ASSEMBLY - BASIC SYNTAX

An assembly program can be divided into three sections:

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

The *data* Section

The **data** section is used for declaring initialized data or constants. This data does not change at runtime. You can declare various constant values, file names or buffer size etc. in this section.

The syntax for declaring data section is:

```
section .data
```

The *bss* Section

The **bss** section is used for declaring variables. The syntax for declaring bss section is:

```
section .bss
```

The *text* section

The **text** section is used for keeping the actual code. This section must begin with the declaration **global _start**, which tells the kernel where the program execution begins.

The syntax for declaring text section is:

```
section .text
   global _start
_start:
```

Comments

Assembly language comment begins with a semicolon (;). It may contain any printable character including blank. It can appear on a line by itself, like:

```
; This program displays a message on screen
```

or, on the same line along with an instruction, like:

```
add eax ,ebx     ; adds ebx to eax
```

Assembly Language Statements

Assembly language programs consist of three types of statements:

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

The **executable instructions** or simply **instructions** tell the processor what to do. Each instruction consists of an **operation code** (opcode). Each executable instruction generates one machine language instruction.

The **assembler directives** or **pseudo-ops** tell the assembler about the various aspects of the assembly process. These are non-executable and do not generate machine language instructions.

Macros are basically a text substitution mechanism.

Syntax of Assembly Language Statements

Assembly language statements are entered one statement per line. Each statement follows the following format:

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

The fields in the square brackets are optional. A basic instruction has two parts, the first one is the name of the instruction (or the mnemonic) which is to be executed, and the second are the operands or the parameters of the command.

Following are some examples of typical assembly language statements:

```
INC COUNT        ; Increment the memory variable COUNT
MOV TOTAL, 48    ; Transfer the value 48 in the
                 ; memory variable TOTAL
ADD AH, BH       ; Add the content of the
                 ; BH register into the AH register
AND MASK1, 128   ; Perform AND operation on the
                 ; variable MASK1 and 128
ADD MARKS, 10    ; Add 10 to the variable MARKS
MOV AL, 10       ; Transfer the value 10 to the AL register
```

The Hello World Program in Assembly

The following assembly language code displays the string 'Hello World' on the screen:

```
section .text
    global _start   ;must be declared for linker (ld)
_start:             ;tells linker entry point
    mov edx,len     ;message length
    mov ecx,msg     ;message to write
    mov ebx,1       ;file descriptor (stdout)
    mov eax,4       ;system call number (sys_write)
    int 0x80        ;call kernel

    mov eax,1       ;system call number (sys_exit)
    int 0x80        ;call kernel

section .data
msg db 'Hello, world!', 0xa  ;our dear string
len equ $ - msg              ;length of our dear string
```

When the above code is compiled and executed, it produces following result:

```
Hello, world!
```

Compiling and Linking an Assembly Program in NASM

Make sure you have set the path of **nasm** and **ld** binaries in your PATH environment variable. Now take the following steps for compiling and linking the above program:

- Type the above code using a text editor and save it as hello.asm.
- Make sure that you are in the same directory as where you saved **hello.asm**.
- To assemble the program, type **nasm -f elf hello.asm**
- If there is any error, you will be prompted about that at this stage. Otherwise an object file of your program named **hello.o** will be created.

- To link the object file and create an executable file named hello, type **ld -m elf_i386 -s -o hello hello.o**
- Execute the program by typing **./hello**

If you have done everything correctly, it will display Hello, world! on the screen.