905 MACRO ASSEMBLER (MASIR).

The 905 macro assembler MASIR was developed as a successor to SIR with additional facilities to support programming for the 905 specifically and more generally for all 900 series machines, particularly those with 16K or more stores.

It is an extension of SIR with facilities for macro processing and addressing across store boundaries.

MASIR has two associated programs: the 900 LOADER which takes reads in tapes in the MASIR relocatable binary output format and builds executable programs in store, and MAPLOD a utility for listing the names of program units resident in store. (Note that the MASIR relocatable format is different to that used by SIR. The 900 LOADER cannot read SIR relocatable binary tapes).

The macro facilities of MASIR can be used independently of the assembler facilities, for example to process text in other programming languages, data generators, reports and so forth.

A MASIR program consists of one or more "program units". A program unit consists of a text terminated by a % symbol on a line by itself. A program unit may be headed by the *PROG directive and given a name. Subject to the extensions and restrictions noted in this chapter, the text of a program unit uses the same notation as SIR, described in the previous chapter.

Each program unit of a system is assembled independently and the code generated from this unit must be loaded into one store module, thus code generated plus local workspace cannot exceed 8192 words.

The 900 LOADER maintains a free store pointer for each store module assigned to it (Set LODSET below) and a default module for loading assembled program units. Programs are loaded downwards from the next free location, either in the default module, or where a patch is module patch is used (see below) in the selected module.

Identifiers.

Identifiers may declared in a MASIR program in one of five ways:

1. As macro names (using *DEFINE, *GDEFINE, *ADDMAC)
2. As a program unit name (using *PROG). This declares the name as a global label for the first location of the unit
3. Use in a global label list, as in SIR
4. Use as a label to instruction or data items
5. Use as a label in statements of the form
   LABEL=absolute address

Absolute addresses.

Absolute addresses may be written in the form m^n where m < 8192 and n is a module number (n<16), as in LABEL2=200^1 which sets LABEL2 to 200+8192=8392.

Repeated data.

A special form of skip can be used to set a particular value into a number of consecutive store locations. It is written in the form:

```
>n:data
```

where data is any valid data element. It is assembled as though n occurrences of the data element had been given.

The form >m: NAME data is equivalent to NAME >m: data.

Titles.

Titles are a special kind of comment optionally displayed on the teletype at assembly time. They should be enclosed in double round brackets (n.b., note different convention to ACD SIR), as in

```
((TAPE 2 VERSION 6 18-10-71))
```

Title listing can be suppressed by the *NOTITLE directive and allowed again by the *TITLE directive.

Module patches.

Module patches specify the store module into which a program unit is to be loaded, without fixing it's address within that module. They are written in the form:

^^module number or ^^global label

If a global label is used it must have an absolute address allocated to it before the patched unit is loaded.

Macro facilities.

Macros are defined by a *DEFINE directive of the form

*DEFINE MACRONAME (formal parameter list) [TEXT]

MACRONAME must be separated from *DEFINE by at least one space or newline character. Any characters between MACRONAME and the formal parameter list will be ignored.

The formal parameter list consists of identifiers separated by commas. If there are no parameters to be listed the parenthesis () should be omitted.

The text must be enclosed within square brackets even if null. Further square brackets may form part of the text provided that they are paired. The text must be made up from characters in the SIR internal code.

When the macro is called any occurrence of a formal parameter in the text will be replaced the parameters specified in the actual parameter list.

A macro may also be defined as having its text on a peripheral: when the macro is called, a demand is sent to the appropriate input peripheral and the text may be read in enclosed between [and]. The directive is as follows:

*DEFINE MACRONAME (formal parameter list) [PERIPH n]

where n=1 to read from paper tape or n=3 to read from teleprinter.

A macro may be redefined by using a new *DEFINE directive, however this must not take place within a call of that macro.

Once a macro has been defined, it can be called at any point by writing its name (MACRONAME).

If the macro has formal parameters, then the call must be followed by an actual parameter list. The actual parameters can be arbitrary characters, excluding commas).

The actual parameters may themselves constitute a macro call. In this case the formal parameter will be replace by the text of the macro called.
The macro call is replaced by the formal parameters and *$ being replaced by an incremented numeric value (see Numeric label generation below).

An existing macro may be modified by the use of the *ADDMAC directive. *ADDMAC MACRONAME [TEXT] adds the text to the end of the existing macro text. (If the macro does not exist, it is created; *ADDMAC cannot be applied to a macro whose text comes from a peripheral).

Macro text can contain macro calls and these will be expanded each time the parent macro is called, but recursive macros are not allowed.

*ADDMAC can be used within a macro to add further text to a macro each time it is called, for example:

```
     *DEFINE EXTEND[
     +0
     *ADDMAC EXTEND[
     +1]]
```

will generate on successive calls:

```
     +0

     +0
     +1
```

Macros can be "global" or "local". If *GDEFINE" is used in place of *DEFINE, the declared macro will be global, otherwise local. The *DELETL directive deletes all currently defined local macros: global macros are unaffected.

The *CHANGE directive can be used to substitute alternative characters for use in macro definitions and calls. *CHANGE abcd will substitute a for *, b for , (comma), c for (and d for). The default settings can be restored by using *CHANGE *,(). The special characters can be separated by spaces if desired. (Note: there appears to be a bug in handling substitutions for "(" in actual parameter lists with MASIR Issue 3.)

The space allocated by MASIR for dictionaries and stacks can be controlled using the *SETDIC m n or *SETDIC s m n directive, where s, m and n are integers separated by spaces.

m is the size of the macro dictionary in words
n is the size of the assembler dictionary in words
s is the size of the stack for macro processing.

Assuming a standard stack size of 128 words, if m+n is less than 1600 locations (approx.) both dictionaries will be placed in store module 0 and MASIR will run on an 8K system. If m+n is greater than 1600 locations, but m is less than 1600 locations, the macro dictionary will be placed in module 0 and the assembler dictionary in locations 8192 onwards. If both m+n and m are greater than 1600 locations, the macro dictionary will be placed from 8192 to 8192+m and the assembler dictionary from 8192+m to 892+m+n. Wither dictionary may occupy more than one module, so m or n can be given values greater than 8192.

Numeric label generation.

Each time a macro is called and a pair of characters *$ is contained in the text, the pair is replaced by a set of digits giving an incremental value, starting at 1 and incremented by 1 for each call. This facility is useful for generating unique labels. For example:

```
DEFINE NEWLAB (ROOT) [ROOT*$]
```

Will generate on successive calls:

```
ROOT1, ROOT2, etc
```

(with the actual parameter substituted for ROOT).

Conditional assembly.

The directives *IF and *IFNOT are used to assemble text conditionally.

The *IF directive takes the form

```
*IF NAMEONE=NAMETWO [TEXT]
```

After macro processing NAMEONE and NAMETWO should be reduced to a single identifier, if not an error will occur. If they reduce to the same identifier, the text will be assembled, otherwise the text will be ignored.

The *IFNOT directive takes the form

```
*IFNOT NAMEONE=NAMETWO [TEXT}
```

and is processed as for the *IF directive, except that the text is not assembled if the condition is true.

Conditions can be concatenated using "&", as in:
*IF MODE=TEST & *IFNOT STATE=ONLINE [...]

Function mnemonics.

MASIR includes built in mnemonics as alternatives for function codes. These can be suppresed using the *NOMEM directive and re-enabled using the *MEM directive. (The *NOMEM facility is helpful when including older code that might have used function mnemonics as labels).

MNEMONIC	CODE	FUNCTION
LDB	0	Load B Register
ADD	1	Add to A register
NEG	2	Negate A register and add
STQ	3	Store Q (auxiliary) register
LD	4	Load A Register
ST	5	Store A Register
COL	6	Collate A Register
JZ	7	Jump if Zero
J	8	Jump
JN	9	Jump if negative
INC	10	Increment
STS	11	Store S Register
MUL	12	Multiply
DIV	13	Divide
SH	14	Shift
SHL	14	Shift Left
SHR	14	Shift Right
IPO	15	Input/Output
TER	15 7168	Terminate current program level
ATB	15 7174	A to B Register (905 only)
BTA	15 7175	B to A Register (905 only)
ATQ	15 7172	A to Q Register (905 only)
QTA	15 7173	Q to A Register (905 only)
SKS	15 7169	Skip if standardized (905 only)
SKB	15 7170	Skip if B Register is zero after count (905 only)
RWG	15 7171	Read word generator (905 only)
SRL	15 7176	Set relative addressing (905 only)
SAB	15 7177	Set absolute addressing (905 only)

Additional addressing facilities.

MASIR provides alternative methods for assembling suitable addresses for programs requiring to communicate between different store modules.

```
+LABEL +LABEL+/-increment     Module Relative Address
     put module relative address in current placing position
     (Note: /15 LABEL has the same effect as +LABEL)
:LABEL :LABEL+/-increment     Absolute Address
     put absolute address in current placing position
+LABEL/                       Module Address Relative
     put relative address of the first location of the store
     module containing LABEL in the current placing position
:LABEL/                       Module Address Absolute
     put absolute address of the first location of the store
     module containing LABEL in the current placing position.
```

Each of these forms can be used as a literal, as in

```
     0 +XL
     /8 0
```

MASIR Subroutines.

Subroutine calls between program units should be called using the CALLG macro, which correctly handles calls between store modules.

To call the subroutine at label SUB, simply write CALLG(SUB).

If the called routine is in the same module the following code is generated:

```
     4    +0
     11   SUB
     8    SUB+1
```

whereas if the called routine is in a different module the generated code becomes:

```
     4    +SUB
     11   QMC
     8    QMC+11
```

The module code QMC is duplicated in each module in which program code is stored and comprises:

```
WORD      CONTENTS
 0;       QMC +0    (link)
1-10;     (reserved for FORTRAN use)
11;       5    W    (store relative address)
12;       0    W
13;       6    &760000
14;       2    QMC
15;       /5   0    (store adjusted link)
16;       6    &76000
17;       /8   1    (jump to subroutine entry)
```

The called subroutine should have the normal form:

```
SUB  +0        (link)
     ...       (entry point following link)
     ...
     ...       (body of subroutine)
     ...
     0    SUB  (exit)
     /8   1
```

Note CALLG can only be used on the base priority level (4).

If SUB has parameters these should be set up using the QPARAM macro. The actual parameters may be any number of literals, labels or any valid MASIR address field elements. These are laid down in a manner expected by the 905 FOTRAN compiler (q.v.): each parameter is assembled as /0 n or 0 +LABEL. The second form is used only for global labels not yet located in the current unit. (Note QPARAM is a reserved macro that must not be redefined by the user).

If the subroutine has n formal parameters QPARAM will lay down n words of addresses, one for each actual parameter in order. Exit from the subroutine should jump to the word following these addresses.

For each parameter address, if bit 18 = 1 (i.e., the word is negative) bits 17-1 hold the direct address of the parameter, relative to the current module. If the word is positive, the address is indirect and points to a location of store in the current module holding the actual address of the parameter relative to the calling module.

The use of CALLG, QPARAM and parameter address decoding is illustrated in DEMO6.DAT.

Notes.

There is a *CHECKW nnnnnn directive (where nnnnnn is a set of octal digits). This is for the internal use of the 905 FORTRAN library system and its effects are undefined.

The 903 programming documentation refers to built in MQCHOP and QFPCALL macros as being built-in. These do not appear to be available with the versions of MASIR available in the simulator.

Differences between MASIR and SIR facilities.

1. Obeyed instructions (e.g., '8 1024`) are not available.
2. Options (e.g., *19) are not available: these are replaced by directives.
3. The use of patch facilities (e.g., ^40986, ^LABEL) are restricted: a patch to a global label or absolute address may only occur at the beginning of a program unit.
4. The restore facility ($) is not available.
5. Program unit facility (e.g., *PROG).
6. Addition of title facility (e.g., ((A TITLE)).
7. The use of the instruction /15 NAME is restricted to communication between store modules where NAME is a global name.
8. LABEL=address facility is extended to allow the construction of any absolute address, such as LABEL=500^2 which would associated LABEL with address 500+8192*2 = 16884.
9. New address forms +XL (module relative), :XL (absolute), +XL/ (module relative address), :XL/ (module absolute address)
10. Addition of module patch facility (e.g., ^^2).

11. Addition of repeated data facility (e.g., >5:+2).

12. Addition of macro facilities (*DEFINE etc.).

13. Addition of conditional assembly facility (*IF / *IFNOT).

Store Used.

The MASIR assembler occupies the following store locations in module 0:

16 to 18
128 to approx. 6600
8110 to 8135

The remainder of module 0 up to 8110 is used for dictionaries, unless modified by use of a *SETDIC directive.

The standard version of MASIR uses module 1 for dictionaries. If MASIR is required to run on an 8K machine, the *SETDIC directive must be used as described earlier.

MASIR Operating Instructions.

1. Load MASIR by initial instructions.
2. Run out blanks and ensure input and output select switches are on AUTO. (Use simulator SELECT command).
3. Enter at 16. The program should reply with a _ character.
4. Type in the form On where n is one of:
   0) Normal assembly: read source text and output relocatable binary paper tape.
   1) Check context: as for 0, but no binary is generated.
   2) Macro processing only: the source text is expanded by macro generation and the output text punched.
   3) Macro processing with context check. The expanded source text is punched and checked for MASIR errors, but no relocatable binary is produced.
   4) Output loader halt sequence. This option should be used BEFORE assembly of programs to be loaded in one sequence. It causes a loader halt sequence to be punched which will precede any following relocatable code, and therefore be at the end of the tape when reverse for loading. After punching the halt sequence, input will be processed as for option 0.

5. Load the first tape to be assembled and type M to continue.
6. If a halt code is encountered, type C to allow assembly of a further tape.
7. When % is encountered, the tape will stop. Return to step 4 to assemble further programs.
8. When one or more programs have been assembled, run out blank tape, tear off the tape and rewind BACKWARDS, i.e., the part first punched should be at the end of the tape. (use simulator REWIND command).

The value following O is step 4 is actually an octal number: if bit 4 (&10) is added, the macro dictionary will be preserved between program units.

Labels may be listed to the teleprinter using the *LISTLA directive, and suppressed by the *NOLIST directive. Similarly using titles through the *TITLE and *NOTITLE directives.

Note: there is an additional directive *CHECKW and built in macro QPARAM to support 905 FORTRAN. These are described in the Chapter on 905 FORTRAN.

Error indications.

Unlike SIR, MASIR prints reasonably descriptive error messages so these are not listed here, with the exception of "EUL" which indicates a local identifier has not been declared as a label within the block containing it's use, or as a global label at the head of the block.

For MASIR demonstrations, see the description in the following chapter on the 900 Loader.

900 LOADER.

The 900 Loader is used to read in relocatable binary tapes produced by the MASIR assembler and/or the 905 FORTRAN system.

Store used by Loader.

The Loader uses store locations 15 to 19 and 128 to approx.. 2700. Store above 2700 (approx.) is used for dictionary with 5 locations for every global label and distinct increment value.

Store used by loaded programs.

Programs are stored downwards in each module between the freestore limits set by LODSET (see below).

Subsequent programs are loaded into the highest address position in which there is sufficient space.

Blank COMMON (see 905 FORTRAN) is allocated space from 128 of module 0 upwards and may extend past location 8191 into module 1 and beyond if not already occupied by a program.

Program and named COMMON will be allocated from the highest addressed end of store downwards. Program units must be contained with one store module. COMMON blocks may extend across module boundaries.

The Loader records the highest and lowest addresses of freestore in each module, and adjusts them as program is loaded. If a program unit containing an absolute patch is loaded, the freestore addresses are not adjusted: it is the user's responsibility to lay out store and prevent overwriting).

900 Loader Operating Instructions.

1. Load the tape 900 LOADER by initial instructions.

2. Enter at 16. A _ should be displayed.

3. Type an option in the form letter O followed by an octal number – see below for a list of available options.

4. Load the first relocatable binary tape to be read into the reader and type L to enter the loader.

5. If the tape unloads at the end (i.e., no halt sequence present) the next tape may be loaded and reading continued. If a change of option is required, or no further tapes are to be read re-enter at 16.

6. However _ is output, either return to step 93) to load a new tape, or type M if all tapes have been read.

7. If there are unlocated labels, these are printed out each preceded by *ULW. If no unlocated labels then GO is displayed.

8. If after output of GO loading was direct into store the program may now be started by typing M again. (An octal number may be typed before the M and this number will be held in the A register on entry).

9. If there were unlocated labels, either return to step 3) to load further tapes or type M to run the program despite missing labels (and OV will be displayed to indicate override).

   If output was to paper tape and override is used the output will then be completed. If loading was direct to store, the program may be started again by typing M (for a third time).

The program will be entered at the first location of the first tape loaded (after loader entry with bit 1 = 0), unless one of the program units is MAIN, in which case this will be entered irrespective or the order of loading tapes.

Loader options.

The loader option is input as an octal number made up of a combination of bits as follows:

001	0: initialize loader
	1: do not initialize loader
002	0: everything read is to be loaded

	1: library scan (only program units called by a previously loaded unit are loaded)
004	0: store program in core
	1: output program to paper tape or backing store
010	0: output to backing store if 004 is also set
	1: output to paper tape if 004 is also set
020	0: if program requires use of "built-in" routines, set by previous use of 040
020	1: program does not require built-in routines
040	0: ignore
	1: freeze current dictionary and store layout to build in routines read so far. Built-in programs are not removed when options with bit 1 = 0 are used.
100	0: ignore
	1: list labels
200	0: print FIST LAST messages
	1: suppress FIST LAST messages
400	0: halt after warnings, *CLW, *COM
	1: continue after warnings.

Note: while it is not obvious from the description, the 020 bit SHOULD be set unless a previous run has used the 040 bit to build in routines to the Loader. No routines are built into the standard Loader and not setting the 020 bit leads to the Loader running amok.

Loader error reports.

These are of the form

*LDR eeeeee NAME1 NAME2

Where eeeee is an octal number describing the error, NAME1 is the current program unit and NAME2 is the name of the item which caused the error.

ERROR	CAUSE
00	First code number incorrect – generally tape loaded backwards, or not RLB.
01	Sum check failure.
02	COMMON block not located. (See 905 FORTRAN)
03	Not enough room for program unit.
04	Label declared twice.
05	Store full.
06	COMMON block name same as a subroutine name (se 905 FORTRAN).

07	Program unit larger than *K, or program unit would cross module boundary.
10	Second or subsequent definition of named COMMON block is of greater size than first definition (see 905 FORTRAN).
11	Code number not in dictionary.
12	Unlocated label in patch.
13	Illegal format of RLB.
14	Not enough room for named COMMON block (see 905 FORTRAN).
15	Forward reference table overflow.

All errors cause loading to stop.

Loader warning messages.

*CLW NAME1 NAME2 – check words not equivalent (see 905 FORTRAN).
*COM NAME1 NAME2 – second or subsequent definition of named COMMON block is of smaller size than first definition. First definition is assumed. (Compare error 10).

After a warning message has been output, loading may be continued by typing C.

When M is typed after all units have been loaded, any unlocated labels are printed thus:

*ULW NAME.

Loader setting for various store sizes.

The Loader contains a loader setting routine (LODSET) which allows users to produce new loaders tailored to their own requirements.

The standard tape is preset to 16384 words of store, however if other store sizes are used, LODSET must be used initially. LODSET is overwritten by operation of the Loader, and is not included on tapes punched out by LODSET.

LODSET provides four facilities identified by the letters S, F, D and L which are entered manually by the operator.

S+nnnnnn; is used to set up the total store size available to the loaded programs. It must be a multiple of 8192 less than or equal to the actual store size and may include "unusable" modules.

F +nn; +mmmmmm; is used to indicate the free store in each store module. nn is the module concerned and mmmmm is the address of the highest free location + 1. The store available in the module is assumed to extend from relative address 0 of the module up to but not including the given address.

L+nn; is used to position the Loader itself in module nn.

D after typing D a sum checked binary tape of the Loader is output, with the new settings.

Example of input.

```
S
+24576;
F+2;
+8192;          (0^2 to 8192^2 available)
+1;
+0;             (module 1 unavailable)
+0;
+8130;          (128 to 8129 available)
D
```

MAPLOD (label listing program).

MAPLOD is a program used with the 900 Loader to list program labels and their addresses, unlocated labels and references to them, and the amount of free store still available in each module.

MAPLOD is available in to forms:

1. A version with a start address of 4096^L, where L indicates the store module containing the Loader. This is the version normally used.
2. A version with a start address of 256^L for use when version 1 would overwrite the loader dictionary or loaded program.

ELLIOTT 900 SERIES SIMULATOR

Operating instructions.

1. Using the 900 Loader, load LB tapes as usual.
2. Load MAPLOD, version 1) or 2) as appropriate by initial instructions. It will be stored in the same module as the 900 Loader and will self trigger.
3. At the _ prompt on the teleprinter, type L to list located labels and their addresses, U for unlocated labels and R for references to unlocated labels. A list is free store is displayed at the end of each of the above lists. A * on a line by itself indicates a parity error on teleprinter input.