Basic Elements of Assembly Language

Assembly language was designed to run in little memory and consists of mainly low-level, simple operations. Then why does it have the reputation of being difficult to learn? After all, how hard can it be to move data between registers and do a calculation? Here's a proof of concept—a simple program in assembly language that adds two numbers and displays the result:

```
main PROC
mov eax,5 ; move 5 to the EAX register
add eax,6 ; add 6 to the EAX register
call WriteInt ; display value in EAX
exit ; quit
main ENDP
```

We simplified things a bit by calling a library subroutine named **WriteInt**, which itself contains a fair amount of code. But in general, assembly language is not hard to learn if you're happy writing short programs that do practically nothing.

Next, we describe basic elements of Microsoft Macro Assembler (MASM) syntax. Knowing these elements will help you to write your first programs in assembly language.

Integer Constants

An integer constant (or integer literal) is made up of an optional leading sign, one or more digits, and an optional suffix character (called a radix) indicating the number's base:

[{+| -}]digits[radix]

Microsoft syntax notation is used throughout this chapter. Elements within square brackets [_] are optional and elements within braces {_} require a choice of one of the enclosed elements (separated by the | character). Elements in *italics* denote items that have known definitions or descriptions.

Radix may be one of the following (uppercase or lowercase):

h	Hexadecimal	r	Encoded real
q/o	Octal	t	Decimal *(alternate)*
d	Decimal	y	Binary *(alternate)*
b	Binary		

If no radix is given, the integer constant is assumed to be decimal. Here are some examples using different radixes:

26	Decimal	42o	Octal
26d	Decimal	1Ah	Hexadecimal
11010011b	Binary	0A3h	Hexadecimal
42q	Octal		

A hexadecimal constant beginning with a letter must have a leading zero to prevent the assembler from interpreting it as an identifier. e.g 0xFFAC

Integer Expressions

An integer expression is a mathematical expression involving integer values and arithmetic operators. The expression must evaluate to an integer, which can be stored in 32 bits (0 through FFFFFFFF). The arithmetic operators are listed in Table according to their precedence order, from highest (1) to lowest (4). The important thing to realize about integer expressions is that they can only be evaluated at assembly time. These are not runtime expressions.

Operator	Name	Precedence Level
( )	Parentheses	1
+, –	Unary plus, minus	2
*, /	Multiply, divide	3
MOD	Modulus	3
+, –	Add, subtract	4

Real Number Constants

Real number constants are represented as decimal reals or encoded (hexadecimal) reals. A decimal real contains an optional sign followed by an integer, a decimal point, an optional integer that expresses a fraction, and an optional exponent:

[sign]integer.[integer][exponent]

Following are examples of valid real number constants:

2.

+3.0

-44.2E+05

26.E5 . At least one digit and a decimal point are required.

Character Constants

A character constant is a single character enclosed in single or double quotes. MASM stores the value in memory as the character's binary ASCII code. Examples are 'A', "d"

String Constants

A string constant is a sequence of characters (including spaces) enclosed in single or double quotes:

'ABC'

'X'

"Good night, Gracie"

'4096'

Embedded quotes are permitted when used in the manner shown by the following examples:

"This isn't a test"

'Say "Good night," Gracie'

Reserved Words

Reserved words have special meaning in MASM and can only be used in their correct context. There are different types of reserved words:

•Instruction mnemonics, such as MOV, ADD, and MUL

•Register names

•Directives, which tell MASM how to assemble programs

•Attributes, which provide size and usage information for variables and operands. Examples are BYTE and WORD

•Operators, used in constant expressions

•Predefined symbols, such as @data, which return constant integer values at assembly time

Identifiers

An identifier is a programmer-chosen name. It might identify a variable, a constant, a procedure, or a code label. Keep the following in mind when creating identifiers:

•They may contain between 1 and 247 characters.

•They are not case sensitive.

•The first character must be a letter (A..Z, a..z), underscore (_), @ , ?, or $. Subsequent characters may also be digits.

•An identifier cannot be the same as an assembler reserved word.

Here are some valid identifiers:

myFile, xVal,_12345, myvar, count123 etc.

Directives

A directive is a command embedded in the source code that is recognized and acted upon by the assembler. Directives do not execute at runtime. Directives can define variables, macros, and procedures. They can assign names to memory segments and perform many other housekeeping tasks related to the assembler. In MASM, directives are case insensitive. For example, it recognizes **.data**, **.DATA**, and **.Data** as equivalent. The following example helps to show the difference between directives and instructions. The DWORD directive tells the assembler to reserve space in the program for a doubleword variable. The MOV instruction, on the other hand, executes at runtime, copying the contents of **myVar** to the EAX register:

myVar DWORD 26 ; DWORD directive

mov eax,myVar ; MOV instruction

Although all assemblers for Intel processors share the same instruction set, they have completely different sets of directives. The Microsoft assembler's REPT directive, for example, is not recognized by some other assemblers.

Defining Segments

One important function of assembler directives is to define program sections,orsegments

. The .DATA directive identifies the area of a program containing variables:

.data

The .CODE directive identifies the area of a program containing executable instructions:

.code

The .STACK directive identifies the area of a program holding the runtime stack, setting its size:

.stack 100h