Macro (1)

• Macro Definition
  – A macro definition is a way to give a name to a piece of text.
  – After a macro has been defined, the programmer can write the macro name instead of the piece of program.

• Basic Parts in Macro Definition
  – A macro header giving the name of the macro being defined
  – The text comprising the body of the macro
  – A pseudo instruction marking the end of the definition (e.g., ENDM).

• Macro Call and Expansion
  – When the assembler encounters a macro definition, it saves it in a macro definition table for subsequent use.
  – From that point on, whenever the name of the macro appears as an opcode, the assembler replaces it by the macro body.
  – The use of a macro name as an opcode is known as a macro call and its replacement by the macro body is called macro expansion.
## Macro (2)

### Assembly language code for interchanging P and Q twice.

**(a) Without a macro.**

<table>
<thead>
<tr>
<th>MOV EAX, P</th>
<th>MOV EBX, Q</th>
<th>MOV Q, EAX</th>
<th>MOV P, EBX</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV EAX, P</td>
<td>MOV EBX, Q</td>
<td>MOV Q, EAX</td>
<td>MOV P, EBX</td>
</tr>
</tbody>
</table>

**(b) With a macro.**

<table>
<thead>
<tr>
<th>MOV EAX, P</th>
<th>MACRO</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV EAX, P</td>
<td>MOV EBX, Q</td>
</tr>
<tr>
<td>MOV Q, EAX</td>
<td>MOV Q, EAX</td>
</tr>
<tr>
<td>MOV P, EBX</td>
<td>MOV P, EBX</td>
</tr>
</tbody>
</table>

(a) *(Without a macro.)*  
(b) *(With a macro.)*
Macro (3)

• Macro vs. Procedure
  – Used to repeat sequences of instruction within a program.
  – Procedure has the disadvantage of requiring a procedure call instruction and a return instruction to be executed every time a sequence is needed. This procedure call overhead may significantly slow the program down.
  – Macros provide an easy and efficient solution to this problem

• Basic Difference
  – Macro calls should not be confused with procedure calls.
  – The basic difference is that a macro call is an instruction to the assembler to replace the macro name with the macro body.
  – A procedure call is a machine instruction that is inserted into the object program and that will later be executed to call the procedure.
Macro (4)

- Macro vs. Procedure

<table>
<thead>
<tr>
<th>Item</th>
<th>Macro call</th>
<th>Procedure call</th>
</tr>
</thead>
<tbody>
<tr>
<td>When is the call made?</td>
<td>During assembly</td>
<td>During program execution</td>
</tr>
<tr>
<td>Is the body inserted into the object program every place the call is made?</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Is a procedure call instruction inserted into the object program and later executed?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Must a return instruction be used after the call is done?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>How many copies of the body appear in the object program?</td>
<td>One per macro call</td>
<td>One</td>
</tr>
</tbody>
</table>
Macro (4)

• Macro Expansion
  – Macro expansion occurs during the assembly process and not during execution of the program. This point is important.
  – Looking only at the machine language program, it is impossible to tell whether or not any macros were involved in its generation.
  – The reason is that once macro expansion has been completed, the macro definitions are discarded by the assembler.

• Assembly Process: Two Pass
  – We can think of the assembly process as taking place in two passes.
  – On pass one, all the macro definitions are saved and the macro calls expanded.
  – On pass two, the resulting text is processed as though it was in the original program.
  – The source program is read in and is then transformed into another program from which all macro definitions have been removed, and in which all macro calls have been replaced by their bodies.
  – The resulting output, an assembly language program containing no macros at all, is then fed into the assembler.
Macros with Parameters

<table>
<thead>
<tr>
<th></th>
<th>MOV EAX,P</th>
<th>MOV EBX,Q</th>
<th>MOV Q,EAX</th>
<th>MOV P,EBX</th>
<th>MOV EAX,R</th>
<th>MOV EBX,S</th>
<th>MOV S,EAX</th>
<th>MOV R,EBX</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>MOV EAX,P</td>
<td>MOV EBX,Q</td>
<td>MOV Q,EAX</td>
<td>MOV P,EBX</td>
<td>MOV EAX,R</td>
<td>MOV EBX,S</td>
<td>MOV S,EAX</td>
<td>MOV R,EBX</td>
</tr>
<tr>
<td>(b)</td>
<td>CHANGE</td>
<td>MACRO P1, P2</td>
<td>MOV EAX,P1</td>
<td>MOV EBX,P2</td>
<td>MOV P2,EAX</td>
<td>MOV P1,EBX</td>
<td>ENDM</td>
<td>CHANGE P, Q</td>
</tr>
</tbody>
</table>

Nearly identical sequences of statements.
(a) Without a macro.  (b) With a macro.

Formal parameters: P1 and P2
Actual parameters: P, Q, R, S
Advanced Features

• Why?
  – These advanced features will make life easier for the assembly language programmer.

• Label duplication
  – Suppose that a macro contains a conditional branch instruction and a label that is branched to. If the macro is called two or more times, the label will be duplicated, causing an assembly error.
  – One solution is to have the programmer supply a different label on each call as a parameter.
  – MASM allows a label to be declared LOCAL, with the assembler automatically generating a different label on each expansion of the macro.

• Macros can call other macros
  – Macros can call other macros, including themselves.
  – If a macro is recursive, that is, it calls itself, it must pass itself a parameter that is changed on each expansion and the macro must test the parameter and terminate the recursion when it reaches a certain value.
  – Otherwise the assembler can be put into an infinite loop. If this happens, the assembler must be killed explicitly by the user.
Macro Facility Implementation

• Two functions
  – To implement a macro facility, an assembler must be able to perform two functions: save macro definitions and expand macro calls.

• Maintaining a Table
  – The assembler must maintain a table of all macro names.
  – Each name has a pointer to its stored definition so that it can be retrieved when needed.
  – Some assemblers have a separate table for macro names and some have a combined opcode table in which all machine instructions, pseudoinstructions, and macro names are kept.

• Encountering a macro definition
  – When a macro definition is encountered, a table entry is made giving the name of the macro, the number of formal parameters, and a pointer to another table: the macro definition table, where the macro body will be kept.
  – A list of the formal parameters is also constructed at this time for use in processing the definition. The macro body is then read and stored in the macro definition table.
  – Formal parameters occurring within the body are indicated by some special symbol.
  – When a macro is called, the assembler temporarily stops reading input from the input device and starts reading from the stored macro body instead.
Exercise

- **True or False**
  - Macro expansion occurs during the execution of the program.
  - An expert on machine language programming may tell whether or not any macros were involved from the corresponding machine code.
  - P and Q are formal parameters of macro
  - FFF