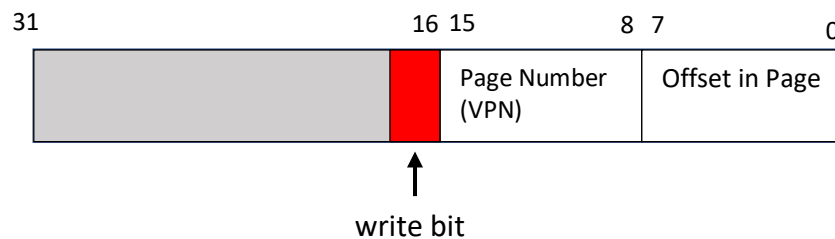


## Virtual Memory Assignment

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 and a write bit. However, you need only be concerned with 16-bit addresses and the write-bit of the 32-bit address, where the rightmost 16 bits represents the 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 ( $256 \text{ frames} \times 256\text{-byte frame size}$ )

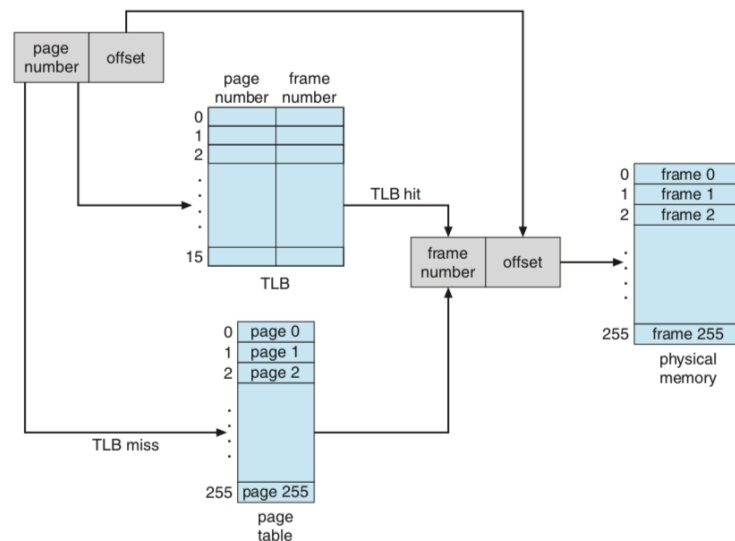
### For part 1

For part one of your assignment your program

- will read logical addresses and translate them to their corresponding physical addresses
- examine the "Write bit" (bit 16), of an address, and if the bit is "1", modify the contents of the address by add "1" to the contents of the memory location referenced by the offset.

## Address Translation

Your program will translate logical to physical addresses using a TLB and page table as outlined in class and in the text. First, the page number is extracted from the logical address (virtual 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 as described in the text and class. The backing store (swap disk) 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. You are expected to use the standard C library functions for performing I/O, including `fopen()`, `fread()`, `fseek()`, and `fclose()`.

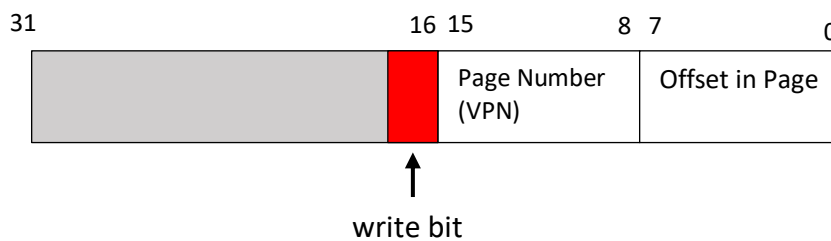
For part one, the size of physical memory is the same as the size of the virtual address space — 65,536 bytes — so you do not need to be concerned about page replacements during a page fault.

## Input Address File

The file `addresses.txt` is provided, which contains integer values representing logical addresses ranging from 0 - 65535 (the size of the virtual address space) and instruction (bit 16). 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 and update the contents if the write-bit is present.

## How to Start

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. Use the FIFO policy for updating your TLB.

## Running 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. Additionally, if the write bite for the

address was set, update the contents of memory after reading its contents. (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 (integer value being read from addresses.txt). print out as a fixed with hex value).
2. The corresponding physical address (what your program translates the logical address to). Print out as a fixed with hex value)
3. The signed byte value stored in physical memory at the translated physical address.
4. If the page is dirty (i.e. the page was written to).

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 by the TLB.
3. Number of dirty pages.

## Part 2

### 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 (Section 10.4) to resolve page faults when there is no free memory. **Note, if you replace a page that has been write to (a dirty page) you must write the dirty page to the backstore (swap disk).**