Fall 2010/2011 – Lecture Notes # 5

Outline of the Lecture

- **Basic Elements of Assembly Language.**
- **Data Types.**
- **Reserved Words, Identifiers and Directives.**

Basic Elements of Assembly Language

Simple program in assembly language:

```
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
```

Data Types

The fundamental data types of the IA (Intel Architecture) are bytes, words, doublewords, and quadwords. A byte is eight bits, a word is 2 bytes (16 bits), a doubleword is 4 bytes (32 bits), and a quadword is 8 bytes (64 bits). The Pentium® III processor introduced a new data type, a 128-bit floating-point numbers data type.

Integer Constants:

```
[{+ | -}] digits [radix]
```

[..] are optional and elements within braces **{..}** require a choice of one of the enclosed elements (separated by the | character)
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		

Examples

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

Integer Expressions

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

Examples

```
4 + 5 * 2                    Multiply, add
12 - 1 MOD 5                 Modulus, subtract
-5 + 2                       Unary minus, add
(4 + 2) * 6                  Add, multiply
```

Real Number Constants

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 the syntax for the sign and exponent:

```
sign {+,-}
exponent E[{+,-}]integer
```

Following are examples of valid real number constants:

```
2.
+3.0
-44.2E+05
26.E5
```

Character Constants

A character constant is a single character enclosed in single or double quotes.

String Constants

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

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**.
- **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.

Directives

A directive is a command embedded in the source code. Directives do not execute at run time, whereas instructions do. Directives can define

- Variables
- Macros
- Procedures
- Memory segments

Directives are case insensitive. It recognizes .**data**, .**DATA**, and .**Data** as equivalent.

Example

The **DWORD** directive tells the assembler to reserve space in the program for a **doubleword** variable. The **MOV** instruction executes at run time, copying the contents of **myVar** to the EAX register:

```
myVar DWORD 26 ; DWORD directive
      mov eax,myVar ; MOV instruction
```

Defining Segments

One important function of assembler directives is to define program sections or segments.

.data

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

.code

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

.stack 100h

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

Data Labels: a data label identifies the location of a variable, providing a convenient way to reference the variable in code. The following, for example, defines a variable named count:

```
count DWORD 100
```

The assembler assigns a numeric address to each label. It is possible to define multiple data items following a label. In the following example, array defines the location of the first number (1024). The other numbers following in memory immediately afterward:

```
array DWORD 1024, 2048
DWORD 4096, 8192
```

Code Labels: a label in the code area of a program (where instructions are located) must end with a colon (:) character. In this context, labels are used as targets of jumping and looping instructions.
For example, the following JMP (jump) instruction transfers control to the location marked by the label named target, creating a loop:

```
target:
mov ax,bx
...
jmp target
```