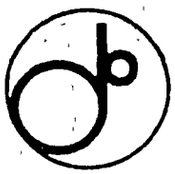


KEK 82-11
November 1982
TRISTAN (A/D)

NODAL INTERPRETER FOR CP/M

Katsunobu OIDE

NATIONAL LABORATORY FOR
HIGH ENERGY PHYSICS

KEK Reports are available from

Technical Information Office
National Laboratory for High Energy Physics
Oho-machi, Tsukuba-gun
Ibaraki-ken, 305
JAPAN

Phone: 0298-64-1171
Telex: 3652-534 (Domestic)
(0)3652-534 (International)
Cable: KEKOHO

NODAL interpreter for CP/M

Katsunobu OIDE

National Laboratory for High Energy Physics
Oho, Tsukuba, Ibaraki, 305 Japan

Abstract

A NODAL interpreter which works under CP/M operating system is made for microcomputers. This interpreter language named NODAL-80 has a similar structure to the NODAL of SPS, but its commands, variables, and expressions are modified to increase the flexibility of programming. NODAL-80 also uses a simple intermediate code to make the execution speed fast without imposing any restriction on the dynamic feature of NODAL language.

1. Language overview

The outline of NODAL-80 interpreter is based on SPS NODAL[1], and some modifications, simplifications, and extensions are done on various parts of the language.

The syntax of the original NODAL of SPS seems a little complicated. For example, in SPS NODAL the meaning of space character differs with commands. It is used to separate parameters in some commands, but in other commands to concatenate two character strings. Exclamation mark (!) means taking a new line in string expressions, error branch specification in DO command, or alternation of a match string in $MATCH command. On the other hand, specifications of similar functions are denoted by different characters. The error branch specification is denoted by exclamation mark (!) in DO command, but by colon (:) when INPUT or OUTPUT function is used. These inconsistencies in command syntax are not only inefficient in the coding of interpreter, but troublesome to users.

In NODAL-80 the language and command syntax are so simplified that an operator or a special character has a definite meaning through the entire command as far as possible. (For instance, All parameters of a command are separated by commas.) The meanings of some special characters are changed from those in SPS, and several characters like '&', '|', and '~' are added to use as logical operators.

The command syntax is also modified in NODAL-80 that it is possible to write an expression as a parameter of a command where a constant can be written. For example, in SPS NODAL ERASE command can erase both program lines and variables, so user

cannot specify the line number by an expression. The following command of SPS NODAL:

```
SET A=1.1; ERASE A
```

does not erase line 1.1 but variable A. (In NODAL-80 a new command Kill is made to erase variables, and ERase command only erases program lines.) In general this rule increases the flexibility of programing, but in some commands it brings inconveniences. For example, in RUn command of NODAL-80 the file name can be any string expression, so user has to use quotation marks to specify a simple file name in the form of string constant, i.e. user has to write

```
RUn 'TEST'
```

to run the program of file 'TEST', and

```
RUn TEST
```

does not run the program of the file 'TEST' but the file whose name is stored in the variable TEST.

Most of file oriented commands are simplified. Some of those commands in SPS are not implemented, but user can do the same function as SPS commands by combining commands or using defined functions. Unlike SPS, NODAL-80 cannot save/load variables directly to/from a file by SAve/LOad command. On the one hand this causes inconvenience, on the other hand this enables to edit a NODAL-80 program file using other text editors of CP/M because the file becomes a usual ASCII file.

The program flow control commands Goto, Do, RETurn, For, ROf, While, WEnd, Repeat, and Until are also modified from those

of SPS (The last three commands do not exist in SPS NODAL). In RETurn command, a line number can be specified to jump after the returning. The repetition abortion commands (Rof, WEnd, and Until) can be accompanied by logical expressions which specify the condition of abortion, because those commands are usualy used to exit conditionally from the current repetition. Goto out of the block which is called by Do command does not terminate Do, so user can extend the range of Do over 99 lines.

In NODAL-80, the length of a name, length of a string value, length of a command line, number of arguments of a defined function, and an array dimension have no restriction except the computer memory size. In particular, the limitlessness in string length increases the power of string manipulation and the ability of run-time modifying of program lines.

Defined function and its related commands are also modified from SPS NODAL. The detail will be mentioned later.

This interpreter works under CP/M[2] operating system, which is a single computer, single user, single task system. So in NODAL-80 there are no network facilities, which is the most excellent characteristic of SPS NODAL. In future, if we have some network of microcomputers, or we use microcomputers as the front end proccessor of a minicomputer which has NODAL, we will install remote-execution commands to NODAL-80.

The main part of this interpreter, which is about a half amount of the whole object code, is coded in FORTRAN-80[3], and the rest in MACRO-80[3] assembler. The program size is almost 28 kilobytes including installed functions. The floating operations and some functions use the library FORLIB[3] of FORTRAN-80.

2. Variables and expressions

(data types)

NODAL-80 has three data types, those are real number, character string, and logical. Integer is not supported in today's version.

Real number has 32 bits length and the same format as the single precision floating of FORTRAN-80. The magnitude is in the range almost from 10^-38 to 10^38, and the precision is almost 7 digits.

Character string is a string of any number of characters. All 7-bit ASCII characters including non-printable characters except null-code (ASCII 00) can be used as the component of character string.

Logical has one byte length and represents logical "true" or "false".

(variables)

NODAL-80 has four kinds of variables, those are

1. simple variable
2. array
3. function
4. defined function.

A variable name is represented by any length of alphabets, digits, ".", "_", and "`". Digits and "." cannot be used as the beginning character of a variable name.

Type of a simple variable and an array is defined when it is

created by commands. (Simple variables are created by set, Ask, $Ask, and For commands, and arrays by DImension, $DImension, and LDImension commands.) Type of a function is defined at the installation time, so it cannot be changed in program. Each function has an attribute defined by installation which is read/call, write, or both. There is a special kind of function, which can return a value of any type, for example the function EVAL belongs to this kind.

A defined function has no fixed type, so the types of returned or received values are determined dynamically by the program.

(constants)

Real constants are denoted by one of the following forms:

a aEe a.Ee a. a.b a.bEe .b .bEe

where a, b, and c are each representing integer, fraction, and exponent respectively. e lies between -38 and 38, and plus (+) sign is optional.

String constants are denoted by one of the following forms:

'c1c2c3 .... cn'

or

"d1d2d3 .... dn"

where c1 ... cn and d1 ... dn represent any characters other than ' or " respectively.

There are no logical constants in NODAL-80, but user can create logical variables which have "true" and "false" in the

start-up command file STARTUP.NOD in the following way:

```
TRUE  = 0 =  0; GLobal TRUE
FALSE = 0 <> 0; GLobal FALSE
```

After this procedure, user can use two logical variables TRUE and FALSE like as constants in all programs including defined functions.

(operators)

The table in the next page shows the operators, operand types, and result types. The term 'any' in this table means they can accept all types, and they always convert the value to string representation before operation. The string representations of logical values are 'true' and 'false'. No type conversion occurs except those cases.

(expressions)

The expression of NODAL-80 is composed of a single operand or series of operands connected by operators. Any number of spaces or tabs can be placed between an operator and an operand. Single operand is one of constant, simple variable, array element, function or defined function reference, and parenthesized expression. Array name without subscripts cannot be used in expression.

operator	operand left	operand right	result
unary -	-	real	real
+	real	real	real
-	real	real	real
*	real	real	real
/	real	real	real
^ (exponentiation)	real	real	real
>	real string	real string	logical logical
<	real string	real string	logical logical
=	real string logical	real string logical	logical logical logical
>= or =>	real string	real string	logical logical
<= or =<	real string	real string	logical logical
<> or ><	real string logical	real string logical	logical logical logical
& (logical and)	logical	logical	logical
| (logical inclusive or)	logical	logical	logical
~ (logical not)	-	logical	logical
space (concatenation)	any	any	string
@ (tabulation)	any or none	real string	string
\ (ASCII conversion)	any or none	real	string
! (new line)	any or none	any or none	string

Array subscripts and function or defined function arguments are also expressions. Those are refered by the form

```
ABC(exp1,exp2, ... ,expn),
```

and spaces or tabs cannot be placed between the name and opening parenthesis (, i.e.

```
ABC (exp1,exp2, ... ,expn)
```

is illegal. There are also such functions which require no arguments, and can be used like as simple variables.

An expression is evaluated with the following order:

1. single operand
2. exponentiation (^)
3. unary minus (-)
4. multiplication (*) and division (/)
5. addition (+) and subtraction (-)
6. concatenation (space), tabulation (@), ASCII conversion (\), and new line (!)
7. comparison (> < = >= <= <>)
8. logical complement (~)
9. logical and (&)
10. logical inclusive or (|)

If two evaluations have the same order, the left one is evaluated first.

There are additional facilities in NODAL-80 expressions:

1) name indirection

Variable names can be indirected in any depth using simple string variables. The indirection is denoted by placing

dollar sign ($) at the head of simple string variable name. For example,

```
A='ABC'
$A='XYZ'     has the same effect as     ABC='XYZ'
$$A=1        has the same effect as     XYZ=1
```

String array cannot be used for indirection, namely, in the following sequence,

```
$DImension A(10); A(1)='ABC'
$A(1)=1
```

the second command line does not mean

```
ABC=1
```

but it causes an error.

2) substring notation

Substring of a string expression is denoted by the following way.

```
string_expression [position]
```

or

```
string_expression [from, to]
```

where position, from, and to are real or string expression representing character position. A real expression specifies the character position directly, counted from the top character as position 1. If a string expression is specified as the position specifier, the first position where the specified string appears is taken.

Example:

```
'ABCDEFG'[2]          means     'B'
'ABCDEFG'[3,5]        means     'CDE'
'ABCDEFG'['BC','EF'] means     'BCD'
'ABCDEFG'[3,'EFG']    means     'CD'
```

3) hexadecimal notation

A hexadecimal number less than four hexadigits are denoted by '#' like as

```
#FF10     #12       #A0A0
```

3. Command syntax

All NODAL-80 program consists of command lines. A command line is an ASCII character string of any length. There are two types of commands, those are number command (for creating or deleting program line) and immediate command. In one line any number of commands can be written using semi-colons (;) as the separator. Capitals and small letters have no differences in a command line except in a string constant. In a command line, a tab character has the same meaning as a space. Other control characters are ignored.

Number command has the following form:

```
line_number command
```

for example,

```
1.1 A=PI; TY A, A^2
```

where line_number is a decimal number between 1.01 to 99.99, and

the fraction part cannot be zero. 'command' is any command including number command itself. This command creates/replaces specified program line of the specified line number. If the command part is null, the specified program line is erased.

Immediate command has the following form:

```
command_specifier [ param1 [,param2 ... ]] [; command ...]
```

for example,

```
DEF ABC(X,Y,Z),10,20  (define ABC) ; T ABC(1,2,3),'Gev'
```

where command_specifier is a command name or its abbreviation. There must be one or more spaces or tabs between the command specifier and the first parameter if exists. The parts of command line enclosed by (and) are regarded as comments, and ignored in the execution of the command.

In almost commands, parameters are separated by commas. (The exception of this rule is the I/O unit specification in the Ask, $Ask, List, and Type commands.) Any expression can be written as the command parameter where a constant is acceptable as the parameter. For instance,

```
List 1, 2.1
```

and

```
A=0; B=10
List COS(A), B-7.9
```

have the same effect on the List command.

The rule of abbreviation of command names is quite simple. First, all command names can be abbreviated from its end as far

as it is possible to distinguish them from other commands. (This is the same way as SPS NODAL.) For example,

```
T    'abcde'
TY   'abcde'
TYP  'abcde'
TYPE 'abcde'
```

have the same meaning. Moreover, some command names can be abbreviated more. See page 14 of this report, where all command names are listed and the shortest form of each command is denoted by capitals. In particular, the shortest forms of set and call commands are null, and the dollar sign is the shortest form of $do command.

(command input and editing)

NODAL-80 reads its commands from terminal or CP/M files. Generally there are no differences between reading from terminal and from a file except for editing.

NODAL-80 has a simple line-editing facility. Those key functions listed below are more powerful than those of common BASIC interpreter, and sufficient to edit one line. Those functions are valid on all inputs from terminal, command input, data input with Ask or $Ask command, and program editing with EDit command.

^A	move cursor to beginning of line
^C	cancel editing
^D	move cursor right
^F	move cursor to end of line

^G	delete character at cursor position
^H (backspace)	move cursor left
^I (tab)	insert tab
^M (return)	end edit and input command or data line
^R	restore the last input line
^S	move cursor left
^T	delete from cursor to end of line
^U	delete from begining of line to cursor
^Y	delete entire line
delete or rub	delete character before cursor

All the other control characters are ignored. All printable characters are inserted at the cursor position. These line-editing functions suppose that the terminal is a video terminal.

The files used in NODAL-80 to store commands, data, or programs are normal ASCII files of CP/M, so user can edit them using other powerful editors which work under CP/M. There are no special rules to edit NODAL-80 files other than 'just like terminal'.

(start-up command file)

NODAL-80 always executes a start-up command file STARTUP.NOD when it is started by CP/M. User can use this file to initialize user's NODAL-80 system, for example to load defined functions, to create global variables, or to set date and time.

4. Command abstract

NODAL-80 has the following 46 immediate commands.

Ask	Bye	call	CANcel
CLEar	CLose	COpy	DEfine
DImension	Do	EDit	ENd
ENTer	ERase	ERRor	EXit
For	GLobal	Goto	If
Kill	LDef	LDImension	List
LOad	LOCal	Merge	Open
RECeive	Repeat	RETurn	ROf
RUn	SAve	SDef	set
Type	Until	Value	WEnd
While	$Ask	$DImension	$do
?	??		

General commands

(set)

set variable = expression

Sets value of the expression to the variable.
If the variable is undefined and is a simple variable, this command creates a new variable which has the specified name, value, and type.

Remarks: The expression must have the same type with the variable. No type conversion occurs.

(Goto)

Goto line

where line : real expression

Moves control to the specified line.

(Do)

Do block [,error ...]

Do line [,error ...]

where

block, line, error : real expression

Executes specified block or line.
One or more alternative blocks or lines can be specified for error handling. If an error occurs inside the first block or line, the second block or line is executed, and so on. This error handling has a priority to the specification of ERRor command.

Remarks: Do command is terminated by end of block (if Do block), end of line (if Do line), or RETurn command. Goto out of the block or line does not terminate Do, so execution continues in the jumped block or line.
ENd, RUn, and Value commands always terminate Do. ROf, WEnd, and Until also terminates Do if the last For, While, and Repeat has been done before the Do command respectively.

(RETurn)

RETurn [line]

where line : real expression

Exits from the last Do command.

If a line number is specified, control goes to the line after exit.

(If)

If condition

where condition : logical *expression*

Executes rest of line when the value of the condition is true.

(For)

For real_variable = initial, final [,step]

where

initial, final, step: real expression

Sets/creates real variable (simple or array element) to the initial value, then repeats commands of the rest of the line. The real variable is increased by the step value (the default is 1), and when the variable becomes larger (if the step is positive) or *smaller (if negative) than the final value, the iteration is over.*

Remarks: *Function and defined function are not allowed to* use as the control variable of For loop.

(ROf)

ROf [condition]

where condition : logical expression

Aborts the last For loop if the condition is true. If the condition is false, control goes to the rest of line.

If no condition is specified, it unconditionally aborts For.

(While)

While condition

where condition : logical expression

Repeats commands of the rest of line while the condition is true.

(WEnd)

WEnd [condition]

where condition : logical expression

Aborts the last While if the condition is true. If the condition is false, control goes to the rest of line.

If no condition is specified, it unconditionally aborts While.

(Repeat)

Repeat [count]

where count : real expression

Repeats the rest of line by the specified count.

The count must be positive or zero, and less than 32767.

If no count is specified, repeats infinitely

Remarks: If the count is specified, the current count of

repetition can be read using the function RPT.

(Until)

Until [condition]

where condition : logical expression

Aborts the last Repeat if the condition is true. If the condition is false, control goes to the rest of line.

If no condition is specified, it unconditionally aborts Repeat.

(call)

call function_reference

Calls the specified function or defined function.

(ENd)

ENd

Terminates program execution.

(Bye)

Bye

Exits to CP/M after closing all files.

($do)

$do string

Executes a string expression as a command string.

Remarks: Writing multi commands in the string is allowed.

(DImension)

DImension name1(subscript1 ...) [,name2(subscript2 ...)...]

where name1,2 : unused names

subscript1,2: real expressions

Creates real arrays with the specified dimensions and sizes.

($DImension)

$DImension name1(subscript1 ...)[,name2(subscript2 ...)...]

where name1,2 : unused names

subscript1,2: real expressions

Creates string arrays with the specified dimensions and sizes.

(LDImension)

LDImension name1(subscript1 ...) [,name2(subscript2 ...)...]

where name1,2 : unused names

subscript1,2: real expressions

Creates logical arrays with the specified dimensions and sizes.

(ERase)

ERase line_range

Erases specified program lines.

The format of line_range is same as List command.

Remarks: Active lines can be ERased. After that, any line specification cannot refer the ERased line, but the commands in the line will be executed until the line becomes inactive.

(Kill)

Kill name1 [,name2 ...]

Kills the variables specified by name1, name2 ...

(CLEar)

CLEar

Kills all local variables.

Remarks: The control variables currently used in For are not killed.

(EDit)

EDit line_range

Edits the specified program lines.
The key commands of EDit are similar to those in the other terminal input except ^R which restores the old line.
The line number cannot be changed by this command. (Use COpy command.)

(COpy)

COpy from, to
where from, to : real expression

Copies from the line specified by 'from' to the line 'to'.
Old line at 'from' remains. If there already exists 'to' line, the old one is erased before copying.

(?)

?

Sets trace ON.

(??)

??

Sets trace OFF.

(ERRor)

ERRor line
where line : real expression

Specifies error branch line.
This specification is canceled when actual branch occurs, or another ERRor command is done. If 0 is specified as the line number, no error branch occurs.

File oriented commands

(Open)

Open unit, file_name
where unit : real expression
file_name : string expression

Opens the specified file with the unit number. If the specified file does not exist, it creates a new file. The unit number ranges from 1 to 127.
The default file type is '.DAT'
The file name for printer is 'LST:'

Remarks: If the same unit number as the already Opened file is specified, the previously opened file is hidden until the new file is closed.

(CLose)

CLose [unit1 [,unit2 ...]]

where unit1,2 : real expressions

Closes the files of the specified unit number.

If no unit number is specified, it closes all files.

(CANcel)

CANcel [unit1 [,unit2 ...]]

where unit1,2 : real expressions

Cancels processing of the currently opened file which has the specified unit number.

On output file, all records previously written are discarded. Old files remain unchanged, and new files are not created.

On input file, CANcel has the same effect as CLose.

If no unit number is specified, it cancels all files.

(Type)

Type [unit:] expression ...

where unit : real expression

Outputs values of expressions on the specified unit (the default is terminal). Unit number 0 is used to specify terminal (same as Ask, $Ask, and List command).

It converts real or logical values to string representations before output. If two or more expressions are specified, a tab character is sent as the delimiter.

(Ask)

Ask [unit:] [prompt,] variable ...

where unit : real expression

prompt : expression except for sigle term variable

Reads values of variables from specified unit (the default is the current command input unit). Unit number 0 is used to specify terminal (same as Type, $Ask, and List command).

If the unit is terminal, prompting string expression (default is ': ') is typed before input.

If an undefined single variable name is specified, Ask creates a new variable.

Remarks: Input string is treated as an expression. User has to use quotation marks to denote string constants (see also $Ask command).

User can input two or more expressions separated by commas in one line.

($Ask)

$Ask [unit:] [prompt,] variable ...

where unit : real expression

prompt : expression except sigle term variable

Reads value of string variable from the specified unit (the default is the current command input unit). Unit number 0 is used to specify terminal (same as Type, Ask, and List command).

If the unit is terminal, prompting string expressions

(default is ': ') are typed before input.

If an undefined single variable name is specified, $Ask creates a new variable.

Remarks: Input line is treated as a string, not as an expression. User can input only one string in one line.

(RUn)

RUn [file_name]

where file_name : string expression

Runs program. If file name is specified, the old program is replaced with that in the specified file.
All local variables are killed before execution.

(List)

List [unit:] line_range

where unit : real expression

Lists program lines on the specified unit (the default is terminal). Unit number 0 is used to specify terminal (same as Ask, $Ask, and Type command).
The format of the line_range is (same as DEfine, EDit, ERase commands):

begin [,end]

where begin and end are real expressions between 0 to 99.99.
The usage of line_range is shown in the next page:

(none)	entire program
1.3	line 1.30 only
1	block 1 (from 1.01 to 1.99)
1.3, 2.57	from 1.30 to 2.57
1.3, 2	from 1.30 to 2.99
0, 1.3	from the beginning of program to 1.30
1.3, 0	from 1.30 to the end of program

(SAve)

SAve file_name

where file_name : string expression

Saves program to the specified file.
The default file type is '.NOD'.

(LOad)

LOad file_name

where file_name : string expression

Changes command input unit to the specified file. This command is usualy used in order to load a program from a file.
The default file type is '.NOD'.

Remarks: Command input file can be nested in any depth. The input unit returns to the original one when an error which is not handled by the program occurs, or the current input file encounters end of file.

Defined function oriented commands

The defined function of NODAL-80 has the following characteristics.

1) The defined function and its arguments have no fixed types. Their types are dynamically determined at execution time. For example, the following defined function

```
1.1 RECeive X; Type X;Value Y X
1.2 Value Y
DEfine ABC(Y)
```

can accept any value of any type as the argument Y. If ABC is used in write mode, ABC can receive any value of any type as the substituted value. Any of the following usages is allowed.

```
A = ABC(1.1E-2)
A = ABC(1 = 1)
A = ABC('Defined function has no fixed type.')
ABC(1.1) = 0.23
ABC(1.1) = 1 = 0
ABC('ABC') = 'Arguments have also no types.'
```

If user requires to distinguish the types of defined function and its arguments, compare them to real, logical, or string constants with the error handling facility.

2) As the arguments of a defined function, user can pass only single values. User cannot pass variables, but the global variables previously declared by GLobal command can be refered by any defined functions. Functions and

defined functions are always global. Real or logical global variables can be read and written, but string global variables execpt function can be only read.

Other local variables are created and refered inside the defined function, so there can exist such local variables which have the same name as those of the main program or the other defined functions. Those local variables created in the defined function are killed when the defined function is terminated.

These features on the local variables and arguments make it completely possible to call a defined function recursively.

3) There are several restrictions on the usage of defined function:

a. A defined function must be terminated by Value command.
b. In the defined function, the program lines of itself cannot be modified.
c. If a defined function is used in write mode, the first command in the defined function must be RECeive command.
d. All files which are opened in the defined function are closed when the defined function is terminated.

(DEfine)

DEfine name[(arg1 ...)] [,line_range]

where name : undefined name
arg1 : dummy argument name

Creates a defined function with the specified name, the arguments, and the lines. The default line_range is the

entire program. The format of line range is same as List command.

Remarks: After DEfine, specified lines are erased from original program. See also ENTer and Merge command.

(RECeive)

RECeive name

Receives right hand value of set command when the defiend function is called in write mode.

If the defined function is called in write mode, this command creates a single variable with the specified name and the received value, then executes commands in the rest of line, otherwise control moves to the next line.

Remarks: Receive command must be placed at the top of the defined function.

(Value)

Value expression

Exits from defined function.

If the defined function has been called in read/call mode, an expression is required to return a value.

(ENTer)

ENTer defined_function_name

Changes program lines to the lines of the specified defined function. After this command all line specification in all commands refer lines of the defined function until another

ENTer command or EXit command is done.

(EXit)

EXit

Changes program lines to the lines of main program. See also ENTer command.

(Merge)

Merge defined_function_name

Merges program lines of the specified defined function to the original program.

(GLobal)

GLobal name1 [,name2 ...]

Changes the variables specified by name1 ... to global variables.

Remarks: Global variables are not killed by RUn or CLEar commands.

(LOCal)

LOCal name1 [,name2 ...]

Changes the global variables specified by name1 ...to local variables.

(SDef)

SDef file_name,defined_function1 [,defined_function2 ...]

where file_name : string expression

Saves the specified defined functions to the specified file. The default file type is '.NOD'.

(LDef)

LDef file_name

where file_name : string expression

Loads defined functions in the specified file.

The default file type is '.NOD'.

5. Intermediate code and working area

(intermediate code)

NODAL-80 uses a simple intermediate code in its execution time to increase the speed. In order to avoid restricting the dynamic feature of NODAL (deleting or creating variables and program lines at the program execution time), the conversion from source to the intermediate code is done line by line. The intermediate code has variable names by characters and has no global informations concerning in other lines. Only the followings are done on the conversion.

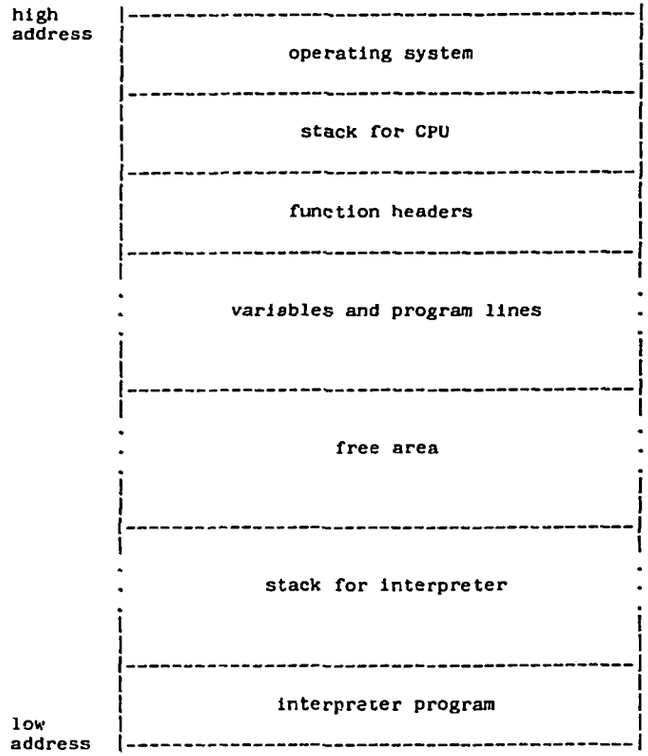
a. to remove comments, trailing spaces, and tabs
b. to convert command names to one-byte codes
c. to discriminate variables and constants
d. to convert real constants into the internal format
e. to convert operators to one-byte codes
f. to convert installed function names to their header addresses

The conversion is done when the line is read from terminal or a file. If a conversion error occurs, an error message is sent to terminal. The source line always remains even if it

causes conversion errors. The intermediate code is processed inside the interpreter, and invisible to user. There are no facilities to save/load the intermediate code to/from a file.

(working area)

NODAL-80 uses the memory as shown in the following figure:



The interpreter program is loaded from #100, and the size is around 28k bytes. The stack for CPU and function headers have

fixed sizes determined at the installation time, and are located under the operating system. The area between the function headers and the interpreter program is dynamically used as the storage of variables and program lines (used from the high address toward the decending direction), and as the stack for the interpreter (used from the low address toward the ascending direction).

Defined functions use the inner free part of memory, when they are called, to allocate their local variables and stacks in the same way as the main program. So the defined functions have to run in the smaller area than the main program, and the size of the available area depends on where and when they are called. During the execution of a defined function, the outer areas are not affected except the contents of global variables. The area used by a defined function is released when it is terminated.

6. Installed functions

real = SIN(real)	sine of a real
real = COS(real)	cosine of a real
real = ATAN(real)	arctangent of a real
real = EXP(real)	exponential of a real
real = LOG(real)	natural logarithm of a real
real = LOG10(real)	common logarithm of a real
real = SQR(real)	square root of a real
real = ABS(real)	absolute value of a real
real = MOD(real1,real2)	modulo of real1 by real2
real = MAX(real,real)	the greater of two reals
real = MIN(real,real)	the smaller of two reals

In the following twelve functions, the real arguments range from -32768 to 32767. Except for INT and FRAC, the arguments are rounded to 16 bit integers before operation.

real = INT(real)	integer part of a real
real = FRAC(real)	fraction of a real
real = ROUND(real)	rounded value of a real
real = AND(real,real)	bitwise AND
real = OR(real,real)	bitwise OR
real = NOT(real)	1's complement
real = IADD(real,real)	integer addition of two reals
real = PEEKB(real)	byte value of specified address
real = PEEKW(real)	word value of specified address
real = PEEKR(real)	real value of specified address
string = PEEKS(real)	string value of specified address
string = HEX(real)	hexadecimal conversion

real = LEN(string)	length of a string
real = INDEX(string1,string2)	character position of string2 in string1
string = SMALL(string)	conversion from capital to small
string = CAPITAL(string)	conversion from small to capital
real = ASCII(string)	the ASCII value of the first character of a string
string = INKEY	senses a character input from terminal, and returns the input character or a null string
real = RPT	current repetition count of Repeat
real = ERROR	reads the last error number
ERROR = real	causes an error of specified number
string = NODLIN(real)	program line of the specified line number
any = EVAL(string)	evaluates string: type of the value returned depends on the argument string
string = TIME	reads/sets time of day with the
TIME = string	format 'hh:mm:ss'
string = DATE	reads/sets date with the format
DATE = string	'yy-mm-dd'

Appendix

(typical execution time with Z80A 4MHz CPU)

NODAL BENCHMARK TEST 82-10-25 21:25:49

LOOP COUNT: 10000

REPEAT	3E-04 SEC
FOR LOOP	7E-04 SEC
SET A = 1	1.2E-03 SEC
SET A = B	1.5E-03 SEC
SET A = A + 1	2.5E-03 SEC
SET A = A - 1	2.6E-03 SEC
SET A = A * 0.9999	3.4E-03 SEC
SET A = A / 0.9999	4E-03 SEC
SET A = A ^ 0.9999	2.61E-02 SEC
SET D1(4) = 1	2.5E-03 SEC
SET D2(4,4) = 1	4E-03 SEC
SET D3(4,4,4) = 1	5.4E-03 SEC
IF B > 1	1.5E-03 SEC
WHILE B <= 1	1.8E-03 SEC
DO 33.3	
33.30 RETURN	3.3E-03 SEC
DO 54	
54.30 GOTO 54.4	
54.40 RETURN	5.4E-03 SEC
SET C = 'A'	2.1E-03 SEC
SET C = 'A' 'B'	3.5E-03 SEC
SET DS(4) = 'A'	3.5E-03 SEC

```
IF C  = 'X'              1.5E-03 SEC

SET $C = 1               2E-03 SEC

$DO "SET A = 1"          8.7E-03 SEC

SET A = SIN(1.001)       1.27E-02 SEC

SET A = SQR(1.001)       1.57E-02 SEC

SET A = DEF(1.001)       8.7E-03 SEC
```

(test program)

```
 1.10 {                                          }
 1.20 {          NODAL BENCHMARK TEST            }
 1.30 {                                          }
 1.40 {               26-Aug-82                  }
 1.50 {                                          }
 1.60 {             Katsunobu  Oide              }
 1.70 {                                          }
 1.80 {          KEK, TRISTAN CONTROL            }
 1.90 {                                          }

 2.10 CLOCK='EVAL(TIME[7,8])+60*EVAL(TIME[4,5])+3600*EVAL(TIME[1,2])'

 3.10 DEF DEF(X),3.3
 3.30 VALUE 0

 5.01 FNAME='';$A 'Output file? ',FNAME;FN=1;I FNAME='';FN=0
 5.02 I FN=1;O 1,FNAME
 5.10 { BENCHMARK MAIN }
 5.20 F I=0,FN;T I: !!              NODAL BENCHMARK TEST      '
DATE ' ' TIME !
 5.40 ASK 'LOOP COUNT? ' ,N
 5.50 IF N <= 0; END
 5.60 TYPE FN: ! 'LOOP COUNT: ' N

 6.10 DIM D1(4),D2(4,4),D3(4,4,4)
 6.20 $DIM DS(4)
 6.30 $DIM ME(25) { MESSAGE BUFFER }

 7.01  ME(1) = ! 'REPEAT                    '
 7.02  ME(2) = ! 'FOR LOOP                  '
 7.04  ME(3) = ! 'SET A = 1                 '
 7.06  ME(4) = ! 'SET A = B                 '
 7.08  ME(5) = ! 'SET A = A + 1             '
 7.10  ME(6) = ! 'SET A = A - 1             '
 7.12  ME(7) = ! 'SET A = A * 0.9999        '
 7.14  ME(8) = ! 'SET A = A / 0.9999        '
 7.16  ME(9) = ! 'SET A = A ^ 0.9999        '
 7.18  ME(10)= ! 'SET D1(4) = 1             '
 7.20  ME(11)= ! 'SET D2(4,4) = 1           '
 7.22  ME(12)= ! 'SET D3(4,4,4) = 1         '
```



```
 7.24  ME(13)= ! 'IF  B > 1                 '
 7.26  ME(14)= ! 'WHILE B <= 1              '
 7.28  ME(15)= ! 'DO 33.3' !     '33.30 RETURN              '
 7.30  ME(16)= ! 'DO 54'!'54.30 GOTO 54.4'!'54.40 RETURN
                                            '
 7.32  ME(17)= ! "SET C = 'A'               "
 7.34  ME(18)= ! "SET C = 'A' 'B'           "
 7.36  ME(19)= ! "SET DS(4) = 'A'           "
 7.38  ME(20)= ! "IF C  = 'X'               "
 7.40  ME(21)= ! 'SET $C = 1                '
 7.42  ME(22)= ! '$DO "SET A = 1"           '
 7.44  ME(23)= ! 'SET A = SIN(1.001)        '
 7.46  ME(24)= ! 'SET A = SQR(1.001)        '
 7.48  ME(25)= ! 'SET A = DEF(1.001)        '

10.10 SET TF=0
10.20 F K=1,25;D 11;F J=0,FN; T J:ME(K) T1' SEC';I K=1;TF = T1
10.30 T !! 'TEST END       ' TIME ! ; END

11.10 SET TO=EVAL(CLOCK)
11.20 DO K+18
11.30 SET T1=(EVAL(CLOCK)-TO)/N-TF;IF K=2;T1=T1+TF
11.40 RETURN

19.10 REPEAT N

20.10 FOR I= 1,N

21.10 REPEAT N; SET A = 1

22.10 SET B = 1; REPEAT N; SET A = B

23.10 REPEAT N; SET A = A + 1

24.10 REPEAT N; SET A = A - 1

25.10 REPEAT N; SET A = A * 0.9999

26.10 REPEAT N; SET A = A / 0.9999

27.10 SET N=N/4; REPEAT N; SET A = A ^ 0.9999

28.10 SET N=N*4; REPEAT N; SET D1(4) = 1

29.10 REPEAT N; SET D2(4,4) = 1

30.10 REPEAT N; SET D3(4,4,4) = 1

31.10 SET B = 0; REPEAT N; IF  B > 1

32.10 REPEAT N; WHILE B <= 1

33.10 REPEAT N; DO 33.2
33.20 RETURN

34.10 REPEAT N; DO 54
```

```
35.10 REPEAT N; C = 'A'
36.10 REPEAT N; C = 'A' 'B'
37.10 REPEAT N; DS(1) = 'A'
38.10 REPEAT N; IF C  = 'X'
39.10 REPEAT N; SET $C = 1
40.10 SET N = N/10; REPEAT N; $DO "SET A = 1"
41.10 REPEAT N; SET A = SIN(1.001)
42.10 REPEAT N; SET A = SQR(1.001)
43.10 REPEAT N; SET A = DEF(1.001)
54.30 GOTO 54.4
54.40 RETURN
```

References

1) M. C. Crowley-Milling and G. C. Shering: CERN 78-07, 1978

2) DIGITAL RESEARCH: An introduction to CP/M features and facilities, 1978

3) Microsoft: FORTRAN-80 reference manual, 1977

Acknowledgement

The author would like to express his sincere thanks to Dr. Tadahiko Katoh and Dr. Shin-ichi Kurokawa for valuable advice on the developement of this interpreter.