Designing a Virtual Memory Manager

This project consists of writing a program that translates logical to physical addresses for a virtual address space of size 216 = 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.

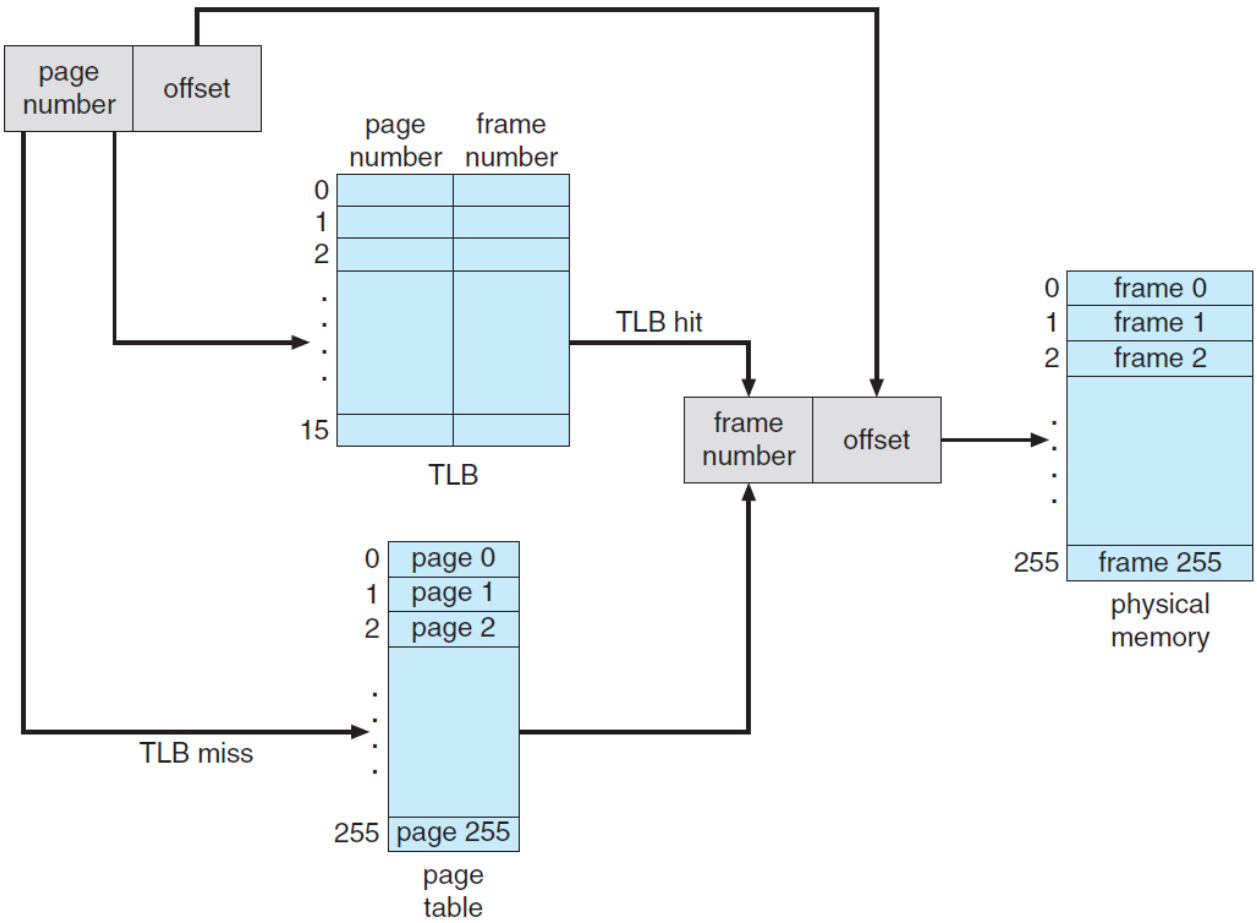
Specific

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.

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 9.3. 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 as described in Section 10.2. 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.

Solution

TLB

1. 从TLB获取物理地址，否则返回-1，代码如下：

```
int TLB_lookup(uint8_t logical_address_page){
    unsigned int i;
    for (i = MAX((TLB_index-TLB_SIZE),0); i < TLB_index; i++){
        struct TLB_Entry_* entry =  &( TLB[i % TLB_SIZE] );

        if (entry->logical_address == logical_address_page){
            return entry-> ;
        }
    }
    return -1;
}
```

2. 用FIFO把逻辑地址加到TLB中，代码如下：

```
void TLB_addition(uint8_t logical_address, uint8_t physical_address){
    struct TLB_Entry_* entry = &( TLB[TLB_index % TLB_SIZE] );
    TLB_index++;
    entry->logical_address = logical_address;
    entry->physical_address = physical_address;
    return;
}
```

3. 加载地址并计算

```
        if (physical_page != -1){
            hits++;
    }
        else{
            physical_page = page_table[logical_page];
            if (physical_page == -1){
                faults++;
                physical_page = unallocated_page;
                unallocated_page++;
                memcpy(

                    memory + physical_page * PAGE_SIZE ,
                    mapped_backing + logical_page * PAGE_SIZE ,
                    PAGE_SIZE
                );

                page_table[logical_page] = physical_page;
            }

            TLB_addition(logical_page, physical_page);
        }
```

运行结果

运行./memory addresses.txt

结果如下：