THE NODAL COMPILER FOR THE M6800 USERS' GUIDE

edited by

SPS/ACC-Software Section

CONTENTS

INTRODUCTION

1. THE M6800 NODAL COMPILER
   1.1 Compiler operating procedure
   1.2 Compiler list file format
   1.3 Symbol table file
   1.4 System variable file
   1.5 Compiled code particulars

2. MEMORY LAYOUT IN M6800
   2.1 Initialization program

3. LOADING-LINKING PROCEDURE

REFERENCES

APPENDIX I: LIST OF COMMANDS AND FUNCTIONS AVAILABLE TO NODAL COMPILED CODE

Page

1
1
2
3
4
5
7
7
8
10
13
14
INTRODUCTION

A NODAL compiler for the M6800 has been developed and is at present available on the TSS running under the SINTRAN-III system. The compiler is very similar to the one developed for the TMS-9900, but all the additional facilities created around the ACC do not exist for the M6800.

Another difference concerns the output format. For the TMS-9900, the Texas object code is generated, while for the M6800 the CUFOM object code is produced.

Section 1 will describe the operation of the NODAL compiler. The memory layout and initialization of the NODAL program are given in Section 2. Section 3 describes the linking procedure for a compiled NODAL program. Finally, the appendix will list all the commands and functions available to the NODAL compiler.

1. THE M6800 NODAL COMPILER

The NODAL compiler can be called by typing the command

@ NODCOMP-6800

The compiler facility is built up of two parts:

- the compiler itself;
- symbol and system variables predefinition processor.

The compiler compiles a subset of the NODAL language. The subset consists of the DATA handling facilities and mathematical operations. It does not support interactive programming facilities (a feature of an interpreter) and does not support string handling.

The compiler converts the NODAL commands into a set of virtual instructions. As is the case in the NODAL interpreter, the instructions act on data stored in a floating accumulator.

For each separate machine, the virtual instructions have to be converted to the machine code of the target machine involved. For the M6800, a completely N-10 compatible floating format has been used.

The symbol and system variable predefinitions processor allows three operations:

- The definition of system variables with predefined values. They will be located into an area common to all routines compiled in one pass.
- The predefinition of symbol values. In contrast to the variables, their values cannot be changed during the execution of a program. They represent the ASCII representation of certain numbers.
- The predefinition of routines which can be called from the compiled code and which are user dependent.
1.1 Compiler operating procedure

To start the compiler from SINTRAN-III, one types the following:

```
@ NODCOMP-6800 cr
M6800 NODAL COMPILER 1.0
$
```

The command processor is now ready to accept commands. Whenever the command processor expects the operator to enter a command it outputs a dollar sign ($). A command consists of a command name followed by zero or more parameters. Several commands, along with all required parameters, may be written on the same line.

The command name consists of one or more parts separated by hyphens (-). Each part of the command name may be abbreviated as long as the command can be distinguished from all other command names.

- The standard editing characters are available while typing commands.
- The collection of parameters is done in a standardized way as follows.
  - Parameters are separated by either a comma or any number of spaces or a combination of comma and spaces.
  - Parameters may be null, in which case a default value is assigned.
  - When a parameter is missing (as opposed to null) it is asked for, and the command processor expects the operator to supply the required parameter plus more parameters if he so wishes.
  - When a parameter syntax error is detected, an error message is printed and the parameter is asked for.

Excess parameters are ignored.

Commands can be given directly to the SINTRAN-III command processor by preceding them with an @ sign. In this case commands to the local command processor following the SINTRAN-III command are ignored.

HELP (command name)

The HELP command lists available commands on the terminal. Only those commands that have (command name) as a subset are listed. If (command name) is null then all available commands are listed.

EXIT

The EXIT command returns control to the SINTRAN-III command processor.

LINES (lines per page)

The LINES command enables the operator to specify the number of lines per page on the listing device.
LIST (list directive)

The LIST command is used to select various listing options. If a list directive has not been given, then only the source program will be listed. A LIST command with an empty parameter will cause all available list directives to be listed on the terminal.

The following are legal list directives:

SYMBOLS
enables the listing of the symbol table;

GENERATED-CODE
enables the listing of the intermediate code in symbolic form;

DEFAULT
sets the listing mode to the default list mode. In other words, listing of symbols and generated code is disabled.

COMPILE (main program) (functions) (list file) (object file)

The COMPILE command is used to compile a NODAL program on the main program file along with a set of defined functions on the function file. A listing will be generated on the listing file. If a listing file is not supplied, then no listing will be generated, but error messages will be printed on the terminal. Object output will be generated on the object file, if supplied. The default file type for the two input files is :NOD. The default file type for the listing file is :LIST. The default file type for the object file depends upon the individual code generator connected to the compiler.

The main program file may be left out if it is desired to compile only a set of defined functions. Similarly, the function file need not be specified.

SYTBL (input file) (list file)

The SYTBL command is used to process a symbol table file, as will be described in the Symbol Table File section.

SYSVR (input file) (list file) (object file)

The SYSVR command is used to process a system variable file, as will be described in the System Variable File section.

1.2 Compiler list file format

The compiler listing always contains the NODAL program lines. If the GENERATED-CODE option has been specified, then the intermediate, or virtual, code is listed in symbolic form after each program line. At the end of each NODAL program or defined function, a symbol table is output if the SYMBOLS option has been specified.
Page heading

The first three lines of a page constitute the page heading. Before the
heading lines are printed, the listing device is advanced to a new page. If the
listing device is the terminal, a blank line is printed instead of advancing to
the next page. The heading consists of the following fields.

- Compiler name and version number.
- Current date and time.
- Page number.
- The name of the module currently being compiled. This is MAIN for the main
  program and the function name along with formal parameters for a defined
  function.
- One blank line.

Virtual code listing

The listing of the intermediate code, if enabled, consists of several fields
for each line:

- The octal address of the intermediate code. This address is relevant only to
  the implementer of a code generator.
- The label field for a LABEL instruction. An "L" character is output in front
  of the label index value.
- The name of the virtual instruction (see Ref. 1).
- The operands of the virtual instruction in symbolic form.

Error messages

If an error is detected in a line, the error message is output following the
line in error. The error message is preceded by five asterisks (****). If no
listing device has been specified, then the error message also includes the line
number and the name of the defined function or MAIN for the main program in which
the error occurred.

Internal symbol table

If the SYMBOLS option has been supplied, then a symbol table will be listed
after each main program or defined function. This symbol table includes inform-
ation about each parameter, local variable, and local array used in the program.

1.3 Symbol table file

The NODAL compiler provides a facility for adding function and constant defi-
nitions to the internal symbol table SYTBL.
The SYTBL file may contain statements as described below. Statements are
terminated either by end-of-line or semicolon.

% comment
The remainder of the line is taken to be a comment.

CONSTANT name=value, ...
This statement defines the specified name to be synonymous with the specified
value. The value may be any constant expression. Whenever the compiler encounters
this name the associated constant value will be substituted for it.

DEF-MATH fname(nops)=rname, ...
This statement defines the name "fname" to be a mathematical function. The number
of operands "nops" must be one or two. The runtime subroutine which implements
the function is specified by "rname".

DEF-RW fname(parameter list)=rname, ...
DEF-WO fname(parameter list)=rname, ...
DEF-RO fname(parameter list)=rname, ...
DEF-CALL fname(parameter list)=rname, ...
These statements are used to define read/write (type 8), write only (type 9),
read only (type 10), or call (type 11) assembly language functions. The parameter
list may contain from zero to eight of the following parameter type names:

RVAL real value corresponds to 6VAL
IVAL integer value corresponds to 6VAL
SVAL string value not available in compiler
NREF NODAL reference not available in compiler
RREF real reference corresponds to 6VRF
IREF integer reference corresponds to 6VRF
RARR real array corresponds to 6ARF
IARR integer array corresponds to 6ARF
NAME NODAL name corresponds to 6NAM

The runtime subroutine which implements the function is specified by "rname".

END
This statement specifies the end of the symbol table file.

1.4 System variable file

The NODAL compiler provides a facility whereby the user may redefine and
preset variables and arrays which are to be used by the compiled main program and
defined functions. These "system variables" are written, by the code generator,
to a specified object file as a standard NODAL list. This means that other pro-
grams may easily share these system variables by simply searching the NODAL list.
The system variable file may contain statements as described below. Statements are terminated either by end of line or semicolon.

% comment

The remainder of the line is taken to be a command.

CONSTANT name=value, ...
This statement defines the specified name to be synonymous with the specified value. The value may be any constant expression. Whenever the compiler encounters this name the associated constant value will be substituted for it.

REAL name, ...
The specified name is defined to be a system variable. Its value is initialized to zero.

DIM name(number of elements), ...
The specified name is defined to be a floating-point one-dimensional system array. Each element of the array is initialized to zero.

DIM name(index1,index2), ...
The specified name is defined to be a floating-point two-dimensional system array. Each element of the array is initialized to zero.

DIM-I name(number of elements), ...
The specified name is defined to be an integer one-dimensional system array. Each element of the array is initialized to zero.

DIM-I name(index1,index2), ...
The specified name is defined to be an integer two-dimensional system array. Each element of the array is initialized to zero.

SET name=value, ...
The system variable is initialized to the specified value.

SET name(index)=value, ...
The specified one-dimensional system array element is set to the specified value.

SET name(index1,index2)=value, ...
The specified two-dimensional system array element is set to the specified value.

SET name=(value 1, ..., value n), ...
The specified array is initialized, starting from the first element, to the set of specified values. For two-dimensional arrays the first index increases most rapidly.
END

This statement specifies the end of the system variable file.

If the symbol table option is enabled while the system variable file is being processed, then a symbol table of the defined constants, variables, and arrays will be output to the listing file.

1.5 Compiled code particulars

The list of commands and functions available to the NODAL compiler is shown in Appendix I.

The differences between the NODAL interpreter running in the NORD computer and the NODAL compiled code for the M6800 are enumerated below.

i) The function BIT works only on single variables. So when bit 10 should be set in AR(9) the code should look as follows:

   SET B=AR(9); SET BIT(10,B)=1; SET AR(9)=B

ii) An array can only be defined once and should be of a fixed length. For example, the following is not allowed:

   a) DIM-I A(10)   } double definition
   b) DIM-I A(9)    

   a) SET I=9       } variable definition
   b) DIM A(I)      

iii) The GOTO and DO commands are only allowed with fixed numbers. For example, the following is not allowed:

   a) SET I = 2.10
   b) SET K = 3.30
   c) DO I
   d) GOTO K

   This should have been coded as

   a) DO 2.1
   b) GOTO 3.3

iv) The DO 5!6 commands with error return are also not available.

2. MEMORY LAYOUT IN M6800

NODAL defined functions require dynamic memory allocation since they may be used recursively. Therefore, memory allocation in the run-time system is done using a stack as in standard NODAL.
The run-time stack is allocated using the M6800 hardware stack mechanism. A typical memory layout for a M6800 compiled NODAL program is as follows.

```plaintext
<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>
\|---\|---\|---\|---\|---\|---\|---\|---\|---\|---\|---\|---\|---\|---\|---\|
\| Global Area |
\|-----------\|
\| #3FE      |
\|-----------\|
\| Initialization Program |
\|-----------\|
\| Compiled NODAL CODE |
\|-----------\|
\| MAIN      |
\|-----------\|
\| Run-time Routines |
\|-----------\|
\| SWORK     |
\|-----------\|
\| Run-time Stack |
\|-----------\|
\| END FREE MEMORY |
```

The global area may be located anywhere in the address space by changing the definition of the symbol G in the file COMMON-6800, but this means a reassembling of all run-time files and is discouraged. The default value of G is zero. If the global area is located in the first 256 bytes of the address space, then the M6800 assembler produces more efficient code for references to global variables.

The initialization program need only initialize the stack register and then call the compiled code with a JSR MAIN instruction.

2.1 Initialization program

In order to execute the code produced by the M6800 NODAL code generator, an initialization program is required. This program must perform the following functions:

i) initialize the stack register;

ii) call the main program with JSR MAIN;
iii) determine, after return from the NODAL code, whether an error has occurred and take appropriate action.

The following is an example of an initialization program which is designed to be run in conjunction with Motorola's MICRObug monitor.

```
TTL "INIT - INITIALIZE AND START COMPILLED CODE"
NAM INIT
INT INIT
EXT MAIN
INF COMMON

* INIT - INITIALIZE AND START COMPILLED CODE
* RETURN TO MICROBUG

INIT
LDS '5FFFF
JSR MAIN
LDS '03FF                      STACK SPACE OVERWRITES GLOBAL AREA
TSTA
BNE INIT1                      YES
LDX MSGA                      NO
JSR MSG                       OUTPUT "DONE"
JMP MICBUG                    GOTO MICROBUG

INIT1
STAA ERCOD                   OUTPUT "ERROR"
LDX MSGB                   OUTPUT "ERROR"
JSR MSGB                   OUTPUT ERCOD
LDX 'ERCOD                   OUTPUT "AT"
LDX 'LINE                   OUTPUT LINE NUMBER
LDX MSGD                   OUTPUT "IN"
LDAA NAME                   OUTPUT NAME
JSR TCO                      .
LDAA NAME+1                 JSR TCO
LDAA NAME+2                 LDAA NAME+3
JSR TCO                      JSR TCO
LDAA NAME+4                 LDAA NAME+5
JSR TCO                      JSR TCO
JMP MICBUG                    GOTO MICROBUG

MSGA FDB MSG1
MSGB FDB MSG2
MSGC FDB MSG3
MSGD FDB MSG4
MSG1 FCB #0D,#0A,"DONE",#04
MSG2 FCB #0D,#0A,"ERROR",#04
MSG3 FCB "AT",#04
MSG4 FCB "IN",#04
END
```
Upon successful completion of a NODAL program the M6800 A-register will contain zero. If an error occurs in the program, then the A-register will contain an error code. Furthermore, LINE will contain the NODAL line number of the offending statement and NAME will contain the name of the corresponding function.

ERCOD should be used to store the error number.

The following is a list of the error codes as currently defined.

* RUNTIME FATAL ERRORS

<table>
<thead>
<tr>
<th>ERR</th>
<th>EQU</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERR1</td>
<td>1</td>
<td>STACK OVERFLOW</td>
</tr>
<tr>
<td>CERR2</td>
<td>2</td>
<td>ILLEGAL ARRAY DESCRIPTOR</td>
</tr>
<tr>
<td>CERR3</td>
<td>3</td>
<td>ILLEGAL INDIRECT ADDRESS</td>
</tr>
<tr>
<td>CERR4</td>
<td>4</td>
<td>ARRAY INDEX OUT OF BOUNDS</td>
</tr>
<tr>
<td>CERR5</td>
<td>5</td>
<td>WRONG ARRAY TYPE</td>
</tr>
<tr>
<td>CERR6</td>
<td>6</td>
<td>ATTEMPT TO DIVIDE BY ZERO</td>
</tr>
<tr>
<td>CERR7</td>
<td>7</td>
<td>BAD PARAMETER TYPE</td>
</tr>
<tr>
<td>CERR8</td>
<td>8</td>
<td>VALUE OUT OF RANGE</td>
</tr>
</tbody>
</table>

3. LOADING-LINKING PROCEDURE

The object output produced by the M6800 NODAL code generator consists of CERN standard CUFOM object code records. An object module is produced for each main program or defined function and the system variable list if required.

The object code produced by the code generator must be linked together with various run-time modules before the code can be run. This can be done using CERN's CUFOM LINKER program.

The run-time library consists of the CUFOM library file (MOTOROLA)LIBRARY:CUF which is composed of the run-time modules of the following files:

- RUNTIME-6800:CUF  Execution routines for each instruction type.
- NDLIB-9900:CUF  Mathematical function library.
- NDFUN-6800:CUF  Assembly language function library.
- FLOAT-6800:CUF  Floating point package.
- UTILITY-6800:CUF Utility routines for runtime.

If a system variable file has been compiled, then the corresponding object file must also be loaded. If such a file is not loaded then the symbols SYSVR and ESYSV are automatically defined to specify a dummy NODAL list.

The following pages contain a listing of the complete procedure for compiling and loading a sample program which implements Algorithm 423 (Linear Equation Solver) taken from Collected Algorithms from ACM.
@NODCOMP-6800

M6800 NODAL COMPILER 2.0

$SYTBL SYTBL, 1

M6800 NODAL COMPILER 2.0 SYTBL FILE

DEF-CALL CRLF( )=CRLF
DEF-CALL FOUT(RVAL)=FOUT
END

NO ERRORS DETECTED

$SYSVR SYSVR,1,SYSVR

M6800 NODAL COMPILER 2.0 SYSVR FILE

% SYSTEM VARIABLES

DIM A(5,5), B(5)
SET A=(3,5,8,5,6,
  6,7,9,2,4,
  1,8,3,6,4,
  5,7,2,7,4,
  5,2,3,7,7)
SET B=(9,5,6,9,2)
END

NO ERRORS DETECTED

$COM TEST,TFUN,1,TEST

M6800 NODAL COMPILER 2.0 MAIN PROGRAM

1.10 SET N=5
1.20 CALL SOLVE(N,A,B)
1.30 FOR I=1,N; CRLF; FOUT(B(I))

M6800 NODAL COMPILER 2.0 DEFINE-CALL SOLVE(V-N,R=A,R=B)

1.10 DIM IP(10); DECOMP(N,A,IP); IF IP(N)=0; SET ERROR=100
1.15 IF N=1; GO 1.95
1.20 SET N1=N-1
1.25 FOR K=1,N1; DO 2
1.30 FOR KB=1,N1; DO 3
1.95 SET B(1)=B(1)*A(1,1)
1.99 END

2.10 SET K1=K+1; SET M=IP(K)
2.20 SET T=B(M); SET B(M)=B(K); SET B(K)=T
2.30 FOR I=K1,N; SET B(I)=B(I)+A(I,K)*T
3.10 SET K1=N-KB; SET K=K1+1
3.20 SET B(K)=B(K)*A(K,K)
3.30 SET T=-B(K)
3.40 FOR I=1,K1; SET B(I)=B(I)+A(I,K)*T

M6800 NODAL COMPILER 2.0  FRIDAY 18 APRIL 1980 1142:01  PAGE 3
DEFINE-CALL DECOMP(V-N,R-A,R-IP)

1.10 SET IP(N)=1
1.20 FOR K=1,N; DO 2
1.30 END

2.10 IF K=N; GO 2.95
2.15 SET K1=K+1; SET M=K
2.20 FOR I=K1,N; IF ABS(A(I,K))>ABS(A(M,K)); SET M=I
2.25 SET IP(K)=M
2.30 IF M<K; SET IP(N)=-IP(N)
2.35 SET T=A(M,K); SET A(M,K)=A(K,K); SET A(K,K)=T
2.40 IF T=0; GO 2.95
2.45 FOR I=K1,N; SET A(I,K)=-A(I,K)*T
2.50 FOR J=K1,N; DO 3
2.95 IF A(K,K)=0; SET IP(N)=0

3.10 SET T=A(M,J); SET A(M,J)=A(J,J); SET A(K,J)=T
3.20 IF T=0; RET
3.30 FOR I=K1,N; SET A(I,J)=A(I,J)+A(I,K)*T

NO ERRORS DETECTED
TIME USED IS 12 SECS
$EXIT

\[\text{LINKER}\]

ENTER THE NAMES OF THE FILES TO BE LINKED.
ONE NAME PER LINE (REPLY <CR> TO FINISH THIS LIST).
===== INIT
==== TEST
==== SYSVR
====
OBJECT FILE (<CR> = OBJECT:CUF) =====> OBJECT:CUF
LISTING FILE (<CR> = TERMINAL) =====>
DO YOU WANT TO MODIFY SOME OPTIONS? (<CR> = NO) =====> Y
ENTER YOUR OPTIONS:
L=(MOTOROLA)LIBRARY

.. THE LINKAGE EDITION STARTS........

<load map from LINKER>

<<<<<< O ERROR DETECTED IN INPUT MODULES <<<<<<<<

\[\text{PUSH-MOTOROLA}\]

INPUT FILE (<CR> = OBJECT:CUF) =====> OBJECT:CUF
OBJECT FILE (<CR> = OBJECT:MT) =====> OBJECT:MT
LISTING FILE (<CR> = TERMINAL) =====>
DO YOU WANT TO MODIFY SOME OPTIONS? (<CR> = NO) =====> Y
ENTER YOUR OPTIONS:
R=<load address>
.. THE PUSHER STARTS TO WORK....... 

=1=> FROM MODULE : ""
=1=> LINKED (VERSION 1.1) ON NORDSINTRANIII (18 APR 1980)

>>>>>> 0 ERROR DETECTED IN THE CUFOM INPUT FILE <<<<<<<<

<LOAD ADDRESS> (MUST BE BIGGER THAN #3FF)

REFERENCES

2. NITTEDATA, NODAL CODE GENERATOR FOR THE M6800.
## APPENDIX I

LIST OF COMMANDS AND FUNCTIONS AVAILABLE TO NODAL COMPILLED CODE

### COMMANDS

<table>
<thead>
<tr>
<th>Command</th>
<th>Command</th>
<th>Command</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET</td>
<td>CALL</td>
<td>GOTO</td>
<td>DJ</td>
</tr>
<tr>
<td>IF</td>
<td>FOR</td>
<td>RETURN</td>
<td>WHILE</td>
</tr>
<tr>
<td>END</td>
<td>DIMENS</td>
<td>ROF</td>
<td>VALUE</td>
</tr>
</tbody>
</table>

### FUNCTIONS

- ABS
- INT
- FPT
- SGN
- MOD
- LOC
- BIT
- ARG
- SHIFT
- IOR
- AND
- NEG
- ERROR