*Name-Abhi Mehta*
*Reg No.-221080001*

# Experiment-08

**AIM:** Write a C program that is passed a virtual address (in decimal) on the command line and have it output the page number and offset for the given address.

**THEORY:**

**Page-Based Memory Organization:** Within systems employing paging, the primary memory (RAM) undergoes segmentation into fixed-size units known as pages. This division facilitates more efficient memory management by the operating system. The size of each page, typically determined by hardware, commonly adheres to powers of 2 such as 4KB, 8KB, or 16KB.

**Determining Page Number:** When a program accesses memory, it relies on a virtual address comprising both a page number and an offset within the page. To derive the page number from a virtual address, the virtual address undergoes division by the page size. The resulting integer portion serves as the page number within the virtual address space. For instance, if a virtual address is 19986 and the page size is 4KB, the page number is calculated as 4.

**Address Offset:** The offset, denoting the memory location within a page, specifies the position of the memory within that page. For example, an offset of 338 indicates the memory location's position within the page.

**Physical Frames:** Frames, contiguous blocks of physical memory, represent the physical memory space. Unlike pages, frames correspond to physical memory rather than virtual memory. When memory allocation occurs for a process, the operating system assigns physical memory in terms of frames.

**Logical Address :** Generated by the CPU during memory access, a logical address includes both a page number and an offset within the page. This allows programs to reference memory without necessitating knowledge of its physical location.

**Physical Address :** Representing the actual data location in physical memory, a physical address comprises a frame number and an offset within the frame. The translation from a logical address to a physical address is managed by the memory management unit (MMU) in hardware.

**Page Number:** A constituent of the logical address, the page number indicates the virtual memory page where the desired data resides.

**Offset:** Another component of the logical address, the offset specifies the memory location's position within the page. Together with the page number, it determines the precise memory location accessed by the program.

**Logical-to-Physical Address:** The process of converting a logical address to its corresponding physical address, termed address translation or mapping, is conducted by the MMU. The MMU translates the logical address's page number into the corresponding frame number in physical memory. The frame number is then combined with the offset to produce the physical address.

**Code-**

```
abhi@abhi-VivoBook-ASUSLaptop-M3400QA-M3400QA:~/coc$ cat os7.c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define PAGE_SIZE (4 * (1ULL << 10))
#define VIRTUAL_MEMORY_SIZE ((1ULL << 48))

#define NUMBER_OF_PAGES (VIRTUAL_MEMORY_SIZE / PAGE_SIZE) // Number of pages in virtual memory

#define BITS_FOR_PAGE_NUMBER 52
#define BITS_FOR_OFFSET 12
int main(int argc, char *argv[]) {
if (argc != 2) {
printf("Error: Please provide a single argument as the virtual address\n");
return 1;
}

// Converts the command-line argument to an unsigned long integer
unsigned long long VA = strtoull(argv[1], NULL, 10);
// Calculate the page number by dividing the virtual address by the
//page size
unsigned long long Pg_no = VA / PAGE_SIZE;
// Calculate the offset by taking the remainder of the virtual address
//divided by the page size
unsigned long long offset = VA % PAGE_SIZE;
// Prints the virtual address, page number, and offset in binary format

printf("Virtual Address is of size 64 bits\n");
printf("Virtual Address: %llu (binary: ", VA);
for (int i = 47; i >= 0; i--) {
printf("%llu", (VA >> i) & 1);
}
printf(")\n");
printf("Page size is of size %d bits\n", BITS_FOR_PAGE_NUMBER);
printf("Page Number: %llu (binary: ", Pg_no);
for (int i = BITS_FOR_PAGE_NUMBER - 1; i >= 0; i--) {
printf("%llu", (Pg_no >> i) & 1);
}
printf(")\n");
printf("Offset is of size %d bits\n", BITS_FOR_OFFSET);
printf("Offset: %llu (binary: ", offset);
for (int i = BITS_FOR_OFFSET - 1; i >= 0; i--) {
printf("%llu", (offset >> i) & 1);
}
printf(")\n");
printf("Program Ends here!\n");
return 0;
abhi@abhi-VivoBook-ASUSLaptop-M3400QA-M3400QA:~/coc$
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define PAGE_SIZE (4 * (1ULL << 10))
#define VIRTUAL_MEMORY_SIZE ((1ULL << 48))

#define NUMBER_OF_PAGES (VIRTUAL_MEMORY_SIZE / PAGE_SIZE) // Number of pages in virtual memory

#define BITS_FOR_PAGE_NUMBER 52
#define BITS_FOR_OFFSET 12
int main(int argc, char *argv[]) {
if (argc != 2) {
printf("Error: Please provide a single argument as the virtual address\n");
return 1;
}

// Converts the command-line argument to an unsigned long integer
unsigned long long VA = strtoull(argv[1], NULL, 10);
// Calculate the page number by dividing the virtual address by the
//page size
unsigned long long Pg_no = VA / PAGE_SIZE;
// Calculate the offset by taking the remainder of the virtual address
//divided by the page size
unsigned long long offset = VA % PAGE_SIZE;
// Prints the virtual address, page number, and offset in binary format

printf("Virtual Address is of size 64 bits\n");
printf("Virtual Address: %llu (binary: ", VA);
for (int i = 47; i >= 0; i--) {
printf("%llu", (VA >> i) & 1);
}
printf(")\n");
printf("Page size is of size %d bits\n", BITS_FOR_PAGE_NUMBER);
printf("Page Number: %llu (binary: ", Pg_no);
for (int i = BITS_FOR_PAGE_NUMBER - 1; i >= 0; i--) {
printf("%llu", (Pg_no >> i) & 1);
}
printf(")\n");
printf("Offset is of size %d bits\n", BITS_FOR_OFFSET);
printf("Offset: %llu (binary: ", offset);
for (int i = BITS_FOR_OFFSET - 1; i >= 0; i--) {
printf("%llu", (offset >> i) & 1);
}
printf(")\n");
printf("Program Ends here!\n");
return 0;
}
```

**Output-**

```
abhi@abhi-VivoBook-ASUSLaptop-M3400QA-M3400QA:~$ cd coc
abhi@abhi-VivoBook-ASUSLaptop-M3400QA-M3400QA:~/coc$ gcc os7.c -o exec
abhi@abhi-VivoBook-ASUSLaptop-M3400QA-M3400QA:~/coc$ ./exec 19986
Virtual Address is of size 64 bits
Virtual Address: 19986 (binary: 0000000000000000000000000000000000100111000010010)
Page size is of size 52 bits
Page Number: 4 (binary: 0000000000000000000000000000000000000000000000000100)
Offset is of size 12 bits
Offset: 3602 (binary: 111000010010)
Program Ends here!
abhi@abhi-VivoBook-ASUSLaptop-M3400QA-M3400QA:~/coc$
```

```
^C
abhi@abhi-VivoBook-ASUSLaptop-M3400QA-M3400QA:~/coc$ ./exec 24498
Virtual Address is of size 64 bits
Virtual Address: 24498 (binary: 0000000000000000000000000000000101111110110010)
Page size is of size 52 bits
Page Number: 5 (binary: 0000000000000000000000000000000000000000000000000101)
Offset is of size 12 bits
Offset: 4018 (binary: 111110110010)
Program Ends here!
abhi@abhi-VivoBook-ASUSLaptop-M3400QA-M3400QA:~/coc$
```

```
Program Ends here!
abhi@abhi-VivoBook-ASUSLaptop-M3400QA-M3400QA:~/coc$ ./exec 298764
Virtual Address is of size 64 bits
Virtual Address: 298764 (binary: 0000000000000000000000000001001000111100001100)
Page size is of size 52 bits
Page Number: 72 (binary: 0000000000000000000000000000000000000000000001001000)
Offset is of size 12 bits
Offset: 3852 (binary: 111100001100)
Program Ends here!
abhi@abhi-VivoBook-ASUSLaptop-M3400QA-M3400QA:~/coc$
```

```
abhi@abhi-VivoBook-ASUSLaptop-M3400QA-M3400QA:~/coc$ ./exec 345673
Virtual Address is of size 64 bits
Virtual Address: 345673 (binary: 0000000000000000000000000001010100011001001001)
Page size is of size 52 bits
Page Number: 84 (binary: 0000000000000000000000000000000000000000001010100)
Offset is of size 12 bits
Offset: 1609 (binary: 011001001001)
Program Ends here!
abhi@abhi-VivoBook-ASUSLaptop-M3400QA-M3400QA:~/coc$
```

**Conclusion-**

To conclude, the effective management of memory and the seamless translation of addresses stand as crucial pillars in the architecture of modern computer systems, facilitating efficient resource utilization and providing a consistent interface for applications. The exploration of memory organization has been akin to unraveling the layers of a complex puzzle, where each page and frame serves a distinct function within the broader structure. Similarly, delving into address translation has provided insights akin to understanding the intricacies of a sophisticated sorting system, ensuring the smooth flow of data between logical and physical memory spaces. Moreover, the profound impact of optimizing memory management algorithms underscores the significance of meticulous fine-tuning, akin to enhancing the performance of a well-oiled machine through precision adjustments. Finally, navigating the trade-offs inherent in memory management strategies resembles a delicate balancing act, where careful consideration must be given to optimizing efficiency while minimizing potential overhead.