INTRODUCTION TO MASM

The Microsoft macro assembler is an x86 high level assembler for DOS and Microsoft windows. It supports wide varieties of macro facilities and structured programming idioms including high level functions for looping and procedures. A program called **assembler** is used to convert the mnemonics of instructions along with the data into the equivalent object code modules, these object code may be further converted into executable code using linker and loader programs. This type of program is called as ASSEMBLY LANGUAGE PROGRAMMING. The assembler converts an Assembly language source file to machine code - the binary equivalent of the assembly language program. In this respect, the assembler reads an ASCII source file from the disk and program as output. The major difference between compilers for a high-level language like PASCAL and an Assembler is that the compiler usually emits several machine instructions for each PASCAL statement. The assembler generally emits a single machine instruction for each assembly language statement. Attempting to write a program in machine language is not particularly bright. This process is very tedious, mistakes, and offers almost no advantages over programming in assembly language. The major disadvantages of programming in assembly language over pure machine code are that you must first assemble and link a program before you can execute it. However, attempting to assemble the code by hand would take far longer than the small amount of time that the assembler takes the perform conversion for you.

An assembler like Microsoft Macro Assembler (MASM) provides a large number of features for assembly language programmers. Although learning about these features takes a fair amount of time, they are so useful that it is well worth the effort. Microsoft MASM version 6.11 contains updated software capable of processing printing instructions. Machine codes and instruction cycle counts are generated by MASM for all instructions on each processor beginning with 8086.

To assemble the file PROG.ASM use this command: (better to use DOS command line)

MASM PROG.ASM

The MASM program will assemble the PROG.ASM file. (To create PROG.OBJ from PROG.ASM)

To create PROG.EXE from PROG.OBJ, use this LINK command:

LINK PROG.OBJ

It converts the contents of PROG.OBJ into PROG.EXE.

To link more than one object file use + signs between their file names as in:

The following is a list of MASM reserved words:

ASSUME-- assume definition
CODE-- begin code segment
DATA-- begin data segment
DB --define byte
DD --define double word
DQ --define quad word
DS --define storage
DUP --duplicate
DW --define word
END --end program
ENDM --end macro

ENDP --end procedure
ENDS --end segment
EQU --equate
FAR --far reference
MACRO-- define macro
NEAR --near reference
OFFSET-- offset
ORQ --origin
PROC-- define procedure
PUBLIC-- public reference
SEGMENT-- define segment

ASSEMBLER DIRECTIVES: The limits are given to the assembler using some predefined alphabetical strings called Assembler Directives which help assembler to correctly understand the assembly language programs to prepare the codes.
DB GROUP EXTRN
DW LABEL TYPE
DQ LENGTH EVEN
DT LOCAL SEGMENT
ASSUME NAME
END OFFSET
ENDP ORG
ENDS PROC
EQU PTR
DB-Define Byte: The DB drive is used to reserve byte of memory locations in the available on memory.
DW-Define Word: The DW drive is used to reserve 16 byte of memory location available on memory.
DQ-Define Quad Word (4 words): The DB directives is used to reserve 8 bytes of memory locations in the memory available.
DT-Define Ten Byte: The DT directive is used to reserve 10 byte of memory locations in the available memory.
ASSUME: Assume local segment name the Assume directive is used to inform the assembler. The name of the logical segments to be assumed for different segment used in programs.
END: End of the program the END directive marks the end of an ALP.
ENDP: End of the procedure.
ENDS: End of the segment.
EQU: The directive is used to assign a label with a variable or symbol. The directive is just to reduce recurrence of the numerical values or constants in the program.
OFFSET: Specifies offset address.
SEGMENT: The segment directive marks the starting of the logical segment.

DOS FUNCTION CALLS

01H	Read Keyboard with Echo
Entry	AH = 01h
Exit	AL = Character read from standard input device
Description:	This function will read a single character from the standard input device. If no character is waiting to be read, it will wait until a character becomes available.

02H	Write to Standard Output Device
Entry	AH = 02h DL = Character to write to standard output device
Exit	None
Description:	This function will write the specified character on the standard output device.

09H	Display A Character String
Entry	AH = 09h DS:DX = Address of the string to print
Exit	None
Description:	This function will write a character string to the standard output device. On entry, DS:DX contains the address of the string. The string must be terminated by a dollar character '$'. All characters up to, but not including, the terminating dollar character will be written to the standard output device.

4CH	Terminate Program
Entry	AH = 4Ch AL = Program status code
Exit	Does not return
Description:	This function is used to terminate execution of a program. The value passed in AL is a status code that will be saved by DOS and can be queried by the parent program using function 4Dh.