UNI-ASM

and

MICRO-LINK

A common relocating cross assembler package
and a linking loader
for a wide range of microprocessors
implemented for NORD-10/100 computers

by

Hans Eriksson, Pär Höglund, Anders Rundgren

Department of Computer Technology
Institute of Technology
University of Uppsala
Uppsala, Sweden

Date: 80.10.11

REVISION-A

Göran Lundström
UPNOD

Table of contents:

Abstract

This user's guide presents the UNI-ASM package consisting of a number of relocating cross assemblers and MICRO-LINK, a single linking relocating loader supporting all the assemblers.
Although the assemblers are all derived from the same basic package they incorporate a high degree of compatibility with the microprocessor manufacturers assemblers. All the assemblers behave in a similar manner and emphasis has been put on such things as, ease of use, simple but efficient design and good diagnostic messages.
Macros are not supported in the basic assembler package but will later be introduced with the help of an universal macro pre-processor now under development.
The linking loader is similar to the NRL from ND and this should make the transition a simple task for present ND user's.
Both the assemblers and the linking loader are written in PS-Pascal[1,2] implemented by R. Cailliau and M. Krüger at CERN, Geneva.

Introduction

This package is the result of a long term committment of the Institute of Technology to supply high quality software for use in education and by various research groups in the university community. The work on this project started already in late 1977 by Lars-Gunnar Bäcklin and Göran Lundström[3] who worked out the principles for the first generation of absolute assemblers.

Why use a minicomputer for program development ?

There are several reasons for this, but the ones most often heard of are:

1. Cost
2. Flexibility

Cost is a rather obvious reason with litterally hundreds of students and researchers doing things with microprocessors. Faced with a situation where many people are involved the usual solution, the dedicated development system, becomes almost impossible. The penalty you have to pay when you are using a minicomputer is mainly speed (single user versus timesharing) and in some cases the hardware emulation facilities. The latter is not always true since there now exists a wide range of products[4] on the market which actually are intended to work with a host computer.

Flexibility is maybe not so obvious but the dedicated systems have until very recently not supported several simultaneous user's and large discs, which is nescessary when a large number of people are using the same files or programs. A flexibility factor is also that the very same system can, as in the case of ours, be used for "everything" from simple student programs in assembly language to complex research systems using LISP.
A mainframe is of course also a possibility (if you can afford it), but since the use of a host computer as a development station for microprocessor programs, normally involves very high amount of terminal I/O, because of all downloading of code, at least the "standard", the IBM 370 with its heavy focus on batch is not very pratical. The availability (uptime) of a minicomputer installation is normally also higher than that of a mainframe. But things are changing very rapidly in this field, so today's solution may not be the best tomorrow.

Why have we developed this package ?

It has always been possible to buy cross support software from the chip-manufacturers and others, but the problem has been (and it still is) that they are most often written in FORTRAN for DEC, DG and IBM computers. It is of course possible to convert those products, but in an environment where a lot of very different processors from various manufacturers are used, it vill be rather complicated especially when you want all system products including the assemblers to behave in a uniform manner which is very important when dealing with non professionals (i. e. students). Every time you get a new release, rework of the system has to be done which is very awkward.
Another important factor is that most chip-manufacturers are only interested in selling development systems and therefore their support of cross software is often minimal. Some American software houses[5] have interesting products to offer but in most cases only available in object form (at reasonable cost) and <u>very</u> seldom for <u>ND</u> computers.

Other microprocessor cross support software developed by us for use on NORD computers includes, at the time of writing, two non-relocating cross assemblers (for the RCA 1802 microprocessor and the Intel 8748/8741A single chip microcomputers) and two crosscompilers (MISCAL, a Pascal subset and PL/M, a copy of the Intel product with the same name) both producing code for the Intel 8080/8085 microprocessors.

Some currently running software projects includes: The already mentioned macro pre-processor, relocating cross assemblers for the RCA 1802, Motorola MC6809 and Intel 8086 processors, and a Pascal cross compiler for the Motorola MC68000 microprocessor.

General caracteristics of the assemblers

All the assemblers are of the two pass type written as single programs and none of them uses any extra files for storing intermediate results.

This document is only in conjunction with the referenced manuals specified in the special sections a complete description of the assembly language for a particular processor. That is there are no complete instruction lists included in this manual, consequently the user must have the manuals specified in order to fully utilize the system.

Since our intention was to develope a system using a common base, some compromises had to be done with respect to compatibility with the microprocessor manufacturers assemblers. Especially the wish to use one linker for all assemblers did put restrictions on the relocation mechanisms, which are in fact a subset of the ones offered by Intel corp. for their 8085 processor. This is not as bad as it may look, since the 8085 relocation directives are powerful enough for most user's needs. What is then compatible and what is not ? The table below gives an idea about the degree of compatibility that can be found in this package.

Compatible

Machine instructions (names and syntax)
Constant defining directives (names and syntax)
Space allocation directives (names and syntax)
Delimiters
Labels
Constants
The basic operators (+,-,*,/)
ORG:s and EQU:s
Absolute output format (MICRO-LINK)

Not compatible

UNI-ASM Relocation
Extended operators (in some cases)
Conditional assembly
Assembler options (list control etc.)
Macros (no macros at all!)

Source line format

A source line can consist of:

1. A blank line (a single CR or spaces or tabs followed by a CR).
2. A single comment (see special section for syntax details).
3. A single label in some cases followed by a comment (see special section for syntax details). The label will be assigned the value and type of the current location counter (PLC).
4. A statement (see special section for syntax details).
5. <$><filename> This is a directive to the assembler which enables the user to include files in the source code. The dollar sign must be located to the first position of a source line and no extra characters are allowed between the $ and the filename. The filename should be terminated with a CR. Included files can be nested to a maximum level of three.

A source line can be of any length but only the first 80 characters are significant. Tabs are not expanded which means that if comments containing lots of spaces are to be correctly listed, then when editing with QED, you must use the MTO(0) option.

Parity on files

The output from the assemblers as well as from MICRO-LINK always have the parity bit reset. That means that if files created by those programs are to be read by QED, a MPI(0) must be issued before the read command. The parity bit on input files is ignored.

Object format

The object format is the same for all assemblers because the object is meant to be processed by the MICRO-LINK. The object format is described in the section covering MICRO-LINK. The output from MICRO-LINK is an absolute format which in most cases is compatible with the manufacturers debugger programs (monitors).

How to start up an assembler

Two modes of operation are available:

1. Interactive mode
2. SCRATCH mode

In the interactive mode the assemblers prompts the user for input files (source), list files, object files and options. In the SCRATCH mode the assemblers takes all directives, list file, object file and the source code from the SCRATCH file (file 100). This mode is mainly intended for use with cross compilers and other pre-processors.

Interactive mode (user input is underlined):

```
@XXXXX    All extra parameters except "!" are ignored.

   XXXXX ASSEMBLER  VXX.XX.XX

SOURCE FILE=XXXX   Default type = :SYMB
LIST FILE=XXXX   Default type = :SYMB
OBJECT FILE=XXXX   Default type, see special sections
OPTIONS:  X,X,X
```

Filenames can be created using the standard Sintran conventions and the file types can be overiden in all cases. The object file type is uniqe for every assembler which enables the user to easy manage objects for different processors. If the user do not want a list or object file, then the user should answer with a single CR (carriage return). If no list file is provided, assembly errors will be printed on the terminal.
The available options are:

C: A cross reference and symbol table is produced. Default is no table output. The C option is equvivalent to the LSTXRF directive.

T: The output list is not formatted into pages. Default is a paged output list. The T option is equvivalent to the PAGOFF directive.

N: Only one line is listed for each source line. Default is that all code is listed. The N option is equvivaent to the LST1LN directive.

F: List <total lines><nest level><local line>. Default is that only the total number of source lines read is listed. The F option is equvivalent to the LSTWID directive.

P: After the P option is given the user is asked to specify the wanted number of lines/page. Default is 69 lines/page. The P option is equvivalent to the PAGSIZ directive.

?: Gives the user a menue of the available options and then asks the user again to specify the wanted options.

SCRATCH mode:

@XXXXX !

When a "!" has been found in the command input string the assembler assumes that the SCRATCH file contains the following:

line	Content
1	{<list file name>}
2	{<code file name>}
3	. The source code
4	.
.	.
n	END (last line)

Both files are optional and their type is by default the same as for the interactive mode. The file names must be located to the first position of the line and should be terminated with only a CR. If options are to be used they can only be given as directives in the source code.

User defined symbols

All user defined symbols can be up to 60 characters long and all characters are significant. The allowed characters in a symbol are given in the special section for each processor.

Error messages

All errors are complete messages and not a number only, i. e. the messages should be self-explanatory. Error messages are as few as possible and errors are reported as soon as possible. This means that if an error is flagged in the first pass it will in most cases not be reported in the second pass.

Relocation directives

The assemblers support three sements, two relocatable (CSEG and DSEG) and one absolute (ASEG). The relocatable items can only be 16 or 32-bit values, i. e. no special base page (0-255) is supported. The purpose of the absolute segment is mainly:

1. For systems written as single modules.
2. To supply trap or interrupt vectors.

The other two segments which are relocatable are identical to use with one exception, only the code segment (CSEG) can have a program entry address in it. Although segments can be used in any way the following use is probably the most logical in most microprocessor systems:

CSEG: Program code and constant data.
DSEG: Variable data.

This division reflects the physical hardware of a typical system, i. e. the code is put into non volatile storage (ROM/PROM) and the variables in read/write memory (RAM). The relocation is controlled by the following directives:

NAME <symbol> assigns a name to the module, and if NAME is used it must be the first non comment in the source code. If NAME is not used the module will be assigned the default name "MAIN". The symbol specified in the name directive has no other function after the directive is used. That is the symbol is undefined and may be used as any other symbol including the declaration of the symbol as PUBLIC or EXTRN. The latter means that an entry and a module may have the same name.

PUBLIC <symbol>{,<symbol>}... makes symbols available to other modules. The symbols declared as PUBLIC can only be assigned values using them as labels (i. e. not using EXTRN) and the symbols are said to belong (be relocatable or absolute) to the segment where they are defined, or in the case of SET and EQU directives, to the expression type.
Although the symbols are declared as PUBLIC they can be used in expressions with the same rules as for any other non external symbols.
The PUBLIC directive can appear anywhere in the source code, but the symbol(s) specified in the PUBLIC directive must not be defined in the first pass when the PUBLIC directive is encountered but always in the last (second) pass. That means that the order between the PUBLIC directive and the source line where the PUBLIC declared symbol(s) is actually defined, depends on which pass the symbol(s) is defined in.

EXTRN <symbol>{,<symbol>}... defines symbols as external to this module. The symbols defined as EXTRN can be used in expressions although the allowed arithmetic is limited to adding or subtracting constants (useful for common block in FORTRAN).
The EXTRN declaration must precede any usage of the symbols defined by it.

ASEG
CSEG
DSEG All instructions and data following a ASEG, CSEG or DSEG directive are assembled and made relocatable (CSEG and DSEG) or absolute (ASEG) to the specified segment. The assemblers have three separate location counters, one for each segment which makes it possible to switch segment anytime without the need to save the current location counter for the current segment.

ORG <expression> sets the location counter of the current segment according to the value of the expression.
The result of the expression must be of the same type as the current segment (i. e. it is not valid to use ORG 10 under CSEG since the expression is absolute).
The expression must be completely defined in both passes (i. e. must not contain any non defined symbols), otherwise the statement will be flagged as invalid. Some assemblers have more than one ORG-directive, but the rules stated above are valid for those types too. The ASEG (absolute) mode is the default mode of all the assemblers in this package.
All the location counters (PLC:s) are by default set to zero at the beginning of an assembly.

END {<expression>} terminates the assembly, and this is the only directive which cannot be disabled using the conditional assembly directives.
If the optional expression is supplied it will be output in the object code as a transfer (program entry) address, and the transfer address should be relocatable to the code segment (CSEG).

<symbol> SET <expression>
<symbol> EQU <expression> assigns a value to the symbol specified in the label field.
Besides the value some other qualities of the expression are assigned to the symbol and these are:

1. Expression type (ASEG, CSEG, DSEG or EXTRN)
2. A flag which tells if the expression contains any not yet defined symbols.
3. If the directive is a SET the symbol is marked as variable which means that it may be redefined.

Because this implementation also tranfers the type of the expression, a label can be set equal to an external symbol with an optional displacement added to it. All further use of the assigned symbol will be treated as references to the external symbol in the expression. This will also show up in the cross reference listing.

Syntactic description of the relocation directives

```
{<label>}     NAME    <symbol>  {<comment>}
{<label>}     PUBLIC  <symbol>{,<symbol>}..   {<comment>}
{<label>}     EXTRN   <symbol>{,<symbol>}..   {<comment>}
{<label>}     ASEG    {<comment>}
{<label>}     CSEG    {<comment>}
{<label>}     DSEG    {<comment>}
{<label>}     ORG     <expression>  {<comment>}
{<label>}     END     {<expression>}  {<comment>}  *
<label>       SET     <expression>  {<comment>}
<label>       EQU     <expression>  {<comment>}
```

On directives where labels are optional (i. e. not EQU and SET) a label will be assigned the value and type of the location counter <u>after</u> the directive is in effect.
All delimiters between different items in the source line are the same as for any other source line for the particular assembler with one exception (*). In assemblers which do not have a special character (not only space or tab) to separate comments from operands the END-directive can only have a comment if a transfer address is specified.

List control directives

LSTON enables listing (Default active).

LSTOFF disables all listing except for error messages. Code generation is not affected.

LSTALL will force the assembler to list all source, even the parts of the source which is not generating code because of a previous conditional statement (IF,ELSE or ENDIF). LSTALL is active by default.

LSTCND is the opposite of LSTALL (i. e. lists only used source).

LSTACD will expand the listing of the output code to more than one line if needed (i. e. long ASCII strings will generate several lines of listing because of all code they generate).
LSTACD is by default active.

LST1LN will only list one line of code.

PAGON formats the assembly output list into pages.
This directive is active by default.

PAGOFF suppresses the paging of the assembly output list.

PAGSIZ <expression> sets the number of lines per page in the assembly output list according to the expression.
The number of lines per page is default set to 69.
The expression must be absolute and in the range of 10-150.

PAGE will generate a new page in the output list if paging is active.

TITLE 'header' fills the page header whith the string in the operand field.
The header string cannot exceed 60 characters (including the "'").

PTITLE 'header' is equivalent to TITLE+PAGE.

LSTNAR formats the assembly output list in the following way (default active):

```
<TL><AF><CF><SF>.
```

LSTWID formats the assembly output list in the following way:

```
<TL><NL><LL><AF><CF><SF>

      <TL> ::= Total number of lines of source read
      <NL> ::= Nesting levels of files
      <LL> ::= Local linenumber in current file
      <AF> ::= Address field
      <CF> ::= Code field
      <SF> ::= Source field
```

LSTXRF will generate a cross reference table at the end of the assembly output list. The table displays besides values and linenumbers also the type of the symbol. The types are:

Xhh	External with relative declaration number (in hex) = hh.
A	Local symbol in the absolute segment.
C	Local symbol in the code segment.
D	Local symbol in the data segment.
AE	Entry in the absolute segment.
CE	Entry in the code segment.
DE	Entry in the data segment.
V	Variable type (SET has been used).
U	Undefined symbol.

Syntactic description of the list control directives

```
{<label>}    LSTON    {<comment>}
{<label>}    LSTOFF   {<comment>}
{<label>}    LSTALL   {<comment>}
{<label>}    LSTCND   {<comment>}
{<label>}    LSTACD   {<comment>}
{<label>}    LSTILN   {<comment>}
{<label>}    PAGON    {<comment>}
{<label>}    PAGOFF   {<comment>}
{<label>}    PAGSIZ   <expression>  {<comment>}
{<label>}    PAGE     {<comment>}
{<label>}    TITLE    <'header'>  {<comment>}
{<label>}    PTITLE   <'header'>  {<comment>}
{<label>}    LSTNAR   {<comment>}
{<label>}    LSTWID   {<comment>}
{<label>}    LSTXRF   {<comment>}
```

All delimiters between items follows the same rules for the list control directives as for any other source line for the particular assembler used.
The optional label will be assigned the value and type of the current location counter.

Conditional assembly

The conditional assembly facilities makes it possible to control the assembly process at assembly time using the directives IF, ELSE and ENDIF.
The effect of an IF directive is that if the condition is not true, the code that follows will not generate any code (i. e. it will not be assembled) until an ENDIF or ELSE directive is found. Otherwise the IF directive has no effect.
The conditional assembly directives are:

```
{<label>}   IF  <expression>  {<comment>}
{<label>}   ELSE   {<comment>}
{<label>}   ENDIF  {<comment>}
```

The assembler evaluates the expression (must be absolute) in the operand field of the IF directive. If the expression is equal to zero it will be considered false and the following statements will not be assembled.
All assembler directives with the exception of END may be disabled with the conditional directives.
An IF directive must be terminated with an ENDIF directive.
The ELSE directive is optional and if used it must be inside of a block IF-ENDIF.
The optional label will be assigned the current value and type of the location counter only if the assembly is not disabled after the directive is in effect, otherwise the label will not be defined.
All delimiters follows the same rules for the conditional assembly directives as for any other source line for the particular assembler used.
The IF-ENDIF and IF-ELSE-ENDIF blocks may be nested to any level.

Example 1. Simple IF-ENDIF Block:

```
IF  OPTION=3
.
.         Assembled if OPTION=3
.
ENDIF
```

Example 2. IF-ELSE-ENDIF Block:

```
IF OPTION=3
.
.         Assembled if OPTION=3
.
ELSE
.
.         Assembled if OPTION<>3
.
ENDIF
```

Expressions and operators

An expression consists of operands and operators. The possible operands in an expression are:

1. User defined symbols (labels).
2. Constants (numeric and ASCII).
3. The location counter (PLC) symbol.

In order to keep a high degree of compatibility with the microprocessor manufacturers assemblers, the syntax of operands and operators is <u>not</u> equivalent for the different assemblers in this package, and the user is advised to read the special sections covering the particular assembler. This assembler package will accept a wide range of expressions involving arithmetic and logical operations. All arithmetic and logical operations are using a <u>32-bit</u> integer format and rangechecking is <u>only</u> done when an expression is used to generate code.
Expressions will be evaluated from left to right but the order depends on which operator used.
The available operators are:

OPERATOR	FUNCTION	ORDER
-	Unary minus	1
NOT	Logical NOT	1
LOW	Lowbyte of	1
HIGH	Highbyte of	1
*	Multiplication	3
/	Division	3
+	Addition	4
-	Subtraction	4
MOD	Modulo	3
SHR	Logical shift right	3
SHL	Logical shift left	3
AND	Logical AND	5
OR	Logical OR	6
XOR	Logical XOR	6
EQ or =	Equals	7
NE or <>	Not equals	7
GE or >=	Greater or equal	7
LE or <=	Less or equal	7
GT or >	Greater than	7
LT or <	Less than	7
UGT	Unsigned greater than	7
ULT	Unsigned less than	7

The operators can be divided into aritmetic, logical and shift operators. All operators except LOW, HIGH, NOT and unary minus are binary (i. e. need operands on both sides of the operator). Parantheses can be used to overide the order of operators.

The modulo (MOD) operator is defined as:

```
   X MOD Y   =   X-Y*(X/Y)
   (using integer division)
```

The shift operators (SHL and SHR) shifts the first argument right or left by the number of positions given in the second argument. Zeros are shifted into the high or low order bits, respectively. An example:

```
    45 SHR 2    will generate a result equal to 11
```

The 8 comparison operators (EQ, NE, GE, LE, GT, LT, UGT, ULT) will evaluate to a logical True (all ones) if the comparison is true, else the result will be a logical false (all zeros).

The operators LT, GT, GE and LE compares signed arguments whereas UGT and ULT assume unsigned arguments.

Arithmetic operations and relocation

Besides a value an expression also have a type associated to it. The possible types are:

Absolute
Relocatable (CSEG or DSEG)
External

Not all operations are possible to use with relocatable and external arguments.
The allowed operations on external symbols are:

```
<external symbol>+<absolute expression>
<external symbol>-<absolute expression>
```

The result of one of these operations on an external symbol is a value (offset) with the type external.

The allowed operations on relocatable symbols and the resulting types are:

OPERATION	X(rel),Y(rel)	X(rel),Y(abs)	X(abs),Y(rel)
X + Y	invalid	relocatable	relocatable
X - Y	absolute	relocatable	invalid

When two relocatable arguments are involved they <u>must</u> be relocatable to the <u>same segment</u>.

Reference manuals:

1. 8080/8085 Assembly language programming manual.
   Order Number: 9800301C
2. MCS-85 User's Manual.
   Order Number: 9800366D

Both of these manuals can be obtained from the manufacturer, Intel corporation (Santa Clara, Calif.) and their local distributors.

Delimiters

The format in the 8085 assembly language is a free format, i. e. labels and instructions can be located anywhere in a source line. The items in a source line must be separated with delimiters and in the 8085 assembly language the following delimiters are valid:

For a source line with only a label:

1. The label may be preceded with tabs or spaces.
2. The label should be terminated with a colon or only a CR.
3. If a comment is used it must be separated from the label with a semicolon.

For a comment only source line:

1. Spaces or tabs may precede the comment.
2. A semicolon must precede the comment string.

For an ordinary statement:

1. Tabs or spaces may precede the label or instruction.
2. If a label is used it must be separated from the instruction with a colon, spaces or tabs.
3. Instructions and operands must be separated with tabs or spaces.
4. Operands must be separated with commas only.
5. Comments must be separated from instructions or operands with a semicolon.

Label syntax

The first character of a label must be alphabetic (A-Z) or the special characters "?", "_" or "@". The following characters (if any) can be any of the already mentioned characters or the decimal digits (0-9).
All instructions, registers and operators are reserved words and cannot be used as labels.

Constants

This implementation conforms to the Intel assembler manual which means:

'A'	= ASCII constant
'AB'	= ASCII constant
78	= Decimal constant
56H	= Hexadecimal constant
10110B	= Binary constant
457Q	= Octal constant

Expressions

The available operators are the same as the ones described in the general section of this user's guide. A difference compared to Intel products is that registers and instructions are not allowed in expressions. Spaces or tabs are ignored in expressions.

Pseudo directives

Besides all the directives listed in the general section the DS, DB and DW directives are also implemented. They are identical to the Intel definition.

Object file type

The assembler object file type is by default equal to :R80.

Reference manual:

1. Z80 Assembly language programming manual.
   REL. 2.1 3.0 d.s (1977)

This manual can be obtained from the manufacturer, Zilog inc. (Cupertino, Calif.) and their local distributors.

Delimiters

The items in a source line must be separated by delimiters and in the Z80 assembly language the following delimiters are valid:

For a source line with only a label:

1. The label must begin in the first position.
2. The label should be terminated with a colon or only a CR.
3. If a comment is used it must be separated from the label with a semicolon.

For a comment only source line:

2. A semicolon in the first position must precede the comment string.

For an ordinary statement:

1. Instructions cannot begin in the first position.
2. If a label is used it must be separated from the instruction with a colon, spaces or tabs. If there is no label the instruction should be preceded with tabs or spaces.
3. Instructions and operands must be separated with tabs or spaces.
4. Operands must be separated with commas only.
5. Comments must always be separated from instructions or operands with a semicolon.

Label syntax

The first character of a label must be alphabetic (A-Z, a-z) or the special characters "?", "@" or "_". The following characters (if any) can be any of the already mentioned characters or the decimal digits (0-9).
All registers and flags are reserved words and cannot be used as labels.

Object file type

The assembler object file type is by default equal to :RZ80.

Constants

This implementation conforms to the Zilog assembler manual but one extension have also been included:

'A'	= ASCII constant
'AB'	= ASCII constant
78	= Decimal constant
56H	= Hexadecimal constant
10110B	= Binary constant
457Q	= Octal constant
"1.0E3	= Floating point constant (Extension)

Floating point constants are stored in the proposed IEEE standard for single precision (32-bit) numbers and the syntax for the constants is with the exception of the first character identical to the Pascal standard. Floating points constants can be used as any other number but no operators are implemented (i. e. "6.0*"4.0E-20 is not invalid but vill not generate the correct answer).

Expressions

Everywhere a constant is specified in the Zilog assembler manual the constant can be substituted with an absolute expression with a valid range. This applies for instance to bit instructions where the bit number can be an expression. The available operators are the same as the ones described in the general section of this user's guide but symbolic operators (names) must have periods "." attached to each end. An example:

```
      AND    =>     .AND.
```

Spaces or tabs are not allowed in expressions.

Pseudo directives

All the directives listed in the general section are implemented but since SET is a machineinstruction this directive has been renamed to DEFL which is the original Zilog mnemonic for that directive.
The DEFB, DEFW, DEFS and DEFM directives are also implemented. The definition of DEFB and DEFW is compatible with the Zilog manual but also adds an extension: The directives may have more than one operand where the operands should be separated with commas.
In addition to DEFB and DEFW a new directive DEFD (define double word) has been defined. This directive works like DEFB and DEFW but the operands must be absolute expressions only. The DEFD directive stores 4 bytes in reversed order and is specially useful for defining floating point constants.

Reference manual:

1. Model 990 Computer and
   TMS9900 Microprocessor
   Assembly Language Programmer's Guide
   Order Number: 943441-9701

This manual can be obtained from the manufacturer, Texas Instruments (Austin, Texas.) and their local distributors.

Delimiters

The items in a source line must be separated by delimiters and in the TMS9900 assembly language the following delimiters are valid:

<u>For a source line with only a label:</u>

1. The label <u>must</u> begin in the first position.
2. The label should be terminated with a colon or only a CR.
3. No comments are allowed on this type of source line.

<u>For a comment only source line:</u>

1. The first position in the line must be "*".

<u>For an ordinary statement:</u>

1. Instructions <u>cannot</u> begin in the first position.
2. If a label is used it must be separated from the instruction with a colon, spaces or tabs. If there is no label the instruction should be preceded with tabs or spaces.
3. Instructions and operands must be separated with tabs or spaces.
4. Operands must be separated with commas only.
5. Comments must be separated from instructions or operands with tabs or spaces.

Label syntax

The first character of a label must be alphabetic (A-Z) or the special characters "?" or "_". The following characters (if any) can be any of the already mentioned characters or the decimal digits (0-9).
The registers R0-R15 are <u>predefined</u> symbols with values according to the register number.

Constants

This implementation conforms to the TI assembler manual but some extensions are also included:

'A'	= ASCII constant
'AB'	= ASCII constant
78	= Decimal constant
>56	= Hexadecimal constant
10110B	= Binary constant (Extension)
457Q	= Octal constant (Extension)

Expressions

The available operators are the same as the ones described in the general section of this user's guide, but symbolic operators (names) must have periods "." attached to each end. An example:

```
        AND   =>   .AND.
```

Since the character ">" is used to identify hexadecimal constants, the following operators are only available in symbolic form:

```
        >     =>     .GT.
        >=    =>     .GE.
```

Spaces or tabs are <u>not</u> allowed in expressions.

Object file type

The assembler object file type is by default equal to :R99.

Pseudo directives

Besides all the directives listed in the general section the following directives are also available:

```
     BSS, BES <absolute expression> {<comment>}
     DATA, BYTE <operand(s)> {<comment>}
     EVEN {<comment>}
     TEXT {<->}<'string'> {<comment>}
     DXOP <symbol>,<expression> {<comment>}
     NOP {<comment>}
     RT  {<comment>}
```

The DXOP directive requires a symbol which cannot contain more than <u>six</u> characters. Instructions that must reside on a word boundary are <u>automatically</u> aligned by the assembler. Register numbers, Shift counts, CRU bits and XOP levels can always be coded as <u>absolute</u> expressions with the range 0-15. AORG and RORG are replaced (not implemented) with ORG, ASEG, CSEG and DSEG.

TMS9940 and TMS9985 operation

As the assembler only accepts the basic 9900 instructions and the 9940 is to be used, the additional instructions (DCS, DCA and LIIM) must be defined with DXOP:s.

Reference manual:

1. MC6800/6801/6805/6809 Macro Assembler
   Reference Manual.
   Order Number: M68MASR(D)

This manual can be obtained from the manufacturer, Motorola Semiconductors (Austin, Texas.) and their local distributors.

Delimiters

The items in a source line must be separated by delimiters and in the MC6800/6801 assembly language the following delimiters are valid:

For a source line with only a label:

1. The label must begin in the first position.
2. The label should be terminated with a colon or only a CR.
3. No comments are allowed on this type of source line.

For a comment only source line:

1. The first position in the line must be "*".

For an ordinary statement:

1. Instructions cannot begin in the first position.
2. If a label is used it must be separated from the instruction with a colon, spaces or tabs. If there is no label the instruction should be preceded with tabs or spaces.
3. Instructions and operands must be separated with tabs or spaces.
4. Operands must be separated with commas only.
5. Comments must be separated from instructions or operands with tabs or spaces.

Label syntax

The first character of a label must be alphabetic (A-Z) or the special characters "?", "." or "_". The following characters (if any) can be any of the already mentioned characters, "$" or the decimal digits (0-9).
The index register X and the accumulators A and B, are reserved words and cannot be used as labels.

Constants

This implementation conforms to the Motorola assembler manual except for ASCII constants which means:

'A'	= ASCII constant
'AB'	= ASCII constants must <u>always</u> be surrounded with "'"
78	= Decimal constant
$56	= Hexadecimal constant
45H	= Hexadecimal constant
%10001	= Binary constant
10110B	= Binary constant
457O	= Octal constant
457Q	= Octal constant

Expressions

The available operators are the same as the ones described in the general section of this user's guide but symbolic operators (names) must have exclamation points "!" attached to each end. An example:

```
        AND  =>   !AND!
```

Spaces or tabs are <u>not</u> allowed in expressions.

Object file type

The assembler object file type is by default equal to :R68.

Pseudo directives

Besides all the directives listed in the general section the following directives are also available:

```
     RMB <absolute expression> {<comment>}
     FCB, FDB <operand(s)> {<comment>}
     FCC <constant>,<ASCII characters> {<comment>}
     FCC <'string'> {<comment>}
     MC6801 {<comment>}
```

The RMB, FCB, FDB and FCC directives works as described in the manual with one exeption: The string delimiters in the FCC directive must only be "'". As the assembler only accepts the basic 6800 instructions, the user must in order to assemble code for the 6801 (applies also to 68701 or 6803) use, an option "1" given at startup, or the directive <u>MC6801</u> in the source code.

Addressing modes and relocation

Since the MICRO-LINK does not support the relocation of 8-bit values the use of relocatable symbols in addresses always forces the assembler to select the extended addressing mode. The only way to get the direct addressing mode is:

1. The address expression is absolute.
2. The expression is completely defined in the first pass (i. e. no undefined symbols).
3. The value of the expression is in the range (0-255).

Note that all those conditions above must be fulfilled if the direct addressing mode is wanted.

Reference manuals:

1. MC68000 Cross Macro Assembler Manual.
   Order Number: M68KXASM(D3)
2. MC68000 User's Manual.
   Order Number: MC68000UM(AD)

Both of these manuals can be obtained from the manufacturer, Motorola Semiconductors (Austin, Texas.) and their local distributors.

Delimiters

The items in a source line must be separated by delimiters and in the MC68000 assembly language the following delimiters are valid:

<u>For a source line with only a label:</u>

1. The label <u>must</u> begin in the first position.
2. The label should be terminated with a colon or only a CR.
3. No comments are allowed on this type of source line.

<u>For a comment only source line:</u>

1. The first position in the line must be "*".

<u>For an ordinary statement:</u>

1. Instructions <u>cannot</u> begin in the first position.
2. If a label is used it must be separated from the instruction with a colon, spaces or tabs. If there is no label the instruction should be preceded with tabs or spaces.
3. Instructions and operands must be separated with tabs or spaces.
4. Operands must be separated with commas only.
5. Comments must be separated from instructions or operands with tabs or spaces.

Label syntax

The first character on a label must be alphabetic (A-Z) or the special characters "?", "@" or "_". The following characters (if any) can be any of the already mentioned characters or the decimal digits (0-9).
<u>All</u> registers (D0-D7, A0-A7, SP, PC, USP, CCR and SR) are <u>reserved</u> words and cannot be used as labels.

Constants

This implementation conforms to the Motorola assembler manual but some extensions are also included:

'A'	= ASCII constant
'AB''C'	= ASCII constant
78	= Decimal constant
$56	= Hexadecimal constant
10110B	= Binary constant (Extension)
457Q	= Octal constant (Extension)
"1.0E3	= Floating point constant (Extension)

Floating point constants are stored in the proposed IEEE standard for single precision (32-bit) numbers and the syntax for the constants is with the exception of the first character identical to the Pascal standard. Floating points constants can be used as any other number but no operators are implemented (i. e. "6.0*"4.0E-20 is not invalid but vill not generate the correct answer).

Expressions

The available operators are the same as the ones described in the general section of this user's guide but symbolic operators (names) must have periods "." attached to each end. An example:

```
        AND  =>   .AND.
```

Spaces or tabs are not allowed in expressions.

Object file type

The assembler object file type is by default equal to :R68K.

Pseudo directives

Besides all the directives listed in the general section the following directives are also available:

```
DS, DS.B, DS.W, DS.L <absolute expression> {<comment>}
DC, DC.B, DC.W, DC.L <operand(s)> {<comment>}
RORG  <expression> {<comment>}
ORG   <expression> {<comment>}
ORG.L <expression> {<comment>}
ORG.W <expression> {<comment>}
RELAD  {<comment>}
LONGAD {<comment>}
ABSAD  {<comment>}
WORDAD {<comment>}
```

The DS and DC directives works as described in the manual with one exception: No extra zero is inserted after a DC.B generating an odd address. Instructions that must reside on a word boundary are automatically aligned by the assembler.

Addressing modes and relocation

This assembler will in the same way as the original Motorola cross assembler choose the addressing mode of an operand which is an expression only and where the operand is allowed to have more than one possible mode. Since this is a relocating assembler and the Motorola cross assembler is not, some differences concerning the address mode selections exist, namely the term relative expression does not exist but instead expressions can be relocatable which in most cases are equvivalent. All the ORG:s specified by the Motorola manual are included as well as four extra directives (RELAD, ABSAD, WORDAD and LONGAD) which further enables the user to specify the default addressing modes. The function of the extra directives are:

RELAD	Generate position independent code.
ABSAD	Generate only direct addresses.
WORDAD	Generate short addresses when the direct mode is used.
LONGAD	Generate long addresses when the direct mode is used.

In addition to the default addressing mode selection an extension has been implemented which enables the user to overide the selections. The syntax for these extensions are:

```
expression(PC)  gives PC-relative adr.
<expression    gives short direct adr.
>expression    gives long direct adr.
```

When the position independent mode is selected (with overide or by default) the assembler will flag the line as invalid if the expression is not of the same type as the current segment or if the range is exceeded. The position independent mode is by default on. 16-bit relocatable items should be in the range -$8000 to $7FFF but this is in the current version of MICRO-LINK not completely checked.

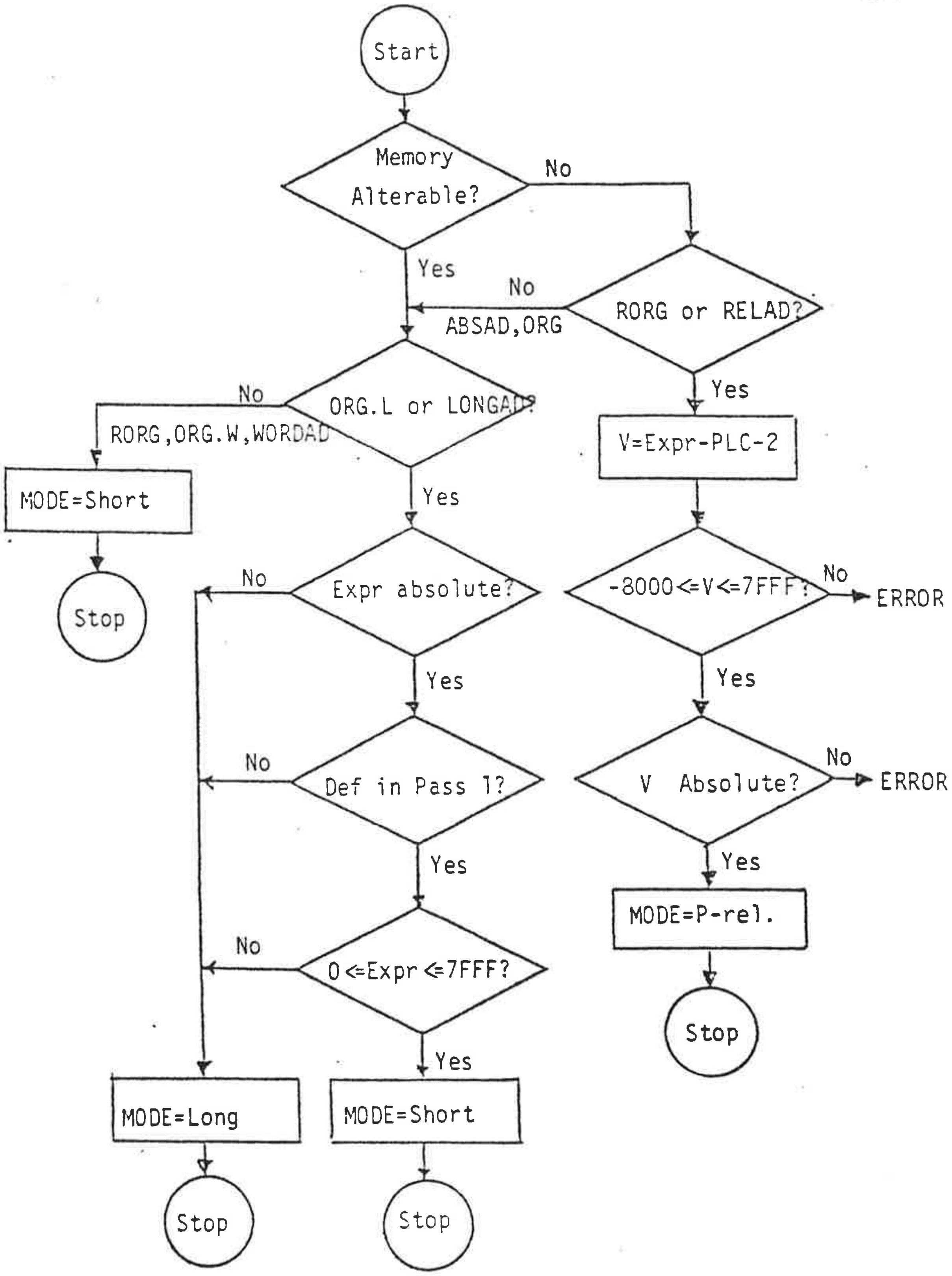
All addressing modes which are program counter relative (i. e. BRA Expr, JMP Expr(PC) and JMP Expr(A4.L)) must have an expression which is of the same type as the current segment.

The address register indirect indexed mode (Expr(A5,D4.W)) requires an absolute expression in the range of -128 to 127.

An alternative syntax for the program counter relative indexed mode is: Expr(PC,Rn).

The exact <u>default</u> selection done by the assembler is demonstrated in the the flow chart below:

Default on: [RORG *+0,WORDAD,ASEG]



Reference manuals:

1. AIM 65 USER'S GUIDE (ROCKWELL)
2. KIM-1 USER'S GUIDE (MOS TECHNOLOGY)

This manuals can be obtained from the manufacturerers. and their local distributors.

Delimiters

The items in a source line must be separated by delimiters and in the 6502 assembly language the following delimiters are valid:

For a source line with only a label:

1. The label must begin in the first position.
2. The label should be terminated with a colon or only a CR.
3. No comments are allowed on this type of source line.

For a comment only source line:

1. The first position in the line must be "*".

For an ordinary statement:

1. Instructions cannot begin in the first position.
2. If a label is used it must be separated from the instruction with a colon, spaces or tabs. If there is no label the instruction should be preceded with tabs or spaces.
3. Instructions and operands must be separated with tabs or spaces.
4. Operands must be separated with commas only.
5. Comments must be separated from instructions or operands with tabs or spaces.

Label syntax

The first character of a label must be alphabetic (A-Z) or the special characters "?" or "_". The following characters (if any) can be any of the already mentioned characters, "$" or the decimal digits (0-9).
The index registers X and Y and the accumulator A, are reserved words and cannot be used as labels.

Constants

This implementation conforms to the 6502 standard except for ASCII constants which means:

'A'	= ASCII constant
'AB'	= ASCII constants must always be surrounded with "'"
78	= Decimal constant
$56	= Hexadecimal constant
45H	= Hexadecimal constant
%10001	= Binary constant
10110B	= Binary constant
457O	= Octal constant
457Q	= Octal constant

Expressions

The available operators are the same as the ones described in the general section of this user's guide but symbolic operators (names) must have exclamation points "!" attached to each end. An example:

```
        AND  =>   !AND!
```

Spaces or tabs are not allowed in expressions.

Object file type

The assembler object file type is by default equal to :R65.

Pseudo directives

Besides all the directives listed in the general section the following directives are also available:

```
     <label>=<expression> equivalent with the EQU directive
     *=<expression> equivalent with the ORG directive
     *=*+<absolute expression> {<comment>}
     .BYTE,.WORD <operand(s)> {<comment>}
```

The . DBYTE, .SKIP, . PAGE and . OPT directives are not implemented.
The *= *+, .BYTE and . WORD directives works as described in the manual

Addressing modes and relocation

Since the MICRO-LINK does not support the relocation of 8-bit values the use of relocatable symbols in addresses always forces the assembler to select the extended addressing mode. The only way to get the direct addressing mode is:

1. The address expression is absolute.
2. The expression is completely defined in the first pass (i. e. no undefined symbols).
3. The value of the expression is in the range (0-255).

Note that all those conditions above must be fulfilled if the direct addressing mode is wanted.

Introduction

MICRO-LINK handles the step between the assembly of a program and the execution of it. The relocatable code produced by the assemblers is read by MICRO-LINK and transformed to a format readable by the monitors of the various systems. The transformation process includes the addition of library routines referenced by the program. Via commands the user can place the program at any address legal for the actual cpu.

The input file consists of at least one module. A module is a undividable piece of code and data. The header contains information such as the name of the module, type and relocation base. Then the actual code and data follows with specifications about public symbols (entry points), external references, absolute data and so on. The tail may have an address which is the start address for the program.

To make it easy to separate code and data, code is relocatable to three segments onto which the code is placed. One segment, Absolute, is handled completely by the user. The code is placed where the user has specified. As for the other two segments, Code and Data, the user decides where they start and then MICRO-LINK will place modules in increasing addresses from there on.

For example, the interrupt routines (or vectors) in a system could be placed via the absolute segment. The user routines are relocated to the code and data segments, whose addresses the user may decide at load time.

In all examples, user input is underlined.

Commands

How to load a simple program

Let's assume that you have a program written in 8085-code and it has been processed by the R8085 cross assembler. The only commands you have to know is DEFINE-CPU, LOAD and DUMP. The LOAD command will read the file produced by the cross assembler and relocate your program and the DUMP command will produce executable code in 8085 format on another file. The procedure is as follows:

```
@MICRO-LINK

    Micro Linking Loader. V80.06.26

*DEFINE-CPU
CPU: 8085
Code seg. start: 0000
Data seg. start: 4000
*LOAD
File: MAIN
Code: 02D4   Data: 4150
*DUMP
File: MY-PROG
*EXIT
```

On the file MY-PROG you will now have an executable version of your program. It may be entered into an 8085-system and run.

Command format

The commands may be abbreviated a'la Sintran and since the system also uses the edited command buffer of Sintran all control characters have the same function in MICRO-LINK as in Sintran.
MICRO-LINK also prompts the user for missing parameters in the same way as Sintran does.
Only spaces or commas may be used to separate commands and parameters.
When a parameter is surrounded by "{" and "}" it is optional.
A "..." after a parameter means that it may be repeated.
Some examples:

```
     LOAD,<file>{,<file>}...       definition
     LOAD,TEMP                     one file to load
     LOAD,TEMP,MATH,IOLIB          several files to load
```

Command to define the CPU

Before any loading is done, the target CPU must be defined. Also the start addresses for the Code and Data segments must be specified.

LIST-CPU-TYPES will list all CPU types handled by the system. This command will also list input and output filetypes and output format used by the loader for the different CPU:s.

DEFINE-CPU,< cpu type>,<code start>,< data start> will define the target CPU and set up the start addresses for code and data segments. These addresses should be hexadecimal numbers. Also the default Sintran file types and the <u>default</u> output format will be defined.

CPU			Input	Output	Format
Zilog	-	Z80	:RZ80	:Z80	I
Intel	-	8085	:R80	:8085	I
Motorola	-	6800-1	:R68	:6800	M
Texas Instr.	-	9900	:R99	:9900	T
Motorola	-	68000	:R68K	:68K	M
MOS/Rockwell	-	6502	:R65	:6502	I

Commands to load files

There exist two commands to load files, namely LOAD and LIBRARY-LOAD. The difference between them is that the LOAD command will load all modules in the file but the LIBRARY-LOAD command will only load those modules which has a referenced entry point.

LOAD,<file>{,<file>}... will load all modules in the specified file(s).

LIBRARY-LOAD,<file>{,<file>}... will load only those modules in the file(s) which has a referenced entry point.

LIBRARY-SCR-FILE,<file> must be specified before any LIBRARY-LOAD can be issued. The file type is by default :IMAG.

Command to specify load addresses

The command SET-LOAD-ADDRESS enables you to specify the starting load addresses for the code and data segments. This is seldom needed since the addresses normally are given in the DEFINE-CPU command.

SET-LOAD-ADDRESS,<segment>,<address> will set the load
    address for <segment> to <address>. <segment> should be
    "C" or "D".
    <address> should be a hexadecimal number.

Commands to examine the symbol table

MISSING-ENTRIES{<,file>} will print the names of external references not yet resolved and the names of the modules which reference them. The output will be directed to the terminal if no <file> is specified.

Ex. *MISSING-ENTRIES

```
          IOHANDLER is undefined
             Referenced by:
               MONITOR
```

WHAT-IS,<symbol> will print information about that symbol.

Ex. *WHAT-IS,IOHANDLER

```
          Entry  in IO
          Extern in MONITOR
          *WHAT-IS,IO
          Module
```

SET-SYMBOL-VALUE,<symbol>,<value> will assign a value to the specified symbol. The symbol must be an external with no entry. The symbol will be added to the last loaded module and the type will be absolute. This command is useful for setting system constants at loadtime.

MAP{,<file>} prints the modules defined, their load addresses in the absolute, code and data segments, the entry points defined in that module with their addresses and types. The output will be directed to the terminal if no <file> is specified.

Ex. *MAP

```
                               A-seg.      C-seg.      D-seg.

     MONITOR                     -        0200-122F   2000-3324
        INTERRUPT              C 0200
        MONITOR                C 0273
        IODATA                 D 2000

     IO                      0000-0002    1230-17D3   3325-3D86
        IOHANDLER              C 1230
        TTYIN                  C 1301
        TTYOUT                 C 1350
        TTYDATA                D 3F66
```

Commands to dump the memory image

When your loading is finished, i. e. all routines needed are loaded you should dump the memory image onto a file.

DUMP,<file> will dump the memory image onto <file> in a format readable by the loader for the actual cpu.

Miscellaneous Commands

The command RESET will reset all pointers and clear all tables, i. e. restart the program for another loading or when your current load has been messed up.

When you are ready the command EXIT will terminate MICRO-LINK.

To get all available commands enter HELP{,< command>}. If no parameter is given in the HELP command only a simple list of all commands will be displayed, else all commands which matches the parameter will be displayed with a syntax and function description.

LIST-CURRENT-CPU prints the name of the target CPU and the output format.

SET-NEW-FORMAT,<format> enables the user to overide the default output format. The available formats are listed in the section MICRO-LINK Output format and the format parameter should be I, M or T.

Input format

Syntax

```
<input file>         ::=<module> | <module><input file>

<module>             ::=<header><body><tail> | <nothing>

<header>             ::=QM<symbol><type><rel base>

<body>               ::=<absolute item> | <relocatable item> |
                        <entry item> | <external item> |
                        <org item> | <nothing>

<tail>               ::=QZ<transfer flag><transfer address>

<symbol>             ::=<symbol length><ASCII string>

<type>               ::=<0>

<rel base>           ::=<address>

<absolute item>      ::=<hex byte>

<relocatable item
in code segment>     ::=QR<address>

<relocatable item
in data segment>     ::=QS<address>

<relocatable
external item>       ::=QT<rel id><address>

<negative
relocatable item
in code segment>     ::=QN<word>

<negative
relocatable item
in data segment>     ::=QO<word>

<neagative
relocatable
external item>       ::=QP<rel id><word>

<entry item>         ::=QE<symbol><seg id><address>

<external item>      ::=QX<symbol><rel id>

<org item>           ::=Q<seg id><address>

<transfer flag>      ::=0 | 1
```

```
<transfer address>::=<address>

<symbol length>   ::=<hex byte>

<ASCII string>    ::=<ASCII char> | <ASCII char>
                     <ASCII string>

<rel id>          ::=<hex byte>

<seg id>          ::=A | C | D

<word>            ::=<hex byte><hex byte>

<address>         ::=<word> | L<word><word>

<hex byte>        ::=<hex digit><hex digit>

<hex digit>       ::=0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
                     A | B | C | D | E | F

<nothing>         ::=
```

Semantics

<symbol> is 1 to 80 characters consisting of any ASCII characters which all are significant but in the symbol table (MAP) only the leftmost 16 characters will be displayed. Module names and names of entries have no interrelationship which means that they may have the same name.

<type> in header may seem redundant but it is there for future extensions.

<rel base> is also zero in most cases but some assemblers, as Norsk Data's MAC, start assembly at location one, IBM:s starts at any specified address. It is subtracted from all addresses relocatable to the module.

<rel id> is the relative number of an external in a single module. Beacause <rel id> is a hex byte the number of external declarations in a module cannot excede 256 (00-FF), but the total number of symbols that can be loaded is more in the range of 1000.

QR,QS and QT are the normal id:s for relocatable items but in those cases when negative addresses (displacements) are used, a special id QN,QO or QP is used. QN,QO and QP records can only be 16-bit values. The reason for this special treatment of negative values is that range checking becomes almost trivial since you only have to check if there was a carry to detect an error-condition.

<address> for an entry point is relative to the specified segment. For an absolute entry point, segment A is specified.

If <transfer flag> is not zero, then <transfer address> is the start address of the relocated program. If several modules has <transfer address> specified, the first encountered is valid. The <transfer address> is supposed to be relative to the code segment.

In the file, carrige return and line feed is ignored.

Output format

"I" - Intel 8080, 8085; Zilog Z80; MOS/Rockwell 6502:

```
:nnppppffddddddddcc
```

where:
nn is number of data bytes (dd) in record.
pppp is load address.
ff is a flag: 00 normal data record. 01 end of file record.
dd are the data bytes which are 1 to 16 in a data record.
cc is the checksum, and the sum of all bytes nn, pppp, ff, dd and cc should be zero.

"M" - Motorola MC6800/6801, MC68000:

S1nnppppddddddddcc	Data record with a 16-bit address.
S2nnppppppddddddddcc	Data record with a 24-bit address.
S9030000FC	End of file record.

where:
nn is the number of bytes (pp+dd+cc) in record.
pppp is the load address.
dd are the data bytes which are 1 to 16.
cc is the checksum and
    cc := (255+(SUM(dd)+SUM(pp)+nn)) MOD 256.

"T" - Texas Instruments TMS9900:

9aaaaBddddBddddBddddBdddd7ccccF	Data record.
:	End of file record.

where:

9aaaa	Load address
Bdddd	Data word
7cccc	Checksum
F	End of line

The checksum (cccc) is the negated sum of all characters in the line upto and including the checksum tag (7).

Building your own library

One of the greatest advantages of having a relocation and linking facility is that it pushes a more structured way of programming i.e. it helps the user create software in modules instead of in one big program. After some time most users realises that it would be nice to save the most frequently used routines so that they could be used in the next project. One obvious way to do that is of course to save each piece of software on a file in source form, but that approach has (at least) three drawbacks. First you have the problem associated with keeping track of an ever increasing number of files but a more disturbing problem is that in order to use a particular module you often have to edit the internal labels since the possibility of duplicate labels increases with the total amount of source code.

The third type of problem you may run into cannot be solved without using separately assembled modules and that is the fact that the symbol table in our cross assemblers as well as in most others is not unlimited. The usable size in our assemblers is in the range of 1000-2000 symbols (with a mean length of six characters) depending on which particular assembler you are using and whether you are using the crossreference option which also takes some space.

So the cure is to assemble and document each reusable piece of software and put the object code in a library. Later you or your colleages can load the library with the LIBRARY-LOAD command (Do not use LOAD since then you always get all the stuff in the library and that is not what librarys are for !).

Since every assembly only produces one module on one file you must merge several files (modules) into one file in order to create a library. This is easily accomplished with the standard ND editor QED.

```
@QED                    % Get the editor
*MPI(0)                 % Turn off parity
*R FILE-1:R80           % Get the first file (module)
*R FILE-2:R80           % Get the next file (module)
  .                     %
  .                     %
*W "MY-LIB:R80"         % Write the modules on a file
                        % Don't forget the filetype!
*EX                     % It's done !
```

Error messages

All errors will be printed on the terminal with a short message (not a meaningless number only!) and errors can be split into three categories which should be dealt with differently.
The categories are:

1. Sintran errors
2. Command errors
3. Fatal loading errors

Sintran errors

The only Sintran errors you should get is from the file system (No such filename etc.) and those errors have no effect on the loaded modules (if any) so the loader does not terminate and if possible you can just retype the aborted command.

Command errors

These are:

1.	No such command	(self-explanatory)
2.	Ambiguous command	(self-explanatory)
3.	**** No such CPU	(self-explanatory)
4.	**** Ambiguous CPU-name	(self-explanatory)

5. **** CPU-type undefined
   If you try to load or dump files without first using the DEFINE-CPU command, the loader don't know what format to expect.

6. **** Cannot change CPU-type now
   If you already have loaded modules you must stick to the CPU you have chosen. After RESET you can start all over again.

7. **** Value out of range
   An address parameter is to large for the CPU type specified (for 8085 0<=val<=FFFF)

8. **** Illegal parameter
   Bad parameter in the HELP command or non hexdigits in an address parameter.

9. **** Illegal segment
   Segment specifier in SET-LOAD-ADDRESS is not C or D

10. **** Unknown symbol
If you specify an non existing symbol in the WHAT-IS and SET-SYMBOL-VALUE commands you will get this message.

11. **** Already defined
If you try to redefine a symbol with SET-SYMBOL-VALUE you naturally get an error.

12. **** Not an external
Only non resolved externals can be assigned values with SET-SYMBOL-VALUE.

13. **** Can't get the HELP-file!
Check if you have the file MICRO-LINK-HELP:DATA.

14. **** Forgotten to DUMP ?
This message will be printed if you do EXIT after loading one or more modules and not proceded with a DUMP. If you don't want to dump repeat EXIT.

15. **** Use LIB-SCR before any LIB-LOAD!
See command description of LIBRARY-LOAD

Command errors have no effect on loaded modules so you can always continue since the faulty command is just ignored.

Fatal loading errors

These are:

1. **** Illegal hex digit: 'x'
2. **** Record doesn't begin with 'Q'
3. **** Illegal record id: 'x'
4. **** Illegal segment id: 'x'
5. **** Record out of sequence
6. **** Illegal symbol length
7. **** External not found
If you use files created by our assemblers and <u>until now</u> have not got any fatal loading errors you should not get these errormessages.

8. **** Doubly defined modulename: XXXXX
9. **** Doubly defined entrypoint: XXXXX
(self-explanatory)

10. **** Address out of range in module: XXXXX
A module is loaded outside the address range or the used address mode of the target CPU.

11. **** Word or Long on uneven address in module: XXXXX
This message only applies to CPU:s which demand that Word or Long data must be located on even addresses. (i.e. TMS9900 and MC68000)

12. **** Overflow in reference to: XXXXX in module: YYYYY
A reference to an external is relocated out of range during LOAD or LIBRARY-LOAD.

13. **** Overflow in reference to: XXXXX
A reference to an external is relocated out of range during DUMP.

14. **** Data item out of range in module: XXXXX
15. **** Code item out of range in module: XXXXX
A relocatable item is relocated out of range.

16. **** Can't reopen the LIB-SCR
Someone have stolen your file!

17. **** Extra program entry in module: XXXXX
Only one module is allowed to have a transfer address.

18. **** Entry: XXXXX out of range in module: YYYYY
An entry point is relocated outside the address range of the target CPU.

19. **LOADER ABORTED**
If you have got more than 32 errors the loader terminates.

After a fatal error the symbol table may be affected so in order to start again you must issue a RESET.
The loader immediately aborts the current command when a fatal error is encountered.

A real session with the system

A complete example which shows how to use the relocation directives in the source code and how they are used by the loader is given on the following pages. Although the example uses 8085 assembly language it is completely relevant for the other assemblers as well.

```
@CC  ASSEMBLE THE FIRST MODULE (THE MAIN PROGRAM )

@R8085

    8085  ASSEMBLER  V80.06.26

SOURCE FILE=MAIN
LIST FILE=TERMINAL
OBJECT FILE=MAIN
OPTIONS:   C,T


Pass 1 complete,  no errors.

   1  0000                          NAME   MYPROG
   2  0000                          EXTRN  LOGX,COSX
   3  0000                          CSEG
   4  0000  210700     START:       LXI    H,7
   5  0003  CD0000                  CALL   LOGX
   6  0006  220000                  SHLD   BUFFER
   7  0009  CD0000                  CALL   COSX
   8  000C  2A0000                  LHLD   BUFFER
   9  000F  CF                      RST    1
  10  0000                          DSEG
  11  0000             BUFFER       DS     2
  12  0002                          END    START


 Assembly complete,  no errors.  16 bytes of code generated.

                       Symbol and cross-referencetable
                       ===============================

 Symbol Value Type  Defline   Referenced in line

 BUFFER     0000  D       11          6          8
```

```
COSX       0000   X01       2          7

LOGX       0000   X00       2          5

MYPROG     0000   A         1

START      0000   C         4         12
```

```
@CC ASSEMBLE THE SECOND MODULE

@R8085

    8085  ASSEMBLER  V80.06.26

SOURCE FILE=T-1:TEST
LIST FILE=TERMINAL
OBJECT FILE=T-1
OPTIONS:   C,T


Pass 1 complete,  no errors.

   1  0000                             NAME  MATH1
   2  0000                             CSEG
   3  0000                             PUBLIC LOGX,SINX,COSX
   4  0000                             EXTRN FLOAT
   5  0000  3E00       LOGX:           MVI   A,0
   6  0002  C9                         RET
   7  0003  CD0800     SINX:           CALL  COSX
   8  0006  AF                         XRA   A
   9  0007  C9                         RET
  10  0008  CDFCFF     COSX:           CALL  FLOAT-4
  11  000B  C9                         RET
  12  000C                             END

 Assembly complete,  no errors.  12 bytes of code generated.
                        Symbol and cross-referencetable
                        ================================

 Symbol  Value  Type  Defline   Referenced in line

 COSX      0008   CE      10          3         7

 FLOAT     0000   X00      4         10

 LOGX      0000   CE       5          3

 MATH1     0000   A        1

 SINX      0003   CE       7          3
```

```
@CC ASSEMBLE THE THIRD MODULE

@R8085

    8085  ASSEMBLER  V80.06.26

SOURCE FILE=T-2:TEST
LIST FILE=TERMINAL
OBJECT FILE=T-2
OPTIONS:   C,T


Pass 1 complete,  no errors.

   1  0000                      NAME  MATH2
   2  0000                      PUBLIC READ,WRITE
   3  0000                      PUBLIC BUFFER
   4  0000                      CSEG
   5  0000 DB77      READ:      IN     77H
   6  0002 C9                   RET
   7  0003 D345      WRITE:     OUT    45H
   8  0005 C9                   RET
   9  0000                      DSEG
  10  0000           BUFFER     DS     80
  11  0050                      END


 Assembly complete,  no errors.  6 bytes of code generated.

                    Symbol and cross-referencetable
                    ===============================

 Symbol Value Type  Defline   Referenced in line

 BUFFER    0000  DE      10          3

 MATH2     0000  A        1

 READ      0000  CE       5          2

 WRITE     0003  CE       7          2

 @CC ASSEMBLE THE LAST MODULE

 @R8085

     8085  ASSEMBLER  V80.06.26

 SOURCE FILE=T-3:TEST
 LIST FILE=TERMINAL
 OBJECT FILE=T-3
 OPTIONS:   C,T


 Pass 1 complete,  no errors.
```

```
   1  0000                         NAME   MATH3
   2  0000                         PUBLIC FLOAT
   3  0000                         CSEG
   4  0000 CD0600      FLOAT:      CALL   MODIF
   5  0003 3E07                    MVI    A,7
   6  0005 C9                      RET
   7  0006 0A          MODIF:      LDAX   B
   8  0007 C9                      RET
   9  0008                         END


Assembly complete,  no errors.  8 bytes of code generated.

                     Symbol and cross-referencetable
                     ===============================

Symbol  Value  Type  Defline    Referenced in line

FLOAT     0000  CE        4             2

MATH3     0000  A         1

MODIF     0006  C         7             4
```

```
@CC CREATE A LIBRARY FILE

@QED
QED 4.1
*MPI(0)
*R T-1:R80
58 WORDS READ
*R T-2:R80
51 WORDS READ
*R T-3:R80
33 WORDS READ
*L1,$
 M05MATH100000QA0000QC0000QX05FLOAT003E00QE04LOGXC0000C9CDQR0
008QE04SINXC0003AF
C9CDQP00FFFCQE04COSXC0008C9QZ00000
 M05MATH200000QA0000QC0000DB77QE04READC0000C9D345QE05WRITEC00
03C9QD0000QD0050
QE06BUFFERD0000QZ00000
 M05MATH300000QA0000QC0000CDQR0006QE05FLOATC00003E07C90AC9QZ0
0000
*W MATH-LIB:R80
143 WORDS WRITTEN
*EX
```

```
@CC LINK MAIN WITH THE LIBRARY

@MICRO-LINK

    Micro Linking Loader V80.06.26

*DEF-CPU 8085 1000 8000
*LIB-SCR I
*LOAD MAIN
Code: 100F     Data: 8001
*LIB-LO MATH-LIB
Code: 1023     Data: 8001
*MAP
                                A-seg.      C-seg.      D-seg.

         MYPROG                   -        1000-100F   8000-8001

         MATH1                    -        1010-101B       -
              COSX              C 1018
              SINX              C 1013
              LOGX              C 1010

         MATH3                    -        101C-1023       -
              FLOAT             C 101C

         Program entry at location: 1000 in module: MYPROG

*DUMP CODE
*EXIT

@CC AS YOU CAN SEE IN THE MAP, MATH2 WAS NEVER

@CC LOADED AS IT IS NOT REFERENCED

@CC DISPLAY RESULT!

@COPY TER CODE:8085

:10100000210700CD1010220080CD18102A0080CFBB
:101010003E00C9CD1810AFC9CD1810C9CD22103E61
:0410200007C90AC929
:00000001FF

END OF FILE
```

References

[1] The Pascal language is defined in:
Pascal User Manual and Report (Wirth,Jensen 1975)
Springer Verlag, Heidelberg

[2] The CERN implementation of Pascal on NORD-10/100 computers is described in:
PS-Pascal User's Guide (R. Cailliau, M. Kruger, J. Mc Cullough 1979)
PS division CO group CERN, Geneva

[3] The principles for the original universal assembler is described in:
Riktlinjer for utveckling av generell korsassemblator (Lars-Gunnar Backlin, Goran Lundstrom oct. 1977) (in swedish)
UPTEC 77 64 R (Institute of Technology, Uppsala)

[4] Some available hardware emulation products which can be used with a host computer:

The 8001 Microprocessor dev. lab.
Tektronix inc. (Beaverton,Oregon)

The Micro System Analyser
and the Micro System Designer, both available from:
Millenium Systems Inc.(Cupertino,Calif)

[5] Some available micro computer cross software:

Interpreters, Compilers etc. for 8080, Z80 and 8086 based systems.
Manufacturer: Microsoft Co. (Bellevue,Washington)

BSAL (Block-Structured-Assembly-Language) for the 8080.
Manufacturer: Mupro Inc. (Sunnyvale,Calif)

SMAL (Structured-Macro-Assembly-Language) for the 8080.
Manufacturer: Chromod Associates (New York)

Forth (high level language available for many types of processors)
Manufacturer: Forth Inc. (Manhattan Beach,Calif)

PLMX (machine independent high level language)
Manufacturer: System Consultants Inc. (San Diego,Calif)

PL/W (high level language for the 6800)
Manufacturer: Wintek Corp. (Lafayette,Ind)

UCSD-Pascal (a machine independent Pascal system implemented on a varity of computers)
Distributor: Softech Inc. (San Diego,Calif)

Installation guide

Although MICRO-LINK and the assemblers runs on any NORD-10/100 under the Sintran III operating system, some features must be available in the system for the package to run correctly, and these are:

1. The output file terminal must have exactly the name TERMINAL and not TERM as it is called in some systems.

2. The SCRATCH file system (file 100) is used by MICRO-LINK for temporary storage.

Both MICRO-LINK and the assemblers may be placed as <u>reentrant segments</u>.

MICRO-LINK is normally delivered on one floppy disc containing:

1. MICRO-LINK-XXXXX:PROG The linking loader.

2. MICRO-LINK-HELP:DATA This file contains the text used by the HELP command.

3. MANUAL:INF This file contains this manual and it can be output to a DIABLO, QUME or lineprinter.

4. MAIN:R80, T-1:R80, T-2:R80, T-3:R80 These files are the ones used in the sample session and they are included only for demonstration and testing purposes.

An assembler is normally delivered on one floppy disc containing:

1. a PROG file with the cross assembler.

   R8085-XXXXX:PROG for the 8085 processor.
   RZ80-XXXXX:PROG for the Z80 processor.
   R9900-XXXXX:PROG for the TMS9900 processor.
   R6800-1-XXXXX:PROG for the MC6800/6801 processors.
   R68000-XXXXX:PROG for the MC68000 processor.
   R6502-XXXXX:PROG for the 6502 processor.

2. One or more test programs, where the start file has the same name as the PROG file. In those cases when more than one test file is provided the other files are included by the start file.
   The names of the test programs are:
   R8085:TEST
   RZ80:TEST
   R9900:TEST
   R6800-1:TEST
   R68000:TEST
   R6502:TEST

"XXXXX" on the file names contains the release date of the delivered program.

Bug reports

If a bug is discovered, we would very much appreciate a report on it, sent to the following address:

Computer dept.
Institute of Technology
P.O. Box 534
S-751 21 Uppsala
SWEDEN