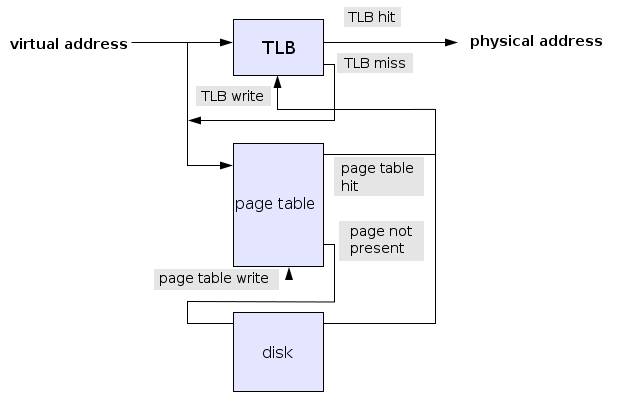
**Program 3**

Last updated: May 3, 2016

This assignment **may not** be done in groups

# Virtual Memory Simulator (memSim)

This assignment is about material covered in the memory Chapters 8 and 9 of the Silberschatz 9th edition book. You are well advised to read ahead to get started on this project. The project is the same “Programming Project” listed in Chapter 9, 9th edition book pages 458-461 with a few small modifications. Read those pages and make sure you understand the background and diagrams, but keep in mind the modifications listed.



## 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. We only care about read operations for this assignment. No need to implement writes.

* **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 number. In this assignment, the TLB gets populated on a first-come-first-serve (FCFC) basis, meaning the slot populated earliest, is the one that should be replaced by a new entry. You will implement a TLB with 16 entries.
* **Page Table.** The page table is a much larger structure that keeps track of every page of a process, including those currently loaded and not currently loaded in main memory. It provides the same page to frame translation as the TLB. When there is a TLB miss, that induces a lookup the page number in the page table. If the entry in the page table is not valid, we incur a page fault. Which means, we have to load the page from the backing store and then update both the page table and TLB. In this assignment the page size is 256 bytes.
* “**Backing store” or disk device**. This is where all pages can be found. This will be given to you in form of a file called [BACKING\_STORE.bin](https://drive.google.com/file/d/0B0Ogk2egn6BfcFFkVm0xNlhhRWM/view?usp=sharing) of length 65,536 bytes, addressable by 256 byte blocks. Note: You should test with your own backing store 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 main memory. It is also addressable in 256 byte frames. Most often there is not enough frames to store all the logically addressable pages for a process. Thus, we have to manage (using a page-replacement algorithm) which pages are loaded into which frames. In this assignment, the size of the physical memory (given as the number of frames) is passed into memSim via the command-line.
* **Reference sequence.** This is a sequence of logical address requests. A text file, containing ASCII integers (one per line), will be given on the command line. Each line is a logical address that needs to be translated to a physical address. Here’s [an example](https://drive.google.com/file/d/0B0Ogk2egn6BfcXM2NFhoZUhRaWM/view?usp=sharing) along with an [example output](https://drive.google.com/file/d/0B0Ogk2egn6BfYTVIcUpRU0QzUjA/view?usp=sharing).

## 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 pages. Once all pages are loaded into physical memory, there is no need ever to load any more pages or evacuate existing pages. That’s boring. In this version, we will work with a variable physical memory size (in terms of frames). And, because 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](#_12uuh6hmkd28) section, below).

The specifications for this assignment are as follows:

* 28 entries in the page table (same as the book’s version)
* 16 entries in the TLB (same as book)
* Page size = frame size = block size = 256 bytes (same as the book)
* Variable number of frames (FRAMES given as command line argument)
* Support for multiple page replacement algorithms (given as command line argument)
* Physical memory of size in bytes is 256\*FRAMES
* Page table is to have a “present” bit associated with it. If the bit is set, it means that page is currently valid, and is present in 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 of the byte referenced (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:

memSim <reference-sequence-file.txt> <FRAMES> <PRA>

* reference-sequence-file.txt 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,” representing first-in first-out, least recently used, and optimal algorithms, respectively.
* FRAMES and PRA or just PRA may be omitted. If so, use 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. For any credit, tests must pass on the CSL infrastructure (e.g. unix1, unix13.csc.calpoly.edu).

* Implementation without any modifications from the book. (*i.e.* hard-coded 256 frames, and no replacement algorithm): 10 points
* Support for a variable number of frames and FIFO page replacement algorithm: 4 points
* Support for LRU approximate (last used) page replacement algorithms: 4 points
* Support for OPT (optimal) page replacement algorithm: 2 points

## Deliverables

Submit a gzip’d tar archive 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.)
3. A README file that contains:

* Your name
* Any special instructions.
* Any other thing you want me to know while I am grading it.
* The README file should be plain text, i.e, not a Word document, and should be named “README”, all capitals with no extension.