A LECTURE NOTE

ON

ASSEMBLY LANGUAGE PROGRAMMING (PART 1)

(CSC 303)

COURSE LECTURER:

DR. ONASHOGA S.A (MRS.)

COURSE OUTLINE

SECTION 1

1. **Introduction to Programming languages**
    - Machine Language
    - Low-Level Language
    - High-Level Language
2. **Data Representation & Numbering Systems**
    - Binary Numbering Systems
    - Octal Numbering Systems
    - Decimal Numbering Systems
    - Hexadecimal Numbering Systems
3. **Types of encoding**
    - American Standard Code for Information Interchange (ASCII)
    - Binary Coded Decimal (BCD)
    - Extended Binary Coded Decimal Interchange Code (EBCDIC)
4. **Mode of data representation**
    - Integer Representation
    - Floating Point Representation
5. **Computer instruction set**
    - Reduced Instruction Set Computer (RISC)
    - Complex Instruction Set Computer (CISC)

SECTION TWO

6. **Registers**
    - General Purpose Registers
    - Segment Registers
    - Special Purpose Registers

7. **80x86 instruction sets and Modes of addressing.**
    - Addressing modes with Register operands
    - Addressing modes with constants
    - Addressing modes with memory operands
    - Addressing mode with stack memory
8. **Instruction Sets**
    - The 80x86 instruction sets
    - The control transfer instruction
    - The standard input routines
    - The standard output routines
    - Macros
9. **Assembly Language Programs**
    - An overview of Assembly Language program
    - The linker
    - Examples of common Assemblers
    - A simple Hello World Program using FASM
    - A simple Hello World Program using NASMS
10. **Job Control Language**
    - Introduction
    - Basic syntax of JCL statements
    - Types of JCL statements

- The JOB statement
- The EXEC statement
- The DD statement

CHAPTER ONE

1.0 INTRODUCTION TO PROGRAMMING LANGUAGES

Programmers write instructions in various programming languages, some directly understandable by computers and others requiring intermediate translation steps. Hundreds of computer languages are in use today. These can be divided into three general types:

a. Machine Language
b. Low Level Language
c. High level Language

1.1 MACHINE LANGUAGE

Any computer can directly understand its own machine language. Machine language is the "natural language" of a computer and such is defined by its hardware design. Machine languages generally consist of strings of numbers (ultimately reduced to 1s and 0s) that instruct computers to perform their most elementary operations one at a time. Machine languages are machine dependent (i.e a particular machine language can be used on only one type of computer). Such languages are cumbersome for humans, as illustrated by the following section of an early machine language program that adds overtime pay to base pay and stores the result in gross pay.

+1300042774

+1400593419

+1200274027

Advantages of Machine Language

i. It uses computer storage more efficiently
ii. It takes less time to process in a computer than any other programming language

Disadvantages of Machine Language

i. It is time consuming
ii. It is very tedious to write
iii. It is subject to human error

iv. It is expensive in program preparation and debugging stages

1.2 LOW LEVEL LANGUAGE

Machine Language were simply too slow and tedious for most programmers. Instead of using strings of numbers that computers could directly understand, programmers began using English like abbreviations to represent elementary operations. These abbreviations form the basis of **Low Level Language.** In low level language, instructions are coded using mnemonics. E.g. DIV, ADD, SUB, MOV. Assembly language is an example of a low level language.

An **assembly language** is a low-level language for programming computers. It implements a symbolic representation of the numeric machine codes and other constants needed to program a particular CPU architecture. This representation is usually defined by the hardware manufacturer, and is based on abbreviations (called mnemonics) that help the programmer remember individual instructions, registers, etc. An assembly language is thus specific to a certain physical or virtual computer architecture (as opposed to most high-level languages, which are usually portable).

A utility program called an **assembler** is used to translate assembly language statements into the target computer's machine code. The assembler performs a more or less isomorphic translation (a one-to-one mapping) from mnemonic statements into machine instructions and data. (This is in contrast with high-level languages, in which a single statement generally results in many machine instructions.)

Today, assembly language is used primarily for direct hardware manipulation, access to specialized processor instructions, or to address critical performance issues. The following section of an assembly language program also adds overtime to base pay and stores the result in gross pay:

```
Load basepay
Add overpay
Store grosspay
```

Advantages of Low Level Language

i. It is more efficient than machine language
ii. Symbols make it easier to use than machine language
iii. It may be useful for security reasons

Disadvantages of Low Level Language

i. It is defined for a particular processor
ii. Assemblers are difficult to get
iii. Although, low level language codes are clearer to humans, they are incomprehensible to computers until they are translated to machine language.

1.3 HIGH LEVEL LANGUAGE: Computers usage increased rapidly with the advent of assembly languages, but programmers still had to use many instructions to accomplish even the simplest tasks. To speed up the programming process, **high level language** were developed in which simple statements could be written to accomplish substantial tasks. Translator programs called **compilers** convert high level language programs into machine language. High level language allows programmers to write instructions that look almost like everyday English and contain commonly used mathematical notations. A payroll program written in high level language might contain a statement such as

grossPay=basePay + overTimePay

Advantages of High Level Language

i. Compilers are easy to get
ii. It is easier to use than any other programming language
iii. It is easier to understand compared to any other programming language

Disadvantages of High Level Language

i. It takes more time to process in a computer than any other programming language

CHAPTER TWO

1.0 DATA REPRESENTATION AND NUMBERING SYSTEMS

Most modern computer systems do not represent numeric values using the decimal system. Instead, they use a binary or two's complement numbering system. To understand the limitations of computer arithmetic, one must understand how computers represent numbers.

1.1 THE BINARY NUMBERING SYSTEM

Most modern computer systems (including the IBM PC) operate using binary logic. The computer represents values using two voltage levels (usually 0v and +5v). With two such levels we can represent exactly two different values. These could be any two different values, but by convention we use the values zero and one. These two values, coincidentally, correspond to the two digits used by the binary numbering system. Since there is a correspondence between the logic levels used by the 80x86 and the two digits used in the binary numbering system, it should come as no surprise that the IBM PC employs the binary numbering system.

The binary numbering system works just like the decimal numbering system, with two exceptions: binary only allows the digits 0 and 1 (rather than 0-9), and binary uses powers of two rather than powers of ten. Therefore, it is very easy to convert a binary number to decimal. For each "1" in the binary string, add in 2**n where "n" is the zero-based position of the binary digit. For example, the binary value 11001010 represents:

1*2**7 + 1*2**6 + 0*2**5 + 0*2**4 + 1*2**3 + 0*2**2 + 1*2**1 + 0*2**0
=128 + 64 + 8 + 2
=202 (base 10)

To convert decimal to binary is slightly more difficult. You must find those powers of two which, when added together, produce the decimal result. The easiest method is to work from the a large power of two down to 2**0. Consider the decimal value 1359:

- 2**10=1024, 2**11=2048. So 1024 is the largest power of two less than 1359. Subtract 1024 from 1359 and begin the binary value on the left with a "1" digit. Binary = "1", Decimal result is 1359 - 1024 = 335.

- The next lower power of two (2**9= 512) is greater than the result from above, so add a "0" to the end of the binary string. Binary = "10", Decimal result is still 335.

- The next lower power of two is 256 (2**8). Subtract this from 335 and add a "1" digit to the end of the binary number. Binary = "101", Decimal result is 79.

- 128 (2**7) is greater than 79, so tack a "0" to the end of the binary string. Binary = "1010", Decimal result remains 79.

- The next lower power of two (2**6 = 64) is less than79, so subtract 64 and append a "1" to the end of the binary string. Binary = "10101", Decimal result is 15.

- 15 is less than the next power of two (2**5 = 32) so simply add a "0" to the end of the binary string. Binary = "101010", Decimal result is still 15.

- 16 (2**4) is greater than the remainder so far, so append a "0" to the end of the binary string. Binary = "1010100", Decimal result is 15.

- 2**3(eight) is less than 15, so stick another "1" digit on the end of the binary string. Binary = "10101001", Decimal result is 7.

- 2**2 is less than seven, so subtract four from seven and append another one to the binary string. Binary = "101010011", decimal result is 3.

- 2**1 is less than three, so append a one to the end of the binary string and subtract two from the decimal value. Binary = "1010100111", Decimal result is now 1.

- Finally, the decimal result is one, which is2**0, so add a final "1" to the end of the binary string. The final binary result is "10101001111"

Binary numbers, although they have little importance in high level languages, appear everywhere in assembly language programs

1.8 THE OCTAL NUMBERING SYSTEM

Octal numbers are numbers to base 8. The primary advantage of the octal number system is the ease with which conversion can be made between binary and decimal numbers. Octal is often used as shorthand for binary numbers because of its easy conversion. The octal numbering system is shown below;

Decimal Number	Octal Equivalence
0	001
1	001
2	010
3	011
4	100
5	101
6	110
7	111

1.3 THE DECIMAL NUMBERING SYSTEM

The decimal (base 10) numbering system has been used for so long that people take it for granted. When you see a number like “123”, you don’t think about the value 123, rather, you generate a mental image of how many items this value represents in reality, however, the number 123 represents”

$1*10^2 + 2*10^1 + 3*10^0$ or 100+20+3