Current Location Counter $

CURRENT LOCATION COUNTER

The **current location counter (LC),** also known as the assembly pointer (AP), is a special symbol or register, in assembly language that represents the current address in memory where the assembler is writing code. It is denoted by the dollar sign ($).

The LC is used by the assembler to assign addresses to variables, labels, and instructions. It is also used by pseudo-ops, such as .data and .code, to define different sections of the program.

When the assembler starts processing a source file, the **LC is initialized to zero.** As the assembler encounters instructions and directives, it increments the LC by the length of the code or data being generated.

The LC can be used in assembly language code to refer to the current address in memory. For example, the following statement declares a variable named selfPtr and initializes it with its own offset address:

```
selfPtr DWORD $
```

This means that the variable selfPtr will contain the address of the first byte of its own definition.

Using the LC to refer to addresses in memory is a common practice in assembly language programming.

In assembly language, this statement is declaring a variable named selfPtr of type DWORD and initializing it with the special symbol $. Here's an explanation:

1. selfPtr: This is the name of the variable. You can think of it as a label that represents a memory location.

2. DWORD: This specifies the data type of the variable. DWORD stands for Double Word, which typically represents a 32-bit value in memory. It's a common data type for integers in assembly language.

3. $: The dollar sign $ is a special symbol in assembly language. In this context, it represents the address of the first byte of the variable's own definition. So, selfPtr will contain the address of the first byte of itself.

In simpler terms, this line of code is declaring a 32-bit integer variable named selfPtr and initializing it with its own memory address. This can be useful in low-level programming when you need to work with memory addresses directly.

In other words, the LC is a way to keep track of where in memory the assembler is currently writing code. This is useful for initializing variables and labels to their correct addresses.

Here is a simpler explanation:

• Imagine you are writing a book. The LC is like the page number you are currently on. As you write more words, the LC increases.

• You can use the LC to refer to specific words or phrases in your book. For example, you could say "See page 10 for more information."
• In the same way, you can use the LC in assembly language to refer to specific variables or labels. For example, you could say "Load the value of the variable selfPtr into the AL register."

KEYBOARD DEFINITIONS

For example, the following statement defines a label named Esc_key and assigns it the value 27, which is the ASCII code for the Esc key:

```
Esc_key = 27
```

Later in the program, the following instruction can be used to move the value of the Esc key into the AL register:

```
mov al, Esc_key
```

This is a better style than using the integer literal 27 directly, because it makes the code more readable and maintainable.

DUP OPERATOR

The DUP operator is used to create duplicate copies of a data item in memory. The counter used by DUP should be a symbolic constant, which is a constant that is identified by a name.

This makes the program easier to maintain because you can change the value of the symbolic constant in one place, and the program will automatically be updated.

Here is an example of how to use DUP to create an array of 5 DWORDs:

```
COUNT = 5
array dword COUNT DUP(0)
```

You can also use DUP to create strings. For example, the following code will create a string containing the ASCII code for the word "Hello":

```
string db "Hello", 0
```

The 0 at the end of the string is a null terminator, which is required by many string functions.

Symbols defined with the = operator can be redefined within the same program. The following code shows how the assembler evaluates the symbol COUNT as it changes value:

```
COUNT = 5
mov al, COUNT ; AL = 5
COUNT = 10
mov al, COUNT ; AL = 10
COUNT = 100
mov al, COUNT ; AL = 100
```

The changing value of the symbol COUNT has nothing to do with the runtime execution order of the statements. Instead, the symbol changes value according to the assembler's sequential processing of the source code during the assembler's preprocessing stage.

In other words, the assembler will first read the entire source file and evaluate all of the symbols. Then, it will generate the machine code for the program. **This means that the value of a symbol can be changed anywhere in the source file,** and the assembler will still be able to generate the correct machine code.