SUBJECT: MICROPROCESSOR LAB (MPL)	
NAME:	
CLASS: SE COMP B	ROLL NO.:
SEMESTER: SEM-II	YEAR: 2023-24
DATE OF PERFORMANCE:	DATE OF SUBMISSION:
EXAMINED:	

Assignment No-01

Title:- Display Accepted Numbers

Assignment Name:- Write an X86/64 ALP to accept five 64 bit Hexadecimal numbers from user and store them in an array and display the accepted numbers.

Objective:-

- To understand the assembly language program
- To understand assembly language programming instruction set
- To understand different assembler directives with example
- To understand macro,procedure concept.

Outcome-

- Students will be able to write ALP code.
- Students will be able to understand difference between procedure and macro .

Prerequisite- System call of Unix for Assembly language Program.
-

Hardware Requirement:-

Requirement- Desktop PC Hardware

Software Requirement:-
Ubuntu14.04,
Assembler: NASM version 2.10.07 Linker:ld

Theory-

Introduction:-Introduction to Assembly Language Programming:

Each personal computer has a microprocessor that manages the computer's arithmetical, logical and control activities. Each family of processors has its own set of instructions for handling various operations like getting input from keyboard, displaying information on screen and performing various other jobs. These set of instructions are called 'machine language instruction'. Processor understands only machine language instructions which are strings of 1s and 0s. However machine language is too obscure and complex for using in software development. So the low level assembly language is designed for a specific family of processors that represents various instructions in symbolic code and a more understandable form. Assembly language is a low-level programming language for a computer, or other programmable device specific to particular computer architecture in contrast to most high-level programming languages, which are generally portable across multiple systems. Assembly language is converted into executable machine code by a utility program referred to as an assembler like NASM, MASM etc.

Advantages of Assembly Language

- An understanding of assembly language provides knowledge of:
- Interface of programs with OS, processor and BIOS;
- Representation of data in memory and other external devices;
- How processor accesses and executes instruction;
- How instructions accesses and process data;
- How a program access external devices.
- Other advantages of using assembly language are:
- It requires less memory and execution time;
- It allows hardware-specific complex jobs in an easier way;
- It is suitable for time-critical jobs;

ALP Step By Step:

Installing NASM:

If you select "Development Tools" while installed Linux, you may NASM installed along with the Linux operating system and you do not need to download and install it separately. For checking whether you already have NASM installed, take the following steps:

- Open a Linux terminal.
- Type ***whereis nasm*** and press ENTER.
- If it is already installed then a line like, *nasm: /usr/bin/nasm* appears. Otherwise, you will see just*nasm:*, then you need to install NASM.

To install NASM take the following steps:
Open Terminal and run below commands:

1. sudo apt-get update
2. sudo apt-get install nasm

Assembly Basic Syntax:
An assembly program can be divided into three sections:

- The **data** section
- The **bss** section
- The **text** section

The order in which these sections fall in your program really isn’t important, but by

convention the
.data section comes first, followed by the .bss section, and then the .text section.

The .data Section
The .data section contains data definitions of initialized data items. Initialized data is data that has a value before the program begins running. These values are part of the executable file. They are loaded into memory when the executable file is loaded into memory for execution. You don’t have to load them with their values, and no machine cycles are used in their creation beyond what it takes to load the program as a whole into memory. The important thing to remember about the
.data section is that the more initialized data items you define, the larger the executable file will be, and the longer it will take to load it from disk into memory when you run it.

The .bss Section
Not all data items need to have values before the program begins running. When you’re reading data from a disk file, for example, you need to have a place for the data to go after it comes in from disk. Data buffers like that are defined in the .bss section of your program. You set aside some number of bytes for a buffer and give the buffer a name, but you don’t say what values are to be present in the buffer. There’s a crucial difference between data items defined in the .data section and data items defined in the .bss section: data items in the .data section add to the size of your executable file. Data items in the .bss section do not.

The .text Section
The actual machine instructions that make up your program go into the .text section. Ordinarily, no data items are defined in .text. The .text section contains symbols called *labels* that identify locations in the program code for jumps and calls, but beyond your instruction mnemonics, that’s about it. All global labels must be declared in the .text section, or the labels cannot be ‘‘seen’’ outside your program by the Linux linker or the Linux loader. Let’s look at the labels issue a little more closely.

Labels
A label is a sort of bookmark, describing a place in the program code and giving it a name that’s easier to remember than a naked memory address. Labels are used to indicate the places where jump instructions should jump to, and they give names to callable assembly language procedures. Here are the most important things to know about labels:

- Labels must begin with a letter, or else with an underscore, period, or question mark. These last three have special meanings to the assembler, so don’t use them until you know how NASM interprets them.
- Labels must be followed by a colon when they are defined. This is basically what tells NASM that the identifier being defined is a label. NASM will punt if no colon is there and will not flag an error, but the colon nails it, and prevents a mistyped instruction mnemonic from being mistaken for a label. Use the colon!
- Labels are case sensitive. So yikes:, Yikes:, and YIKES: are three completely different labels.

Assembly Language Statements

Assembly language programs consist of three types of statements:

- Executable instructions or instructions
- Assembler directives or pseudo-ops
- Macros

Syntax of Assembly Language Statements

```
[label]    mnemonic    [operands]                [;comment]
```

Algorithm:

INPUT: ARRAY OUTPUT: ARRAY

```
STEP 1: Start.
STEP 2: Initialize the data segment.
STEP 3: Display msg1 “Accept array from user. “
STEP 4: Initialize counter to 05 and rbx as 00 STEP 5: Store element in array.
STEP 6: Move rdx by 17.
STEP 7: Add 17 to rbx.
STEP 8: Decrement Counter.
STEP 9: Jump to step 5 until counter value is not zero.
STEP 10: Display msg2.
STEP 11: Initialize counter to 05 and rbx as 00
STEP 12: Display element of array.
STEP 13: Move rdx by 17.
STEP 14: Add 17 to rbx.
STEP 15: Decrement Counter.
STEP 16: Jump to step 11 until counter value is not zero.
STEP 17: Stop
```

CODE AND OUTPUT:

Conclusion:- Hence we implemented an ALP to display accepted Numbers.

Questions:-

1. Explain macro with example?
2. Explain procedure with example?
3. Write difference between procedure and macro?