Macros in ASM

Certainly, I'll simplify and clarify the information about macros in assembly language.
Overview of Macros:

A macro is a named block of assembly language statements. It can be called multiple times in a program.

When you invoke a macro, a copy of its code is inserted directly into the program where it was invoked.

This is known as inline expansion, and there's no actual CALL instruction involved. Placement of Macros:

Macro definitions are usually placed at the beginning of the program's source code or in a separate file included with an INCLUDE directive.

Macros are expanded during the assembler's preprocessing step.

The assembler replaces every invocation of the macro with a copy of the macro's source code. If a program defines a macro but never calls it, the macro code does not appear in the compiled program. Defining Macros:

A macro is defined using the **MACRO and ENDM directives.**

The syntax is as follows:

```
macroname MACRO parameter-1, parameter-2...
statement-list
ENDM
```

Parameters are named placeholders for arguments passed to the macro.

Parameters can be any text, integers, variable names, or other values. The preprocessor treats them as text.

Parameters don't have type information; type checking occurs during assembly.

Example - mPutchar Macro:

The mPutchar macro takes a single parameter named char. It pushes eax, moves the character char into al, and calls the WriteChar procedure.

Finally, it pops eax to restore the original value.

When you use this macro, it replaces the macro invocation with these statements, and the character is displayed on the console.

In essence, macros are a way to create reusable blocks of code that can be inserted directly into your program. They make your code more modular and easier to maintain.

```
mPutchar MACRO char
    push eax       ; Push the value in the eax register
    mov al, char   ; Move the character 'char' into the al register
    call WriteChar ; Call the WriteChar procedure to display the character
    pop eax        ; Pop the original value back into eax
ENDM


;You'd use it like this:


.code
; ... Your code here ...
mPutchar 'A' ; Display the character 'A' on the console
; ... Your code here ...
```

Invoking Macros:

To use a macro, insert its name in your program, and you can provide arguments.

The syntax for invoking a macro is: **macroname argument-1, argument-2, ...** macroname must be a previously defined macro name in your source code.

Each argument replaces a parameter in the macro. The order of arguments must match the order of parameters.

You can pass a different number of arguments than the number of parameters in the macro. If you pass too many arguments, the assembler warns you.

If you pass too few, the unfilled parameters remain empty.

For example, if you have a macro called mPutchar that displays characters on the console, you can invoke it like this: mPutchar 'A'.

The macro call is expanded to the code that displays 'A' on the console.

Debugging Macros:

Debugging programs with macros can be challenging.

Check the listing file (.LST) after assembling to ensure that each macro is expanded as intended.

In the Visual Studio debugger, you can view the disassembly to see how each macro call is expanded into actual code. This can help with debugging.

Additional Macro Features:

You can use the REQ qualifier to specify that a macro parameter is required.

If the macro is invoked without an argument for a required parameter, the assembler shows an error message.

For example, in the mPutchar macro, you can specify that the char parameter is required:

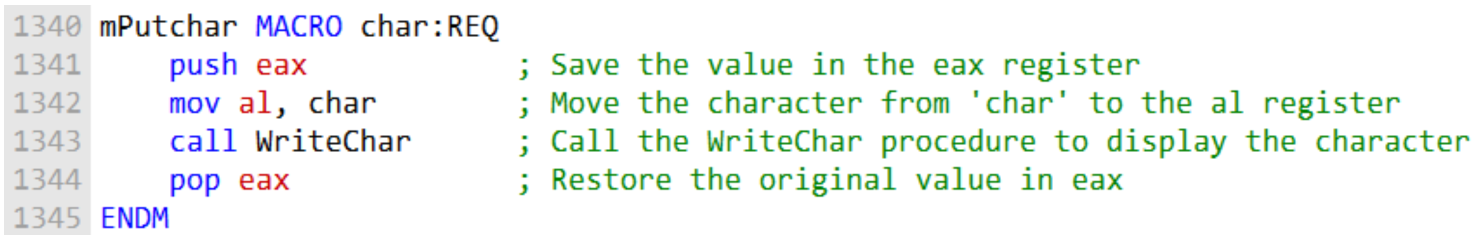
```
mPutchar MACRO char:REQ
```

To exclude comments from appearing in macro expansions, use double semicolons (;;).

This way, comments within a macro definition won't show up when the macro is expanded. For example, you can add comments like this: **;; reminder: char must contain 8 bits** within your macro, and they won't appear in the expanded code.

In summary, invoking macros is like using predefined functions in your program. You provide arguments to the macro, which then replaces them with its predefined instructions.

Debugging macros can be done by checking the listing file and using the debugger. Additionally, you can specify required parameters and control which comments appear in macro expansions.



```
mPutchar MACRO char:REQ
    push eax            ; Save the value in the eax register
    mov al, char        ; Move the character from 'char' to the al register
    call WriteChar      ; Call the WriteChar procedure to display the character
    pop eax             ; Restore the original value in eax
ENDM
```

This macro essentially saves the value in eax, displays the character specified by the char argument, and then restores eax to its original value. Remember that the REQ qualifier indicates that the char parameter is required when invoking the macro.