OPERATING SYSTEM
IT-42033

CHAPTER – 8
Q&A

Q. Address Binding

- Compile time: In compile time where the process will reside in memory, then can be generated. For example, a user process will reside starting at location R, then the generated compiler code will start at that location and extend up from there. If, at some later time, the starting location changes, then it will be necessary to recompile this code. The MS-DOS .COM-format programs are bound at compile time.
- Load time: In compile time where the process will reside in memory, then the compiler must generate In this case, final binding is delayed until load time. If the starting address changes, it need to reload the user code to incorporate this changed value.
- Execution time: If the process can be moved during its execution from one memory segment to another, then binding must be delayed until run time. Special hardware must be available for this scheme to work. Most general-purpose operating systems use this method.

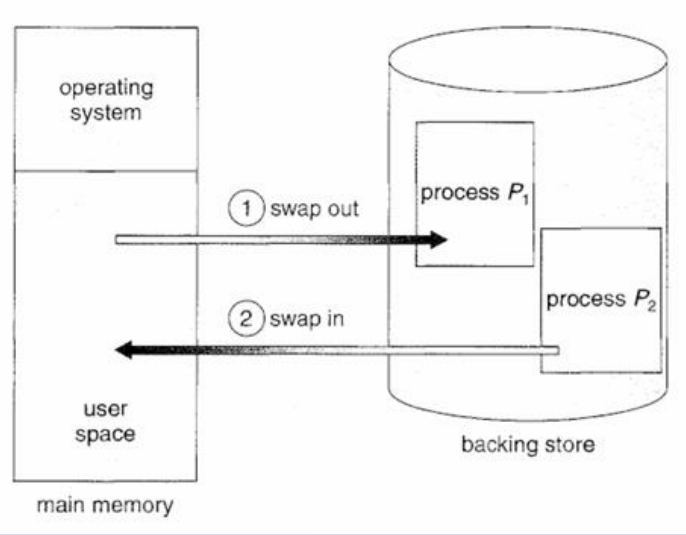
Q. Dynamic Loading

- With dynamic loading, a routine is not loaded until it is called. All routines are kept on disk in a relocatable load format. The main program is loaded into memory and is executed. When a routine needs to call another routine, the calling routine first checks to see whether the other routine has been loaded. If it has not, the relocatable linking loader is called to load the desired routine into memory and to update the program's address tables to reflect this change. Then control is passed to the newly loaded routine. The advantage of dynamic loading is that an unused routine is never loaded. This method is particularly useful when large amounts of code are needed to handle infrequently occurring cases, such as error routines. In this case, although the total program size may be large, the portion that is used may be much smaller. Dynamic loading does not require special support from the operating system.

Q. Swapping

- A process must be in memory to be executed. A process can be swapped temporarily out of memory to a and then brought into memory for continued execution. For example, assume a multiprogramming environment with a round-robin CPU-scheduling algorithm. When a quantum expires, the memory manager will start to swap out the process that just finished and to swap another process into the memory space that has been freed. The CPU scheduler will allocate a time slice to some other process in memory. When each process finishes its quantum, it will be swapped with another process. The memory manager can swap processes fast enough that some processes will be in memory, ready to execute, when the CPU scheduler wants to reschedule the CPU. In addition, the quantum must be large enough to allow reasonable amounts of computing to be done between swaps.

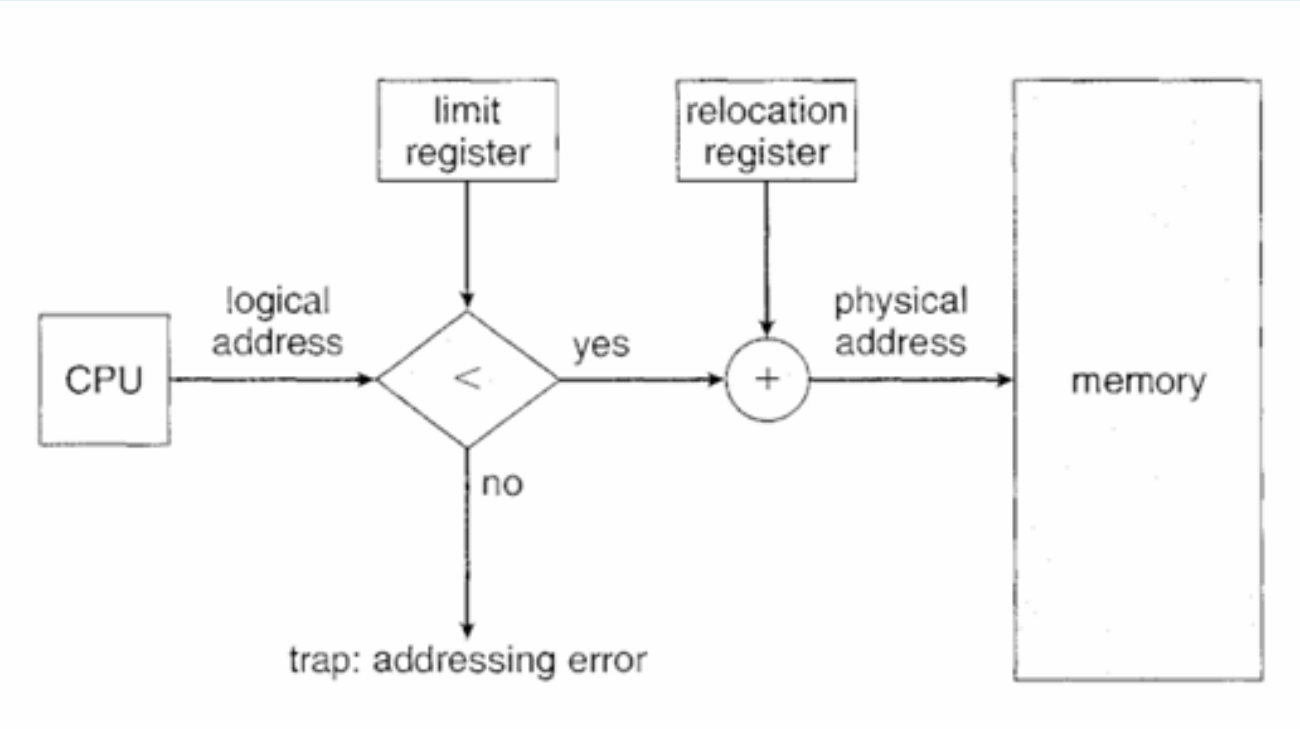
  - Swapping of two processes using a disk as a backing store



Q. Memory Allocation

- In the variable-partition scheme, the operating system keeps a table indicating which parts of memory are available and which are occupied. Initially, all memory is available for user processes and is considered one large block of available memory a Eventually as you will see, memory contains a set of holes of various sizes.
- For dynamic storage allocation problem, which concerns how to satisfy a request of size n from a list of free holes. The first-fit, best-fit, and worst-fit strategies are used to select a free hole.
- First fit: Allocate the first hole that is big enough. Searching can start either at the beginning of the set of holes or at the location where the previous first-fit search ended. We can stop searching as soon as we find a free hole that is large enough.
- Best fit: Allocate the smallest hole that is big enough. We must search the entire list, unless the list is ordered by size. This strategy produces the smallest leftover hole.
- Worst fit: Allocate the largest hole. Again, we must search the entire list, unless it is sorted by size. This strategy produces the largest leftover hole, which may be more useful than the smaller leftover hole from a best-fit approach.

Memory Allocation (cont…)



- Hardware support for relocation and limit registers

Q. Explain the external, internal fragmentation and compaction

1.External Fragmentation

- •Occurs when enough total memory exists to satisfy a request, but the free spaces are non-contiguous.
- •Caused by processes being loaded/removed, breaking free memory into small holes.
- •In worst case, a small block of wasted memory between every two processes.
- •First-fit and best-fit strategies both suffer from it.

2.Internal Fragmentation

- •Occurs when allocated memory is slightly larger than requested memory.
- •Request of 18,462 bytes in an 18,464-byte hole leaves 2 bytes unused, which is too small to track efficiently.
- •It fixed-size memory blocks.

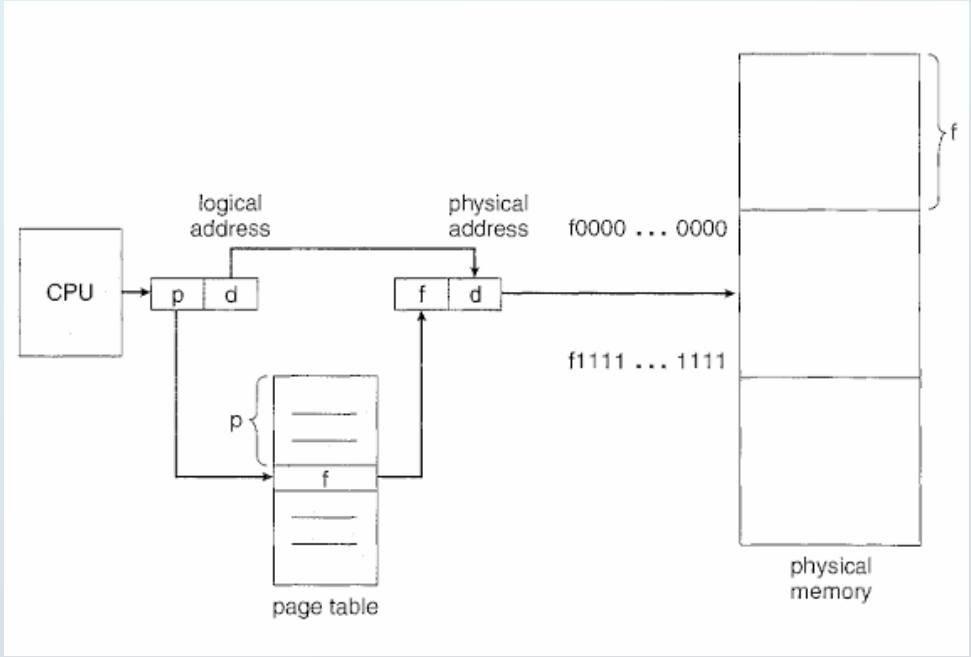
3.Compaction

- •Moves memory contents, so all free memory becomes one large block.
- •Possible only with dynamic relocation at execution time.
- •It moves all processes toward one end of memory, holes to the other.

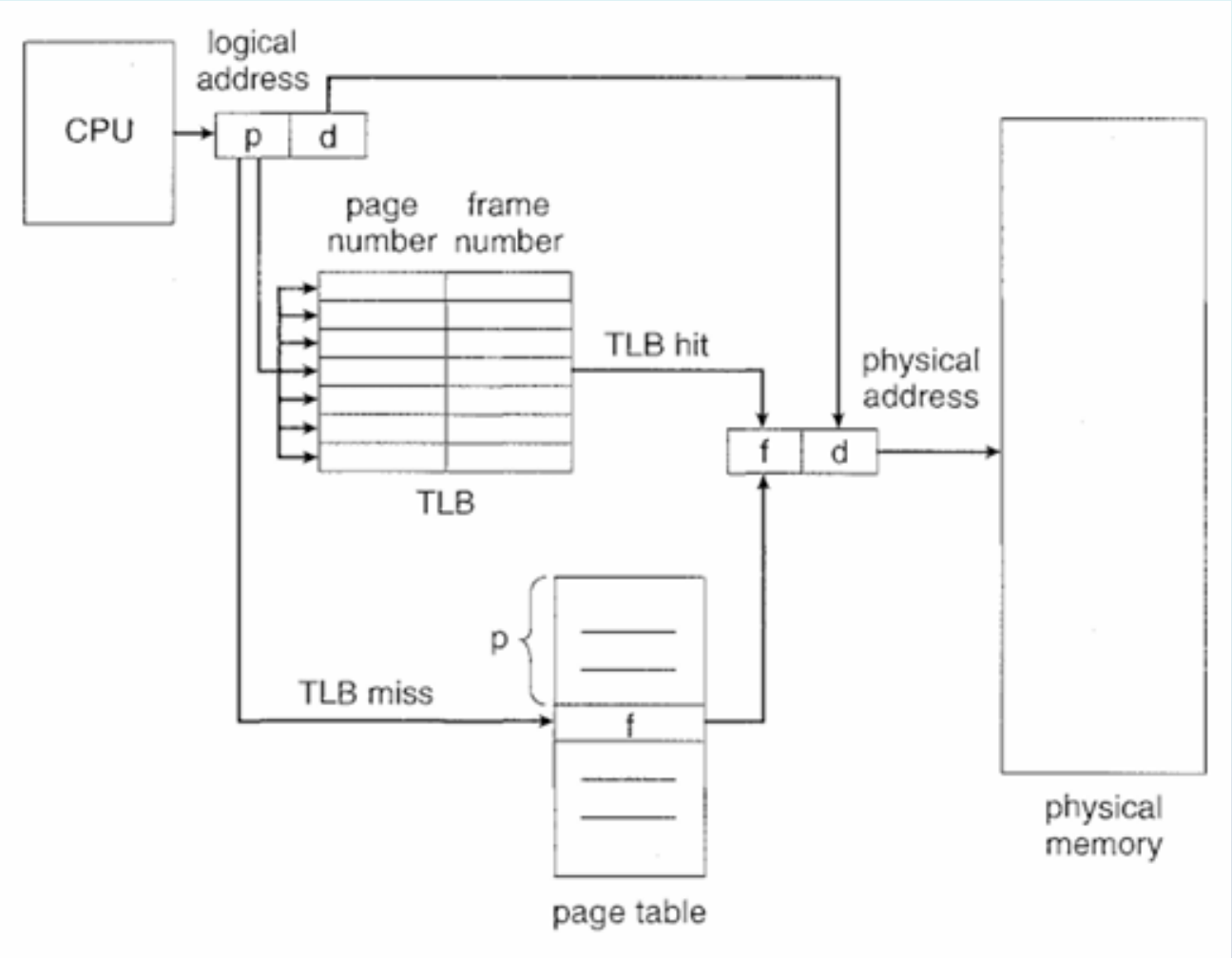
Q. Paging

- Paging is a memory-management scheme that permits the physical address space a process to be noncontiguous. Paging avoids external fragmentation and the need for compaction. It also solves the considerable problem of fitting memory chunks of varying sizes onto the backing store.
- The backing store has the same fragmentation problems discussed in connection with main memory, but access is much slower, so compaction is impossible. Because of its advantages over earlier methods, paging in its various forms is used in most operating systems.

- Paging Hardware



Paging (cont …)



- Paging Hardware with TLB

8.12.

The major advantage of this scheme is that it is an effective mechanism for code and data sharing. For example, only one copy of an editor or a compiler needs to be kept in memory, and this code can be shared by all processes needing access to the editor or compiler code. Another advantage is protection of code against erroneous modification. The only disadvantage is that the code and data must be separated, which is usually adhered to in a compiler-generated code.

8.17.

Since segment tables are a collection of base–limit registers, segments can be shared when entries in the segment table of two different jobs point to the same physical location. The two segment tables must have identical base pointers, and the shared segment number must be the same in the two processes.

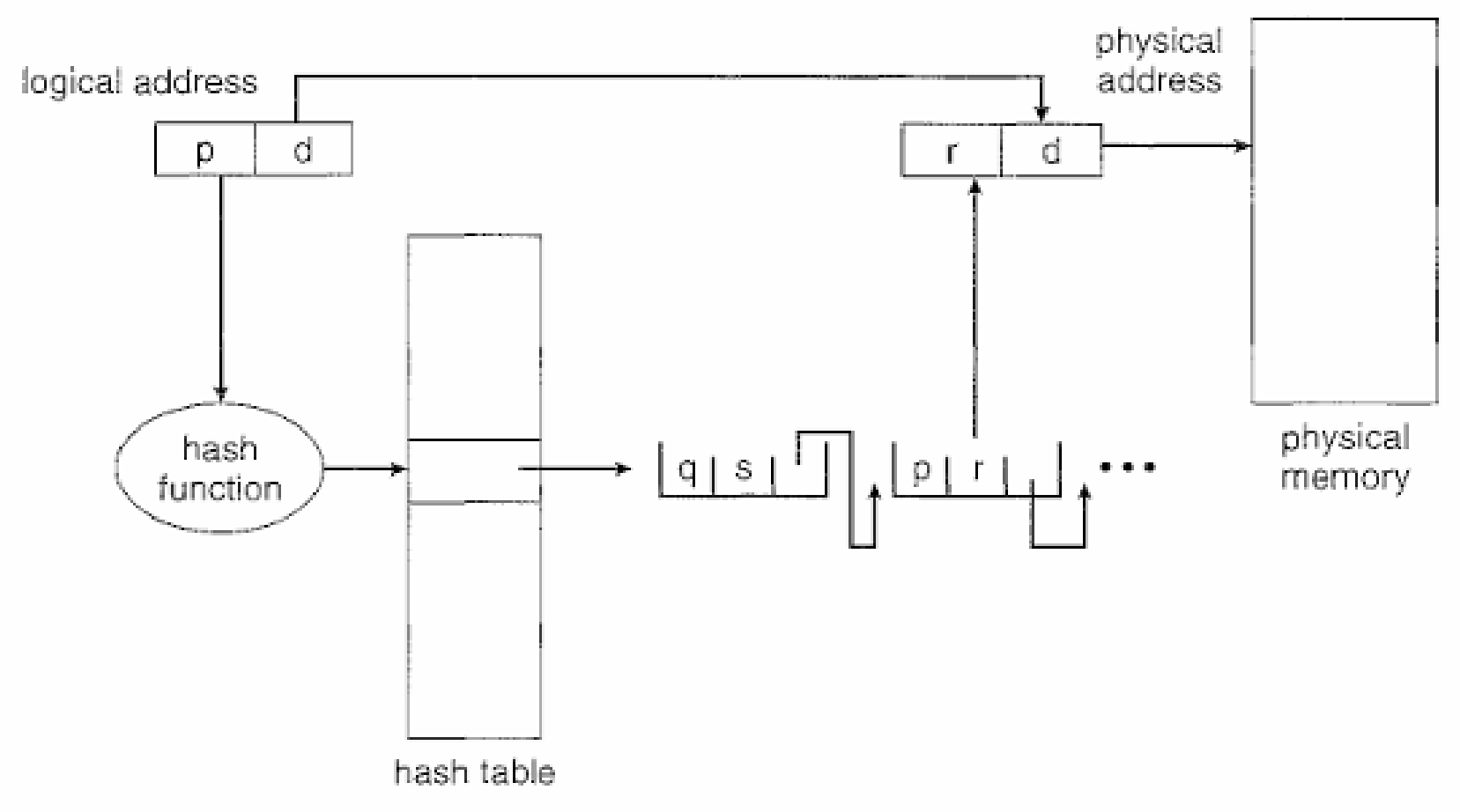
8.23.

Both of these problems reduce to a program being able to reference both its own code and its data without knowing the segment or page number associated with the address. Associating four registers with each process, one register had the address of the current program segment, another had a base address for the stack, another had a base address for the global data, and so on. The idea is that all references have to be indirect through a register that maps to the current segment or page number. By changing these registers, the same code can execute for different processes without the same page or segment numbers.

Q. Hashed Page Tables

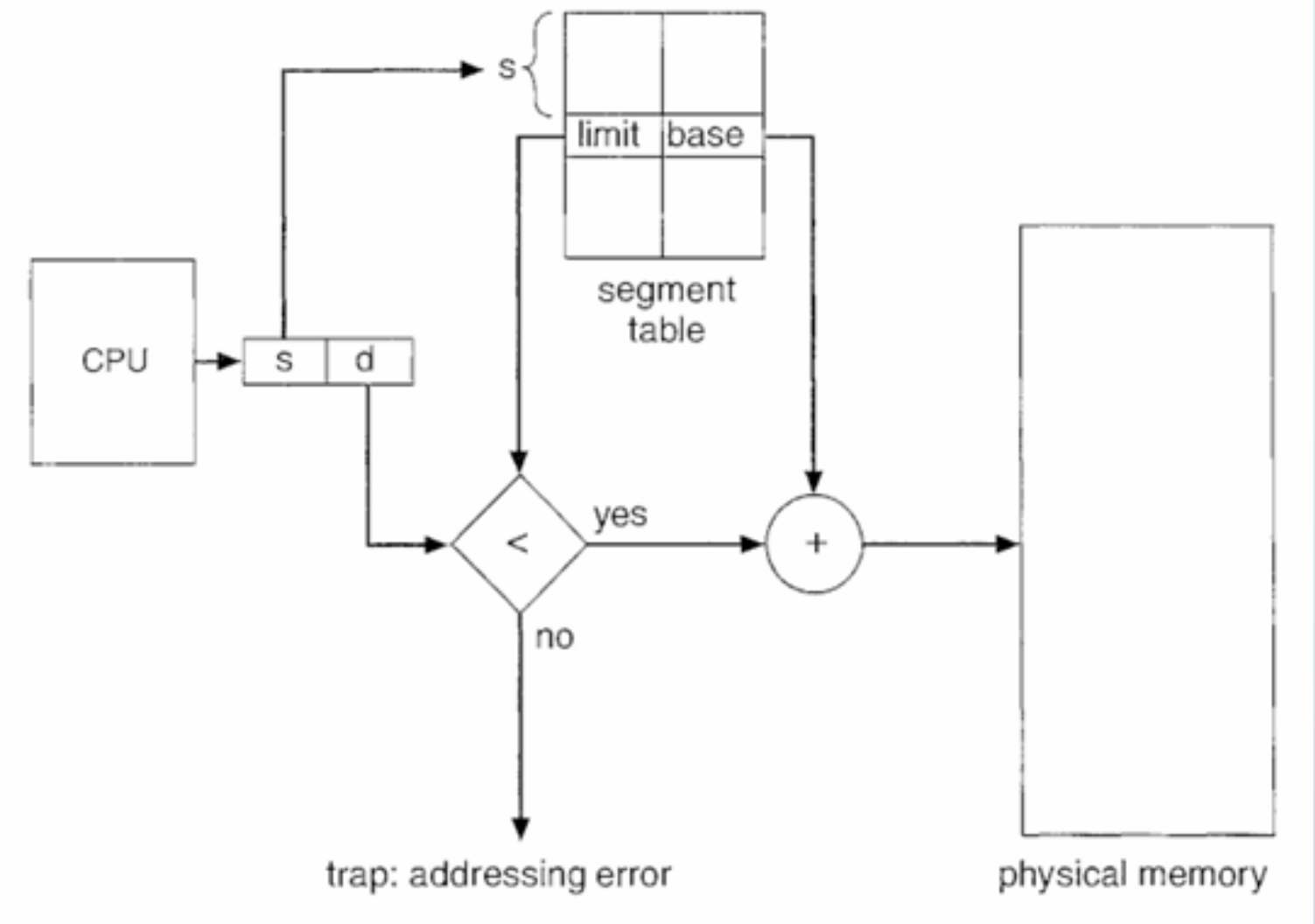
- Hashed page tables are used for address spaces larger than 32 bits.
- The virtual page number is hashed to find its location in the table.
- Each table entry has a linked list to handle collisions.
- Each linked list element stores the virtual page number, the mapped page frame, and a pointer to the next element.
- The system searches the linked list for a matching virtual page number.
- If a match is found, the mapped frame is used to form the physical address.
- Clustered page tables are a variation of hashed page tables.
- Each entry in a clustered page table maps several pages, not just one.
- They are useful for sparse address spaces with scattered memory references.

Hashed Page Tables (cont …)



- Hashed page table

Q. Illustrate Segmentation Hardware



- Segmentation Hardware