STATEMENTS

An Assembly Program consists of a set of statements. Statements are the line-by-line components of source files. The two types of statements are:

1.INSTRUCTION Instructions such as MOV & ADD which the Assembler translates to Object Code.

2.DIRECTIVES
Directives tell the Assembler to perform a specific action, such as define a data item etc.

DIRECTIVES
The directives are commands to the assembler, directing it to perform operations other than assembling instructions. The directives are thus executed by the assembler, not assembled by it. They may affect all the operations of the assembler. Assembly Language supports a number of statements that enable to control the way in which a source program assembles and lists. These Statements are called Directives. They act only during the assembly of a program and generate no machine executable code. The most common Directives are PAGE, TITLE, PROC, and END.

Some Assembler Directives are as:

PAGE DIRECTIVE
The PAGE Directive helps to control the format of a listing of an assembled program. It is optional Directive. At the start of program, the PAGE Directive designates the maximum number of lines to list on a page and the maximum number of characters on a line.
Its format is
PAGE [LENGTH], [WIDTH]
Omission of a PAGE Directive causes the assembler to set the default value to PAGE 50,80

TITLE DIRECTIVE
The TITLE Directive is used to define the title of a program to print on line 2 of each page of the program listing. It is also optional Directive.
Its format is TITLE [TEXT]
TITLE "PROGRAM TO PRINT FACTORIAL NO"

SEGMENT DIRECTIVE
The SEGMENT Directive defines the start of a segment. A Stack Segment defines stack storage, a data segment defines data items and a code segment provides executable code. MASM provides simplified Segment Directive. The format (including the leading dot) for the directives that defines the stack, data and code segment are

```
.STACK [SIZE]
.DATA
………………. ; Initialize Data Variables
 .CODE
```

The Default Stack size is 1024 bytes. To use them as above, Memory Model initialization should be carried out.

MEMORY MODEL DEFINTION

The different models tell the assembler how to use segments to provide space and ensure optimum execution speed.
The format of Memory Model Definition is
.MODEL [MODEL NAME]
The Memory Model may be TINY, SMALL, MEDIUM, COMPACT, LARGE and HUGE

TINY :All DATA, CODE & STACK Segment must fit in one Segment of Size <=64K. **SMALL :** One Code Segment of Size <=64K. One Data Segment of Size <=64 K. **MEDIUM:** One Data Segment of Size <=64K. Any Number of Code Segments.
COMPACT: One Code Segment of Size < =64K. Any Number of Data Segments.
LARGE: Any Number of Code and Data Segments.
HUGE: Any Number of Code and Data Segments.

THE PROC DIRECTIVE

The Code Segment contains the executable code for a program, which consists of one or more procedures, defined initially with the PROC Directive and ended with the ENDP Directive. Its Format is given as:
PROCEDURE NAME **PROC**
………………..
……………….
PROCEDURE NAME
ENDP

ENDP DIRECTIVE

The ENDP Directive indicates the end of a procedure. It is used to" bracket a procedure".

END DIRECTIVE

An END Directive ends the entire Program and appears as the last statement. The assembler will ignore any statements after END . Its Format is
END [PROCEDURE NAME]

PROCESSOR DIRECTIVE

MASM supports a set of directives for selecting processors and coprocessors. Once you select a processor, you must use only the instruction set for that processor. The default is the 8086 processor. If you always want your code to run on this processor, you do not need to add any processor directives.To enable a different processor mode and the additional instructions available on that processor, use the directives .186, .286, .386, and .486.Most Assemblers assume that the source program is to run on a basic 8086 level. As a result, when you use instructions or features introduced by later processors, you have to notify the assemblers be means of a processor directive as .286,.386,.486 or.586 This directive may appear before the Code Segment.

THE EQU DIRECTIVE

It is used for redefining symbolic
names EXAMPLE
NUM DB 25
DATA EQU
NUM

THE .STARTUP AND .EXIT DIRECTIVE

MASM 6.0 introduced the .STARTUP and .EXIT Directive to simplify program initialization and Termination. .STARTUP generates the instruction to initialize the Segment Registers. If you do not use .STARTUP, you must give the starting address as an argument to the END directive.
.EXIT generates the INT 21H function 4ch instruction for exiting the Program. .EXIT generates executable code, while END does not.

DEFINING TYPES OF DATA

The Format of Data Definition is given as
[NAME] DV [EXPRESSION] ; DV represents various data types.

EXAMPLES

STRING DB 'kathmandu$'
NO1 DB 5
NO2 DB 78

DEFINITION DIRECTIVE

DB Used to declare BYTE type variable
DW Used to declare WORD type
variable
DD Used to declare DOUBLE WORD type
variable DF Used to declare FAR WORD type
variable
DQ Used to declare QUAD WORD type
variable DT Used to declare TEN BYTES type
variable

Duplication of Constants in a Statement is also possible and is given by

[NAME] DV [REPEATCOUNT[DUP (EXPRESSION)]
EXAMPLES
NUM DB 5 DUP(11) ; 5 Bytes containing hex 0b0b0b0b0b
VAL DW 10 DUP(?) ; 10 Words Uninitialized ‘?’ represents uninitialized space
VALUE DB 3 DUP(5 DUP(4)) ; 44444 44444 44444

1. CHARACTER STRINGS

Character Strings are used for descriptive data which mainly contains ASCII characters.
Consequently DB is the conventional format for defining character data of any length

```
Example DB 'Management'
        DB "Information"
        DB "Star"
```

2. NUMERIC CONSTANTS

BINARY : NUM DB 0111010B
DECIMAL : DATA DB 478
HEXADECIMAL : VAL1
DB 78H

DOS Interrupt function:

-> These interrupts are used to interface with DOS Screen and is compatible with IBM PC.

INT 21h

INT 21h / AH = 1:
-> read characters from standard input, result is stored in AL.
-> If there is no character in the keyboard buffer, the function waits until any key is pressed.

```
-> e.g.: .MOV AH, 1
         .INT 21h
```

a. INT 21h / AH = 2:
   -> write character to standard output entity. DL = character to write, after execution AL = DL.

```
-> e.g.:              .MOV AH, 2
               .MOV DL, ‘a’
               .INT 21h
```

b. INT 21h / AH = 9:
   -> Output of a string at DS:DX.
   -> String must be terminated by ‘$’

```
-> E.g.:              .MOV DX, offset msg   [msg dB “hello$”]
               .MOV AH, 09h
               .INT 21h
```

INT 10h:

a. INT 10h / AH = 2:
   -> Set cursor position
   -> input: DH = row   DL = column
               BH = page number (0 … 7)

b. INT 10h / AH = 06h: Scroll of window

c. INT 10h / AH = 07h: Scroll down window

# Sample Program

Write an ALP to Print Hello World

```
 .MODEL SMALL
.STACK 100h
.DATA
STRING DB 'HELLO WORLD $'
.CODE
   MAIN PROC
MOV AX,@DATA ; @DATA is predefined symbol for the start of the Data Segment
MOV DS,AX ; Initialize the DATA Segment
MOV DX,OFFSET STRING ; Load the Offset Address into DX
MOV AH,09H ; AH=09H For String Display until $
INT 21H ; DOS Interrupt Function
MOV AX,4C00H ; End Request with AH=4CH or AX=4C00H
INT 21H
MAIN ENDP; End Procedure END
MAIN ; End Program
```