# Writing new functional blocks for FIBULA

## *A block is an assembly text file*

The user may write new custom blocks that will be recognized by the compiler.
Each block is a separate text file with the extension ".asm" . It contains a macro-definition whose name is same as the file name, without extension.

Example: the file "myfilter.asm" has this form

```
;; Block description
;; …
myfilter    macro name,arg2,arg3,…
            . . . .
            <asm instructions>
            . . . .
            endm
```

In order to be found by the compiler, the file must be stored in the same directory as the main program which is to be compiled. Alternatively, one may store it in FIBULA\USERLIB\ if this block is of some utility for other users. A block in USERLIB with the same name as a reference macro (from LIB or SYSLIB) will have priority, that means that you may modify the behavior or customize a block, simply by storing a modified copy of it inside USERLIB.

## *About names*

Each block has a unique name that may not be a reserved name. (See Help|Reserved names)
Each block must have "*name*" as first symbolic argument. At the time the macro is expanded the user must assign a unique *instance name* by substituting a unique string to *name*. Each symbol defined within the macro must contain this *instance name*. This is done in the following manner:
Suppose you defined the symbol "there" as a destination for a jump, just replace it by

**`name\_there`**

the backslash "\" serves as a symbol concatenation operator, which will allow the assembler to substitute the instance name (1st argument of the macro) to *name.*
When blocks are using other blocks in their definition (multi level nesting), the nested blocks must have names beginning with "name_...". While proceeding this way, it becomes easy to the user to access any variable buried into a block by evoking it's unique name. For example, in a radio receiver application, you may attain the local oscillator frequency by evoking

**`radio_rf_mixer_oscillator_freq`**

Internal variables and constants
Do not attempt to reserve variables / constants with DS / DC directives. Use these special macros instead:

**`var`** **`name[,initial_value]`** *reserve a 24 bit variable in X:*
**`vard`** **`name[,initial_value]`** *reserve a 48 bit (double) variable*
**`varc`** **`name[,initial_value_re,initial_value_im,c]`** *reserve a complex var (cartesian)*
**`varc`** **`name[,initial_value_amplitude,initial_value_phase,p]`** *" " " (polar)*

**`const`** **`name,value`** *constant definition (immediate addressing)*
**`coef`** **`name,value`** *create a 24 bit constant in Y:*
**`coefd`** **`name,value`** *create a 48 bit (double) constant in Y:*
**`coefc`** **`name,value_re,value_im,c]`** *create a complex const (Cartesian)*
**`coefc`** **`name[,value_amplitude,value_phase,p]`** *create a complex const (Polar)*

**`buf`** **`name,field,size`** *reserve a buffer (field = x,y,l,p)*
**`bufm`** **`name,field,size`** *reserve a modulo buffer*

### *Debugging and context saving support*

When you are writing assembly instructions in your macro, you have to precise which registers have been modified. For that you just write at the end of your macro:

```
u     a     saves A2, A1, and A0
u     x     saves X0 and X1
u     r0    saves R0 and M0
```

You can also support step by step debugging at the block level:

```
mymac macro name,…
      dbgdisa           disable stepping inside the block
      ….
      dbgena            enable stepping
      dbgw  MYMAC       stop here and display “MYMAC:”
      dbg   name_in     display interesting variables
      dbg   name

      endm
```