NORD PL

User's Guide

A/S NORSK DATA-ELEKTRONIKK
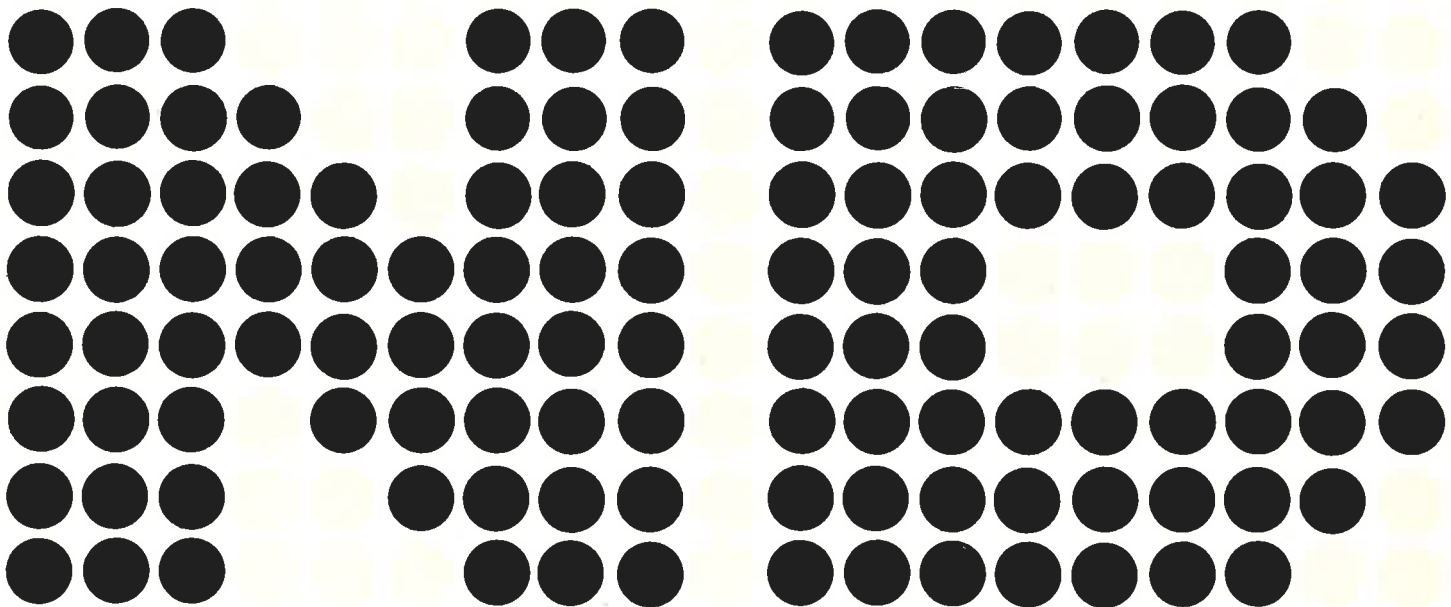
NORD PL
User's Guide

REVISION RECORD

Revision	Notes
8/73	Original printing

TABLE OF CONTENTS

--ooOoo--

--ooOoo--

1 INTRODUCTION

1.1 Machine Oriented Languages

A machine oriented language is a medium level language, standing between the high level languages and assembly code. The syntax resembles that of ALGOL. However. the use is intended to be like that of an assembler, because all facilities of the computer can be reached.

a) The complete instruction set with all addressing modes

b) All registers

c) All available memory locations

Comparing a machine oriented language to assembly code:

a) It is easier to write programs, and the error checking can be more extensive.

b) The programs will be more readable for others.

Comparing a machine oriented language to high level language:

a) A PL-language will give more optimal object code, about the same as for assembly code.

b) The programmer is not dependent on fixed calling sequences or data structures.

One of the main applications of machine oriented language is system programming (operating systems, compilers). where efficiency as well as readability is needed.

1.2 Properties of the NORD PL

The NORD PL is designed for the NORD-1, NORD-10 and NORD-20 computers. The object output is MAC assembler source code. Therefore, including of assembly code sequences is very easy. All the debug facilities of MAC are immediately available, including symbolic references to labels and variables.

The statement set includes:

a) Arithmetical statements, consisting of arithmetical/logical expressions and assignments. Constant expressions are also included. These will be evaluated at compile time.

b) Control statements, including:

```
FOR      - loop control
IF       - conditional branching
GO       - unconditional branching
CALL     - subroutine call
```

c)   Declaration statements, with type specifications and data presetting.

The compiler also includes conditional compiling.

2   ENVIRONMENTS

The compiler needs about 5.7 K of core plus main symbol table
(5 locations per symbol). The text is compiled in one pass. The
compiler and the assembler can be coupled together as co-routines,
so that even the whole compiling-assembling can be performed in one
single pass.

The compiler can be run either as a freestanding system, under
NORD-OPS or under TSS.

3   STRUCTURE OF NORD PL

3.1   Basic Elements

3.1.1   Symbols

A symbol is a string of digits and letters, where the first 5 characters only are significant. The rest will be taken as a comment. At least one of the 5 first must be a letter (not necessarily the very first). A symbol can be used for a variable, a label or a constant.

Examples:

NEW SYMBOL, A1, 1A

3.1.2   Numbers

In decimal mode the compiler will normally regard a string of digits as a decimal number. If the string is immediately preceded by a &, it will be an octal number. In octal mode the digit string will be octal in any case. Decimal/octal mode is set by commands. Initially the compiler is in octal mode.

Floating point numbers have the same syntax as MAC floating point, except that the sign $\#$ is used instead of E; besides the number must always start with a digit.

Example:

0. 33$\#$ - 33

3.1.3   Reserved Combinations

Some symbols are reserved for special use, as operators, statement symbols or register identifiers. Some special characters are also used. A complete list is found in Appendix A.

3.1.4   Character and String Constants

Character constants have the same form as in MAC. $\#$ $\#$ A puts the 7 bits ASCII equivalent of A right adjusted in a word, and $\#$ AB packs the characters A and B into one word. The string has also the same form.

'ABCD' will be packed as:

A	B
C	D
'	

3.1.5   Registers

The registers have fixed names:

P	TAD
X	K
T	Z
A	Q
D	O
L	C
B	M
AD	0

If the number 0 is found in an executable expression, it will be regarded as the zero register, not as a constant.

3.2   Data Structure

Three data types are available:

a)   Integers   (16 bits)

b)   Double   (32 bits)

c)   Real   (48 bits)

These types can be used either as single variables or arrays. Pointers to the actual variables can also be declared.

All data must be declared before they can be used. However, the actual location can be delayed, allowing for instance a data table to be placed after the code using it.

The addressing mode is defined by the context of the declaration.

Static data can be initialized.

3.3   Data Expressions

A data expression is evaluated at compile time. The operands consist of constants and symbols. If labels or variables are used, their address values will be used. The operators are:

+   Add

-           Subtract

*   Multiply

\   Byte separation (equivalent to "* 400_8 +")

The expression is evaluated strictly from left to right. If a label or a variable's reference occurs, only + and - are allowed for the rest of the expression.

Examples:	MAC equivalents:
1+2*10	30
1 \ 1	401
# #A\ # #B (equivalent to #AB)	#AB
5+2+LAB1-VAR2	7+LAB1-VAR2

Data expressions can be found in:

a)   Declaraction statements as initializations

b)   Call statements as parameters

c)   Executable statements as operands, surrounded by quotes (").

3.4   Executable Expression

In general an executable expression specifies a series of operations between the primary operand, which is a register, and different secondary operands, which can be registers, variables or constants. If the expression starts with a register, this register will be the primary operand throughout the expression.

Examples:	MAC equivalents
A+V1+55=:V2	ADD V1
	AAA 55
	STA V2
X+5=:L-10	AAX 5
	COPY SX DL
	AAX -10

The operations are executed strictly from left to right. with no implicit or explicit priority.

Operators

Arithmetical

:=	Load
=:	Store
:=:	Swap
-	Subtract
+	Add
*	Multiply
\	Divide (reals only)

Shift

	SHZ	shift with zero end input
	SH	arithmetical shift
	SHR	rotational shift
	SHL	shift with link end input

Logical

	/\	And
	\/	Or
	XOR	Exclusive or
	-,	one's complement
	BONE	set bit
	BZERO	reset bit

Special unary:

	MIN	Memory increment (MIN instruction)
	GOSW	Switch

Example: MAC equivalent:

A+4 GOSW L1, L2   AAA 4
                                           RADD SA DP
                                           JMP L1
                                           JMP L2

An expression can also start with a variable. Then the Λ, AD or TAD will be the primary operator, if the variable is an integer, double or real. The first operation is assumed to be a load.

Example:

If IX is an integer, then the expression

IX+IY=: IZ is equivalent to A:=IX+IY=:IZ

If a variable is included in quotes ("), it is said to be referenced. Then it is accessed one time less indirect than otherwise. This means that a referenced pointer will be referenced as a variable, and a referenced variable or label will give the address value. A constant will have the same meaning whether it is referenced or not. Inside the quotes even whole data expressions can be placed (see Section 3.3). Then all variables will be represented with their address values (even the pointers).

Examples:

Let PNT1 be a pointer, V1 a variable and LL a label.

NORD PL	MAC equivalents
PNT1	LDA I PNT1
"PNT1"	LDA PNT1
V1	LDA V1
"V1"	LDA (V1
"PNT1+V1+5"	LDA (PNT1+V1+5

The number 0 will be regarded as the zero register. If constant with value zero is wanted, it should be surrounded by quotes (").

Examples:

NORD PL	MAC equivalents
0=:T	COPY DT
0=:VAR	STZ VAR
"0"=:VAR	SAA 0:STA VAR
A BONE "0"	BSET ONE 0 DA

3.5   Statement Structure

A statement is normally terminated by a semicolon or a carriage return. Using a command (ICR), it is possible to set the compiler in "ignore-carriage return" mode, so that the carriage return will be ignored. Then a statement can consist of several lines.

A command begins with the percent sign (%). Then the rest of the line will be ignored.

If a statement starts with an asterisk (*), the rest of the line is regarded as MAC assembly code.

If a statement starts with an at (@), it is regarded as a command to the compiler.

3.6   Program Structure - Scope of Variables

Since the NORD-1/10/20 computers have direct addressing areas of
256 words, the programs will usually be divided into small subroutines.
Therefore, NORD PL has a subroutine feature, where labels and variables
defined within a subroutine are local to that subroutine.   Labels and
variables defined outside the subroutines are global.

There is only one level of subroutines; it is not possible to declare a
subroutine within another subroutine.

Example of layout:
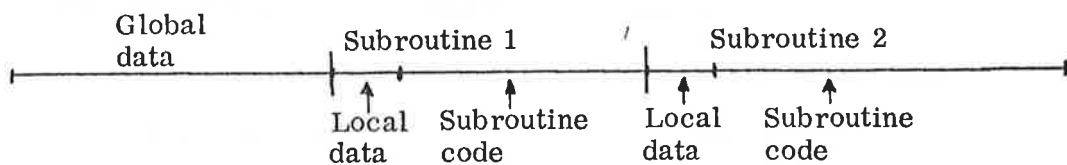
4   TIIE INDIVIDUAL STATEMENTS

4.1   Declaration Statements

4.1.1   <u>Data Declarations</u>

Variables must be declared to be one of the types:

     INTEGER

     DOUBLE

     REAL

In addition two optional declaration symbols can be added:

     ARRAY

     POINTER

These will modify the addressing mode. After the declaration symbols
a list of variables can follow. The variables can be initialized,
either as default zeros, or to specified values.

Examples:	MAC equivalents
INTEGER INT1, INT2	INT1, 0 INT2, 0
REAL FLX	FLX, 0; 0; 0;
DOUBLE SYM, SY2=SYM, S3	SYM, 0; 0 SY2=SYM S3, 0: 0
INTEGER POINTER PVAR:=VAR	PVAR, VAR
REAL ARRAY FX(20), FY(30)	FX=*; *+20+20+20/ FY=*; *+30+30+30/
INTEGER ARRAY TEXT:='STRING'	TEXT, 'STRING'

A declared entity can be initialized by several elements, divided by a
comma, the whole list enclosed in parentheses.

INTEGER ARRAY PARLIST:=(LOGNO, AREA, "100", "15")	PARLI, LOGNO AREA (100 (15

Each element can be a data expression.

Data can also be initialized with no name attached to it by using
the DATA statement:

DATA (44, XY, 5)

is equivalent to

INTEGER DUMMY := (44, XY, 5)

The actual place of data can be delayed, so that data can be placed
after the code physically. Then the variables can be declared equal
to question mark the first time and later be declared a second time.

Example:

```
INTEGER ARRAY TABLE=?
SUBR S
    :
    :
RBUS
INTEGER ARRAY TABLE (1000)
```

4.1.2 Symbolic Contants

Symbols can be defined to be constants, using the SYMBOL statement.

Example: MAC equivalent

SYMBOL L200=200, L210
=L200+10   L200=200
                                             L210=L200+10

The value can be a data expression of numbers and defined symbols.
If the value is omitted, it will be one greater than the former value.

Examples: MAC equivalent

SYMBOL S0,S1,S2,S3   S0=0
                                             S1=1
                                             S2=2
                                             S3=3

SYMBOL CHA=# #A, CHB, CHC   CHA=# #A
                                             CHB=CHA+1
                                             CHC=CHB+1

4.1.3 Addressing Mode Specifications

The addressing mode of a variable is dependent on the context of the
declaration statement. If nothing else is stated, a variable declared
inside a subroutine (between a SUBR and a RBUS statement) is directly
P-relative addressed, otherwise indirect addressed. If a variable is to be
B-relative addressed, its declaration statement must be enclosed by
a BASE -ESAB pair or a DISP -PSID pair.

If the variables are static allocated, BASE can be used, followed by a base-field identifier.

Example: MAC equivalent

BASE BA   BA=*+200
     INTEGER BVAR1, BVAR2   BVAR1,0
                                         BVAR2,0
     INTEGER POINTER PSUB:=SUB   PSUB, SUB
ESAB

The instruction to get a BASE variable can be

     LDA BVAR1 - BA, B

If the variables are dynamic allocated, for instance as variables in an element of a data structure, DISP should be used, followed by the displacement of the first variable. Such variables cannot be initialized.

Example: MAC equivanlent

DISP - 200   D1=-200
     INTEGER D1,D2   D2=-177
     INTEGER ARRAY DARR(10)   DARR=-176
     INTEGER ENDA   ENDA=-166
PSID

It is the user's responsibility to set the B register to the proper value.

Addressing modes:

Type \ Addr.		BASE	DISP	Local	Global
INTEGER	J "J"	LDA J-BA,B LDA (J	LDA J,B	LDA J LDA (J	LDA I (J LDA (J
INTEGER POINTER	PP "PP"	LDA I PP-BA,B LDA PP-BA,B	LDA I PP,B LDA PP,B	LDA I PP LDA PP	Illegal LDA I (PP
REAL	R "R"	LDF R-BA,B LDA (R	LDF R,B	LDF R LDA (R	LDF I (R LDA (R
INTEGER ARRAY	XB "XB"	LDA X B-BA ,B,X LDA (XB	LDA XB,B ,X	LDA I,X (XB LDA (XB	LDA I,X (XB LDA (XB
INTEGER ARRAY POINTER	LL "LL"	LDA I LL-BA ,B,X LDA LL-BA,B	LDA I LL ,B,X LDA LL,B	Illegal "	Illegal "

DISP- or BASE-addressed variables can be forced to be X-addressed instead of B-addressed. Then the variable must be preceded by an X value denotation and a period (.).

Examples:	MAC equivalents
X. DD	LDA DD, X
START. DD	LDX START
	LDA DD, X
START. EL1. VAL	LDX START
	LDX EL1, X
	LDA VAL, X

This access method is useful for addressing data structures.

4.1.4 Subroutine Declarations

A subroutine statement starts with the symbol SUBR. followed by a list of entrypoints. The entrypoints will be global symbols. Other labels and variables declared after the subroutine heading will be killed at the end of the subroutine. The end is marked by the symbol RBUS, which is a formal declaration only.

It is the programmer's responsibility to provide a return jump from the subroutine (using EXIT, EXITA or GO).

Example:

```
SUBR ENT1, ENT2
        :
        :
        :
ENT1:
        :
        :
ENT2:
        :
        :
RBUS
```

4.2 Executable Statements

An executable statement can only be found within a subroutine.

4.2.1 Arithmetical Statements

An arithmetical statement consists of an executable expression, as described in section 3.4. If the expression begins with a variable, the A, AD or TAD register will be used as the primary operand, depending on the type of the variable.

4.2.2 Control Statements

4.2.2.1 GO Statements

The GO statement is used for unconditional branching. It consists of the symbol GO followed by a label or a pointer.

Examples:	MAC equivalents:
INTEGER POINTER RET: = RETX	RET, RETX
LL: GO RET	JMP I RET
GO LL	JMP LL
GO ENTX % EXTERNAL ENTRYPOINT	JMP I (ENTX

If a subroutine is long, it can be useful to force a jump to be indirect, as not to exceed the displacement range. This can be done by placing the symbol FAR after GO.

Example:	MAC equivalent
GO FAR LL	JMP I (LL

4.2.2.2 CALL Statements

The simplest form of a subroutine call is the symbol CALL followed by a subroutine entrypoint, a pointer or a local label. If it is not yet defined, it is assumed to be an entrypoint of a succeeding subroutine. The parameters can be transferred by the registers.

The simple CALL statement can also be followed by a parameter list, being equivalent to the data list of the DATA statement. This can be used by placing the parameter addresses after the subroutine jump.

Examples: MAC equivalents:

INTEGER POINTER PNTR: = SUB0   PNTR, SUB0
CALL SUB1   JPL I (SUB1
CALL PNTR   JPL I PNTR
CALL SUB2 (V1, SX2, WM)   JPL I (SUB2
                                        V1
                                        SX2
                                        WM

4.2.2.3   EXIT Statements

Return from subroutines can be performed by EXIT (which is compiled to EXIT)
or EXITA (which is compiled to EXIT AD1). If these means for subroutine
return are used, the programmer should ensure that the L register contents have
not been destroyed, for instance by a subroutine call within the subroutine.

4.2.2.4   IF Statements

The IF statement has the general form:

IF < conditions> THEN <statements> ELSE <statements> FI

ELSE can be omitted.

Between IF and THEN there can be several conditions, delimited by the OR
or AND symbols. The conditions are evaluated from left to right. If a condition
followed by AND or THEN is not true, the statements after THEN are bypassed.
If a condition followed by OR or THEN is true, the statements after THEN are
executed. Otherwise more conditions will be tested.

There are two types of conditions: relations and bit tests.

A relation consists of two executable expressions with a relational operator
between them:

>   Greater

<   Less

=   Equal

> =   Greater or equal

< =   Less or equal

> <   Not equal

If the first element of the first expression is a variable or a constant, the A
or TAD register is considered as the primary register.

If the first element of the second expression is a variable or a constant, the T register is considered as the primary register. This means that the relation

IF VAR1   VAR2   THEN...   is equivalent to

IF A: = VAR1   T: = VAR2 THEN...

An expression can be empty. Then the present value of the A or T register will be used.

IF > THEN...   is equivalent to

IF A > T   THEN...

This means that a construction like this is possible,

IF VAR1 = 4   OR = 6   OR = 7 THEN...

If the first expression is of type REAL, the second must be equal to zero.

It is also possible to compare absolute values, where the register contents are considered as positive numbers from 0 to 64K. Then the two relational operators can be used:

    < <   Less
    > > =   Greater or equal

In the NORD-1 version the second expression must be equal to zero.

Examples:	MAC equivalents:
IF VAR-D < VAR2 THEN....	LDA VAR
	RSUB SD DA
	LDT VAR2
	SKP IF DA LST ST
	JMP BYPAS
IF A + 10 = 0 THEN...	AAA 10
	JAF BYPAS
IF A - LLIM > > = 0 THEN	SUB LLIM
	BSKP ONE SSC
	JMP BYPAS
	(NORD-1 version)

A condition can be a bit test.

The bit to be tested can be one of the single bit registers. The condition can be inverted by placing the symbol NBIT after the register specification.

Examples: MAC equivalents:

IF K THEN...   BSKP ONE SSK
                                   JMP BYPAS

IF M NBIT THEN...   BSKP ZRO SSM
                                   JMP BYPAS

A bit in the general registers can also be specified. An expression determines
the register. Then one of the symbols BIT or NBIT selects 1 or 0 as true. At
last a constant determines the bit number.

Examples: MAC equivalents:

IF T BIT 7 THEN...   BSKP ONE 70 DT
                                   JMP BYPAS

IF NBIT 1 THEN   BSKP ZRO 10 DA
                                   JMP BYPAS

The construction THEN GO <label> FI can be abbreviated to GO <label>.

For instance IF A < 0 GO ERR

is equivalent to IF A < 0 THEN GO ERR FI

MAC equivalent: JAN ERR

4.2.2.5   FOR Statements

The FOR statement is used for iterative purposes. Between FOR and DO
the iteration specifications. Between DO and OD are the statements to be
executed.

The general version has the form:

FOR <control variable> STEP <step count> TO <limit> DO <statements> OD

The control variable normally specifies a register, which can be initialized
by an expression.

The step count is a constant defining the stop size. STEP can be omitted,
then assuming 1 as a default step. The limit is compared to the control variable
before each execution. If it exceeded, there will be no more executions.

Example:

FOR X: = VAR STEP 3 TO 50 DO A + ARR (X) OD

MAC equivalent:

```
        LDX VAR              % SET INITIAL VALUE
NEXT,   SAT 50
        SKP IF DT GRE SX     % TEST LIMIT
        JMP BYPAS
        ADD I, X (ARR
        AAX 3                % STEP CONTRON VAR
        JMP NEXT
BYPAS,  ......
```

In case of beginning with a variable or constant, the expression after FOR
will take A as the primary register and the expression after TO will take T
as the primary register. However, if the expression after FOR consists of
one single variable, this variable will be taken as the control variable instead
of a register.

Example: MAC equivalent:

FOR J TO T DO CALL INCR OD
```
                LDA J
        NEXT,   SKP IF DT GRE SA
                JMP BYPAS
                JPL I (INCR
                LDA J
                AAA 1
                STA J
                JMP NEXT
        BYPAS,  ......
```

If only simple counting is wanted, the two following special cases are avail-
able:

a)   A single variable between FOR and DO:

        Example: MAC equivalent:

        FOR VAR DO.....OD   NEXT,

```
                                MIN VAR
                                JMP NEXT
```

        This means that if the control variable contains negative number
        before entering FOR, this will be the number of executions.

b)   An X-expression between FOR and DO:

        Example: MAC equivalent:

        FOR X: = -5 DO....OD   SAX -5 % 5 EXECUTIONS
                                      NEXT,

```
                                JNC NEXT
```

The loop can also start with just a single DO.   Then there will
be an unconditional jump back.

Example: MAC equivalent:

DO.....   OD   NEXT,

                                              JMP NEXT

5   ADDITIONAL FEATURES

5.1   Commands

A command starts with a circled alpha (@) followed by the command name.
The command names are not reserved symbols, so that the same symbol
can be used for a command name as well as for a user variable. After
the command name parameters may follow, seperated by commas.

Some of the commands are used for conditional compiling, being described
in Section 5.2. In Section 5.3 in-line assembly coding is treated. The re-
maining commands are described below.

@ ICR   — "Ignore carriage return"-mode.
                  This command is to be used if a statement should need
                  several lines (especially declaration statements). The
                  carriage return is treated as if it were a space.

@ CR   — "Carriage return"-mode.
                  After this command carriage return will have the same
                  effect as the semicolon (;), so that it will terminate the
                  current statement.

@ EOF   — "End of file".
                  This command is used for exit from the compiler to
                  the operating system. The MAC command)LINE is
                  output on the object device.

@CLEAR   — Clear the symbol table of the compiler.

@OCT   — All integer numbers will be treated as octal.

@DEC   — Integer numbers will be treated as decimal, except
                  for those preceded by the "&" sign.

@DEV   — <input device>, <list device>, <object output device>.
                  This command is used for setting device numbers for
                  the compiler. If the list device   0, the error messages
                  will be printed on the output communication device,
                  otherwise on the list device.

                  If list output and object output use the same device
                  number, the object output will appear left adjusted and
                  the source program will be listed 32 columns to the right.
                  The source program will be preceded by "%" signs, so
                  that the mix can be assembled.
                  Example: @DEV 4, 5, 3

                  For the TSS version files and devices can be specified
                  symbolically in the TSS notation. The necessary closing
                  and opening of files will be done. Numeric and symbolic
                  representation can not be mixed in the same DEV-
                  command, except for the single digit 0.

                  If a device is not specified at all, the old one will be used.

Examples:

>   @ DEV T-R, 0, OBJECT FILE
>   @ DEV INP-FILE, L-P
>   @ DEV INP, L-P, L-P

MODE — <input communication device>, <output communication device>. The communication devices will be defined. Normally they will be equal to 1.

5.2 Conditional Compiling

The form of conditional compiling is conceptual somewhat similar to the "Library mode" of the MAC assembler. This means that this facility is especially well suited for extracting modules from a symbolic library.

A module to be possibly included is headed by the command

@ LIB

followed by a logical expression of symbols. For each symbol the compiler maintains an "include"-flag, which is automatically set if the symbol is undefined, and reset when the symbol is defined. However, the pro- grammer can also explicitly put the "include"-flag on or off using the commands

@ STLIB <symbol>   — Set the "library include"-flag

@ NSLIB <symbol>   — Reset the "library include"-flag

The expression after @ LIB can have the operators

/ \   And

\ /   Or

-,   Not

The expression is evaluated from left to right. If the resulting "include"- value is true, the following module will be included, otherwise it will be skipped.

The module is terminated by the command

@ ELIB

The @ LIB - @ ELIB's can be nested. If a module is skipped, it is skipped until its corresponding @ ELIB.

Example:

```
        .
        .
        .
     CALL SUB1
        .
        .
     CALL SUB2
        .
        .
     @ LIB SUB1 \/SUB2        % INCLUDE THE FOLLOWING IF
                              % SUB1 OR SUB2 HAS BEEN
                              % REFERENCED
     SUBR SUB1, SUB2
        .
        .
        .
     @ ELIB                   % UP TO THIS POINT
```

5.3 In-line Assembly Coding

There are two ways of including assembly coding:

a)   If a statement starts with an asterisk (*), the rest of the line
      will be taken as assembly code, being copied to the object
      output stream.

b)   The command

      @MAC

      switches the compiler to assembly mode. The text will pass
      unchanged to the output stream until an at sign (@) is found.

Examples:

```
     *TRA OPR
@ MAC
        BORA 170 DX
@
```

6   USING THE COMPILER

6.1   Preparing NORD PL Programs

The compiler can be used in several contexts, such as coupled to a
MAC assembler to produce immediate binary result, or as a separate
system outputting MAC assembly code to a file or external device.
However, the program itself can look the same in all cases.

In case of absolute programs (not BRF), it is not necessary with any special
heading; the program can start with normal statements.

If on line return to the compiler is wanted, the program should be ended
with the command

@ DEV 1, 0, 0

giving the control back to the operator, which can start a new compilation.
If it is the last part to be compiled, it can instead be ended with

@ EOF   % EXIT FROM COMPILER

Example of program:
% START OF PROGRAM

INTEGER B1, B2

SUBR SUB1
    33=: B1
     5=: B2
RBUS

@ DEV 1, 0, 0

If the resulting program should be output in BRF format, the pertinent MAC
commands)9BEG,)9END,)9ENT and)9EXT should be inserted as assembly
code.

Example:
```
% SUBROUTINE TO PRINT 2 CHARACTERS
        *)9BEG
        *)9ENT OUT2
        *)9EXT OUTBT
SUBR  OUT2
        INTEGER WORD
        INTEGER POINTER LINK
OUT2:       T:=L=: "LINK"
            A=:  WORD SHZ -10        % LEFT BYTE
            T:=5: CALL OUTBT         % LINE PRINTER
            WORD /\ 377; T:=5; CALL OUTBT    % RIGHT BYTE
            GO LINK
RBUS
*)9END
@EOF
```

6.2   Compiling NORD PL Programs

a)   Under TSS:

The compiler is fetched by using the command NORD-PL.
It then writes the message NORD PL < version number >,
waiting for input from the terminal. Then give the DEV
command to set the appropriate devices.

Example:

@ DEV T-R, L-P, F-P

If the compiler is attached to an assembler, the inter-
communication device is number 6.

Example:

```
*6,0,3⌀                    % SET DEVICES FOR THE
                          % ASSEMBLER
@ DEV 2,5,6               % SET DEVICES FOR THE
                          % COMPILER
```

Now the resulting BRF output will come on device no. 3 (punch).

b)   Under NORD-OPS:

The control card is used:

⌀NORDPL (<input device>,<list device>,<output device>)

Example:
⌀NORDPL (2,5,3)

6.3   Diagnostic Messages

6.3.1   Diagnostic Messages from the Compiler

If the compiler detects an error, it prints a diagnostic message on the
list device, preceded by some asterisks. If the list device is equal to
zero, it instead on the communication device prints the name of the last
label and the number of lines after the label, followed by the diagnostic
message. Usually the compilation will continue; however, in a few cases
the compilation has to stop, returning control to the operator (aborting
if NORD-OPS).

Message	Meaning
Error, ill. base	Error in a BASE statement
Error, buffer full	Too long statement or object instruction
Error in command	
Error in compiler	The compiler is destroyed, or may be there is a bug in the compiler
Error, ill. condition	Error in the conditional compiling commands (LIB, SLIB, STLIB or NSLIB)
Error in data expression	Illegal operand or operator in a data expression
Error in decl.	Error in a declaration statement
Error, ill. disp.	Error in a DISP statement
Error, ill. elem.	A basic element is found in a place where it should not be
Error in elem.	An ill-formed basic element
Error, in else/fi	Bad nesting of THEN-ELSE-FI
Error, ill. else/fi/od	Bad nesting of THEN-ELSE-FI or DO-OD
Error in expr.	Error in an executable expression
Error in for	Error in a FOR statement
Error in if	Error in an IF statement
Error in I/O	I/O error signalled by the surrounding system
Error, no FI/OD	Unmatched THEN/ELSE or DO at the end of a subroutine
Error, no (	Missing left parenthesis in a data list
Error, ill. operation	This operation is not implemented in hardware, or non-corresponding operands
Error in output	Error message from the surrounding system

Message	Meaning
Error in relation	Ill-formed relation in an IF or FOR statement
Error in statement start	The statement is illegal in this context, or illegal element in an expression
Frror in subr.	Frror in a SUBR statement ·
Error, table destroyed	Probably overlapping of compiled/assembled program and the compiler's symbol table
Error, table full	Too many symbols in the program
Error, too complex	To complex construction in an executable expression; the backtracking stack is filled
Error, undefined	Undefined local symbols at the end of a subroutine

6.3.2   Diagnostic Messages from the Assembler

Some errors can be detected at assembly time only, because the compiler does not keep track of memory address values. Below is a list of the most usual errors. For more information, see the manual "MAC User's Guide".

Message	Meaning
RANGE EX.	A label or variable is used too far away from where it was defined. It can for example occur for GO to a label defined earlier, or at a OD statement.
POSS. FLT	May be a label has been defined too far after the place where it was used. It can occur for a forward GO or in an ELSE, FI or OD statement. However, this message can occur if an undefined symbol is part of a data expression. Then it can normally be ignored.
(ERROR	Too far between the filling in of literals. The compiler outputs a)FILL command at each RBUS statement. However, the programmer can put *)FILL commands in between.

APPENDIX A

OPERATORS AND RESERVED SYMBOLS

A.1   Non-alphanumeric Elements

<u>Arithmetic Operators</u>

:=	Load
=:	Store
:=:	Swap
-	Subtract
+	Add
*	Multiply
/	Divide
\	Byte separator (Data expressions only)

<u>Logical Operators</u>

/\	And
\/	Or
-,	One's complement

<u>Relational Operators</u>

>	Greater
<	Less
=	Equal
>=	Greater or equal
<=	Less or equal
><	Not equal
>>=	Absolute greater
<<	Absolute less