# Lab05: Designing a Virtual Memory Manager

## Due date

Please refer to the lab assignment requirements.

## Goal

The goal of this project is to practice address translation of virtual memory management.

## Details

This project consists of writing a program that translates logical to physical addresses for a virtual address space of size 2^16 = 65,536 bytes. Your program will read from a file containing logical addresses and, using a TLB and a page table, will translate each logical address to its corresponding physical address and output the value of the byte stored at the translated physical address.

Your learning goal is to use simulation to understand the steps involved in translating logical to physical addresses. This will include resolving page faults using demand paging, managing a TLB , and implementing a page-replacement algorithm.

## Specific

Your program will read a file containing several 32-bit integer numbers that represent logical addresses. However, you need only be concerned with 16-bit addresses, so you must mask the rightmost 16 bits of each logical address. These 16 bits are divided into (1) an 8-bit page number and (2) an 8-bit page offset. Hence, the addresses are structured as shown as:
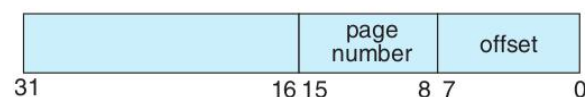


Other specifics include the following:
- 2^8 entries in the page table
- Page size of 2^8 bytes
- 16 entries in the TLB
- Frame size of 2^8 bytes
- 256 frames
- Physical memory of 65,536 bytes (256 frames × 256-byte frame size)

Additionally, your program need only be concerned with reading logical addresses and translating

them to their corresponding physical addresses. You do not need to support writing to the logical address space.

## Address Translation

Your program will translate logical to physical addresses using a TLB and page table as outlined in the textbook. First, the page number is extracted from the logical address, and the TLB is consulted. In the case of a TLB hit, the frame number is obtained from the TLB . In the case of a TLB miss, the page table must be consulted. In the latter case, either the frame number is obtained from the page table, or a page fault occurs. A visual representation of the address-translation process is:
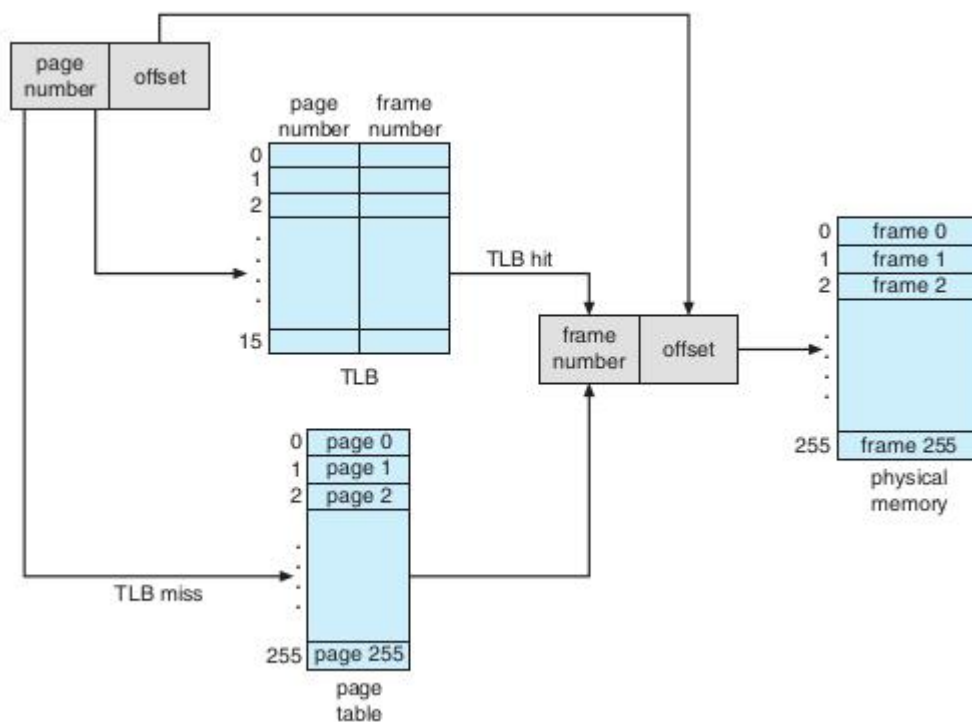


## Handling Page Faults

Your program will implement demand paging strategy. The backing store is represented by the file BACKING_STORE.bin , a binary file of size 65,536 bytes. When a page fault occurs, you will read in a 256-byte page from the file BACKING_STORE and store it in an available page frame in physical memory. For example, if a logical address with page number 15 resulted in a page fault, your program would read in page 15 from BACKING_STORE (remember that pages begin at 0 and are 256 bytes in size) and store it in a page frame in physical memory. Once this frame is stored (and the page table and TLB are updated), subsequent accesses to page 15 will be resolved by either the TLB or the page table.

You will need to treat BACKING_STORE.bin as a random-access file so that you can randomly seek to certain positions of the file for reading. We suggest using the standard C library functions for

performing I/O , including fopen() , fread() , fseek() , and fclose() .

The size of physical memory is the same as the size of the virtual address space—65,536 bytes—so you do not need to be concerned about page replacements during a page fault. Later, we describe a modification to this project using a smaller amount of physical memory; at that point, a page-replacement strategy will be required.

## Test File

We provide the file addresses.txt , which contains integer values representing logical addresses ranging from 0 to 65535 (the size of the virtual address space). Your program will open this file, read each logical address and translate it to its corresponding physical address, and output the value of the signed byte at the physical address.

## How to Begin

First, write a simple program that extracts the page number and offset based on:
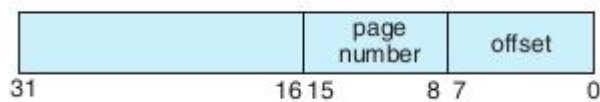


from the following integer numbers:

1, 256, 32768, 32769, 128, 65534, 33153

Perhaps the easiest way to do this is by using the operators for bit-masking and bit-shifting. Once you can correctly establish the page number and offset from an integer number, you are ready to begin.

Initially, we suggest that you bypass the TLB and use only a page table. You can integrate the TLB once your page table is working properly. Remember, address translation can work without a TLB ; the TLB just makes it faster. When you are ready to implement the TLB , recall that it has only sixteen entries, so you will need to use a replacement strategy when you update a full TLB . You may use either a FIFO or an LRU policy for updating your TLB .

## How to Run Your Program

Your program should run as follows:

./a.out addresses.txt

Your program will read in the file addresses.txt , which contains 1,000 logical addresses ranging from 0 to 65535. Your program is to translate each logical address to a physical address and determine the contents of the signed byte stored at the correct physical address. (Recall that in the C language, the char data type occupies a byte of storage, so we suggest using char values.)

Your program is to output the following values:

1. The logical address being translated (the integer value being read from addresses.txt ).

2. The corresponding physical address (what your program translates the logical address to).

3. The signed byte value stored in physical memory at the translated physical address.

We also provide the file correct.txt , which contains the correct output values for the file addresses.txt . You should use this file to determine if your program is correctly translating logical to physical addresses.

## Statistics

After completion, your program is to report the following statistics:

1. Page-fault rate —The percentage of address references that resulted in page faults.

2. TLB hit rate —The percentage of address references that were resolved in the TLB .

Since the logical addresses in addresses.txt were generated randomly and do not reflect any memory access locality, do not expect to have a high TLB hit rate.

**Hints:**

You can try to generate a sequence of logical address with locality, thus a little bit higher hit rate may be expected.

## Page Replacement

Thus far, this project has assumed that physical memory is the same size as the virtual address space. In practice, physical memory is typically much smaller than a virtual address space. This phase of the project now assumes using a smaller physical address space with 128 page frames rather than 256. This change will require modifying your program so that it keeps track of free page frames as well as implementing a page-replacement policy using either FIFO or LRU to resolve page faults when there is no free memory. Don't forget to reserve your previous program as version1 and start this one as another version.

## Submission

Your submission should include (1) the code (a makefile or its equivalent is required), (2) a readme file describing your design and how to compile / use your code, and (3) a report for the following home assignments:

● Your design of the program

● Snapshots of experimental results(statistics) with analysis

● Problems encountered and your solution

● Summarize the different memory management methods listed in our textbook.

● Reference materials

● Your suggestions and comments

## Environment

Linux (Ubuntu 18.04/16.04 is recommended) and C/C++, Java code is also accepted if you like.

# References

Textbook or any other articles you find useful.