Declaring Uninitialized Data

The **.DATA? directive** in MASM is used to declare uninitialized data. This means that the data is not given an initial value when it is declared. Instead, the operating system will allocate memory for the data when the program is run.

Using the **.DATA? directive** to declare large blocks of uninitialized data can reduce the size of the compiled program. For example, the following code declares two arrays:

```
.data
smallArray  DWORD   10      DUP(0)          ;40 bytes
.data?
bigArray    DWORD   5000    DUP(?)          ;20000 bytes, not initialized
```

The smallArray array is declared with a size of 10 DWORDs, and each DWORD is 4 bytes in size. This means that the smallArray array will be 40 bytes in size.

The bigArray array is declared with a size of 5000 DWORDs, and each DWORD is 4 bytes in size. This means that the bigArray array will be 20,000 bytes in size.

The smallArray array is initialized to zero, but the bigArray array is not initialized. This means that the operating system will allocate 40 bytes of memory for the smallArray array and 20,000 bytes of memory for the bigArray array when the program is run.

If the bigArray array were declared using the .DATA directive instead of the .DATA? directive, the compiler would allocate 20,000 bytes of memory for the array when the program is compiled. This would make the compiled program 20,000 bytes larger.

Mixing Code and Data

MASM allows you to switch back and forth between code and data in your programs.

This can be useful for declaring variables that are only used within a localized area of a program.

For example, the following code inserts a variable named temp between two code statements:

```
.code
mov eax, ebx
.data
temp DWORD ?
.code
mov temp, eax
```

The declaration of **temp** appears to interrupt the flow of executable instructions, but MASM will place **temp** in the data segment, separate from the segment holding compiled code.

However, intermixing .CODE and .DATA directives can make a program difficult to read. It is generally best to keep code and data separate whenever possible.

Summary

The .DATA? directive can be used to declare uninitialized data. This can reduce the size of the compiled program, especially for large blocks of data.

MASM allows you to switch back and forth between code and data in your programs, but it is generally best to keep code and data separate whenever possible.

It is not necessary to capitalize .code, .data, and .text in MASM. The capitalization is not significant to the assembler.

However, it is common practice to capitalize these directives for readability. This makes it easier to distinguish between code and data in the source code.

Ultimately, it is up to the programmer to decide whether or not to capitalize these directives. There is no hard and fast rule.

Here are some additional tips for writing readable MASM code:

- Use consistent indentation to make the code structure clear.
- Add comments to explain what the code is doing.
- Use labels to make it easy to jump to different parts of the program.
- Break the code up into logical functions and procedures.

By following these tips, you can write MASM code that is easy to read and maintain.

Question: Create an uninitialized data declaration for a 16-bit signed integer.
Question: Create an uninitialized data declaration for an 8-bit unsigned integer.

Question: Create an uninitialized data declaration for an 8-bit signed integer.
Question: Create an uninitialized data declaration for a 64-bit integer.
Question: Which data type can hold a 32-bit signed integer?

```
.data

variable1    SWORD   ?
variable2    BYTE    ?
variable3    SBYTE   ?
variable4    QWORD   ?
```

Data type	Size
BYTE	8 bits
SBYTE	8 bits, signed
WORD	16 bits
SWORD	16 bits, signed
DWORD	32 bits
SDWORD	32 bits, signed
QWORD	64 bits

To declare uninitialized data, you use the .DATA? directive and the appropriate data type. For example, to declare an uninitialized 32-bit signed integer, you would use the following code:

```
.DATA?
variable1 DWORD ?  ; ? indicates that the variable is uninitialized.
```

The ? indicates that the variable is uninitialized. The assembler will allocate memory for the variable when the program is run.

It is important to note that uninitialized data can contain garbage values. It is important to initialize all data before using it.