EQU Directive

The EQU directive associates a symbolic name with an integer expression or some arbitrary text.

• There are three formats:

```
name EQU expression
name EQU symbol
name EQU <text>
```

• When the assembler encounters name later in the program, it substitutes the integer value or text for the symbol.
• EQU can be useful when defining a value that does not evaluate to an integer, such as a real number constant or a text string.
• A symbol defined with EQU cannot be redefined in the same source code file.

Here are some examples of how to use the EQU directive:

```
; Define a symbolic constant for the value of pi.
PI EQU <3.1416>

; Define a symbolic name for a character string.
prompt EQU <"Press any key to continue...",0>

; Define a symbolic constant for the number of cells in a 10-by-10 integer matrix.
matrixSize EQU 10 * 10

; Use the symbolic constant to define a memory location for the matrix.
matrix WORD matrixSize
```

The EQU directive can also be used to create complex expressions that are difficult to calculate by hand. For example, the following code defines a symbolic constant for the address of a memory location:

```
; Define a symbolic constant for the address of the start of the stack.
stackStart EQU _end + 1024
```

The _end symbol is defined by the assembler to point to the end of the program code. The EQU directive then adds 1024 bytes to this address to define the start of the stack.

The EQU directive is a powerful tool that can make assembly language code more readable and maintainable. However, it is important to understand how it works before using it in your own code.

You've correctly highlighted an important distinction between the EQU and = (or ==) directives in assembly language:

1. EQU Directive:

When you use the EQU directive to define a symbolic constant, that constant cannot be redefined within the same source code file.

This restriction ensures that once you've associated a symbol with a specific value or text using EQU, it cannot be inadvertently changed or redefined elsewhere in the code. For example:

```
MY_CONSTANT EQU 42
; ...
MY_CONSTANT EQU 99 ; This is not allowed and will result in an error
```

----

2. = Directive (or == in some assemblers):

In contrast, the = directive (or == in some assemblers) allows you to redefine a symbol with a new value or expression within the same source code file.

This means you can modify the value associated with a symbol at different points in your code. For example:

```
MY_VARIABLE = 10
; ...
MY_VARIABLE = 20 ; This is allowed and changes the value associated with MY_VARIABLE
```

The key difference is that the **EQU directive enforces immutability**, preventing accidental changes to a symbol's value within the same source file, while the = directive allows for redefinition. Depending on your coding requirements, you can choose the directive that suits your needs.
