Lab 7 – Main Memory - Address Translation
Operating Systems CS SH3 Term 2, Winter 2022
Prof. Neerja Mhaskar

Labs that are not scheduled for a Lab Test are not mandatory. These are practice labs, designed to help you on your assignments.
Lab Format: The practice labs will be posted a day before or on the day of the lab on the course website. You can choose to solve it beforehand and come in with your solutions and check the correctness of your solution with your TA.
The TAs will also be available to answer any questions you might have on your assignments.

Solutions to practice labs will not be posted online.

Outline

Assume that a system has a 32-bit virtual address with a 4-KB (=2^{12}) page size. The physical memory address is also a 32-bit address. Consider a small program that needs only 8 pages of memory. Below is the page table for this program.

Page Table:

Frame Number
6
4
3
7
0
1
2
5

In these labs you are to write a C (lab7.c) program that simulates an MMU's (memory management unit's) address translation capability. To simulate a program's memory address requests, we use the text file named **labaddr.txt**. This file can be downloaded from Avenue -> content -> Practice labs -> Lab 7. This file contains a sample of **20** logical addresses generated for this program. You are to read these addresses and output the following for every address:

Logical/virtual address, its corresponding page number and offset, and its corresponding physical address.

There are three parts to this lab:
1. Reading from a file.
2. Given a logical/virtual address output its page number and offset
3. Given the page table and the logical address output its corresponding physical address.

It is recommended that you approach these labs in the above order; that is, first read all the logical addresses from the file and simply output it to the terminal. Then compute the page number and offset and output these details for each logical address in the labaddr.txt file on the terminal. Finally, compute the physical address and display it on the terminal.

Part I – Reading from a file.

1. Use the C library function `openf()` to open the '**labaddr.txt**' file. Since you will be simply reading from this file choose the 'r' (read) option.

Eg: `FILE *fptr = fopen("labaddr.txt", "r");`

2. To read a logical address from the file use the `fgets()` function. Since the logical addresses are no more than 10 characters long, you can read and store just 10 characters at a time. Sample code is:

```
#define BUFFER_SIZE 10;
char buff[BUFFER_SIZE];
//Read from labaddr.txt till you read end of file.
while(fgets(buff, BUFFER_SIZE, fptr) != NULL){...}
```

3. After reading a logical address print it to the terrminal.

4. It is important that you close the file after you are done reading all the logical addresses from the file.
`Sample code: fclose(fptr);`

Part II – Given a logical address compute the page number and offset.

1. Define `PAGE_NUMBER_MASK`, `OFFSET_MARK` etc. as macro definitions. Sample code is below, where you need to fill in the appropriate values in the blanks in your program.

```
#define OFFSET_MASK ____
#define PAGES ____
#define OFFSET_BITS ____
#define PAGE_SIZE ____
```

2. For each logical address compute, the page number and offset using **bitwise operators in C** (See notes on it at the end of the document).

3. Print the logical address and its corresponding page number and offset to the terminal.

Part III – Given the logical address and page table compute the corresponding physical address.

1. Define the page table as an integer array and store all the frame numbers as shown in the page table under the outline section of this document.

```
Eg: int page_table[PAGES] = {6,4,3,7,0,1,2,5};
```

2. After computing the page number (p) and offset (o), extract the frame number for the page (p) from the page table.

3. Using the frame number compute the corresponding physical address **using bitwise operators in C**.

3. Print the physical address along with the logical address and its corresponding page number and offset from PART II to the console.

4. Compile your program without errors and show the program's output to your TA.

Important:

1. Note that for your program to run correctly (and to avoid segmentation faults) it is important that you use correct data types for page number, frame_number, virtual and physical addresses and offset.
2. Refer to lecture notes on paging. This content is under the lecture notes slides on Main memory (Chapter 9).

<u>Correct Program output: **./lab7**</u>

Virtual addr is 19986: Page# = 4 & Offset = 3602. Physical addr = 3602.
Virtual addr is 16916: Page# = 4 & Offset = 532. Physical addr = 532.
Virtual addr is 24493: Page# = 5 & Offset = 4013. Physical addr = 8109.
Virtual addr is 8198: Page# = 2 & Offset = 6. Physical addr = 12294.
Virtual addr is 20683: Page# = 5 & Offset = 203. Physical addr = 4299.
Virtual addr is 18515: Page# = 4 & Offset = 2131. Physical addr = 2131.
Virtual addr is 28781: Page# = 7 & Offset = 109. Physical addr = 20589.

Virtual addr is 24462: Page# = 5 & Offset = 3982. Physical addr = 8078.
Virtual addr is 16399: Page# = 4 & Offset = 15. Physical addr = 15.
Virtual addr is 20815: Page# = 5 & Offset = 335. Physical addr = 4431.
Virtual addr is 18295: Page# = 4 & Offset = 1911. Physical addr = 1911.
Virtual addr is 12218: Page# = 2 & Offset = 4026. Physical addr = 16314.
Virtual addr is 13000: Page# = 3 & Offset = 712. Physical addr = 29384.
Virtual addr is 12229: Page# = 2 & Offset = 4037. Physical addr = 16325.
Virtual addr is 27966: Page# = 6 & Offset = 3390. Physical addr = 11582.
Virtual addr is 24894: Page# = 6 & Offset = 318. Physical addr = 8510.
Virtual addr is 28929: Page# = 7 & Offset = 257. Physical addr = 20737.
Virtual addr is 27865: Page# = 6 & Offset = 3289. Physical addr = 11481.
Virtual addr is 5000: Page# = 1 & Offset = 904. Physical addr = 17288.
Virtual addr is 2315: Page# = 0 & Offset = 2315. Physical addr = 26891.

Notes on Bit wise Operators in C

1. Bitwise operators work on **bits** and perform bit-by-bit operation.
2. Binary AND Operator (**&**) - copies a bit to the result if it exists in **both** operands.
3. Binary OR Operator (**|**) - copies a bit if it exists in **either** operand.
4. Binary Left Shift Operator (**<<**) - The left operands value is moved left by the number of bits specified by the right operand.
    a. For example, 1100 << 2 = 1100**00**
5. Binary Right Shift Operator (**>>**) - The left operands value is moved right by the number of bits specified by the right operand.
    a. For example, 1100 >> 2 = 11
6. Assume you have a 8 bit logical address and page size = 16 bytes = 2^4.
7. Number of bits to represent page number = 4
8. Number of bits to represent page offset = OFFSET_BITS = 4
9. **Page number = logical address >> OFFSET_BITS**
10. OFFSET_MASK = 2^4 – 1 = 15 (in binary it is 00001111)
11. **Page Offset = logical address & OFFSET_MASK**
12. Suppose the page is stored in the frame “frame_number”, then
    Physical Address = (frame_number << OFFSET_BITS) | offset