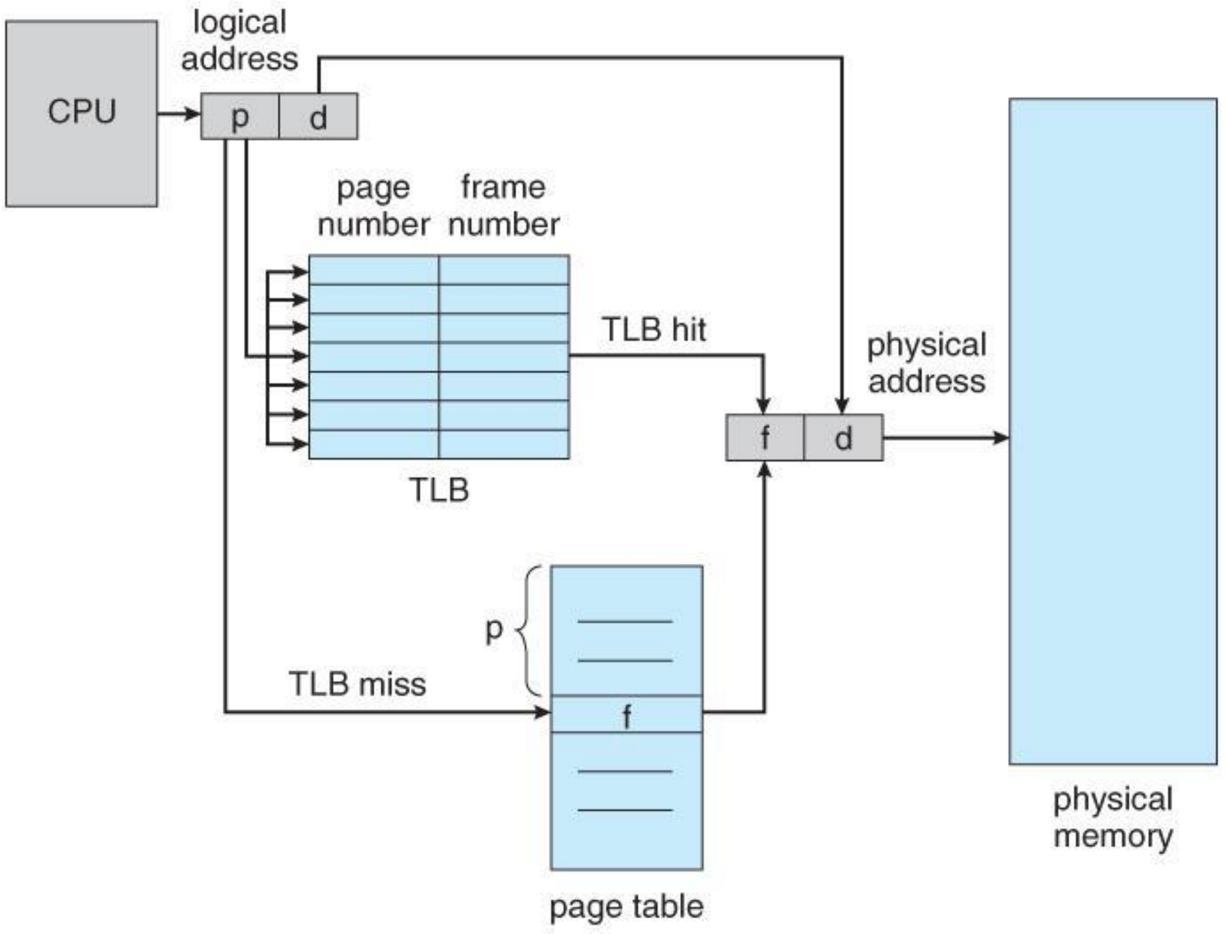
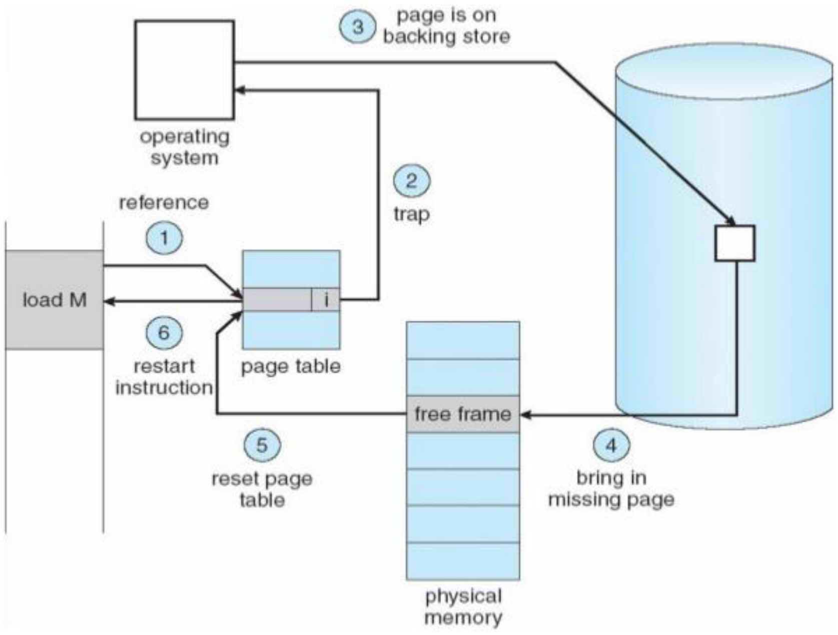
Project 3 | CPE 453
Due: (check schedule for updates)
This assignment may be done in groups of 2-3.

Virtual Memory Simulator (MemSim)

This assignment is about material covered in the memory chapters 9 and 10 of the Silberschatz 10th edition book. This is actually very close to the programming project at the end of chapter 10. You are well advised to read ahead to get started on this project.



Memory access process with TLB



Page fault process (no TLB shown)

Designing a Virtual Memory Manager

As the problem in the book states, you have to translate logical memory to physical and simulate the operations of the main memory structures needed. Below are the units needed to accomplish this. Each of the units below should be modeled as a distinct function, class or data structure in your code. We only care about read operations for this assignment. No need to implement write()s.

- **TLB.** The Translation-Lookaside-Buffer is like a small cache for memory translation. An entry in the TLB consist of a logical memory page number and its corresponding physical memory frame #. TLB always gets populated on first-in-first-out (FIFO) basis, meaning the slot populated earliest, is the one that should be replaced by a new entry. Currently empty slots are not given any special priority. We will implement a 16 entry TLB with a rotating FIFO replacement algorithm.
- **Page Table.** The page table is a much larger structure that keeps track of every page currently loaded in memory (and pages not currently loaded). It provides the same page to frame translation as TLB. When there is a TLB miss, then we lookup the page # in the page table. If it's not here either, we incur a page fault. Which means, we have to load the page from the backing store and then update both page table and TLB.
- "**backing store" or disk device**. This is where the content of all pages can be found. This will be given to you in form of a Unix file called BACKING_STORE.bin of length 65,536 bytes. Every 256 bytes (page size) is a page. Just open it using system calls and read from it as binary. You will need to test with your own version as well as an example one provided. It will not be the same one you are graded on.

- **Physical memory.** This is where the actual content of memory is loaded. It is also stored in page-size chunks (256 Bytes) called frames. Most often there is not enough physical memory to store all the logically addressable pages. That's why we have to manage (using a page -replacement algorithm) which pages are loaded into which frames. The size of the physical memory (in terms of number of frames) is passed in via the command-line to MemSim.
- **Reference sequence.** This is a sequence of logical addresses that need to be read. A text file of ASCII integers (one per line) will be given on the command line. Each line is a logical address that needs to be translated to physical.

Modifications to the book version

In the book version, the size of the physical memory is the same as the size of the logical memory. Therefore, the only kinds of page faults we can expect are the kind that result from empty frames at the beginning. Once all pages are loaded into physical memory, there is no need ever to load any more pages or evacuate existing pages. In this version, we will work with a variable physical memory size (in terms of frames). And since it is possible to have a smaller physical memory, we will need to use a page replacement algorithm. Both of these will be passed in as command line arguments. (See Executable section)

The specifications for this assignment are as follows:

- 2^8 entries in the page table (same as the book's version)
- 16 entries in the TLB (same as book)
- Page size = Frame size = 256 Bytes (same as the book)
- Variable number of frames (FRAMES given as command line argument)
- Support for multiple page replacement algorithms (PRA given as command line argument)
- Physical memory of size 256*FRAMES
- Page table to have a "loaded" bit associated with it. If the bit is set, it means that page is currently loaded into a physical memory frame.
- Your executable must be called **memSim**
- The output of the executable will have these components, printed to standard out.
    - For every address in the given addresses file, print one line of comma-separated fields, consisting of
        - The full address (from the reference file)
        - The value (1 signed integer)
        - The physical memory frame number (one positive integer)
        - The content of the entire frame (256 bytes in hex ASCII characters, no spaces in between)
        - new line character
    - Total number of page faults and a % page fault rate
    - Total number of TLB hits and misses and % TLB miss rate

Executable

The usage of the main executable is this:

```
usage: memSim <reference-sequence-file.txt> <FRAMES> <PRA>
```

- reference-sequence-file contains the list of logical memory addresses
- FRAMES is an integer <= 256 and > 0 which is the number of frames in the system. Note that a FRAMES number of 256 means logical and physical memory are of the same size.
- PRA is "fifo" or "lru" or "opt" standing for a particular page replacement algorithm.
- FRAMES and PRA or just PRA may be omitted. If so, use defaults: 256 for FRAMES and "fifo" for PRA.

Grading

For this assignment, I will run multiple tests. You are welcome to complete only some of the tests for the given number of points shown. For full credit, you must pass all the tests.

- Implementation without any modifications from the book. (i.e. hard-coded 256 frames, and no replacement algorithm): 50 points
- Support for variable number of frames and "fifo" page replacement algorithm: 20 points
- Support for "lru" approximate (last used) page replacement algorithms: 20 points
- Support for "opt" (optimal) page replacement algorithm: 10 points

Resources (on canvas)

BACKING_STORE.bin (Backing store binary file),
example address sequence and (FIFO) results:
addresses.txt, addresses-correct.txt

Deliverables

Submit a **tar.gz** file with all your source code (no binaries). It must include the following:

1. Your source and header file(s).
2. A makefile (called Makefile) that will build the **memSim** executable. (This can be skipped for Python executables, but memSim itself must be submitted as an executable)
3. A 1-2 page report that describes how you modeled the simulator, including each of the objects and functional units.
4. A README file that contains:
    - Your names.
    - Any special instructions and shortcomings.

- The README file should be plain text, should be named “README.txt”