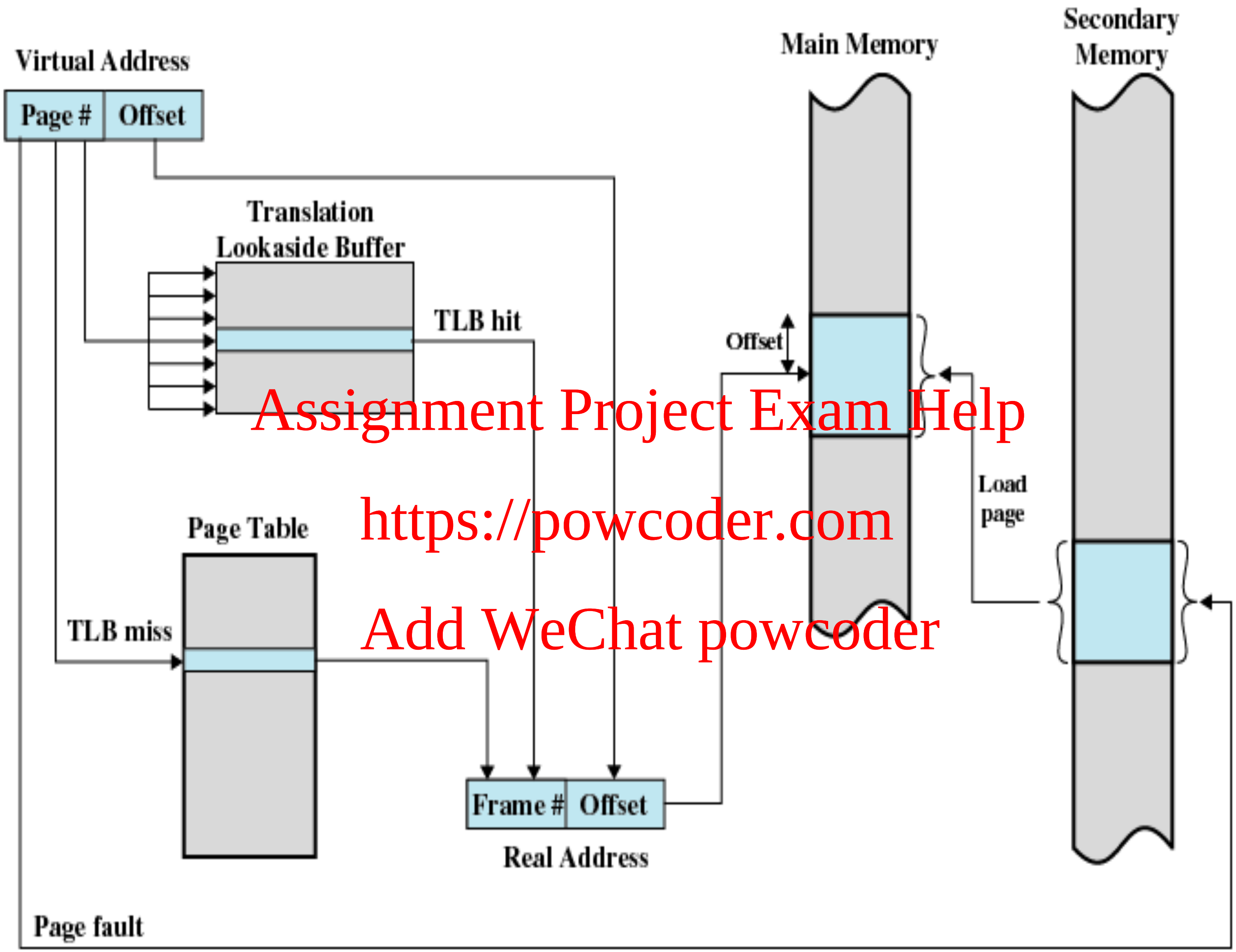


Figure 8.7 Use of a Translation Lookaside Buffer

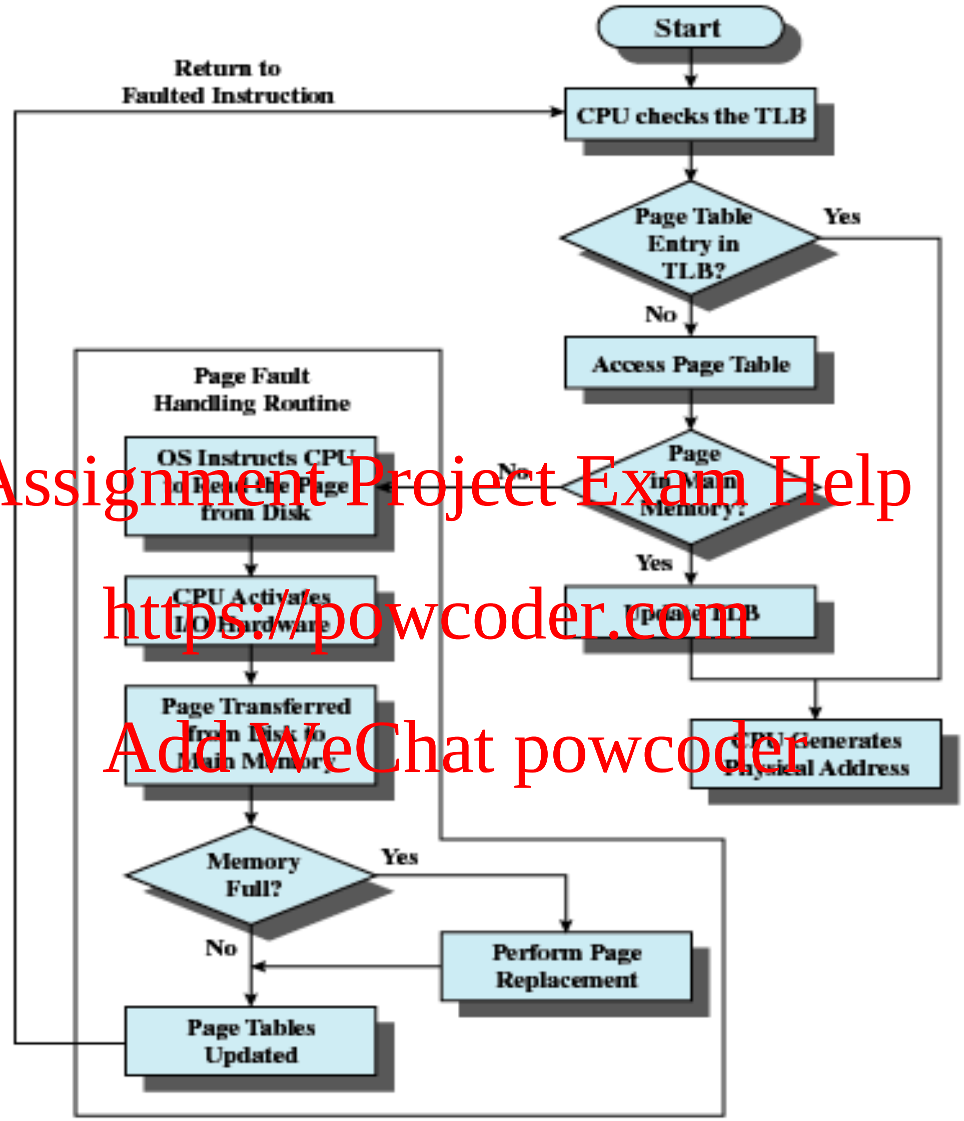


Figure 8.8 Operation of Paging and Translation Lookaside Buffer (TLB) [FURH87]

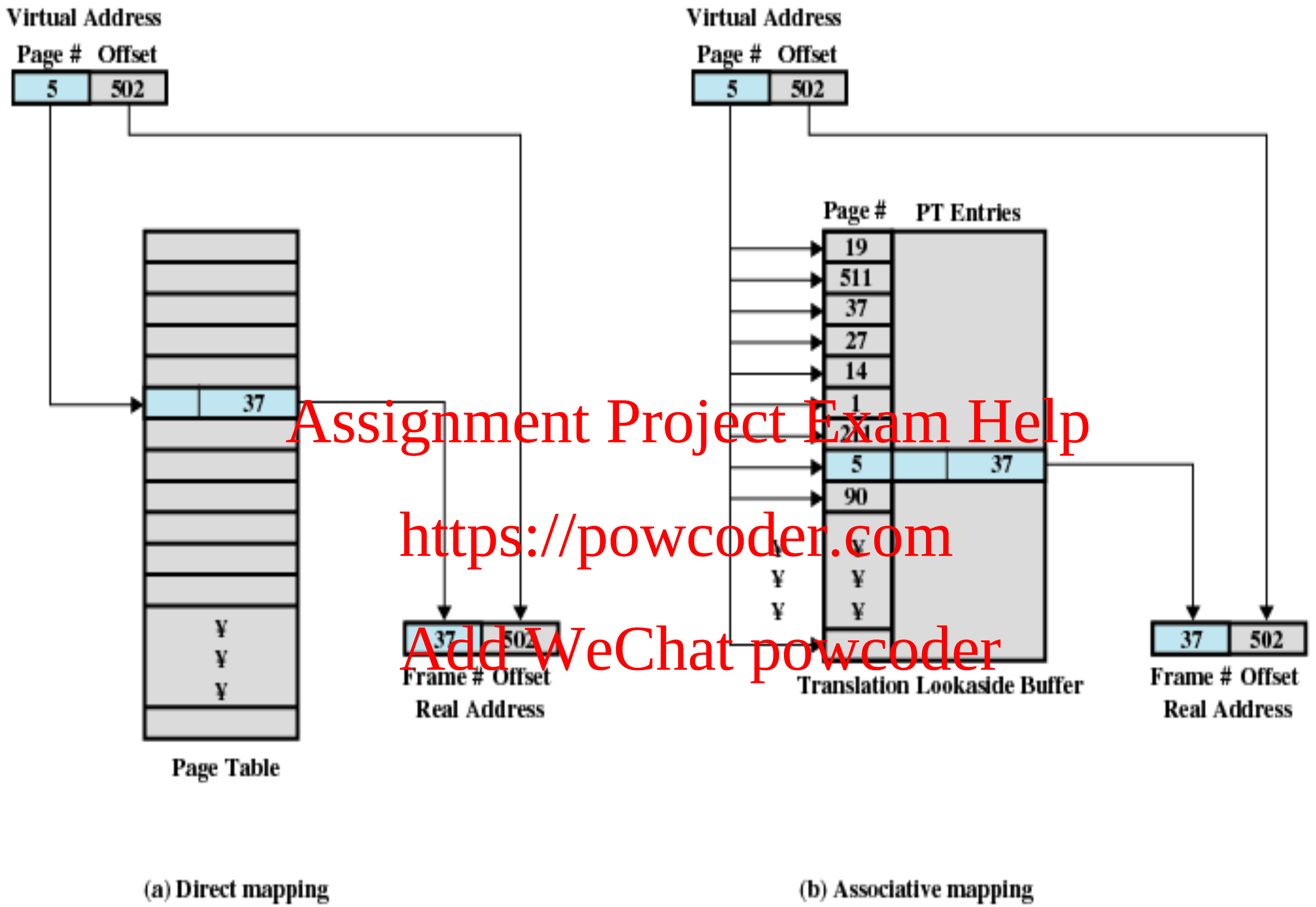


Figure 8.9 Direct Versus Associative Lookup for Page Table Entries

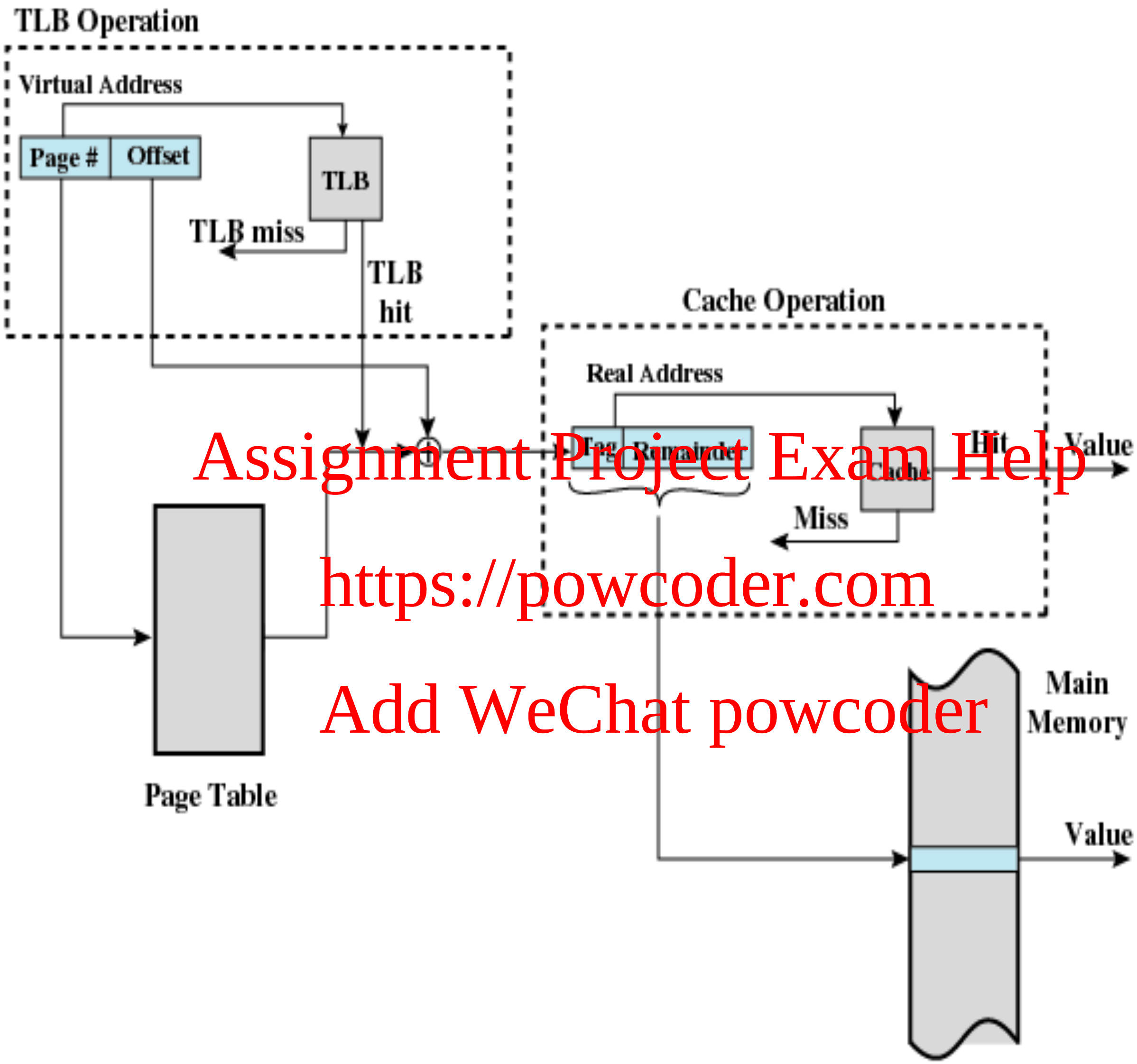


Figure 8.10 Translation Lookaside Buffer and Cache Operation

Page Size

- Small page size, large number of pages will be found in main memory
- As time goes on during execution, the pages in memory will all contain portions of the process near recent references. Page faults low.
- Increased page size causes pages to contain locations further from any recent reference. Page faults rise.

Example Page Sizes

Table 8.2 Example Page Sizes

Computer	Page Size
Atlas	512 48-bit words
Honeywell-Multics	1024 36-bit word
IBM 370/XA and 370/ESA	4 Kbytes
VAX family	512 bytes
IBM AS/400	512 bytes
DEC Alpha	8 Kbytes
MIPS	4 kbyes to 16 Mbytes
UltraSPARC	8 Kbytes to 4 Mbytes
Pentium	4 Kbytes or 4 Mbytes
PowerPc	4 Kbytes
Itanium	4 Kbytes to 256 Mbytes

Virtual Memory Based on Segmentation

- May be unequal, dynamic size
- Simplifies handling of growing data structures
- Allows programs to be altered and recompiled independently
- Lends itself to sharing data among processes
- Lends itself to protection

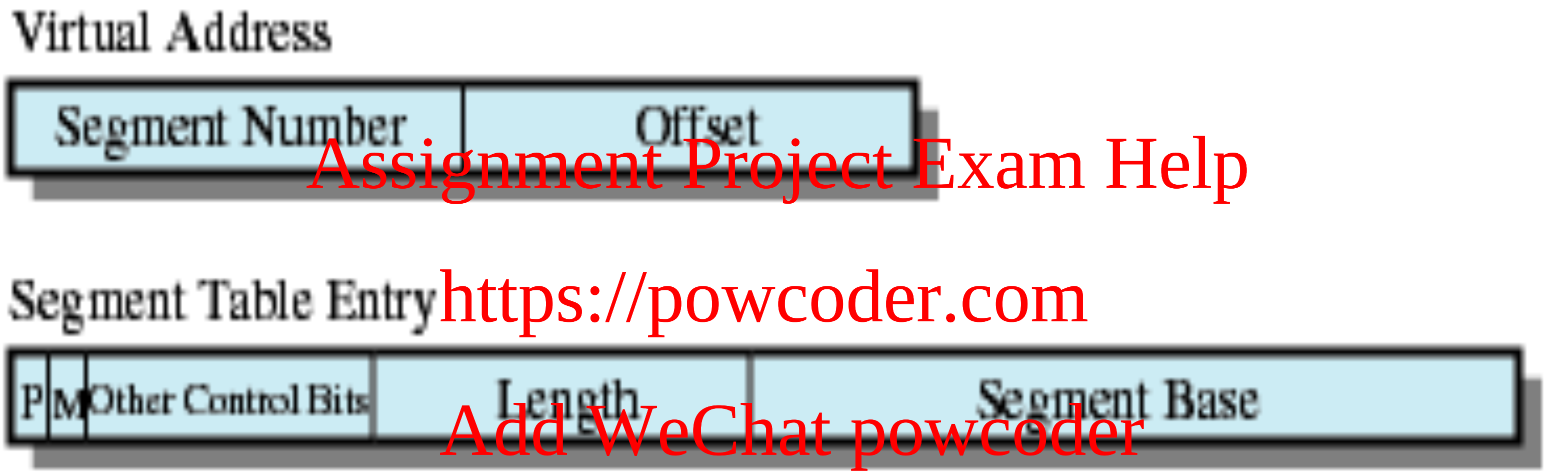
Segment Tables

- Each process has one segment table
- Each entry contains the length and the base address of the segment

- A bit is needed to determine if the segment is already in main memory

- Another bit is needed to determine if the segment has been modified since it was loaded in main memory

Segment Table Entries



(b) Segmentation only

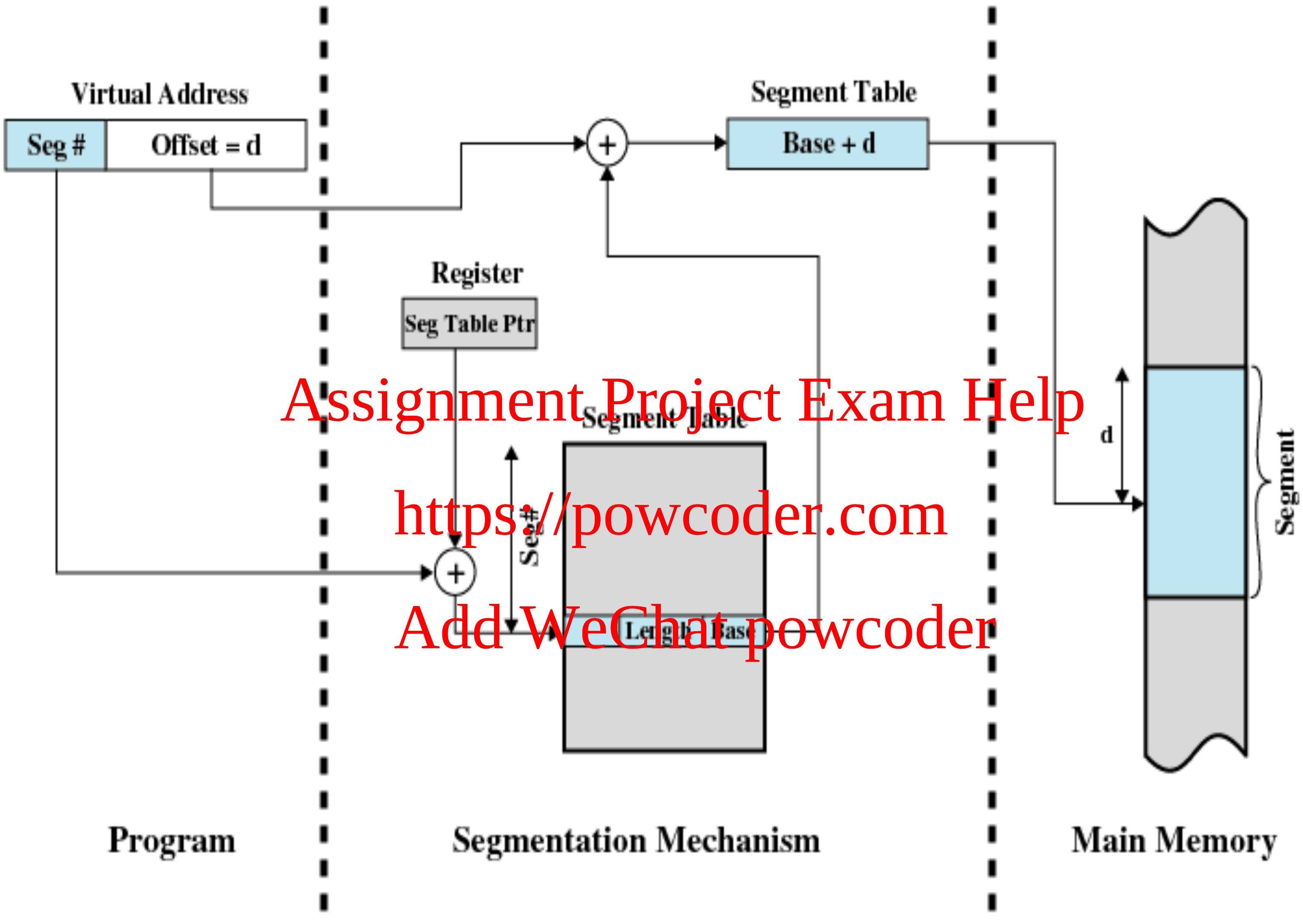
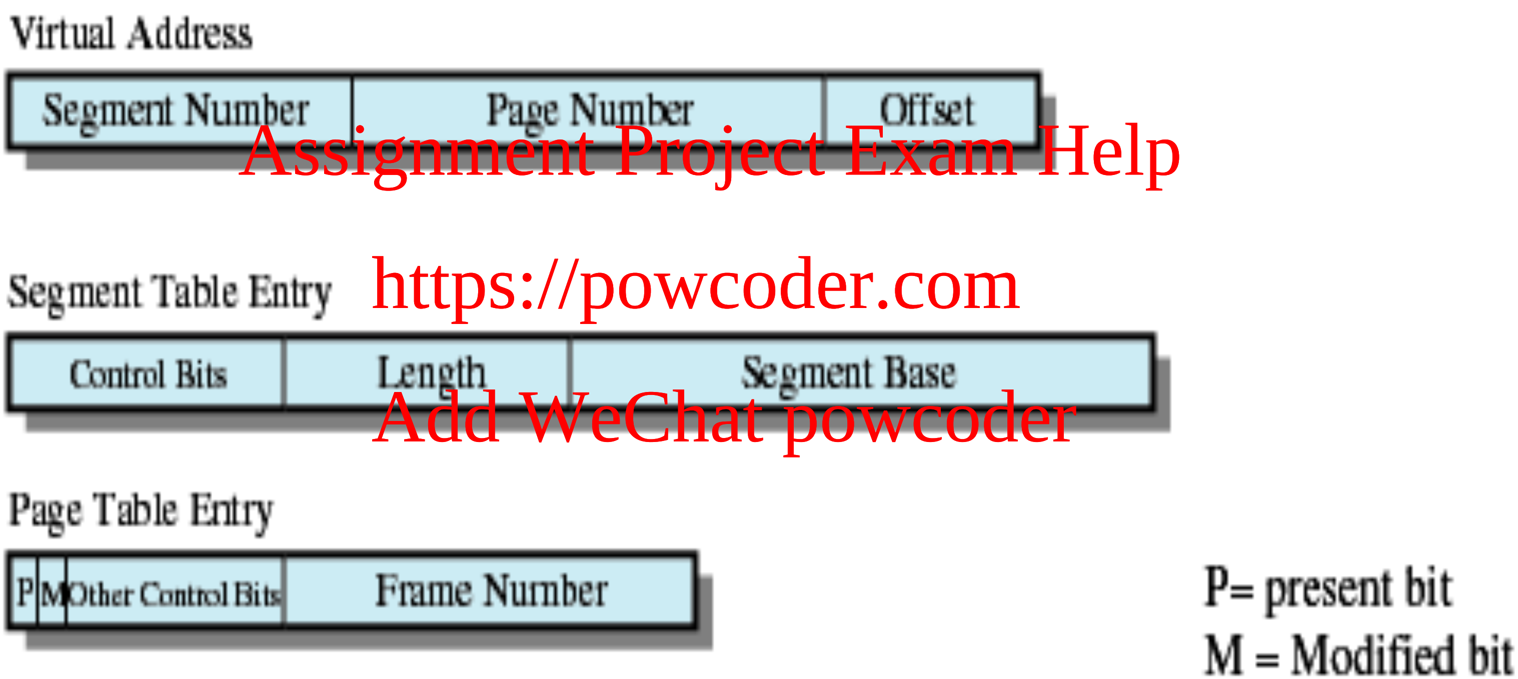


Figure 8.12 Address Translation in a Segmentation System

Virtual Memory Based on Combined Paging and Segmentation

- Paging is transparent to the programmer
- Segmentation is visible to the programmer
- Each segment is broken into fixed-size pages

Combined Segmentation and Paging



(c) Combined segmentation and paging

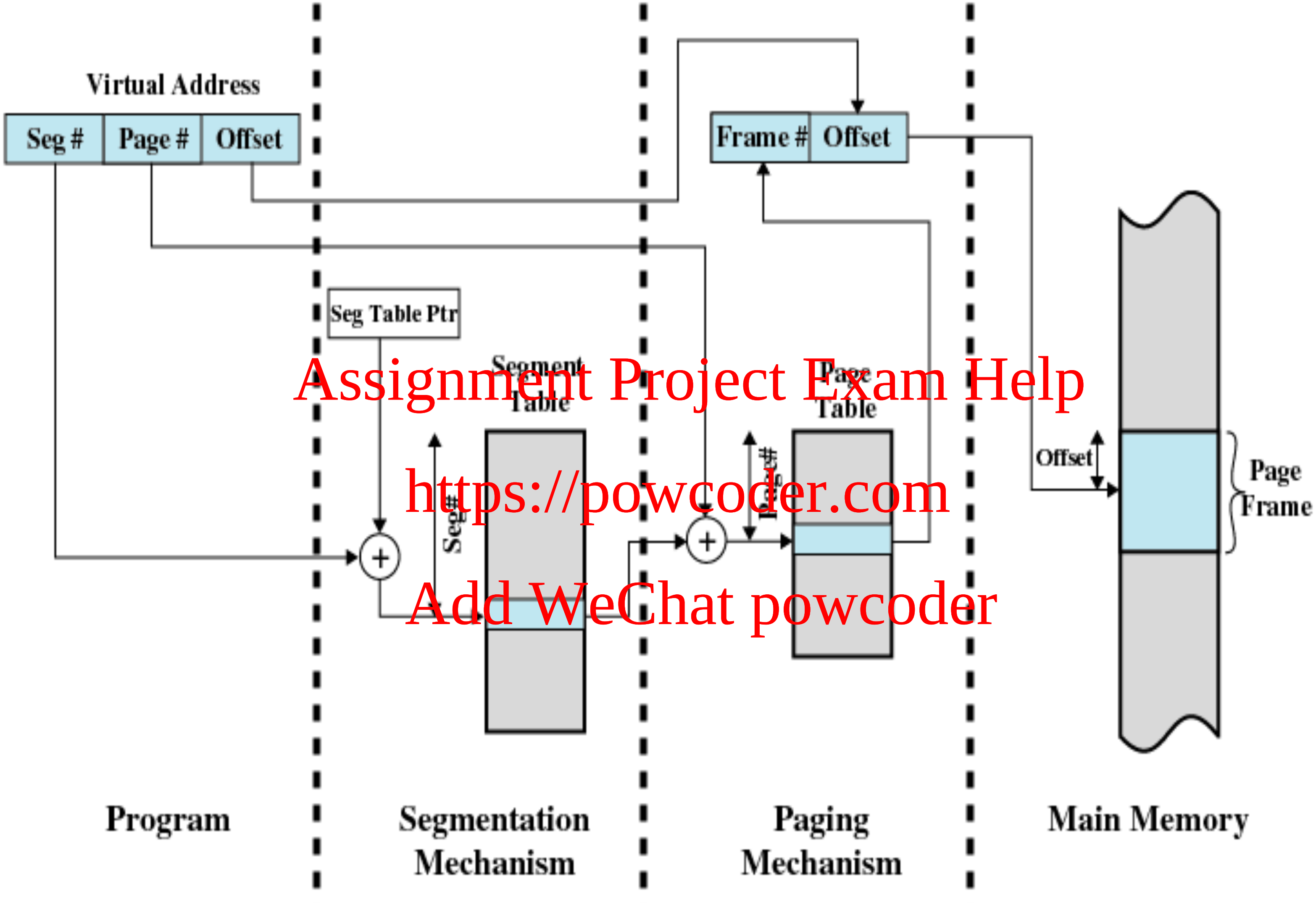


Figure 8.13 Address Translation in a Segmentation/Paging System

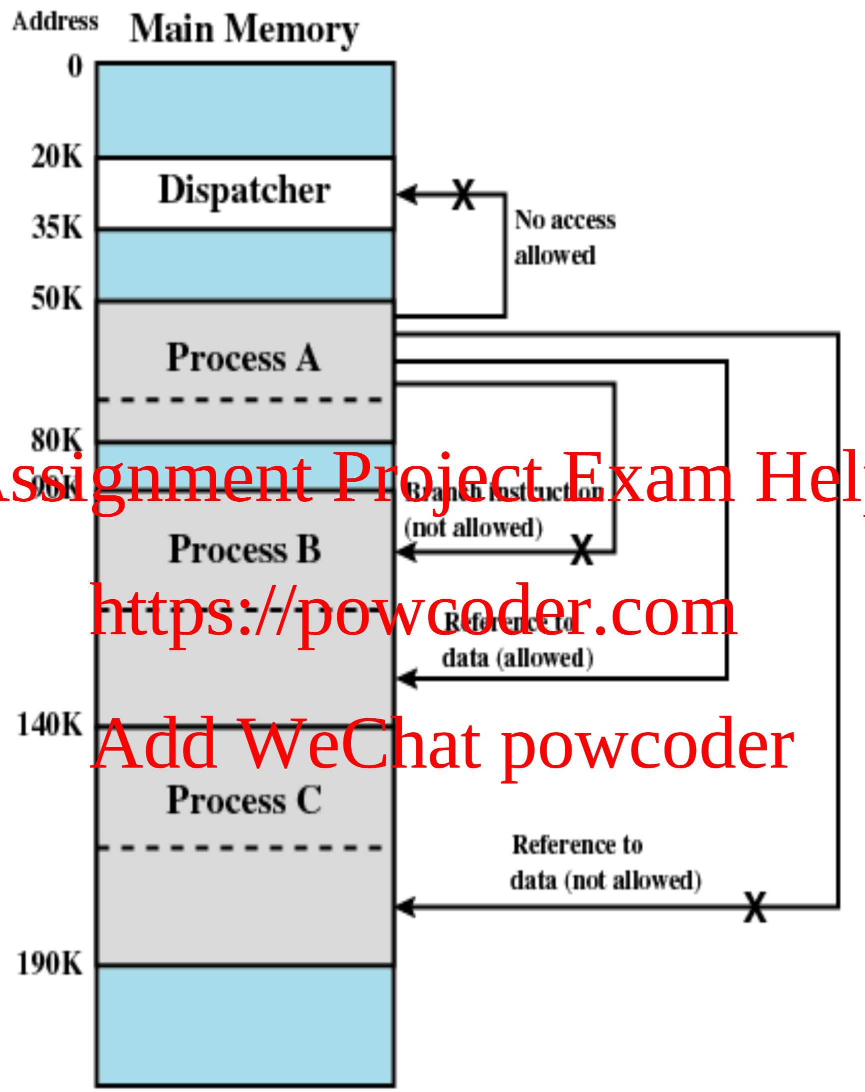


Figure 8.14 Protection Relationships Between Segments

Replacement Policy

- Placement Policy
  - Which page is replaced when there is no free page frame?
  - Page removed should be the page least likely to be referenced in the near future
  - Most policies predict the future behavior on the basis of past behavior

Replacement Policy

- Frame Locking
  - If frame is locked, it may not be replaced
  - Kernel of the operating system
  - Control structures
  - I/O buffers
  - Associate a lock bit with each frame

Basic Replacement Algorithms

- Optimal policy
  - Selects for replacement that page for which the time to the next reference is the longest
  - Impossible to have perfect knowledge of future events

Basic Replacement Algorithms

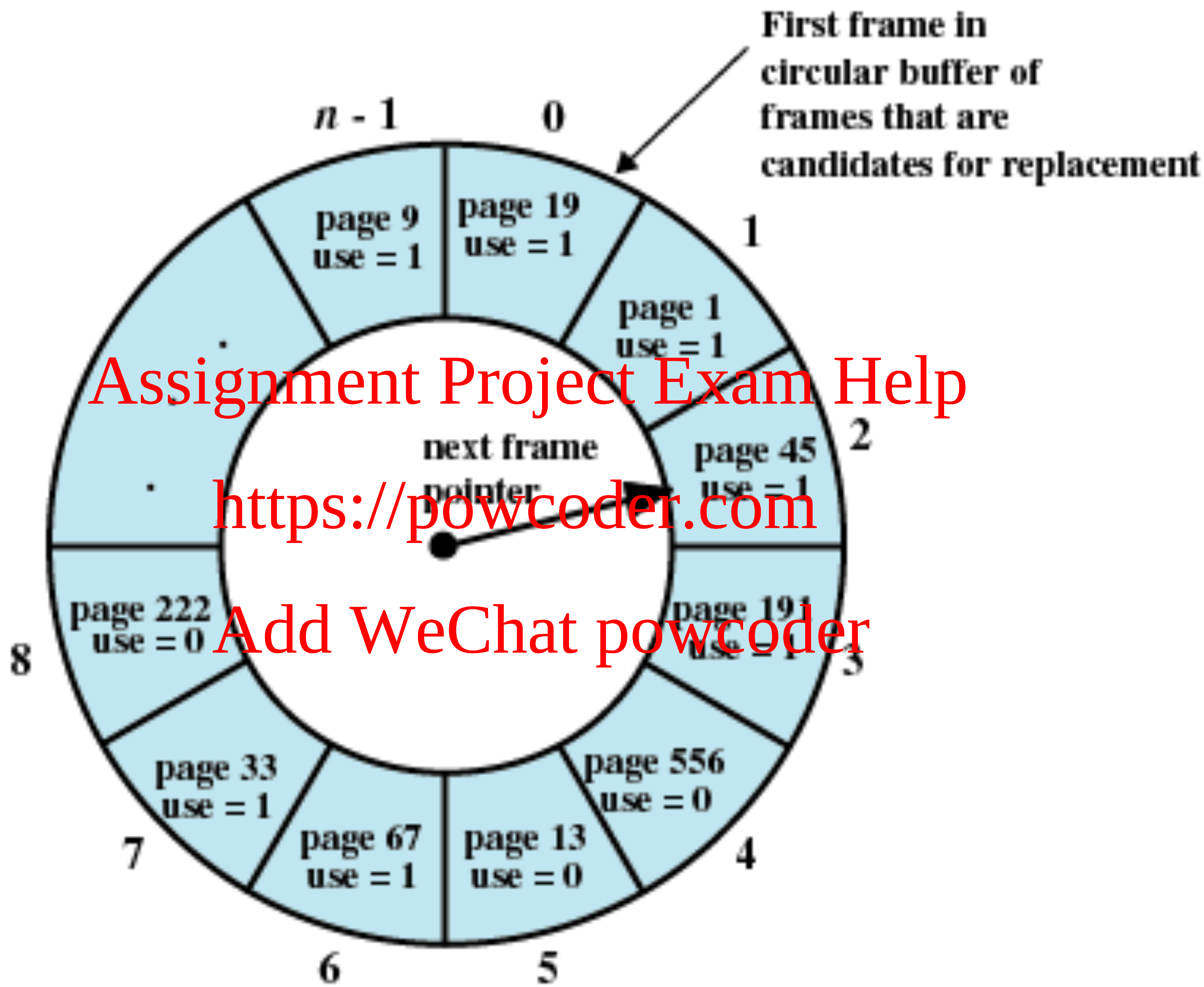
- Least Recently Used (LRU)
  - Replaces the page that has not been referenced for the longest time
  - By the principle of locality, this should be the page least likely to be referenced in the near future
  - Each page could be tagged with the time of last reference. This would require a great deal of overhead.

Basic Replacement Algorithms

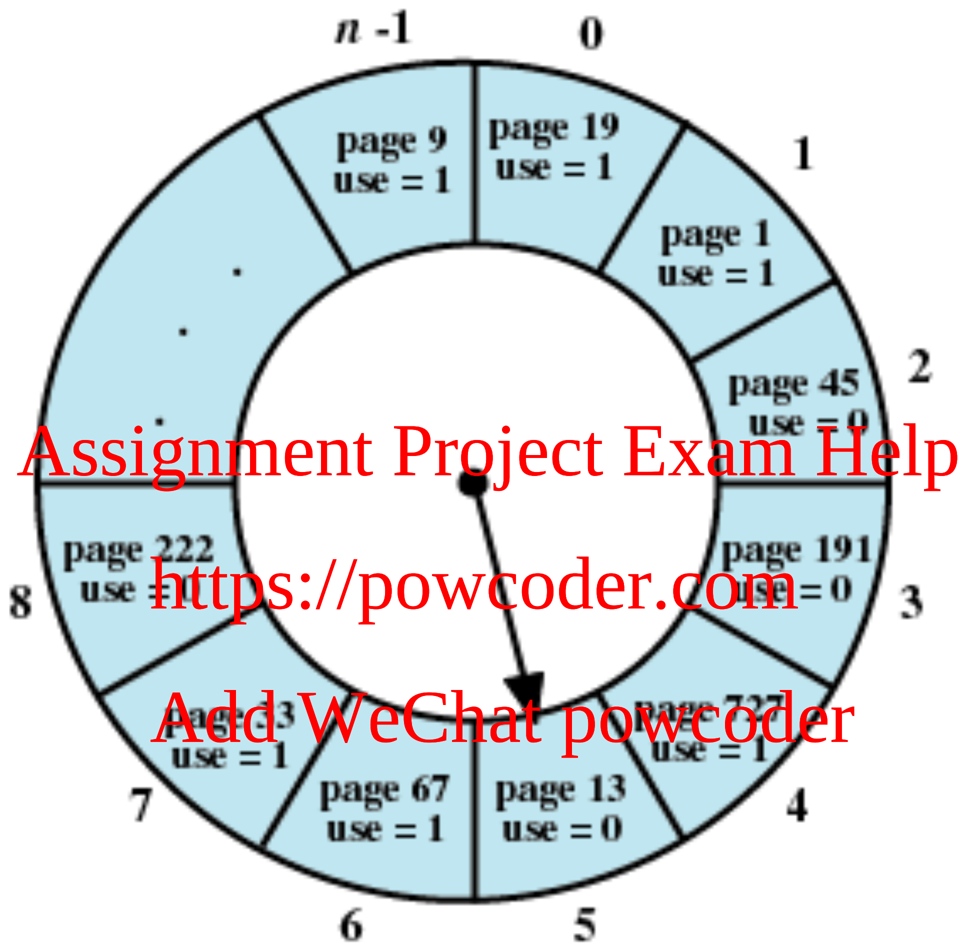
- First-in, first-out (FIFO)
  - Treats page frames allocated to a process as a circular buffer
  - Pages are removed in round-robin style
  - Simplest replacement policy to implement
  - Page that has been in memory the longest is replaced
  - These pages may be needed again very soon

Basic Replacement Algorithms

- Clock Policy
  - Additional bit called a use bit
  - When a page is first loaded in memory, the use bit is set to 1
  - When the page is referenced, the use bit is set to 1
  - When it is time to replace a page, the first frame encountered with the use bit set to 0 is replaced.
  - During the search for replacement, each use bit set to 1 is changed to 0



(a) State of buffer just prior to a page replacement



(b) State of buffer just after the next page replacement

Figure 8.16 Example of Clock Policy Operation

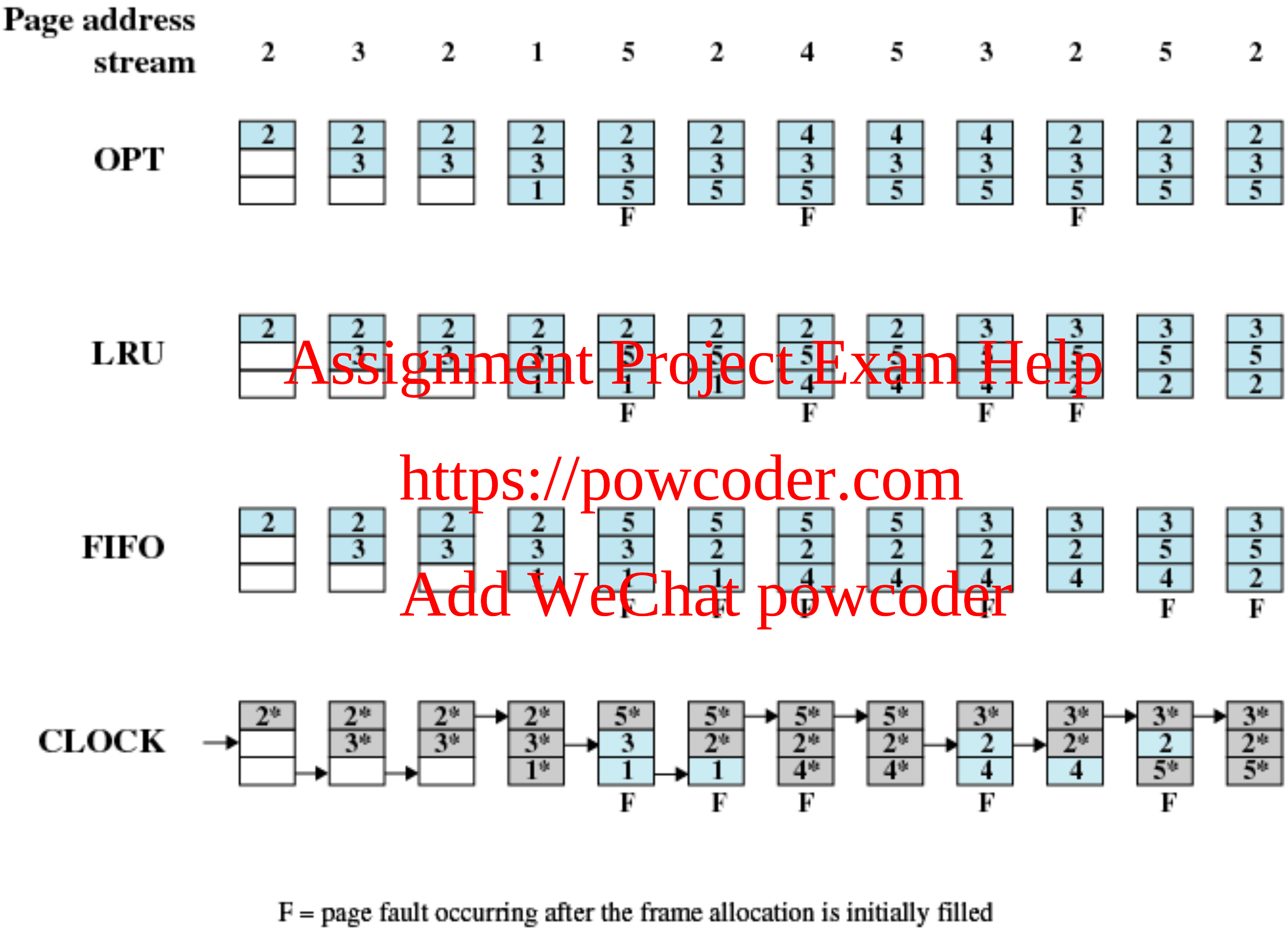


Figure 8.15 Behavior of Four Page-Replacement Algorithms

Comparison of Placement Algorithms

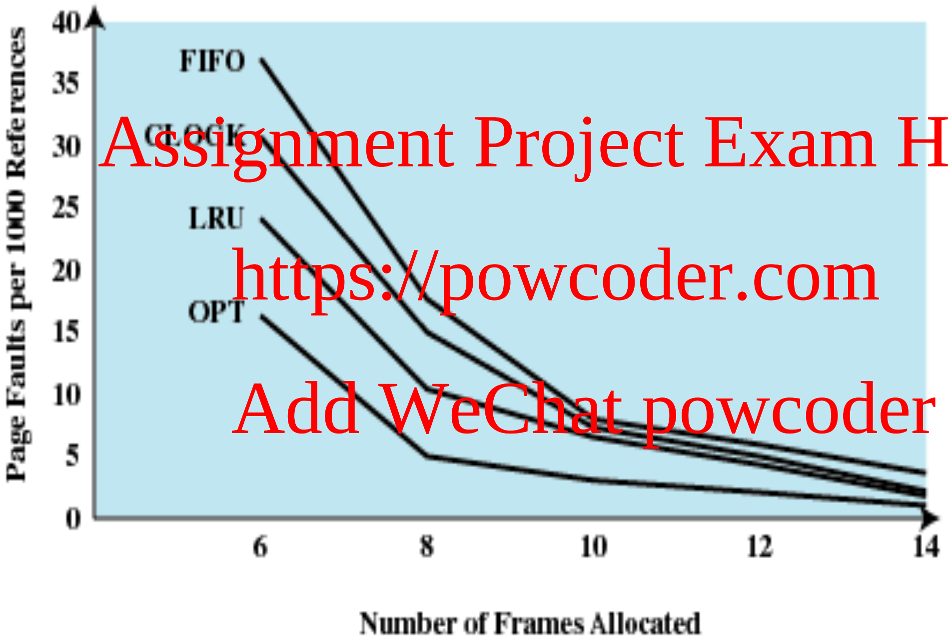


Figure 8.17 Comparison of Fixed-Allocation, Local Page Replacement Algorithms

UNIX and Solaris Memory Management

- Paging System
  - Page table
  - Disk block descriptor
  - Page frame data table
  - Swap-use table

Table 8.5 UNIX SVR4 Memory Management Parameters (page 1 of 2)

Page Table Entry

Page frame number
Refers to frame in real memory.

Age
Indicates how long the page has been in memory without being referenced. The length and contents of this field are processor dependent.

Copy on write
Set when more than one process shares a page. If one of the processes writes into the page, a separate copy of the page must first be made for all other processes that share the page. This feature allows the copy operation to be deferred until necessary and avoided in cases where it turns out not to be necessary.

Modify
Indicates page has been modified.

Reference

Indicates page has been referenced. This bit is set to zero when the page is first loaded and may be periodically reset by the page replacement algorithm.

Valid

Indicates page is in main memory.

Protect
Indicates whether write operation is allowed.

Disk Block Descriptor

Swap device number
Logical device number of the secondary device that holds the corresponding page. This allows more than one device to be used for swapping.

Device block number
Block location of page on swap device.

Type of storage
Storage may be swap unit or executable file. In the latter case, there is an indication as to whether the virtual memory to be allocated should be cleared first.

Table 8.5 UNIX SVR4 Memory Management Parameters (page 2 of 2)

Page Frame Data Table Entry

Page State
Indicates whether this frame is available or has an associated page. In the latter case, the status of the page is specified: on swap device, in executable file, or DMA in progress.

Reference count
Number of processes that reference the page.

Logical device
Logical device that contains a copy of the page.

Block number

Block location of the page copy on the logical device.

Pfdata pointer

Pointer to other pfdata table entries on a list of free pages and on a hash queue of pages.

Swap-use Table Entry

Reference count
Number of page table entries that point to a page on the swap device.

Page/storage unit number
Page identifier on storage unit.

Page frame number	Age	Copy on write	Modify	Reference	Valid	Protect

(a) Page table entry

Swap device number	Device block number	Type of storage

(b) Disk block descriptor

Page state	Reference count	Logical device	Block number	Pfdata pointer

(c) Page frame data table entry

Reference count	Page/storage unit number

(d) Swap-use table entry

Figure 8.22 UNIX SVR4 Memory Management Formats

UNIX and Solaris Memory Management

- Page Replacement
  - Refinement of the clock policy

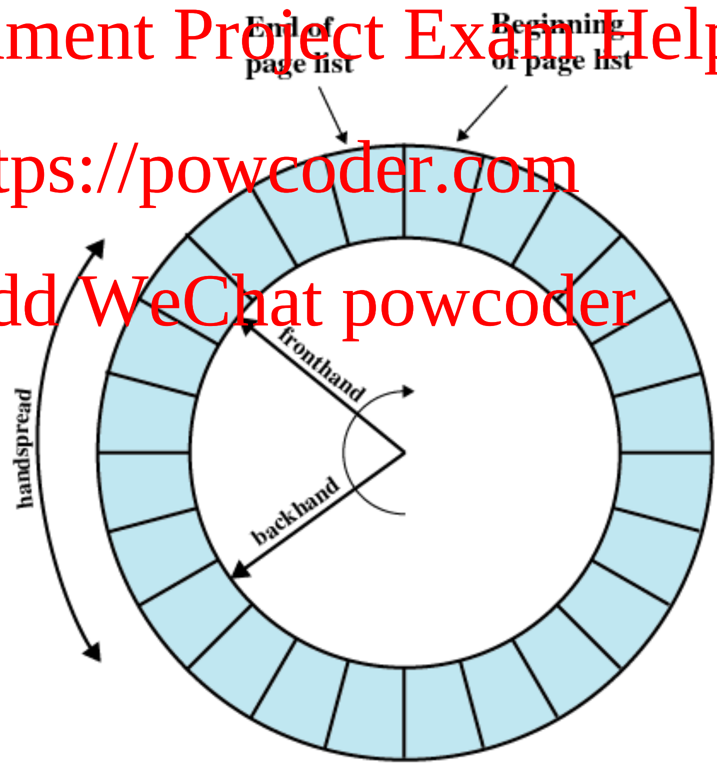


Figure 8.23 Two-Handed Clock Page-Replacement Algorithm

Linux Memory Management

- Page directory
- Page middle directory
- Page table

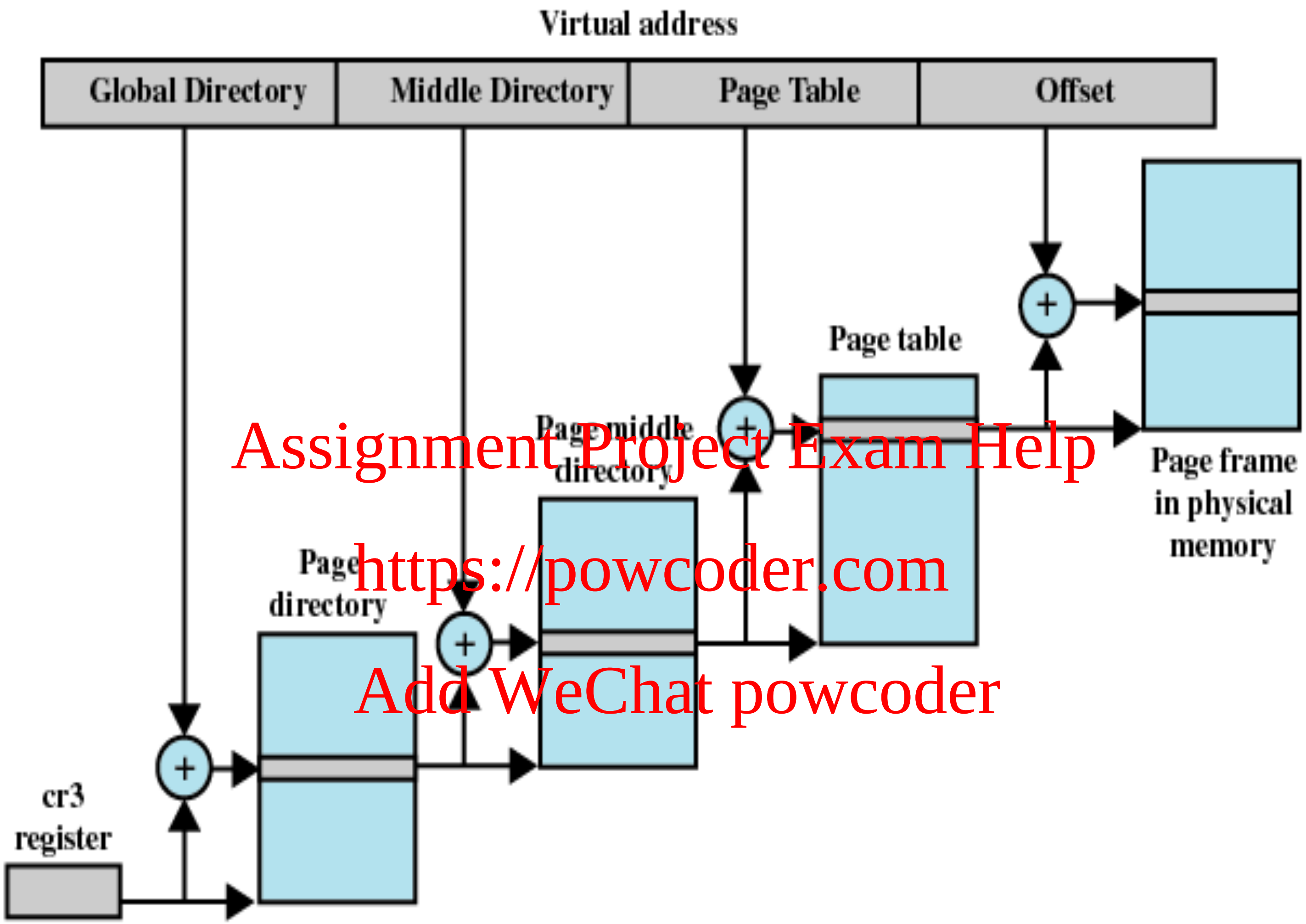


Figure 8.25 Address Translation in Linux Virtual Memory Scheme

Windows Memory Management

- Paging
  - Available
  - Reserved
  - Committed

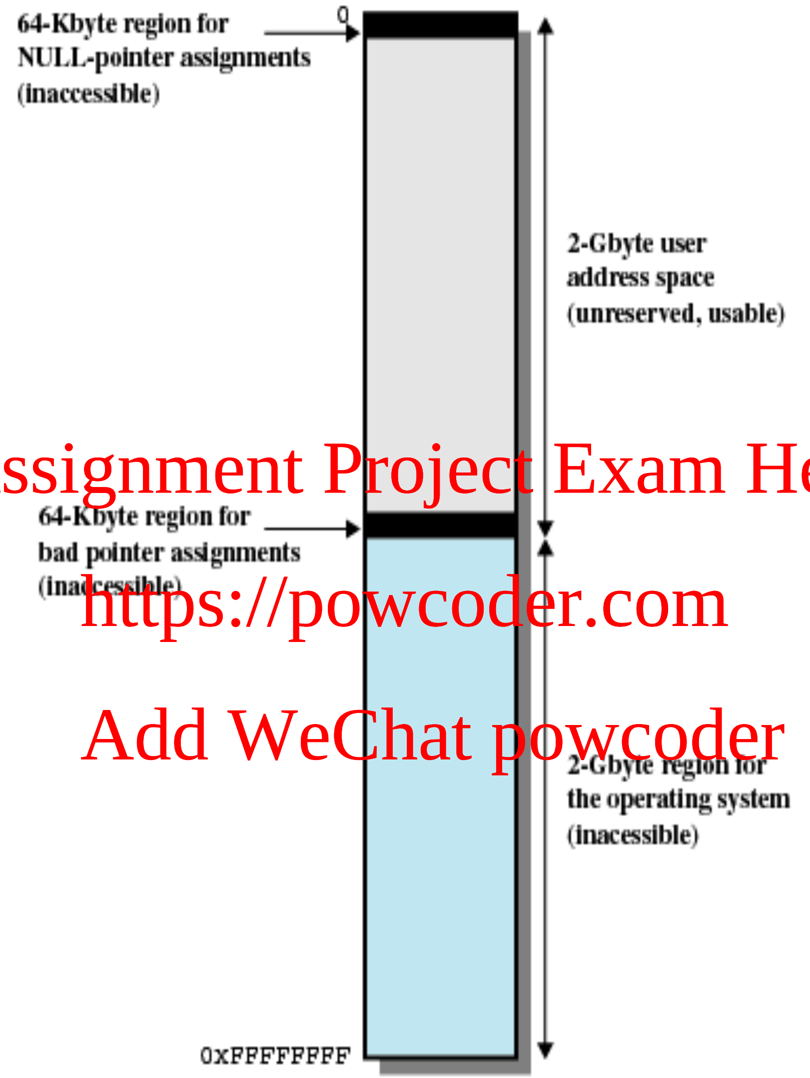


Figure 8.26 Windows Default Virtual Address Space