Assembly Language Macros

- Most assemblers include support for macros. The term macro refers to a word that stands for an entire group of instructions.
- Using macros in an assembly program involves two steps:
  1. Defining a macro:
     The definition of a macro consists of three parts: the header, body, and terminator:

     ```
     <label> MACRO The header
     . . . . The body: instructions to be executed
     ENDM The terminator
     ```

  2. Invoking a macro by using its given <label> on a separate line followed by the list of parameters used if any:

     ```
     <label> [parameter list]
     ```
Differences Between Macros and Subroutines

• Both permit a group of instructions to be defined as a single entity with a unique given label or name called up when needed.

• A subroutine is called by the BSR or JSR instructions, while a macro is called by simply using its name.

• Macros are not a substitute for subroutines:
  – Since the macro is substituted with the code which constitutes the body of the macro into the code, very long macros that are used many times in a program will result in an enormous expansion of the code size.
  – In this case, a subroutine would be a better choice, since the code in the body of the subroutine is not inserted into source code many when called.

• Support for subroutines is provided by the CPU --here, the 68000-- as part of the instruction set, while support for macros is part of the assembler (similar to assembler directives).
A Macro Example

Defining the macro:

AddMul MACRO
ADD.B #7,D0 D0 = D0 + 7
AND.W #00FF,D0 Mask D0 to a byte
MULU #12,D0 D0 = D0 x 12
ENDM End of macro def.

Invoking the macro:

MOVE.B X,D0 Get X
AddMul Call the macro

MOVE.B Y,D0 Get Y
AddMul Call the macro
Macros and Parameters

- A macro parameter is designated within the body of the macro by a backslash "\" followed by a single digit or capital letter:

  \1, \2, \3 . . . \A, \B, \C . . . \Z

- Thus, up to 35 different, substitutable arguments may be used in the body of a macro definition.

- The enumerated sequence corresponds to the sequence of parameters passed on invocation.
  
  - The first parameter corresponds to \1 and the 10th parameter corresponds to \A.
  
  - At the time of invocation, these arguments are replaced by the parameters given in the parameter list.
### Macro Example with Parameter Substitution

#### Defining the macro:

<table>
<thead>
<tr>
<th>Function</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddMul</td>
<td>MACRO</td>
<td>Macro definition</td>
</tr>
<tr>
<td>ADD.B</td>
<td>#7,\1</td>
<td>Reg = Reg + 7</td>
</tr>
<tr>
<td>AND.W</td>
<td>#00FF,\1</td>
<td>Mask Reg to a byte</td>
</tr>
<tr>
<td>MULU</td>
<td>#12,\1</td>
<td>Reg = Reg x 12</td>
</tr>
<tr>
<td>ENDM</td>
<td></td>
<td>End of macro def.</td>
</tr>
</tbody>
</table>

#### Invoking the macro:

- MOVE.B X,D0 Get X
- AddMul D0 Call the macro
- . . .
- MOVE.B Y,D1 Get Y
- AddMul D1 Call the macro
Labels Within Macros

• Since a macro may be invoked multiple times within the same program, it is essential that there are no conflicting labels result from the multiple invocation.

• The special designator "\@" is used to request unique labels from the assembler macro preprocessor.

• For each macro invocation, the "\@" designator is replaced by a number unique to that particular invocation.

• The "\@" is appended to the end of a label, and the preprocessor replaces it with a unique number.
Internal Macro Label Example

Macro SUM adds the sequence of integers in the range:  \( i, \ i+1, \ldots, \ n \)

Macro Definition:

```assembly
SUM MACRO \1 = start \ 2 = stop \ 3 = sum
CLR.W \3 sum = 0
ADDQ.W #1,\2 stop = stop +1
SUM1\@ ADD.W \1,\3 For \( i = \) start \ to \ stop
ADD.W #1,\1 sum = sum + i
CMP.W \1,\2
BNE SUM1\@
ENDM
```

Sample macro SUM invocation:

```
SUM D1,D2,D3 D1 = start D2 = stop D3 = sum
```
Macro Example:
ToUpper, A String Conversion Macro

* ToUpper Address-Register
* This macro converts a string from lower case to upper case.
* The argument is an address register. The string MUST be
* terminated with $0
*
ToUpper macro

convert\@  cmpi.b     #0,(\1)      test for end of string
   beq      done\@
   cmpi.b     #'a',(\1)   if < 'a' not lower case
   blt      increment\@
   cmpi.b     #'z',(\1)   if <= 'z' is a lower case
   ble      process\@

increment\@  adda.w     #1,\1
   bra      convert\@

process\@  subi.b     #32,(\1)+ convert to upper case
   bra      convert\@

done\@  NOP

endm End of macro