Most assemblers include support for macros. The term macro refers to a word that stands for an entire group of instructions. Macro is a text substitution facility. It allows programmer to define their own opcodes and also operands:

- `move.w X,d0`
- `muls d0,d0`  `sqr`
- `move.w d0,X`

Inline subroutines:
- Avoids overhead of subroutine calls (jsr, rts)
- Faster than subroutine

Code is generated when macro is actually used. Additional code is generated during each macro call.
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.

• Simpler to write and use (subroutines are more complex, stacks are used)

• Macros are faster than subroutines (no overheads, no saving of return addresses)
Differences Between Macros and Subroutines

• Macros are not a substitute for subroutines:
  – Since the macro is substituted with the code and additional code is generated every time a macro is called, very long macros that are used many times in a program will result in an enormous expansion of the code size
  • Wastage of storage due to multiple copies
    – 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).
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
```

Example: `sqr` macro

```
move X,d0
muls d0,d0
move d0,X
endm
```
Assembly Language Macros

- Using macros in an assembly program involves two steps:

  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]`

    When macro is called it is replaced by the body of the macro

    Parameters – order of parameters is important
**A Macro Example**

### Defining the macro:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddMul MACRO</td>
<td>Macro definition</td>
</tr>
<tr>
<td>ADD.B</td>
<td>D0 = D0 + 7</td>
</tr>
<tr>
<td>AND.W</td>
<td>Mask D0 to a byte</td>
</tr>
<tr>
<td>MULU</td>
<td>D0 = D0 x 12</td>
</tr>
<tr>
<td>ENDM</td>
<td>End of macro def.</td>
</tr>
</tbody>
</table>

### Invoking the macro:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVE.B</td>
<td>Get X</td>
</tr>
<tr>
<td>AddMul</td>
<td>Call the macro</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>MOVE.B</td>
<td>Get Y</td>
</tr>
<tr>
<td>AddMul</td>
<td>Call the macro</td>
</tr>
</tbody>
</table>
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 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.
  
  – If less number of operands than in the body of macro, null string is assigned to the excess operands in body.
## Macro Example with Parameter Substitution

<table>
<thead>
<tr>
<th>Defining the macro:</th>
<th></th>
<th>Invoking the macro:</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddMul MACRO</td>
<td></td>
<td>MOVE.B X,D0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AddMul D0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>. . .</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MOVE.B Y,D1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AddMul D1</td>
</tr>
</tbody>
</table>

**Macro definition**
- **AddMul**: Uses macro with parameter substitution
  - ADD.B #7,\1: Reg = Reg + 7
  - AND.W #00FF,\1: Mask Reg to a byte
  - MULU #12,\1: Reg = Reg x 12
- **ENDM**: End of macro def.

**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
Another Macro Example with Parameter Substitution

Defining the macro:

```
MACRO Add3
  move.l \1, \4
  add.l \2, \4
  add.l \3, \4
ENDM
```

Invoking the macro:

```
Add3 D2,D5,D6,D0
  move.l D2,D0
  add.l D5,D0  \macro expansion\n  add.l D6,D0

Add3 #2,D2,D3,D7
  move.l #2,D7
  add.l D2,D7 \macro expansion\n  add.l D3,D7
```
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.

BusyWait macro
  movem.l d0-d1, -(a7)
outer move.w \1, d1
  move.w #$FFFF, d0
inner dbra d0, inner
dbra d1, outer
  movem.l (a7)+, d0-d1
endm

If macro in invoked more than once, it will lead to multiple declaration of symbols outer and inner
Labels Within Macros

- Multiple invocation problem can be corrected by using two local symbols and two extra parameters

```
BusyWait macro
  movem.l d0-d1, -(a7)
  move.w \1, d1
  move.w #$FFFF, d0
  dbra d0, \2
  dbra d1, \3
  movem.l (a7)+, d0-d1
endm
```

To invoke the macro, a new set of parameters should be provided.

```
BusyWait x, outer1, inner1
BusyWait x, outer2, inner2
BusyWait x, outer3, inner3
```
Labels Within Macros

• Instead of keeping track of the labels generated, 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. It is replaced by .nnn (number of macro expansions that have already occurred).

• The "\@" is appended to the end of a label.
Labels Within Macros

BusyWait macro
  movem.l d0-d1, -(a7)
outer\@ move.w \1, d1
move.w #$FFFF, d0
inner\@ dbra d0, inner\@
   dbra d1, outer\@
   movem.l (a7)+, d0-d1
endm

If macro in invoked more than once:
  - first invocation will replace it with outer.001 and inner.001
  - second invocation will replace it with outer.002 and inner.002
Internal Macro Label Example

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

Macro Definition:

**SUM**  MACRO  
\( \backslash 1 = \text{start} \)  \( \backslash 2 = \text{stop} \)  \( \backslash 3 = \text{sum} \)

CLR.W  \( \backslash 3 \)  \text{sum} = 0

ADDQ.W  \#1,\2 \text{stop} = \text{stop} +1

SUM1@  ADD.W  \1,\3 \text{For i = start to stop}

ADD.W  \#1,\1 \text{sum} = \text{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\@
cmpeq 'a',(\1) if < 'a' not lower case
blt increment\@
cmpeq '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