# IMPLEMENTATION OF NASTRAN ON THE IBM/370 CMS OPERATING SYSTEM*

Stephen S. Britten
M.I.T. Lincoln Laboratory

Betsy Schumacker
M.I.T. Department of Civil Engineering

## SUMMARY

The NASA Structural Analysis (NASTRAN) computer program is operational on the IBM 360/370 series computers. While execution of NASTRAN has been described (ref. 1) and implemented under the virtual storage (VS) operating systems of the IBM 370 models, the IBM 370/168 computer can also operate in a time-sharing mode under the virtual machine (VM) operating system using the Conversational Monitor System (CMS) subset. This report describes the changes required to make NASTRAN operational under the CMS operating system.

---



*This work is sponsored by the Department of the Air Force

## INTRODUCTION AND BACKGROUND

M.I.T. Lincoln Laboratory first obtained NASTRAN in April 1974, when it purchased Level 15.5 from COSMIC for use on its newly installed IBM 370/168 computer. Minor modifications were made in order to make NASTRAN operational under the virtual storage concept of the IBM 370 system. However, much of the source code did not match the special executable load module code that had been received for direct loading onto IBM 3330 disk drives.

The Laboratory's NASTRAN capabilities were upgraded to Level 15.5.3+ in November 1975 with the assistance of NASA Goddard Spaceflight Center. The new source code and new executable load module code was in complete agreement. This Level 15.5.3+ of NASTRAN has adequately served the computational needs of the Laboratory since that original installation.

However, the Laboratory's computer configuration and operational schedule restrict the original version of NASTRAN to nighttime operation under the virtual storage (VS) batch-processing environment. The daytime operating mode (when engineering users are present) consists of an interactive time-sharing environment of up to 170 users under the virtual machine (VM) operating system using the Conversational Monitor System (CMS) subset.

It was desired to have NASTRAN available for general use in the interactive (CMS) as well as the batch (VS) environment, with commonality of input files. From an engineering user standpoint, the availability of NASTRAN in both environments is highly desirable. The CMS environment can be used for input data syntax checking, plotting of input and/or output data, and execution of relatively small analyses; while the VS environment can be used for execution of large analyses that require substantial computer time. The CMS environment is also useful to the programmer for maintenance and development of the many NASTRAN subroutines and functions.

For these reasons, the decision was made to develop a CMS/370 version of NASTRAN, working from the Laboratory's Level 15.5.3+ source. Compatability between the source files, the source listings, and the executable code under both the VS and CMS operating systems was required. Our primary objective was to make NASTRAN operational with minimal changes in computer coding.

This report describes the changes which were made to the NASTRAN computer program to develop a time-sharing version under 370-CMS and a compatible 370-VS1 batch-processing version. In this report, the following terminology is used:

1. CMS-NASTRAN: the version of NASTRAN developed at M.I.T. Lincoln Laboratory to operate under VM/370 (Virtual Machine facility for the IBM S/370) using the CMS (Conversational Monitor System) subset of VM/370.

2. VS1-NASTRAN: the version of NASTRAN developed at M.I.T. Lincoln Laboratory to operate under the VS1 (Virtual Storage) operating system for the IBM S/370.

The following definitions are based on the hardware and software developed by IBM for the implementation of the virtual storage concept.

Address Space - the set of memory addresses used by a program. VS1 allows a maximum total address space of 16 megabytes. CMS allows each user a possible maximum address space of up to 16 megabytes (limited at MIT-LL to 2 megabytes).

Page - a subdivision of the address space, 2K bytes on VS1 and 4K bytes on CMS.

Real Memory - the set of memory addresses which are physically available on the CPU. (3 megabytes on the Lincoln computer).

Virtual Memory - the address space which can be addressed by a relocating CPU. Physically, the virtual memory exists on a direct access storage device, and although a program may reference virtual memory in a random fashion, the information must be transmitted from virtual memory (direct access) to real memory one page at a time.

## CMS DIFFERENCES AFFECTING NASTRAN

The CMS subset of the VM operating system is noticeably different from its VS operating system counterpart, in several areas:

(1) Input/output file structure,

(2) Load module formation, and

(3) Memory management and open core concepts.

These differences between CMS and VS required design decisions regarding implementation of NASTRAN under CMS. In all cases, we attempted to introduce minimal coding changes into the Level 15.5.3+ version of NASTRAN and tried to remain consistent with the original design philosophy regarding NASTRAN operation on IBM computers.

The following sections will deal in specifics concerning the above-mentioned areas of difference and will describe our solutions to the problems

encountered. The final section will mention some discovered coding errors as well as the implementation of a plotting package using the SC4060 plotter.

## Input/Output File Structure

Execution of NASTRAN under the VS operating systems requires a large number of standard Job Control Language (JCL) cards. Modification of the Procedure (PROC) card that precedes these basic JCL cards allows the user flexibility to define space allocations, physical units, and program libraries to be searched.

The CMS operating system has NO Job Control Language cards. File definitions are performed by issuing FILEDEF commands immediately prior to execution time. The FILEDEF command defines the physical device type and the characteristics of the file. All CMS files are always stored and retrieved in 800 byte blocks. We have chosen 6400 byte blocks for the NASTRAN files - a size which will physically fit two blocks per track on an IBM 3330 disk drive as well as be wholly compatible with CMS block sizes. These FILEDEF commands are stored into an EXEC file which can be invoked by a user along with other commands to load and execute CMS-NASTRAN. (refer to Appendix A).

In order to minimize the coding changes required in GNFIAT (the program that generates the file allocation tables), it was necessary to provide a list of acceptable file names as input to the program. Under the VS operating system, acceptable names are chosen from the JCL cards included in the data deck. In CMS, the first card in the IOLIST file indicates (4I2 format) the number of acceptable (permanent, primary, secondary, and tertiary) file names, and the names themselves follow (refer to Appendix B).

Since file chaining is accomplished by CMS, the SPACE limitations needed for VS operations are no longer pertinent under CMS. For this reason, all IOLIST files were allowed to expand in additional 6400 byte blocks as often as needed to provide file space. The VS1-NASTRAN version uses the same file extension scheme outlined (ref. 2) for NASTRAN on the IBM 360-370 operating system.

The POINT macro differs in its operation under CMS from that under VS and thus coding changes were necessary for successful operation of BSAM I/O processing in CMS-NASTRAN.

## Load Module Formation and Usage

The most significant difference concerning load module formation and usage under the CMS operating system is that no provision exists in CMS for an overlay structure. The NASTRAN load module generation under VS attempts to minimize the core requirements of each control routine LINKNSii, ii = 01, ..., 14 by use of overlay structures. Program core space in a virtual machine no longer becomes so critical, and thus provisions for overlay structure under CMS were felt not to be important.

As a simple first approach, the load module generation of all fourteen control routines LINKNSii was carried out with no overlays. While LINKNS01 required 660K for storage, the next largest, LINKNS13, required only 530K. NASTRAN, the super-link module, needs 45K and the system consumes 129K. Allowing for a reasonable work space in open core and providing FORTRAN with sufficient buffer space, CMS-NASTRAN thus requires a minimal core of 1200K. Our present account structure provides us with 2048K which we fully utilize while running CMS-NASTRAN. However, we are eyeing the possibilities of reducing core requirements to 1024K by splitting the larger LINKNSii control routines into multiple modules.

Load module formation under CMS also differs from the process of linkage editing used by VS. The linkage editor under CMS first searches the user's directory for TEXT files with the same names as CSECT entry names and then searches user-specified libraries called TXTLIBs in a specified order. Unlike partitioned-data sets (PDSs) on VS where member names are user-specified, the CMS main entry in a TXTLIB consists of the name of the first CSECT in the program, irrespective of the original name of the compiled program. For this reason, FORTRAN subroutines in NASTRAN containing only BLOCK DATA statements and named COMMONs have the CSECT name of the first named COMMON when compiled and thus that entry name when stored in TXTLIBs under CMS, but have the assigned member name when stored in OBJECT PDS's under VS.

In VS, the linkage editor has input cards - such as INCLUDE and INSERT - which specify specific members to be included in a module from a PDS and specify specific placement of a CSECT within a load module. It was this ability of the linkage editor to have the user specify placement within a module which enabled the developers of NASTRAN to implement the open core concept on IBM batch systems.

In CMS, there is no linkage editor, per se, but rather a loader with two user functions associated with it - LOAD and INCLUDE. The loader builds its own CSECT list which can be resolved from only two places: TEXT files or TXTLIBs. All TEXT file names and entry names in TXTLIBs must therefore be CSECT names. Since the loading process is primarily a search process, the order of CSECTs in a module will depend on the order in which programs were encountered in the search. This absence of user-specified ordering in CMS loading required the removal of certain CSECTs (CONMSG, MAPFNS, EJDUM2, and some named COMMONs) from TXTLIBs and their storage in TEXT files with file names different from CSECT names. These CSECTs will be unresolved externs after the LOAD function but will then be put at the end of the module in a user-specified order by means of the INCLUDE function, making sure EJDUM2 is last since it represents open core.

The CMS linkage editor also differs from its VS counterpart in that it resolves COMMON CSECTs at execution time rather than at load time. For this reason, a way had to be found to force those FORTRAN subroutines containing only BLOCK DATA statements and named COMMONs to be loaded with the other CSECTs.

This was accomplished by writing additional FORTRAN subroutines LINKiiC that did nothing more than CALL the named COMMONs whose CSECT entries existed in the TXTLIBs. This forced inclusion at load time. (Refer to Appendix C.)

A change was also required to LINKNSii and NASTRAN to reference the proper entries in the current FORTRAN function library (e.g. IHOERRM, etc.).

The lengths of all load modules formed in the CMS system are stored in a MASTER file. These lengths can than be used to control the open core requirements described in the next section. Load module generation under CMS reserves the first 128K (origin hex 20000) for the operating system and CMS. All load modules have the user's lowest address originated to 20000, which for NASTRAN execution meant loading the super-link NASTRAN at location 20000. In order to allow for future growth and/or modifications to the super-link NASTRAN, the origin for all LINKNSii entries was taken to be hex location 2AD00. This LINK origin is stored in the MASTER file along with the total load module lengths of all fourteen LINKNSii modules (refer to Appendix D).

Memory Management and Open Core Concepts

Memory management and open core concepts under the VS operating system have been very clearly presented (ref. 1) and will not be reviewed here. Instead, the manner in which the CMS operating system must handle these tasks will be explained.

Under CMS, the super-link NASTRAN load module is initially loaded into core at origin hex 20000 and execution begins. (Figure 1 depicts address space allocation for CMS-NASTRAN execution.)

1. Within the NASTRAN module, a user request for working storage (GETMAIN) will be issued for all of available memory.

2. The NASTRAN module then releases FOURK (16K words under CMS) high address space for operating system use via the FREEMAIN macro.

3. All of the GETMAINed area will be managed by NASTRAN rather than by CMS for all executions of LINKNSii load modules requested by the super-link program NASTRAN.

4. The load modules LINKNS01 thru LINKNS14 have been generated at origin 2AD00 and are loaded with the CMS macro LOADMOD instruction as they are needed.

The above memory management scheme was required since the CMS simulation of VS LINK, GETMAIN, and FREEMAIN macros did not present a duplicate image of core to NASTRAN of that which VS presented. Thus, the macro for loading a module into core could not be LINK but had to be LOADMOD. This in turn said memory had to be obtained prior to and for the LOADMOD. Since the LOADMOD forced CMS to obtain buffer space for itself, we were forced to resort to a single GETMAIN with total core management performed by NASTRAN so that fragmentation of memory (into NASTRAN blocks intersperced with system blocks) was eliminated.

Since CMS-NASTRAN has now assumed the responsibility of managing memory, the "open core" concept of data management must also be assumed by NASTRAN. Under the overlay load module structure of VS, "open core" at the end of an overlay tree segment was denoted by a dummy named-COMMON section. This named-COMMON could contain an array whose length extended into the open core region but yet would not destroy or overwrite code or data in other segments of the overlay. Unfortunately, under CMS with no overlay structure, we are faced with numerous dummy named-COMMONs, each of which must be located near the end of the load module so that it will not overwrite other code and/or data in the module. The placement of these named-COMMONs is accomplished by INCLUDing newly-written FORTRAN subroutines, LINKiiCC, when doing the load module generation (refer to Appexdix E).

Minor Differences

Several subroutines contain DATA statements which are used to initialize variables which are subsequently modified. The DATA statements were used assuming that the programs containing these statements exist on segments of the overlay tree in VS, implying that a fresh copy of the subroutine will be loaded (and thus reinitialization of DATA variables will occur) each time the segment is needed. Under CMS, the subroutine when it is reentered contains the last values of the DATA variables and they are not reinitialized, thus causing errors. This problem would also occur in a non-overlay VS version of NASTRAN.

The graphics package acceptable to the CMS environment at M.I.T. Lincoln Laboratory utilizes SC4060 meta code to generate the meta code for on-line Tektronix terminals. For this reason, a SC4060 plotting package was added to the NASTRAN package. Since the CMS environment provides disk file facilities for graphics as well as standard I/O, restrictions on graphics files in NASTRAN were removed so that disk file definitions for PLT1 and/or PLT2 files are permissible.

Conclusions

The implementation of CMS-NASTRAN has been completed. While operation has been limited to test examples and several small production runs, it definitely shows promise as a useful program on future projects in which

NASTRAN will be needed. Program enhancements (such as the SC4060 graphics package) are easily made while working in the CMS environment and can be added to the VS-NASTRAN after successful debugging has been completed.

A new VS1-NASTRAN is being generated at the present time. It will embody the design philosophy of CMS-NASTRAN and, thus, will not be in overlay structure. Compatibility of source (except for several assembly language programs) and executable code between the two systems should provide for the reliability needed to assure identical results in either mode.

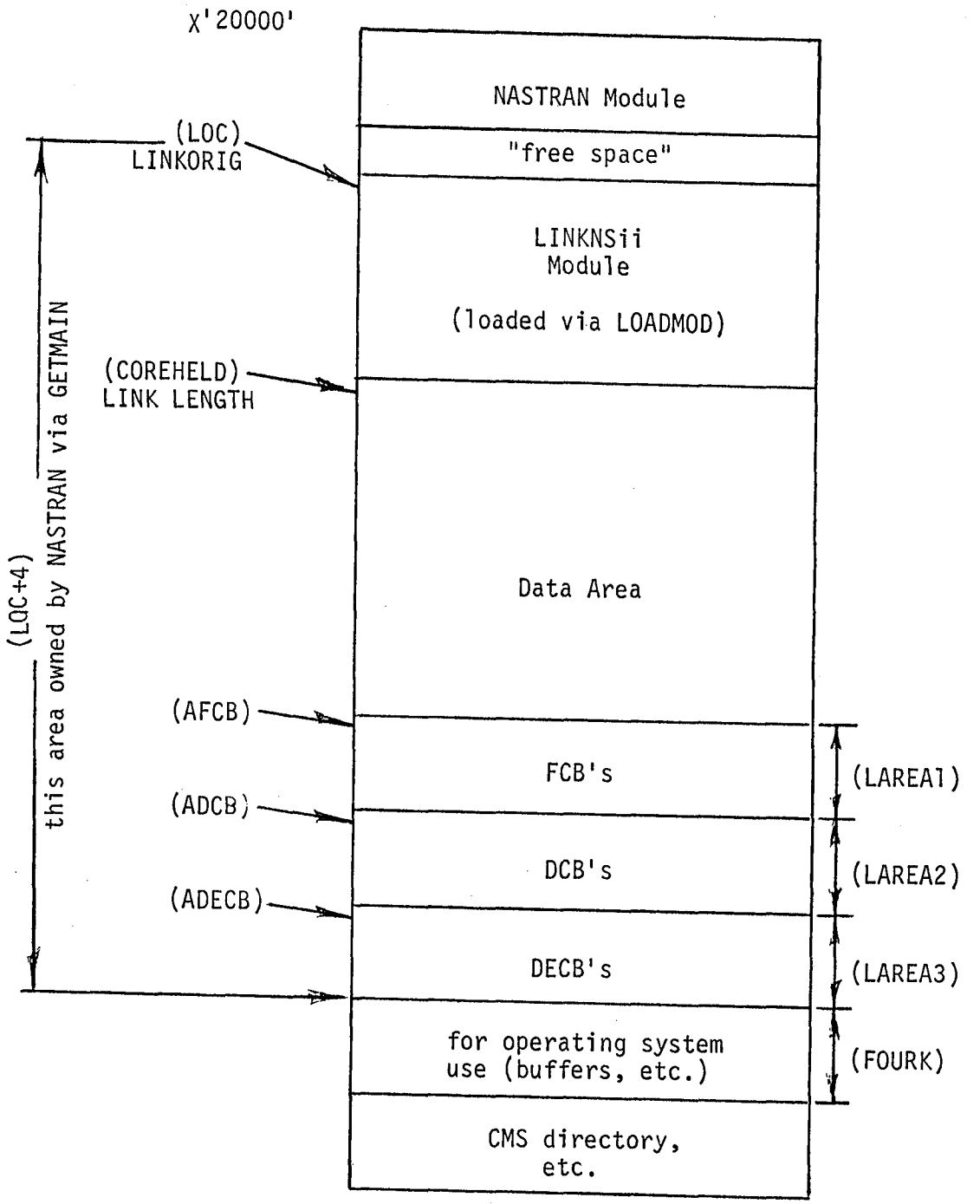
It is anticipated that CHKPNT saves and restarts should be possible between the CMS and VS operating systems. This would allow preprocessing (data syntax checking, undeformed structural plots, etc.) to be performed and check pointed under CMS; restored, executed, and checkpointed under VS; and finally restored and postprocessed (including deformed structural plots) under CMS.

## REFERENCES

1. McCormick, C. W. and Redner, K. K.: "Study of the Modifications Needed for Effective Operation of NASTRAN on IBM Virtual Storage Computers", NASA CR-2527, 1975.

2. Anon. The NASTRAN Programmer's Manual (Level 15.5), NASA SP-223(01), May 1973.

Figure 1 CMS Allocation of Address Space For NASTRAN



## Appendix A: Partial Listing of NASTRAN EXEC File

```
&CONTROL OFF NOMSG
-START &TYPE ENTER FILENAME OF NASTRAN FILE
&READ ARGS
&IF &INDEX NE 1 &GOTO -START
&FN = &1
-CONT &TYPE ENTER FILEMODE FOR NASTRAN OUTPUT FILES
&READ ARGS
&IF &INDEX NE 1 &GOTO -CONT
&FM = &1
FILEDEF 1      DISK    FT01  NAST     &FM    (BLOCK   80 RECFM F LRECL   80
FILEDEF 4      DISK    FT04  NAST     &FM    (BLOCK  133 RECFM F LRECL  133
FILEDEF 5      DISK    &FN   NASTRAN
FILEDEF 6      DISK    &FN   OUTPUT   &FM    (BLOCK  133 RECFM F LRECL  133
FILEDEF 7      DISK    &FN   PUNCH    &FM    (BLOCK   80 RECFM F LRECL   80
FILEDEF POOL   DISK    POOL  NAST     &FM    (BLOCK 6400 RECFM F
FILEDEF NPTP   DISK    NPTP  NAST     &FM    (BLOCK 6400 RECFM F
FILEDEF OPTP   DISK    OPTP  NAST     &FM    (BLOCK 6400 RECFM F
FILEDEF PLT2   DISK    &FN   4060     &FM    (BLOCK  800 RECFM F LRECL  800
FILEDEF PRI01  DISK    PRI01 NAST     &FM    (BLOCK 6400 RECFM F
FILEDEF PRI02  DISK    PRI02 NAST     &FM    (BLOCK 6400 RECFM F
FILEDEF PRI03  DISK    PRI03 NAST     &FM    (BLOCK 6400 RECFM F
          •                                          •
          •                                          •
          •                                          •
          •                                          •

FILEDEF PRI30  DISK    PRI30 NAST     &FM    (BLOCK 6400 RECFM F
FILEDEF PRI31  DISK    PRI31 NAST     &FM    (BLOCK 6400 RECFM F
FILEDEF PRI32  DISK    PRI32 NAST     &FM    (BLOCK 6400 RECFM F
FILEDEF SEC01  DISK    SEC01 NAST     &FM    (BLOCK 6400 RECFM F
FILEDEF SEC02  DISK    SEC02 NAST     &FM    (BLOCK 6400 RECFM F
FILEDEF SEC03  DISK    SEC03 NAST     &FM    (BLOCK 6400 RECFM F
FILEDEF TER01  DISK    TER01 NAST     &FM    (BLOCK 6400 RECFM F
LOADMOD NASTRAN

START
```

## Appendix B: Listing of NASTRAN IOLIST File

```
04320301
PERMPOOL
PERMNPTP
PERMOPTP
PERMPLT2
PRIMPRI01
PRIMPRI02
PRIMPRI03
PRIMPRI04
PRIMPRI05
PRIMPRI06
PRIMPRI07
PRIMPRI08
PRIMPRI09
PRIMPRI10
PRIMPRI11
PRIMPRI12
PRIMPRI13
PRIMPRI14
PRIMPRI15
PRIMPRI16
PRIMPRI17
PRIMPRI18
PRIMPRI19
PRIMPRI20
PRIMPRI21
PRIMPRI22
PRIMPRI23
PRIMPRI24
PRIMPRI25
PRIMPRI26
PRIMPRI27
PRIMPRI28
PRIMPRI29
PRIMPRI30
PRIMPRI31
PRIMPRI32
SECOSEC01
SECOSEC02
SECOSEC03
TERTTER01
```

## Appendix C: Listing of Typical LNKiiC FORTRAN Subroutine

```
      SUBROUTINE LNK01C
C
C     SUBROUTINE RPDABD
            CALL XSFA1
C     SUBROUTINE IFXDBD
            CALL IFPDTA
C     SUBROUTINE IFX0BD
            CALL IFPX0
C     SUBROUTINE IFX1BD
            CALL IFPX1
C     SUBROUTINE IFX2BD
            CALL IFPX2
C     SUBROUTINE IFX3BD
            CALL IFPX3
C     SUBROUTINE IFX4BD
            CALL IFPX4
C     SUBROUTINE IFX5BD
            CALL IFPX5
C     SUBROUTINE IFX6BD
            CALL IFPX6
C     SUBROUTINE IFX7BD
            CALL IFPX7
C     SUBROUTINE IFPABD
            CALL IFP1A
C     SUBROUTINE AXICBD
            CALL IFP3BD
C     SUBROUTINE XGPIBD
            CALL XGPIC
C     SUBROUTINE XMPLBD
            CALL XGPI2
C     SUBROUTINE XSORBD
            CALL XSRTBD
C     SUBROUTINE UMFZBD
            CALL UMFZZZ
C     SUBROUTINE XBSBD
            CALL XLKSPC
C
      RETURN
      END
```

## Appendix D: Listing of NASTRAN MASTER File

```
LINKORIG 02AD00
LINKNS01 0D00A8
LINKNS02 08C580
LINKNS03 092F50
LINKNS04 072030
LINKNS05 094F00
LINKNS06 078218
LINKNS07 078428
LINKNS08 0976B0
LINKNS09 054950
LINKNS10 073AD8
LINKNS11 08D8D0
LINKNS12 05C688
LINKNS13 0AF908
LINKNS14 07A390
```

Appendix E: Listing of Typical LNKiiCC FORTRAN Subroutine

```
C       DUMMY TO FORCE XGPI1 TO COME JUST BEFORE EJDUM2 (OPEN CORE)
        BLOCK DATA
        COMMON DUM4(100)
        COMMON /GINOX/ DUM6(163)
        COMMON /SETUP/ DUM2(7)
        COMMON /IFP3LV/ DUM10(104)
C       BEGINNING OF DUMMY COMMONS THAT MARK START OF OPEN CORE.
        COMMON /IFP1X/ DUM9(371)
        COMMON /ENDSSS/ DUM7(2)
        COMMON /IFPXX/ DUM8
        COMMON /IFP3ZZ/ DUM11
        COMMON /IFP4ZZ/ DUM12
        COMMON /IFP5ZZ/ DUM13
        COMMON /XCSABF/ DUM14
        COMMON /ESOPT/ DUM
        COMMON /UMFXXX/ DUM1
        COMMON /ESFA/ DUM5
        COMMON /XGPI1/ DUM20(5)
        COMMON /EJDUM2/ DUM21
        END
```