Project Two, **Virtual Memory (**from Silberschatz**),** due Saturday, 16 Mar 2019

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 fle containing logical addresses and, using a TLB (Translation look-aside buffer) as well as 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. The goal behind this project is to simulate the steps involved in translating logical to physical addresses.

Specifcs Your program will read a fle 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) 8-bit page offset. Hence, the addresses are structured as shown in Figure 9.33. Other specifcs 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 Section 8.5 of Silberschatz. 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.

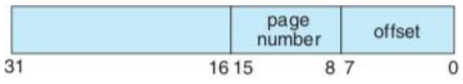

Figure 9.33 Address structure.

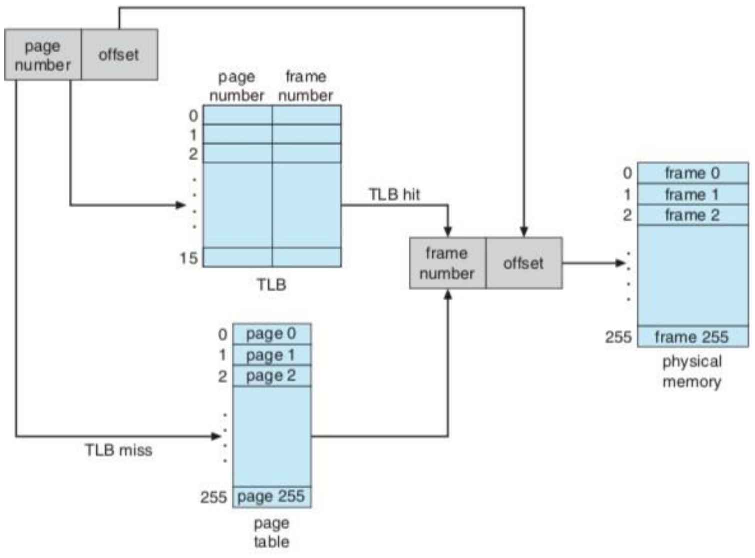

Figure 9.34 A representation of the address-translation process.

Handling Page Faults Your program will implement demand paging as described in Section 9.2 of Silberschatz. The backing store is represented by the fle BACKING STORE.bin, a binary fle of size 65,536 bytes. When a page fault occurs, you will read in a 256-byte page from the fle 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 fle so that you can randomly seek to certain positions of the fle for reading. We suggest using the standard ioo library functions: `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 modifcation 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 fle addresses.txt, which contains integer values represent- ing logical addresses ranging from 0 − 65535 (the size of the virtual address space). Your program will open this fle, 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 (**`getpage(), getoffset()`**, using 's bit operators). Test your new fuctions with the following integers: **1, 256, 32768, 32769, 128, 65534, 33153**

Initially, we suggest that you bypass the TLB and use only a page table. You can implement and integrate the TLB **after** 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 16 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: `./virt_mem addresses.txt`. Your program will read in the fle `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 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 to which your program translates the logical address.
3. The signed byte value stored at the translated physical address.

We also provide the fle **`correct.txt`,** which contains the correct output values for the fle **`addresses.txt`**. You should use this fle 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 refect any memory access locality, do not expect to have a high TLB hit rate.

Modifcations This project assumes 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. A suggested modifcation is to use a smaller physical address space. We recommend using 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 (Section 9.4 of Silberschatz).

Your name and company name: Aatib Abdullah CPSC 351 9:00 AM to 11:45 AM
Repository https://github.com/ aatibabdullah/CPSC-351-project-2-Virtual-Memory---Aatib-Abdullah
Verify each of the following items with a corresponding checkmark. Incorrect items will incur a 5% penalty on the grade.

Complete	Incomplete	CPSC-351: project 2: Virtual Memory
☒❑	❑	Created the Virtual Memory project in C or C++ to translate logical to physical addresses for a virtual address space of 2^{16} (65,536) bytes, using a page table and a TLB, using the specifications in this project assignment.
☒❑	❑	Created a page table of 2^8 entries, with a page size and a frame size of 2^8 bytes, i.e., having a total of 256 frames, and a total physical memory of 65,536 bytes (256 frames times 256 bytes per frame).
☒❑	❑	Program can read logical addresses and translate them to their physical addresses using a page table.
☒❑	❑	Program can use a TLB__and__a page table to translate logical to physical addresses. The page number is extracted, then the TLB is consulted. In the case of TLB hits, extract the frame number from the TLB; for TLB misses, extract it from the page table (see figure).
☒❑	❑	Program implements demand paging, it handles page faults by reading in a 256-byte page from backing store file `backing_store.bin` (a file of 65,536 bytes), and updating the page table and TLB.
☒❑	❑	Uses standard C library functions `fopen(), fread(), fseek(), and fclose().`
☒❑	❑	Your program reads the 1,000 logical addresses (you read that correctly!) from file `addresses.txt`, and translates them correctly to physical addresses using `getpage()` and `getoffset()`. Assertions for correct output against file `correct.txt` all pass.
☒❑	❑	After completion, your program reports the page-fault rate, and the TLB hit rate.
☒❑	❑	Once other elements of your program all work, modify your program to use virtual memory by modifying the physical address space to be smaller (use 128 page frames instead of 256). This will require your program to keep track of free page frames, and to implement the FIFO page-replacement policy (see section 9.4 of Silberschatz).
☒❑	❑	Implement a different virtual memory policy using the LRU page-replacement policy.
❑	❑ ☒	Understand the code structure to the degree that the student could rewrite any section of the code from scratch.
☒❑	❑	Project directory pushed to new GitHub repository listed above using GitHub client.

Your comments (required)

I did the code in a different approach, but It worked. I used fopen() and fclose(), but did not had to use the other two C library functions. Every 5 lines would have a space in between.