Norsk Data

Product Information

Product name	Product number
PLANC for MC68000 compiling on ND-500/5000	*211038J*

Description of product

PLANC is Norsk Data's general programming language. The compiler takes as input a source file (containing PLANC statements), and generates an output file containing the relocatable version of the program. This output file is the input to the loader when creating an executable program. PLANC for MC68000 generates output in NRF format.

Reason(s) for new product or version

New functionality, error corrections.

Table of Contents Page

Product Information

Product name	Product number
PLANC for MC68000 compiling on ND-500/5000	*211038J*

1 Prerequisites

In order to use this product, the following are required.

1.1 Hardware prerequisites

CPU type (any of the following):
- 500
- 5000

1.2 Software prerequisites

The following software must be available on the system before the product is installed:

Operating system:

Name	Version/Revision
SINTRAN III	*I or later*

1.3 Other prerequisites

Minimum Permanent Mass Storage:

User	Space (pages)	Number of files
SYSTEM	*9*	*1*
Domain user, old format	*192*	*3*
Domain user, new format	*192*	*1*
Library user	*11*	*1*
Example user	*90*	*18*

2 Dependencies

None.

3 Documentation

Included with the product

Title	Number
PLANC User Guide and Reference Manual	*ND-860117.6*

4 Installation procedure

<rev> is the current revision number.

- Log in as user: SYSTEM
- Insert the diskette 211038J<rev>-XX-01D in the floppy drive.
- Enter the floppy directory as follows:

```
    @ENTER-DIRECTORY
     DIRECTORY NAME: 211038J<rev>-XX-01D ↵
     DEVICE NAME: <device name> ↵
     DEVICE UNIT: <device number> ↵
```

4.1 Automatic installation

- Start the installation program by typing:

```
    @(211038J<rev>-XX-01D:FLOPPY-USER)IN-PLANC ↵
```

- Follow the instructions given by the installation program.

4.2 Non-automatic installation

- Installation of the runtime system:

```
    @DELETE-FILE ↵
     FILE NAME: "PLANC-68020:NRF" ↵
    @COPY-FILE ↵
     DESTINATION FILE: "PLANC-68020-J<rev>:NRF" ↵
     SOURCE FILE: (211038J<rev>-XX-01D:FLOPPY-USER)PL-68020-J<rev>:NRF ↵
```

- Installation of the compiler (old domain format):

```
    @LINKAGE-LOADER ↵
    ND-Linkage-Loader
    N11: DELETE-DOMAIN PLANC-MC68 ↵
    N11: COPY-DOMAIN ↵
    Source-domain: (211038J<rev>-XX-01D:FLOPPY-USER)PLANC-MC68-J<rev> ↵
    Destination-domain: "PLANC-MC68-J<rev>" ↵
    N11: EXIT ↵
```

- Installation of the compiler (new domain format):

```
    @DELETE-FILE ↵
     FILE NAME: "PLANC-MC68:DOM" ↵
    @COPY-FILE ↵
     DESTINATION FILE: "PLANC-MC68-J<rev>:DOM" ↵
     SOURCE FILE:(211038J<rev>-XX-01D:FLOPPY-USER)PLANC-MC68-J<rev>:DOM ↵
```

```
- Making the compiler a standard domain:
     @ND ↵
     ND-500 MONITOR Version x
     N500: DELETE-STANDARD-DOMAIN ↵
     Standard domain name: PLANC-MC68 ↵
     N500: DEFINE-STANDARD-DOMAIN ↵
     Standard domain name: PLANC-MC68-J<rev> ↵
     Domain name: PLANC-MC68-J<rev> ↵
- Installation of the version-information file:
     @DELETE-FILE ↵
      FILE NAME: "PLANC-J<rev>:HELP" ↵
     @COPY-FILE ↵
      DESTINATION FILE: "PLANC-J<rev>:HELP" ↵
      SOURCE FILE: (211038J<rev>-XX-01D:FLOPPY-USER)PLANC-J<rev>:HELP ↵

- Copying the example programs:
     @BACKUP-SYSTEM ↵
      Ba-sy: COPY-USERS-FILES ↵
         Destination type: DIRECTORY ↵
            Destination directory name '' : ↵
            Destination user name 'SYSTEM' : ↵
         Source type: DIRECTORY ↵
            Source directory name '' : 211038J<rev>-XX-01D ↵
            Source user name: FLOPPY-USER ↵
            Source file name '' : :PLNC ↵

         Manual selection: NO ↵

      Ba-sy: EXIT ↵
```

Product name	Product number
PLANC for MC68000 compiling on ND-500/5000	*211038J*

5 Modifications

5.1 New functions/commands

<u>New functionality:</u>

- 16 characters in names default.
  Note! This may cause problems when loading with old libraries.
  Use the compiler command $LONG-NAMES OFF if you want to change the default name length to the old default length of 10 characters.

- Co-routines are now part of the PLANC language.
  They must be declared inside records. To make a co-routine, you must put the modifier PARALLEL in the routine header. Co-routines may have parameters in the same way as ordinary routines. Each co-routine must contain an Inistack statement. You may use an array declared globally or inside a record, or use IND of an integer array pointer as the stack array. This array must have zero as Minindex and you may not use subranges in the Inistack statement. The new standard routines co_Call, co_Detach and co_Resume are introduced to stop and start execution of co-routines.

- Hexadecimal constants (indicated by H as terminator):

  Example:

  ```
  CONSTANT Hex = 0E07H
  ```
  The value of Hex will be 3591 decimal. Hexadecimal constants must start with a digit.

- Repeat factor in array initialization:
  It is now possible to initialize a whole array to the value of a single element or to repeat an element value a specified number of times.

  Example 1:

  ```
  INTEGER ARRAY : IntArray1(0:7) := 5 % All the elements = 5
  ```
  This is the same as
  ```
  INTEGER ARRAY : IntArray1(0:7) := (5,5,5,5,5,5,5,5)
  ```

  Example 2:

  ```
  INTEGER ARRAY : IntArray2(0:7) := (7,(*4)5,(*3)3)
  ```
  This is the same as
  ```
  INTEGER ARRAY : IntArray2(0:7) := (7,5,5,5,5,3,3,3)
  ```

- Equivalence to array elements:
  It is now possible to allow data elements begin at the same storage location as any element in an array.

  Example 1:

  ```
  INTEGER ARRAY: IntArray(0:6)
  INTEGER: Mid = IntArray(3)
  ```

  The variable Mid will start at the same address as IntArray(3).

  Example 2:

  ```
  BITS: BitArray(0:6)
  BOOLEAN: Mid = BitArray(3)
  ```

  This will result in the error message 'Not implemented'. The compiler is not able to generate the correct code to access Mid, unless BitArray and Mid are components in a packed record.

- Compile-time evaluation of BYTES expressions:

  Example 1:

  ```
  CONSTANT Text = 'Hello' // BYTES(15b,12b)
  ```

  The string constant Text will be Hello followed by cr and lf.

  Example 2:

  ```
  $IF LanguageCode = 'N' $THEN
     % ...
  $ENDIF
  ```

  The statements will be compiled if the compile time constant LanguageCode has the value 'N'.

<u>New commands:</u>

- $INCLUDE-PLANC <SOURCE FILE>
  New include statement that takes a PLANC string as parameter.

Example:

Source file (mainfile:plnc):

```
$IF $PRESENT UserArea AND $PRESENT FileName $THEN
   $INCLUDE-PLANC UserArea // FileName
$ELSE
   $MESSAGE The constants UserArea and FileName must be defined !
   $EXIT
$ENDIF
```

```
@PLANC-MC68-J00
CONSTANT UserArea = '(source)'
CONSTANT FileName = 'includefile:incl'
COMPILE mainfile,listfile,outfile
```

If you want to compile the mainfile under a different operating system, you will only have to set the constants UserArea and FileName.

This command has lower priority than $INCLUDE (if ambiguous).

- $MESSAGE-PLANC <MESSAGE>
  New message statement that takes a PLANC string as parameter.

  Example:

  ```
  CONSTANT CrLf = BYTES(15b,12b)
  $MESSAGE-PLANC 'Including: ' // UserArea // FileName // CrLf
  ```

  $MESSAGE-PLANC does not output a CrLf after the message string, so CrLf has been appended to the message.

  This command has lower priority than $MESSAGE-TO-TERMINAL (if ambiguous).

- $TIME
  This command returns a string on the format hh:mm:ss.
  Example:

  ```
  BYTES : Time := $TIME
  ```

  Time is now a string containing the hour, minute and second this statement was compiled, e.g. '15:36:47'.

- $OPTION O
  This command generates output file in format suitable for loading under UNIX. The default output file type will be O instead of NRF.

- $HINTS <ON/OFF/++/-->
  This command generates hints during compilation.
  Example:

```
@PLANC-MC68-J00
*HINTS ON
*COMPILE 1,1,0
     1  MODULE Ex
     2  ROUTINE VOID,VOID: Rout
     3  INTEGER: Int
     4  ENDROUTINE Rout
*FILE* 1
*HINT* at ROUT.4 Not in use "INT"
     5  ENDMODULE
     6  $EOF
     6 Lines compiled.  No diagnostics. Codesize=14 Datasize=0
```

  The variable Int is not needed, and should be removed, to reduce reduce the stack demand for the routine Rout.

New warnings:

- 'Too large decimal constant'
- 'Too large hex constant'
- 'Too large octal constant'
- 'Too large Ada constant'
  An integer literal is too large to be represented in 32 bits.

  Examples:

```
CONSTANT Decimal = 37_777_777_777, Ada = 16#FFFF_FFFFF#
```

- 'Division by zero during evaluation of constant expression'
  Self-explanatory.

- 'Overflow during evaluation of constant expression'

  Example:

```
CONSTANT LargeInt = 2**32
```

  LargeInt is too large to be represented in 32 bits.

- 'Constant value outside range'
  If you initialize or store into a variable the compiler will check the value of the constant against the range of the variable to be initialized.

Example 1:

```
BYTE : Char := 300
```

will result in a warning, since *300* is outside the range for BYTE.

This check is only done on the bit pattern of the variable and not on the actual range.

Example 2:

```
-120 =: Char
```

is legal because it is 210B = 136 as a bit pattern in byte size which is inside the BYTE range.

It is necessary that at least one of the sign bits fit into the range.

Example 3:

```
-200 =: Char
```

is NOT legal because as a bit pattern this is 470B = 312 which is outside the BYTE range. If we strip off one bit, it will become 70B = 56 which fits in the BYTE range but since we have stripped off the last sign bit it is no longer the original value which was negative.

- 'For-loop with zero iterations'
  This warning is given if the upper bound of an implied range is less than the lower bound of the implied range.

  Example:

  ```
  BYTE: Index
  FOR Index IN 1:Max-1 DO ENDFOR
  ```

  This loop is dummy if the constant Max <= 1.

New error messages:

- 'No code generation for MC68000'
  This error message is given if you give the command $CPU-EXTENSION 0, indicating that you want the compiler to generate code for the MC68000.

- 'Illegal character in numerical literal'

Example:

```
CONSTANT number = 39B
```

Octal constants can only contain the digits 0 to 7, and hexadecimal constants must be terminated by H.

- 'Illegal command in macro/inline routine: EXIT/EOF'
  The compiler was unable to handle $EXIT/$EOF in macros/inline routines.
- 'Illegal with in-value'
  This error message is given for routines declared with the modifiers C or STANDARD, if the in-value type is not VOID.

- 'Illegal macro termination'
- 'Illegal routine termination'
- 'Illegal string termination'
- 'Illegal comment termination'
  A macro, inline routine, string or comment was not terminated before end of file was found.

5.2 Changed functions/commands

- Modified interpretation of arguments on command line:
  If your source file name is equal to (an abbreviation of) a compiler command, PLANC-J interprets this as if you had given the compile command, unless the first non-space character is $ or the command line contains a semicolon (;).

  Example 1:

  ```
  @PLANC-MC68-I CON,LIST,OBJECT
  ```

  Interpreted as
  ```
  *CONSTANT,LIST,OBJECT
  ```
  (an error message is given)

  ```
  @PLANC-MC68-J CON,LIST,OBJECT
  ```

  Interpreted as
  ```
  *COMPILE CON,LIST,OBJECT
  ```

  Example 2:

  ```
  @PLANC-MC68 $HELP
  ```

  Interpreted (PLANC-I and PLANC-J) as
  ```
  *HELP
  ```

Example 3:

```
@PLANC-MC68 CONSTANT LIST=TRUE; COMPILE SOURCE,LIST,OBJECT
```

Interpreted (PLANC-I and PLANC-J) as

```
*CONSTANT LIST=TRUE; COMPILE SOURCE,LIST,OBJECT
```

- Modified command:
  $GENERATE-IMPORTS <FILE NAME/ON/OFF/++/-->
  It is now possible to let the compiler generate imports for only a part of your source (by giving ON/OFF/++/-- as parameter).

5.3 Removed functions/commands

None.

5.4 Other modifications

- New stack layout:
  The stack layout has been changed to be C compatible. This makes it easier to mix programming languages and to use C libraries and system calls. The calling sequence will be faster and more compact. The new stack layout is quite different from the old stack layout, so there will be problems in mixing PLANC code using different stack layouts.

- Warning for undefined compile time constants:
  You now get a warning every time you use an undefined flag in a conditional compilation command. This is to avoid errors caused by misspelling the flag.

  Example:

  ```
  $IF Exno500 $THEN
  $ENDIF
  ```

  If you meant Exon500 instead of Exno500, the lines inside the conditional compilation would never be included.

  In large programs with a lot of $IF commands, it will be annoying to get all of these warnings. You are therefore recommended to do the following in the beginning of your source or in a suitable include file:

  ```
  $IF NOT $PRESENT Exon500 $THEN
     $CONSTANT Exon500 = FALSE
  $ENDIF
  ```

Product Information

If you do this for every conditional compilation flag you will only get warnings if you misspell the flag, which ought to be useful.

- File system error messages are now reported in the same way as for syntax errors:
  Example:

```
@PLANC-MC68 TEST,1,0
     1  MODULE Ex
     2  $INCLUDE NO-NAME:IMPT
*FILE* (PACK-SEVEN-6031:TEST-PLANC)TEST:PLNC;1
*ERROR* at EX.2 NO SUCH FILE NAME "NO-NAME:IMPT"
*ERROR* at EX.2 Illegal module termination
      1 Line compiled. 2 Errors.
```

- Source file closed when file system error is detected:
  In the example above, the source file TEST:PLNC is closed when the error NO SUCH FILE NAME is detected. The second error message is a consequence of the first error.

- The compiler now generates an Execution Inhibit NRF-group (IHB) when errors are detected during compilation. This will cause the error message 'Insufficiently compiled program in module' when you try to load the NRF-file, indicating that the compilation failed.

- INLINE routines inside records now have access to the components of the record, just like normal routines inside records.

- Improved type check on parameters to Monitor_Call statement, which allows additional general expressions to be used.

- The syntax is stricter concerning missing semicolons.

  Example:

```
INTEGER: Int
DO WHILE Int<5 ++Int ENDDO
```

  You will get the warning:

```
   Expects "End of statement" Illegal syntax "++"
```

  To remove the warning message, just insert a ";" between "5" and "++".

- Some warning messages are now given as error messages instead:

  Example:

  ```
  MODULE Ex
  Undef POINTER: UndefPtr
  ENDMODULE
  ```

  In PLANC-J, this results in the error message:

  ```
     Data type not specified "UNDEF"
  ```

  In PLANC-I, the same message was given, but only as a warning.

6 Errors corrected

- The last block was not written to the list file when the compiler terminated after a file system error.

- The completion code was not set properly when the compiler crashed after a syntax error had occurred.

- The completion code was not set properly when the compiler was left by giving $EXIT in the command processor.

- Characters were sometimes duplicated in INLINE routines when the character & (ampersand) was read.

- The compiler gave no error message if the argument to $PRESENT was not a variable name (either user defined or predefined).

- $KILL did not work for string constants.

- Multiple PUBLIC statements were not allowed (inside records).

- Comments/blank lines were not allowed before PUBLIC statements.

- If the $SELECT command was used, the compiler could not handle routine names after ENDROUTINE.

7 Errors known but not corrected

None.