MULTIMISSION MODULAR SPACECRAFT GROUND SUPPORT SOFTWARE SYSTEM (MMS/GSSS) STATE-OF-THE-ART COMPUTER SYSTEMS/COMPATIBILITY STUDY

MAY 1980

NASA
National Aeronautics and Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

SOFTWARE ENGINEERING LABORATORY SERIES SEL-80-003

MULTIMISSION MODULAR SPACECRAFT GROUND SUPPORT SOFTWARE SYSTEM (MMS/GSSS) STATE-OF-THE-ART COMPUTER SYSTEMS/COMPATIBILITY STUDY

MAY 1980

NASA

[illegible]

Goddard Space Flight Center

[illegible]

FOREWORD

The Software Engineering Laboratory (SEL) is an organization sponsored by the National Aeronautics and Space Administration Goddard Space Flight Center (NASA/GSFC) and created for the purpose of investigating the effectiveness of software engineering technologies when applied to the development of applications software. The SEL was created in 1977 and has three primary organizational members:

NASA/GSFC (Systems Development and Analysis Branch)
The University of Maryland (Computer Sciences Department)
Computer Sciences Corporation (Flight Systems Operation)

The goals of the SEL are (1) to understand the software development process in the GSFC environment; (2) to measure the effect of various methodologies, tools, and models on this process; and (3) to identify and then to apply successful development practices. The activities, findings, and recommendations of the SEL are recorded in the Software Engineering Laboratory Series, a continuing series of reports that includes this document. A version of this document was also issued as Computer Sciences Corporation document CSC/TM-80/6154.

Contributors to this document include

Todd Welden	(Computer Sciences Corporation)
Mike McClellan	(Computer Sciences Corporation)
Paul Liebertz	(Computer Sciences Corporation)

Single copies of this document can be obtained by writing to

Frank E. McGarry
Code 582.1
NASA/GSFC
Greenbelt, Maryland 20771

ABSTRACT

This report concerns itself primarily with the compatibility of the Multimission Modular Spacecraft (MMS) Ground Support Software System (GSSS), currently operational on a ModComp IV/35, with the VAX 11/780 system. The compatibility is examined in various key areas of the GSSS through the results of in-depth testing performed on the VAX 11/780 and ModComp IV/35 systems. In addition, the compatibility of the GSSS with the ModComp CLASSIC is presented based upon projections from ModComp-supplied literature.

TABLE OF CONTENTS

SECTION 1 - INTRODUCTION

The Multimission Modular Spacecraft (MMS) Ground Support Software System (GSSS) was developed and is currently operational in a ModComp IV/35 hardware and software environment. The dependencies of the GSSS upon this environment have previously been enumerated in the MMS/GSSS ModComp Device and MAX IV Dependency Study.[1] This compatibility report concerns itself primarily with the compatibility of the currently operational GSSS with two more advanced minicomputers with approximately the same capabilities as the ModComp IV/35: the ModComp CLASSIC and the DEC VAX 11/780. The compatibility is examined through the results of in-depth testing in the various key areas of the GSSS performed on both the ModComp IV/35 and VAX 11/780 systems. In addition, the compatibility of the GSSS with the ModComp CLASSIC is presented based upon projections from ModComp-supplied literature and discussions with ModComp CLASSIC users. The most significant portion of the compatibility study involved the transporting of the STOL module and a few associated "key-in" modules, from the ModComp to the VAX. It was through this vehicle that the important areas of GSSS intertask communication and activation were investigated. Additional differences in the FORTRAN language implementations were also discovered during the transport of the STOL module. Since the STOL module has been

[1] CSC Document# CSC/TM-80/6013, "Multimission Modular Spacecraft Ground Support Software System (MMS/GSSS) MODCOMP Device and MAX IV Dependency Study", T. Welden and M. McClellan, December 1979.

successfully transported to the VAX system, this can be used as a nucleus of the VAX version of the CSSS should the decision be made to transport the entire system.

This report describes the methods used in gathering the data, the compatibility of the peripheral devices, the results of the testing, the compatibility of the application languages, and the compatibility of the vendor supplied software, and annotates pertinent conclusions based upon the data gathered.

SECTION 2 - METHODS USED IN GATHERING DATA

The compatibility of the ModComp implementation of the GSSS was studied primarily through computer-based testing on the ModComp IV/35 and VAX-11/780 computers. These tests consisted of small specific benchmarks - to be referred to simply as the benchmarks - and a larger benchmark: the STOL module implementation.

The benchmarks tested three general areas: I/O and file manipulation, FORTRAN programming language semantics, and timing the overhead of common but sometimes crucial operations. The I/O and file manipulation tests concerned sequential and random disk files, formatted and unformatted tape files, and tape file positioning. The FORTRAN programming language tests concerned the memory representation, manipulation, and comparison of numbers and character strings, as dependent on their definition in DATA statements, variable data types, array EQUIVALENCEing, and compiler options. Also tested in the FORTRAN language was array storage organization. Finally, the timing benchmarks covered polynomial evaluation and the overhead involved in the following operations: READ/WRITE for mailboxes (message passing), OPEN/CLOSE and READ/WRITE for files, FORTRAN subroutine calls, and system services.

The STOL module implementation, while using the knowledge gained from the benchmarks, focused on developing the systems programming techniques needed to run GSSS processes (tasks) under real-time control.

The techniques are:

- Interprocess (intertask) communication
- Process (task) activation
- Shared (global) regions.

All three techniques employ system services and utilities. The first two do so via a developer-written host-system GSSS library routines, while the third uses the host system utilities. Table 2-1 summarizes the information about the MODCOMP software (i.e., library routines, and the REX services they reference) used to realize these techniques; it also gives the corresponding VAX system services (or library routines) used. In this table the VAX term, process, is used in place of the equivalent ModComp term, task.

ORIGINAL PAGE IS
OF POOR QUALITY

Table 2-1. Software for Process Communication/Activation and for Shared Regions (1 of 2)

Function	MODCOMP & VAX Routine Name (REX Service)	VAX System Service Used
Send a message to another process.	SNDMSG (SEND)	CREMBX: create a mailbox QIO: write message to mailbox
Receive a message sent from another process by SNDMSG.	RECVMS (RECEIVE)	GETJPI: retrieve infomation about a process (e.g., process name). ASSIGN: Assign I/O channel to a mailbox. QIO: read message from mailbox.
Activate a process immediately.	ACTIV8 (ACT)	CREPRC: create a process.
Suspend a process for a specified period of time.	WAIT (DELAY)	SETIMR: set an event flag after a specified period of time. WAITFR: place calling process in wait state until event flag is set.
Retrieve information about a process.	INFO4 (GETTASK)	GETJPI: retrieve information about a process. (Not totally compatible with MODCOMP).
Convert 3-character ASCII string to CANCODE.	ISCAN (ATCAN)	GSSS library routine specially written for the VAX.
Convert 6-character ASCII string to CANCODE.	ISDCAN (ATCAN)	GSSS library routine specially written for the VAX.

ORIGINAL PAGE IS
OF POOR QUALITY

Table 2-1. Software for Processing Communication/Activation and for Shared Regions (2 of 2)

Function	MODCOMP & VAX Routine Name (REX Service)	VAX System Service Used
Convert 3-character CANCODE to ASCII string.	DECAN3 (CANTA)	GSSS library routine specially written for the Vax. (This function not needed on MODCOMP GSSS).
Convert 6-character CANCODE to ASCII string.	DECAN6 (CANTA)	GSSS library routine specially written for the VAX. (This function not needed on MODCOMP GSSS).

ORIGINAL PAGE IS OF POOR QUALITY

SECTION 3 - COMPATIBILITY OF PERIPHERAL DEVICES

The standard peripheral devices that are necessary to support the GSSS in the ModComp IV/35 environment consist of a moving head disk, a fixed head disk, two 9 track magnetic tape drives, one KCRT control console, at least two KCRT display consoles, a card reader and a printer. In addition to these standard devices there are specialized devices: four parallel I/O ports for telemetry and commanding, one A/D input port, and 16 D/A output ports.

Since it was not expected that any other machine used for benchmark testing would have the specialized devices attached, this report addresses only the areas of standard peripheral devices.

3.1 MAGNETIC TAPE DRIVES

The ModComp IV/35 system is currently configured with two 9 track, 75 inches per second magnetic tape drives. One operates only at 800 BPI. The other is dual density and operates at either 800 or 1600 BPI. The VAX is configured with one dual density tape drive. Any I/O reference to any tape drive on the ModComp will cause I/O to be attempted to that drive. No special Job Control Language (JCL) commands or coding techniques are required to accomplish this (other than assigning the tape drive).

On the ModComp CLASSIC system the tape drives operate identically to those on the ModComp IV/35 from the users standpoint.

On the DEC VAX 11/780, however, tape access is accomplished with an entirely different philosophy. Any tape is considered a new "volume", just like mounting a different disk pack under supervisor control. This philosophy forces the user (without special coding in assembler) to issue a mount command in order to access the tape drive. In addition, the <u>default</u> tape format is "standard ASCII labeled" format which is not expected or accepted by any module of the GSSS. The GSSS expects all tapes to be unlabeled. This can be accomplished on the VAX by explicitly requesting "UNLABELED" on the mount command. Another aspect of the tape drive configuration that differs from the ModComp on the VAX is tape density (800 or 1600 BPI). On the ModComp the density is set manually by a switch on the tape drive unit. This removes the concern about tape density from the program and its assignments. However, on the VAX the tape density must be specified on the mount command. This implicates different sets of JCL commands for any program accessing tapes that can be of either density.

There is one more difference implicated by the VAX tape philosophy. On the ModComp any number of assignments from one or numerous programs can be made to the same tape drive using only assignment statements. On the VAX this can be accomplished but requires a special JCL sequence as follows:

```
$MOUNT/NOLABEL/DENSITY=800 MTA0: FOR004 FOR004
$ASSIGN FOR004 FOR003
$ASSIGN FOR004 FOR006
$OPEN/WRITE FOR004 MTA0:
```

ORIGINAL PAGE IS
OF POOR QUALITY

This sequence assigns the FORTRAN logical units 3, 4 and 6 to the tape drive MTA0. Note that this tape is opened for writing and reading. If another process wishes to access it or later additional assignments must be made, a $CLOSE JCL command must be issued and a new JCL command sequence input. In addition, all processes accessing the tape must explicitly "OPEN" it as "shared".

3.2 KCRT DISPLAYS

The KCRT displays on both ModComp systems and VAX system appear to be compatible when displaying information. However, due to the different set up of the keyboard there is a compatibility problem. On the ModComp systems the keyboards operate in message mode, transmitting a complete line of text on each transmission. On the VAX they operate in character mode transmitting one character at a time as each is typed. This mode of operation precludes updating display screens while an input request is pending on that device. This problem can probably be overcome with some specialized coding techniques for KCRT input involving queuing the characters until the entire string is input. Further analysis needs to be done in this area.

3.3 DISK FILES

The ModComp IV/35 is currently configured with one 24-megabyte moving head disk and one 2-megabyte fixed head disk, each with a sector size of 256 bytes. The ModComp CLASSIC can have disks of a similar nature. The VAX 11/780 used for the benchmark testing is equipped with one 176 megabyte moving head disk with a sector size of 512 bytes.

ORIGINAL PAGE IS
OF POOR QUALITY

Accessing the disks on any of the machines requires no special coding techniques. On the ModComp IV/35 disk files are a fixed partitioning of the disk determined at "Sysgen" time. The disk files can also be generated this way on the ModComp CLASSIC. On the VAX 11/780, and optionally on the ModComp CLASSIC, disk files can be dynamically allocated. This dynamic allocation of disk files is a desirable feature. However, on the VAX system, disk files are dynamically allocated automatically by simply opening a file for output. Without special coding techniques to avoid this, the proliferation of files could be colossal on a real-time system like the GSSS since many tasks write to disk files and then exit (e.g. OBC dump collector). On the VAX, each time one of these tasks ran, a new file would be created.

On the ModComp IV/35 the access method used on any file can be sequential or random and the physical files can have any number of end-of-file marks. On the VAX system random files cannot have any end-of-file marks. This may effect the command processing in the GSSS. However, the VAX system provides for automatic record blocking to and deblocking from the disk files while the ModComp IV/35 does not.

3.4 LINE PRINTER

The ModComp IV/35 system is equipped with a 600 line per minute printer. The VAX 11/780, used for the benchmark tests, is equipped with a DECWRITER III for printing which is much slower. Other than for speed, these two devices appear to be totally compatible for printing ASCII characters. However, to operate effectively the GSSS will require a printer with at least the capability of 600 lines per minute (100 mls/line).

3.5 CARD READER

The ModComp IV/35 system is equipped with a 300 card per minute card reader. The VAX 11/780 is not equipped with a card reader. Therefore no statement can be made other than that the fully operational GSSS requires a card reader in its current configuration (e.g. for, DBPARS, DBXREF, BLDTAB, BLDHAZ, etc.).

SECTION 4 - RESULTS OF TESTING

During the course of the compatibility study, numerous tests have been run on both the ModComp IV/35 and VAX 11/780 computer systems. All of these benchmark tests have been accomplished using the FORTRAN language. Many obstacles had to be overcome during the running of these tests, partially due to the differences in the two systems, and partially due to incompatibilities in the FORTRAN language. The VAX system is a secure system, isolating each user from the others, and requiring privileges and quotas for each user. The ModComp system requires no privileges for any user and any user can do anything with the system. These differences between the systems impacted the testing greatly in the areas of task activation and communication. The differences between the FORTRAN language implemented on both machines impacted the testing in the areas of Input/Output (I/O), file manipulation, and internal data organization, representation and manipulation.

This section of the report describes these obstacles and annotates the results of the benchmark tests.

4.1 I/O BENCHMARK TESTS

The I/O Benchmark Tests were run on both the ModComp IV/35 and VAX 11/780. They examined all of the areas of I/O that are commonly used by modules in the GSSS. Many of the "normal" I/O methods used on the ModComp system did not operate in the same way on the VAX system. These compatibility problems are accounted for in the following sections. Note that all the benchmark tests were done using the FORTRAN language.

4.1.1 GENERAL I/O INFORMATION

The ModComp IV/35 and VAX 11/780 differ greatly in their I/O philosophy. On the ModComp it is only necessary to assign a logical name to a device or disk file name in order to attempt accesses to that device or file. If the device or file is not available for access, an error message is output by the supervisor and the program is placed into a "held" state. However, on the VAX , access to a device is denied unless the device is explicitly "mounted" in the code or through JCL commands (see section 3.1). If the device is not mounted the program is aborted by the supervisor. On the ModComp, any or all programs can share any device or disk file by simply assigning logical names to the same device or file name, and no program can guarantee that it has exclusive use of that device or file. On the VAX, however, the opposite is true. Programs, by default, obtain exclusive use of a device or file as long as they have it "opened". Only by using special VAX coding techniques can more than one process access the same device or file at the same time. This is accomplished by using the special VAX FORTRAN OPEN statement fully specifying the file name and specifying it as SHARED:

```
(e.g., OPEN (UNIT=3, NAME='FILE.DAT;1', SHARED))
```

ORIGINAL PAGE IS
OF POOR QUALITY

Even this method will work only for input files to multiple processes. Output files cannot be shared for output by more than one process. The resulting data on the file is unpredictable using sequential access. Some of the records are never written to the file. This can have an effect on the dump collection modules in the GSSS.

4.1.2 SEQUENTIAL DISK FILE I/O

Except for shareability and sector sizes, the sequentially organized disk files on both the ModComp and VAX systems operate in much the same way:

- Both systems can have multiple end-of-file marks on one physical disk file.
- Both systems can randomly read a sequentially created disk file as long as the records are of fixed length.
- Both systems can add records at the end of the file.
- Logical record lengths of 256 bytes can be read and written on both systems.

However there are some differences between the two systems. Some things that are done on the VAX system but not done on the ModComp system are:

- Blocking and deblocking is automatically done by the supervisor.

ORIGINAL PAGE IS OF POOR QUALITY

- File sizes are dynamically expanded as records are added to the file.

- The I/O from one process is isolated from other processes (unless special JCL commands are used).

- Files are dynamically created when written to for the first time.

- Evidence of the last updated version of a file is kept in the file directory, including date and time.

In contrast there are some operations effecting sequential files that can be performed on the ModComp system that cannot be done on the VAX:

- Sequential I/O can update individual records on a disk file.

- Random writes can be made to a sequentially created file.

- File pointers (keys) can be positioned to any particular record by either advancing to, or backspacing over, end-of-file marks or records (i.e. AVF, AVR, BKF & BKR utilities).

- Files need not be explicitly opened to indicate access mode (i.e. sequential or random).

Some of these differences can have a great effect on the GSSS software; others are of little consequence. Only through attempting to transport the GSSS to the VAX can all obstacles be found. The VAX philosophy of dynamically allocating files may be the greatest obstacle to overcome since none of the GSSS software expects this to ever occur (see Section 3.3).

4.1.3 RANDOM DISK FILE I/O

Except for the sector sizes and automatic blocking and deblocking, random access disk files operate in much the same way on both the ModComp and VAX system:

- Both systems allow random updates anywhere in the file using the same "key" values.
- Both systems allow sequential reads from a randomly created file (as long as the records are continuous).
- Logical records of 256 bytes can be accessed on both machines.

However, some things can be accomplished on the VAX system that cannot on the ModComp are:

- Fixed record lengths of other than 256 bytes can be accessed.
- Due to automatic file size expansion, records can be added past the end of the file.

In addition the VAX system has one limitation that the ModComp system does not:

- Any file used for random access can have only one end-of-file and this end-of-file must be after the last record in the file.

Conversely, there are things that can be done on the ModComp system and not on the VAX system:

- Sequential updates can be made to a randomly created file.
- Randomly accessed files can have many end-of-file marks.
- Record lengths are fixed at 256 bytes.

Probably only one of these differences will impact greatly the transport of the GSSS. The allowance by the ModComp system for multiple end-of-file marks on a randomly accessed file. In many cases, in the GSSS, files with more than one end-of-file mark are created sequentially but input randomly. This can have an effect on the database and commanding modules of the GSSS.

4.1.4 FORMATTED TAPE I/O

Once a tape is mounted and positioned properly, the records written to tape, or read from tape, with FORTRAN formatted reads and writes, are totally compatible on both the ModComp and VAX systems. However, there are some problems with compatibility in the areas of tape positioning and end-of-file marks.

The FORTRAN REWIND command is not compatible:

- On the ModComp system a REWIND of any tape merely positions the tape at the "load point".
- On the VAX system a REWIND of a tape causes two end-of-file marks to be written at the beginning of tape and the tape is positioned immediately following them, if the tape is "unlabeled".

There were other problems encountered during the formatted tape I/O testing on the VAX system:

- 256 byte records have not been successfully written to unlabeled tapes using FORTRAN formatted writes.
- Tapes must be explicitly mounted using JCL commands before they can be accessed (see Section 3.1).

The physical end-of-file marks written to tapes are compatible to both systems, however:

- End-of-file marks written by the ENDFILE function on the VAX system are <u>not always</u> recognized as end-of-files on FORTRAN read statements with "END=" specified, on the VAX. The end-of-file mark is sometimes ignored by the I/O system if it was the last thing written to the tape. Any module of the GSSS that reads tapes (e.g. PLBK, STOLPH) can be effected by this.

4.1.5 UNFORMATTED TAPE I/O

When using standard FORTRAN reads and writes on both the ModComp and VAX systems, unformatted tape records are totally incompatible. The FORTRAN generated record headers are different as shown below:

- Record headers for 36 byte records:

VAX	ModComp
X'0022', X'003'	X'0700', X'0024'

In addition to the record headers and tape positioning problems previously mentioned in Section 4.1.4, there is a problem with end-of-file marks on unformatted tapes on the VAX:

- Unformatted, unlabeled tapes cannot have an end-of-file mark written to them from FORTRAN using the ENDFILE function. Attempting to do so causes an I/O error and terminates the process.

4.1.6 FILE POSITIONING FUNCTIONS

The ModComp system allows for all the file and record positioning functions using JCL commands:

- Advance File (AVF)
- Advance Record (AVR)
- Backspace File (BKF)
- Backspace Record (BKR)

ORIGINAL PAGE IS
OF POOR QUALITY

The VAX system does not allow for any of them. In addition there is the REWIND problem stated previously in Section 4.1.4. Since many modules within the GSSS (e.g. DBGEN, DBXREF) use these functions this can have a great impact on the transporting of the GSSS to the VAX. However, all of the above functions can be accomplished with simple, specialized library routines, or special $QIO calls.

4.1.7 EVENT PRINTING

The philosophy of printing event messages differs between the ModComp IV and VAX 11/780 computer systems. On the ModComp system event messages are easily output in chronological order simply by closing or endfiling the output stream after each line. When this is done the messages are sent to the spooler and concatenated with all other messages and then actually printed. The ModComp makes no attempt to isolate one task's printout from another's when this method is used.

On the VAX system, the spooler groups each process' output separately and prints the lines as separate listings for each process. Using the ModComp method on the VAX results in each line being output to a "new page" on the printer. However, there is a method that can be used on the VAX system to insure event messages are neatly printed in chronological order.

This can be accomplished by routing all printer event messages to one process through a mailbox. Once a message is received by this process (through the mailbox) it can be printed in the normal manner.

However, there is one more oddity of the VAX system print spooler. Unless this process keeps count of lines printed and closes its spool file, after each page of printout (approximately every 50 lines), or periodically, no lines will be actually printed until this process exits. A preliminary version of this process already exists on the VAX.

4.2 FORTRAN LANGUAGE BENCHMARK TESTS

The FORTRAN language tests fall into three categories: arithmetic and logical operations, array allocation, and the logical and physical organization of numbers and character strings. Some of these tests established compatibilities, while others revealed serious and potentially widespread incompatibilities between the ModComp IV/35 and the VAX 11-780 with respect to the GSSS.

4.2.1 ARITHMETIC AND LOGICAL OPERATIONS

- There exist integer arithmetic calculations which are valid and which work on the ModComp but which cause overflow and abort the process on the VAX (See Section 5.1).
- The LOGICAL*1 (L*1) data type on the VAX works like any other integer date type for integer arithmetic and logical operations, including conversion (i.e., sign extension, etc.). Note that this data type does not exist in ModComp FORTRAN and should not be used when transporting modules to the VAX.

- The FORTRAN supplied functions IOR, IAND, and ISHFT work the same for integers on the ModComp and the VAX. This was verified by extensive bit extraction/insertion and shifting tests. However, their use in character manipulation leads to different results on the two computers (See Section 4.2.3).

- Any differences in the accuracy of floating point calculations between the ModComp and the VAX are too small to affect their application in GSSS. As an example, least squares calculations typical for GSSS were performed on the two machines and agreed to four to six decimal places for 6-term/11-point formulas thru 2-term/5-point formulas. This was two to four places more accurate than the approximation formula to the correct solution.

4.2.2 ARRAY ALLOCATION

- The assignment of the elements of one-dimensional FORTRAN arrays is in their logical order on both machines. For example, the VAX FORTRAN data declarations:

```
LOGICAL*1    W(8)
INTEGER*2    X(4)
INTEGER*4    Y(2)
REAL*8       Z(1)
EQUIVALENCE  (W(1),X(1),Y(1),Z(1))
```

  assign these arrays to the same eight bytes of memory with the following element correspondence:

ORIGINAL PAGE IS OF POOR QUALITY

Table 4-2. Array Elements Correspondence

Byte Addresses:	a	a+1	a+2	a+3	a+4	a+5	a+6	a+7
W:	W(1)	W(2)	W(3)	W(4)	W(5)	W(6)	W(7)	W(8)
X:	X(1)		X(2)		X(3)		X(4)	
Y:	Y(1)				Y(2)			
Z:	Z(1)							

This memory correspondence also holds on the ModComp for arrays X,Y, and Z, but not for W since the L*1 data type doesn't exist on the ModComp. Note that on both the ModComp and the VAX the address of a variable or an array is presented as the address of its lowest addressed byte in memory even though the ModComp is a word addressing machine (16 bit words).

- On both machines the allocation of the elements of two-dimensional arrays is column major order (the FORTRAN standard); that is the elements A(I,J) are taken by varying the leftmost subscript (I) most frequently and the rightmost subscript (J) least frequently.

4.2.3 BYTE COMPOSITION OF NUMBERS AND CHARACTER STRINGS

- The storage of alphanumeric character strings in arrays by FORTRAN statements (i.e., DATA or READ) results in the same physical ordering of bytes in memory for each data type and for both the ModComp and the VAX computers.

ORIGINAL PAGE IS
OF POOR QUALITY

As an example, consider the assignment of the character string "ABCDEFGH" to each of the arrays W, X, Y, Z (declared in Section 4.2.2) by the following FORTRAN statements:

```
      DATA W/'A', 'B',....., 'H'/
or    READ (5, 10) W    with   10 FORMAT (8A1)

      DATA X/'AB', 'CD', 'EF', 'GH'/
or    READ (5,10) X     with 10    FORMAT (4A2)

      DATA Y/ 'ABCD', 'EFGH'/
or    READ (5,10) Y     with    10 FORMAT (2A4)

      DATA Z/ 'ABCDEFGH'/
or    READ (5,10) Z    with   10 FORMAT (A8)
```

The result is summarized in Table 4-3. Recall that the LOGICAL *1 data type (e.g., array W) exists only on the VAX. Also, we are <u>not</u> considering the CHARACTER data type, which exists only on the VAX.

Table 4-3. Byte Ordering for Character Strings ModComp and VAX (Arrays W, X, Y, and Z)

Logical Byte Order:	A	B	C	D	E	F	G	H
Byte Addresses:	a	a+1	a+2	a+3	a+4	a+5	a+6	a+7
Hex Codes:	41	42	43	44	45	46	47	48

- The storage of an integer at the byte addresses for an integer variable differs on the ModComp and VAX computers. Considering a FORTRAN integer as a sequence of bytes, the bytes are stored in decreasing order of significance on the ModComp (i.e., most significant byte at the lowest byte address) and in increasing order of significance on the VAX (i.e., least significant byte at the lowest byte address). For example, consider the following integer (hexadecimal) value assignments to the arrays W, X, and Y from Section 4.2.2:

```
       DATA W/Z41, Z42, Z43, Z44, Z45, Z46, Z47, Z48/
or     READ (5,10) W     with    10 FORMAT (8Z2)

       DATA X/Z4142, A4344, Z4546, Z4748/
or     READ (5,10) X     with    10 FORMAT (4Z4)

       DATA Y/Z41424344, Z45464748/
or     READ (5,10) Y     with    10 FORMAT (2Z8)
```

The READS refer to the hexadecimal string "4142434445464748". Table 4-4 compares the internal storage representations of these integers for the ModComp and VAX computers and also shows the bit-numbering conventions for the two computers (they are the reverses of each other within data items). Note: The sign bit of a data item is denoted by s. Also, the address of a data item for both the ModComp and the VAX is its lowest byte address.

ORIGINAL PAGE IS
OF POOR QUALITY

Table 4-4. Byte Ordering for Integers

L*1 array W (VAX only)

Array Element :	W(1)	W(2)	W(3)	W(4)	W(5)	W(6)	W(7)	W(8)
Logical Byte Order :	s41	s42	s43	s44	s45	s46	s47	s48
Bit Numbering :	0-7	0-7	0-7	0-7	0-7	0-7	0-7	0-7
Byte Addresses :	a	a+1	a+2	a+3	a+4	a+5	a+6	a+7

I*2 array X

Array Element	:	X(1)		X(2)		X(3)		X(4)	
Logical Byte Order	:	(s41	42)	(s43	44)	(s45	46)	(s47	48)
Bit Numbering MOD	:	0-7	8-15	0-7	8-15	0-7	8-15	0-7	8-15
VAX	:	15-8	7-0	15-8	7-0	15-8	7-0	15-8	7-0
Byte Address MOD	:	a	a+1	a+2	a+3	a+4	a+5	a+6	a+7
VAX	:	a+1	a	a+3	a+2	a+5	a+4	a+7	a+6

I*4 array Y

Array Element	:		Y(1)				Y(2)		
Logical Byte Order	:	(s41	42	43	44)	(s45	46	47	48)
Bit Numbering MOD	:	0-7	8-15	16-23	23-31	0-7	8-15	16-23	24-31
VAX	:	31-24	23-16	15-8	7-0	31-24	23-16	15-8	7-0
Byte Addresses MOD	:	a	a+1	a+2	a+3	a+4	a+5	a+6	a+7
VAX	:	a+3	a+2	a+1	a	a+7	a+6	a+5	a+4

- These differences in the allocation of integer data types are transparent to the FORTRAN programmer <u>except</u> when

ORIGINAL PAGE IS OF POOR QUALITY

programming techniques are used which require operating with only part of a data item. One such case is the manipulation of parts of an integer (i.e., sign, most significant part, least significant part, etc.) by equivalencing arrays and variables of different data types. For example, suppose the following equivalences are made for the arrays W,X, and Y,

```
EQUIVALENCE (W(1),X(1),Y(1))
```

Table 4-5 summarizes the semantic differences between the ModComp and VAX computers which occur in the elements of array X. Clearly, programs performing such manipulations will not work the same on both machines.

Table 4-5. Byte Manipulation of Integers

Part of Y(1)	ModComp ref.	VAX ref.
Most Significant Half:	X(1)	X(2)
Least Significant Half:	X(2)	X(1)
Sign:	X(1)	X(2), W(4)
Most Significant Byte:	___	W(4)
Least Significant Byte:	___	W(1)

- In a similar way, the difference in allocation of floating point numbers, coupled with equivalencing variables of other types with floating point type

ORIGINAL PAGE IS
OF POOR QUALITY

variables, can lead to semantic differences in ModComp and VAX FORTRAN programs. For example, consider setting the less significant part of the mantissa of a double-precision floating point number P via an equivalenced I*4 array Y(2): the bytes of element Y(2) will not be in the same order as those of the less significant half of P.

- Another case of semantic differences due to the allocation of integers occurs in character manipulation. Character strings can be constructed or modified in FORTRAN by arithmetic or logical operations. The following FORTRAN statements define the same string in I*2 variables X(1), X(2):

```
   X(1) = 65*256 + 66 , X(2) = 67*256 +68
or
   X(1) = IOR(ISHFT(65,8), 66)), X(2) = IOR(ISHFT(67,8),68)
```

  as do the following in Y(1):

```
   Y(1) = ((65*256 + 66)*256 + 67)*256 + 68
   Y(1) = IOR(ISHFT(IOR(ISHFT(IOR(ISHFT(65,8),66),
            8),67),8),68)
```

However, the logical ordering of the characters within the integers in memory differs:

- On the ModComp the above produce the equivalent of the data statements:

```
  DATA X/'AB', 'CD'/    and
  DATA Y/'ABCD'/
```

ORIGINAL PAGE IS
OF POOR QUALITY

- On the VAX they produce the equivalent of:

  DATA X/'BA', 'DC'/   and

  DATA Y/'DCBA'/

This results in different logical memory layouts as shown in Table 4-6, assuming that the arrays are equivalenced. Note that 65, 66, 67 and 68 are the decimal character codes for A, B, C and D.

Table 4-6. Logical Character Strings Resulting from Arithmetic and Logical Operations

	ModComp				VAX			
Byte Address:	a	a+1	a+2	a+3	a+3	a+2	a+1	a
I*2 X Character String:	A	B	C	D	C	D	A	B
Array Element:	X(1)		X(2)		X(2)		X(1)	
I*4 Y Character String:	A	B	C	D	A	B	C	D

In addition, the logical order of characters within integers (which corresponds to the ordering of bytes in memory) is the same as A-format input and differs on the two machines, as illustrated in Table 4-7.

Table 4-7. Character Strings by Input in A Format

	ModComp				VAX			
Byte Addresses :	a	a+1	a+2	a+3	a+3	a+2	a+1	a
I*2 X Character String:	A	B	C	D	D	C	B	A
Array Element :	X(1)		X(2)		X(2)		X(1)	
I*4 Y Character String:	A	B	C	D	D	C	B	A

The implications of these semantic differences between the ModComp and the VAX are obvious. Since these kinds of programming techniques are allowable in the FORTRAN language, they may be used anywhere in a program and, hence in a large program system like the GSSS there may be numerous ModComp dependencies due to character and number manipulation occurring throughout the software.

4.3 TIMING BENCHMARK TESTS

Several timing tests were performed on the VAX on common but important code sequences. The purpose was to determine if any potential timing problems might occur in the GSSS due to these computations. These tests along with their corresponding average times (over 100 to 100,000 executions as appropriate) are presented in Table 4-8.

Unless otherwise stated, the tests were coded in FORTRAN and the times are in milliseconds (ms). In general these times are comparable to those on the ModComp. The only potential problems are, the outrageously long time for the file OPEN/CLOSE sequence (314 ms), and the time for the mailbox WRITE/READ sequence (.4 ms). We note that the VAX microcoded MACRO instruction for polynomial evaluation is five times as fast as efficient FORTRAN code.

Table 4-8. Timing Benchmarks

Test Performed	Time (ms)
5-th Order Polynomial Evaluation (Horner's Method):	
- FORTRAN DO-loop	0.11
- Special VAX microcoded instruction (POLY)	0.02
FORTRAN subroutine Call Overhead	0.02
System Service Call Overhead	0.07
Blocked File I/O:	
- READ	3.3
- WRITE	4.0
Mailbox Write/Read Sequence	0.40
File OPEN/CLOSE Sequence	314.0

SECTION 5 - COMPATIBILITY OF APPLICATIONS LANGUAGES

In the ModComp IV/35 version of the GSSS only two applications (programming) languages are used: FORTRAN IV and M4A Assembler. The ModComp CLASSIC system is totally compatible with these two programming languages. In fact, the object language from the ModComp IV/35 will execute on the ModComp CLASSIC. On the other hand, the VAX 11/780 system has a different implementation of FORTRAN (FORTRAN IV-PLUS) and its assembler language is entirely different.

5.1 FORTRAN

The difference in the implementation of the FORTRAN language on the ModComp systems (IV/35 & CLASSIC) and the VAX 11/780 system are varied. There are many things that are allowed and done by GSSS modules on the ModComp system that cannot be done or don't work the same in the VAX implementation of FORTRAN:

- Modification of DO LOOP variables within the loop is ok on the ModComp but not on the VAX.
- Using SHIFT, AND and OR logic to manipulate characters works differently on the two systems.
- Attempting to convert X 'FFFFA8AA' to a 16 bit integer causes overflow on the VAX.

- DATA I/Z2031/ and DATA I/' 1'/ are identical for 16 bit variables on the ModComp but not on the VAX. On the VAX it is DATA I/Z2031/ and DATA/'1 '/ that are equivalent.

- The ENCODE and DECODE statements have a slightly different format.

- The following code:

  DIMENSION A(3)

  DATA A/'12 CHARACTERS'/ is ok on the ModComp

  but must be coded as follows on the VAX:

  DIMENSION A(3)

  DATA A/'12CH', 'ARAC', 'TERS'/

- The VAX allows for LOGICAL *1 data type, the ModComp does not.

- The VAX has a special, VAX, CHARACTER data type which must be used to call many system services.

- The equivalencing of arrays is logically different on the two systems (see Section 4.2).

In addition to the above mentioned compatibility problems, there is one major "bug" in the implementation of FORTRAN on the VAX system when it is optimized. The following code sequence produces a value other than the zero for the variable J:

```
   J=0
   B=.TRUE
   DO 100 I=1,10
   IF (B) GOTO 100
   J=I
100 CONTINUE
   PRINT  J
```

The effect of this on the GSSS software will remain unknown until each and every FORTRAN module is thoroughly tested and debugged on the VAX system.

5.2 ASSEMBLER

The ModComp IV/35 and ModComp CLASSIC can use the same Macro Assembler (M4A). The VAX 11/780, by nature, uses a different Macro Assembler (VAX-11 MACRO). If only the languages were different, this compatibility problem could possibly be overcome by writing a cross-assembler to assemble the M4A source code into VAX-11 MACRO code. However, the hardware architecture and language logic differences between the two vendors' machines precludes this approach:

- The VAX system has 21 different addressing modes. The ModComp systems have only 6 addressing modes.

- Data organizations within memory are logically reversed.

- The VAX uses stacks for register and argument save areas extensively. The ModComp systems do not.

- On the ModComp systems, 15 of 16 general purpose registers (R1 - R15) can be used for any purpose, at any time. The

VAX system has 4 dedicated general purpose registers (R12 - R14) and 6 special use registers (R0 - R5) leaving only 6 registers that can be used at any time, for any purpose.

- The instruction logic is entirely different on the ModComp and VAX systems.

- The VAX is a byte addressing machine. The ModComps use word addressing.

This makes the two assembler languages totally incompatible. Since over 33,000 lines of M4A assembler code exists in the ModComp IV/35 version of the GSSS, this is a major compatibility problem with the VAX system.

5.3 LIBRARIES

Except for system service interfaces, the FORTRAN libraries on the ModComp and VAX systems appear to be totally compatible. However, to transport the GSSS to the VAX the entire user written library (a large portion of which is written in M4A assembler) must be rewritten.

SECTION 6 - COMPATIBILITY OF VENDOR SUPPLIED SERVICES

Both ModComp systems appear to have similar, if not identical, vendor supplied services in all areas including JCL commands and system services. The VAX system has an entirely different set of services. The JCL commands on the VAX are more complex and allow the user to accomplish more functions but do not include as a subset all the functions allowed in the ModComp systems. The system services are entirely different but do allow for many of the same functions.

6.1 JOB CONTROL LANGUAGE COMMANDS

All three systems (ModComp IV/35, ModComp CLASSIC and DEC VAX 11/780) provide the user with Job Control Language (JCL) commands that give the user the capability to control the execution of programs and maintain data files. Both ModComp systems provide almost identical sets of JCL commands to the user. However, the VAX system provides an entirely different set of JCL commands. Many of the differences are merely syntactical differences. Others operate in logically different ways. The syntactical differences are easily overcome; but the logical differences are difficult, if not impossible, in some cases to resolve.

Besides the previously mentioned problems with tape mounting and assigning logical units for shared use (see Sections 3.1 and 4.1.1), there is one major difference in the JCL commands that will have a substantial effect on the transportation of the GSSS. It is in the area of the link-editor:

- On the ModComp systems logical unit assignments for a "foreground" task can be specified when the program is linked.
- On the VAX system there is no way of specifying logical unit assignments when a program is linked.

This incompatibility is of paramount importance to the transportation of the GSSS. In the GSSS no standards for logical unit numbers or names have been made and different units are used for the same devices and files throughout the system. If the GSSS is transported without major changes in this area, special JCL command procedures would have to be run before the <u>execution</u> of each module. The necessary changes in this area would involve:

- The examination of each program for the logical units used.
- The examination of all the link decks for the physical equivalents for the logical units.
- Cross referencing all references to the same physical devices.
- Changing all the logical unit assignments to a physical device to the same unit number or name. This requires coding changes within the programs.

6.2 SYSTEM SERVICES

The ModComp systems provide for FORTRAN interface routines in the library for calling system services. The VAX system provides for calling the system services directly from FORTRAN. Most of the major system services available on the ModComp systems have corresponding counterparts in the VAX system as shown in Table 6-1. Some do not. Even some of the corresponding system services on the VAX do not operate in the same way or provide the same information:

- $CREPRC doesn't operate like ACT. A secondary activation can be done on the ModComp but can't using $CREPRC on.the VAX.

- $GETJPI doesn't return the same information as GETTASK.

- $GETTIM returns time in a different format than GETTIME.

- $QIO requires channel number as an argument, not logical name as do the ModComp services.

The total impact of this area on the GSSS software will remain unknown until the transport of the system is attempted. However, secondary activations of tasks are common-place in the GSSS.

Table 6-1. ModComp vs. VAX System Services (1 of 3)

ModComp System Service	VAX System Service	ModComp Purpose
EST	none	Establish a resident task.
DEES	none	De-establish a task.
ACT	$CREPRC	Start execution of a task.
KILL	$FORCEX	Abort another task.
ABORT	$FORCEX	Abort this task.
HOLD	none	Suspend task until resumed by operator.
WAIT	$SUSPND	Suspend task until resumed by any resume.
RES	$RESUME	Resume a task in WAIT.
DELAY	$SETIMR and $WAITFR	Suspend a task for a specified period of time.
CONNECT	none	Allocate timer to schedule any above function at a future time.
GETTASK	$GETJPI	Get information about a task.
SEND	$QIO to a mailbox	Send a message to another task.
RECEIVE	$QIO from a mailbox	Receive a message sent by a SEND service.
ALLOCATE	$CRETVA	Allocate a region of private memory to this task's space.
DEALLOCATE	$DELTVA	Deallocate region of private memory from this task's space.

ORIGINAL PAGE IS
OF POOR QUALITY

Table 6-1. ModComp vs. VAX System Services (2 of 3)

ModComp System Service	VAX System Service	ModComp Purpose
CREPRI	none?	Create shared region from this task's private space.
INSPRI	none?	Insert shared region into this task's private space.
GETTIME	$GETTIM	Get time of day or elapsed time.
DUMP	none	Dump region of memory to line printer
COLLECT	none	Parse next parameter in character string.
ATCAN	none	Convert ASCII string to CAN-CODE.
CANTA	none	Convert CAN-CODE to ASCII string.
ATNUM	none	Convert ASCII string to binary.
BTDEC	none	Convert binary to decimal ASCII string.
BTHEX	none	Convert binary to Hexa-decimal ASCII string.
ASSI	$CRELOG	Assign logical name to device or file.
TASSI	$TRNLOG	Test assignment of a logical name.
REW	$QIO	Rewind a logical device.
HOME	$QIO	Position a logical device at beginning of media (i.e. Rewind)

Table 6-1. ModComp vs. VAX System Services (3 of 3)

ModComp System Service	VAX System Service	ModComp Purpose
WEOF	$QIO	Write an end-of-file mark on a logical device.
TERMIO	$CANCEL	Terminate an I/O request to device.
IOWAIT	none ?	Wait for I/O to complete
BKR	$QIO	Backspace a record on a logical unit.
AVR	$QIO	Advance a record on a logical unit.
BKF	$QIO	Backspace one file on a logical unit.
AVF	$QIO	Advance one file on a logical unit.
READ	$QIO	Read a record from a logical unit.
WRITE	$QIO	Write a record to a logical unit.
MESSAGE	$BRDCST ?	Display a message on the operators console.
MESSAGE/HOLD	none	Display a message on the operator's console and enter a hold state.

SECTION 7 - CONCLUSIONS

This report has presented many interesting peculiarities of the VAX 11/780 system in relationship to the ModComp IV/35 implementation of the GSSS. Some of the software within the GSSS uses ModComp unique coding techniques and logic sequences that are contributing factors to the incompatibilities indicated. However, these incompatibilities still exist with relation to the ModComp version of the GSSS. With this in mind the following conclusions about transporting the GSSS to the VAX are presented:

- All of the tape I/O handling logic will require changes.
- All of the file and device sharing logic will need modification.
- All of the standard (256 byte) tape records will have to be redesigned to a different length.
- The KCRT input handler will have to be redesigned and rewritten.
- Logic and coding techniques will have to be developed and implemented to avoid the proliferation of disk files.
- All end-of-file logic on random access disk files will have to be found and eliminated.
- Any use of FORTRAN unformatted tapes will have to be eliminated.

- Most I/O to files will have to use the OPEN statement.
- JCL logic sequences will require massive changes.
- All use of the FORTRAN REWIND command for tapes will have to be eliminated.
- All event messages will need to be routed through one module (process).
- Timing problems, particularly in the area of file OPEN/CLOSE sequences, will need to be investigated.
- Most, if not all, character DATA statements in FORTRAN will have to be changed.
- All DATA statements in FORTRAN will have to be examined for validity on the VAX.
- All use of SHIFT, AND and OR logic for character manipulation will have to be eliminated.
- Conversions of 32 bit integers to 16 bits will have to be examined and investigated.
- Most ENCODE and DECODE statements in FORTRAN will need modification.
- Array equivalencing will have to be closely examined for validity on the VAX.

- Any modification of DO LOOP variables within the loop will have to be eliminated.

- Any conditionally executed statements within DO LOOPs will have to be carefully examined for valid execution.

- New generalized methods for character manipulations will have to be developed.

- All of the assembler language modules will have to be rewritten using different logic sequences.

- Most of the user written GSSS library routines will have to be recoded.

- A method for specifying logical unit assignments in real-time will have to be developed.

- The inconsistancies between the system services will have to be resolved.

Very few, if any, of the above conclusions will apply if the GSSS is transported to the ModComp CLASSIC system. Therefore, the main conclusion to be drawn from the State-of-the-Art Computer Systems/ GSSS Compatibility Study is:

> Transporting the GSSS to the ModComp CLASSIC system will require very few coding changes. However, attempting to transport the GSSS to the VAX 11/780 will require massive changes, not only in the coding of the modules but also in the design of many

areas of the GSSS, particularly in the areas of I/O and task activation.

In Fact, it may be a futile effort to attempt to implement any real-time satellite ground support system like the GSSS on the VAX 11/780, since the basic philosophy in the VAX environment is one of a multi-user, timesharing, interactive processor.

To summarize the ModComp-to-VAX compatibility analysis, recall the conclusion from the MMS/GSSS MODCOMP Device and MAX IV Dependency Study. Based on the hypothesis of the compatibility of (1) FORTRAN language and compiler, (2) data base structure, (3) CRT page library and tables, (4) telemetry format tables, (5) command structures, and (6) all other internal tables, the approach to moving the GSSS to the target environment would be developed. This would entail at least rewriting the ModComp/FORTRAN library routines (with the aid of a cross-assembler) and moving the FORTRAN code with only the "obvious" changes made, followed by an iterative procedure of load/link/execute, error isolation, and error correction, until working code is produced.

With respect to the VAX as a target machine, this minimal transport effort must, of course, be revised in light of the incompatibilities revealed in this study. The FORTRAN language and compiler incompatibilities (in one case an outright bug in vendor software) obviate hypothesis (1) above and point to vast, detailed, and sometimes obscure changes in the GSSS FORTRAN code. Moreover, the FORTRAN semantic incompatibilities also imply incompatibilities in areas (2) through

(6) above. At the very least, GSSS internal structures involving manipulations of characters and hexadecimal codes will (most likely) be incompatible due to differences in internal byte and bit-string operations (logical or arithmetic). Examples of areas probably affected are the parsing software in the CRT page table and the command structure areas, and data base handling (i.e., do creation and examination of structures "hide" differences between MODCOMP and VAX in data item encoding?). These points can be resolved only by detailed examination of the software in each of areas (2) through (6).

Also in the language area, the significantly different philosophies in the organization of the assembly languages for the MODCOMP and VAX machines makes the cross-assembler approach to MODCOMP/FORTRAN library translation unproductive. Hence, complete, individual re-coding of these routines will have to be done. Furthermore, incompatibilities in the VAX operating system and I/O software (i.e., in JCL, system services, etc.) force re-coding and, in fact, re-design of some areas of the GSSS. Examples we have seen from these areas are: file and device sharing, 256-byte tape records, tape I/O, KCRT input handling, end-of-file handling for random access files, and disk file proliferation as a result of multiple task activations. A major problem lies in certain time-critical system actions: it is almost definite that the large OPEN/CLOSE time will cause severe degeneration of GSSS performance on the VAX, since each task activation requires that the disk file containing the image be opened and closed. Also, the mailbox write/read time may be a problem, since this will occur for all event messages. Still another major problem is presented by secondary task activations: there is no system-supported method to do this on the VAX and it appears that changes will be needed in <u>every</u> such program (i.e., most programs).

Thus, there are quite a few technical problems in the GSSS implementation on the VAX, with <u>no</u> clean, fast solutions. Most can be solved with detailed, item-by-item examination of the individual programs - a time-consuming and by no means guaranteed method of generating error-free software in one pass. Not surprisingly, the manpower estimates presented in the Dependency Study (1 1/2 man-years) must be revised. The best estimate at this stage is based on the STOL module implementation experience. This was a 2-man, 1 1/2-month effort for the transport of approximately 5000 lines of FORTRAN code. Allowing 1/2-month for learning the system, this amounts to a 2-man-month transport effort. The MMS/GSSS consists of approximately 100,000 lines of code, over 33,000 lines of which are Assembler. For Assembler code, doubling the manpower effort per line of code, when compared to the STOL effort, is appropriate. So by extrapolation GSSS transport should require about:

33.2 * 2-man-months = 66.4-man-months = 5.6 man-years.

Thus, as the latest rough estimate of human resources, the MMS/GSSS transport to the VAX-11/780 will require approximately <u>6-man-years</u>. This programmer resource committment could be supported by the equivalent of one terminal, 40 hours per week for up to three programmers, through three terminals, 40 hours per week for six or seven programmers, etc.

Finally, consider the VAX hardware required to support the full GSSS. This is summarized as follows:

Type	DEC Supplied VAX Devices
Tape Drives	Two TU45 (1600/800 BPI, 9 trk, 75 IPS)
Disk Drives	One RM03 (67 MB, 1200 KB/S)
KCRTs	At least two (test conductor & page display)
Printer.	One LP11 (660 LPM)
Card Reader	One CR11 (285 CPM adequate)
Main Memory	1 MB, 600 nanosecond cycle time (The GSSS on the VAX will require about 750 KB since task swapping is not done as efficiently on the VAX due to the file OPEN/CLOSE timing).

In addition, specialized devices will have to be procured to replace the specialized devices that are in the current GSSS ModComp IV/35 environment.

APPENDIX A - Obstacles on the VAX and How They Were or Can be Overcome

Obstacle 1:	Character strings in FORTRAN DATA statements.
Obstacle 2:	FORTRAN ENCODE and DECODE statement differences.
Obstacle 3:	Hexadecimal constants in FORTRAN arithmetic expressions.
Obstacle 4:	Debug option character on FORTRAN statements.
Solution (For obstacles 1 thru 4):	Run the MTOV command procedure on the FORTRAN source. This procedure executes a combination of text editor CMD procedures and a FORTRAN program to modify the source code within the FORTRAN source (see GSSS/VAX Users Guide).
Obstacle 5:	Using the FORTRAN REWIND statement to rewind tapes.
Solution :	Unknown
Obstacle 6:	The proliferation of data files used for output.
Solution :	Use the OPEN statement specifying TYPE='OLD'.

Obstacle 7:	Formatted tape I/O positioning.
Solution :	Use the JCL command procedures, @MOUNT, @REW and @DISMOUNT or Use OPEN statements and $QIO system service calls (exact method unknown).
Obstacle 8:	Unformatted tape incompatibility.
Solution :	Unknown
Obstacle 9 :	Tape record lengths of 256 bytes.
Solution :	Possibly using OPEN statement specifying RECORDSIZE=256, RECORDTYPE='FIXED'.
Obstacle 10:	Carriage Control byte to tape drives.
Solution :	Write a utility program to copy files circumventing the VAX carriage control to tape logic.
Obstacle 11:	Record and file positioning through JCL commands.
Solution :	Write a utility program to accomplish this using $QIO system service calls.

Obstacle 12:	Allowing many end-of-file marks on random access disk files.
Solution :	Unknown
Obstacle 13:	Sharing output files between processes.
Solution :	Unknown - OPEN statement specifying SHARED doesn't work for output files.
Obstacle 14:	Sharing input files between processes.
Solution :	Use OPEN statement fully specifying file name and SHARED.
Obstacle 15:	Integer overflow when converting 32 bit integers to 16 bits.
Solution :	Compile FORTRAN program with /NOCHECK option.
Obstacle 16:	The use of SHIFT, AND & OR logic for character manipulation.
Solution :	Eliminate all such logic and replace with ENCODE and/or DECODE statements.

Obstacle 17:	Modification of DO LOOP variable within the loop.
Solution :	Recode DO LOOP using IF statement to terminate the loop.
Obstacle 18:	Incorrect execution of conditionally executed statements within DO LOOPS when the FORTRAN is optimized.
Solution :	Recode DO LOOPs using IF statement to terminate the loop.
Obstacle 19:	DEBUG can't run with shareable image, making testing while running in real-time very difficult.
Solution :	Unknown
Obstacle 20:	Being denied task activation rights because other tasks have already been activated.
Solution :	Any task (process) that activates another task should receive all the necessary privileges necessary and the maximum quotas allowed.
Obstacle 21:	Retrieving the task status returns different information on the two machines.
Solution :	Status returned is OK in many cases; but for the others the solution is unknown. Some information is not available.

Obstacle 22:	KCRT input works in character mode on the VAX.
Solution :	Queue KCRT input until whole string is input (see Bill Mocarsky for exact details).
Obstacle 23:	Assembler languages totally incompatible.
Solution :	Other than redesigning and recoding all assembler modules the solution is unknown.
Obstacle 24:	Specifying logical unit assignments when linking a module.
Solution :	Unknown
Obstacle 25:	Secondary activation of modules while they are executing.
Solution :	Unknown, unless all modules hibernate or suspend themselves instead of exiting.
Obstacle 26:	Printer output lines written directly from different modules.
Solution:	Modify them to route all printer output through one process.

Obstacle 27:	Differences in array equivalencing in FORTRAN.
Solution :	One by one examination of the use of the arrays equivalenced and modification where necessary. In some cases there may be no known solution.
Obstacle 28:	The length of time it takes to accomplish an OPEN/CLOSE sequence for a disk file on the VAX (314 mls).
Solution :	Unknown
Obstacle 29:	The length of time it takes to pass messages between processes on the VAX using mailboxes (0.4 mls).
Solution :	Unknown
Obstacle 30:	The differences between data types in DATA statements in FORTRAN: e.g. DATA I/Z2031/ = DATA I/' 1'/ on the ModComp but not on the VAX.
Solution:	One by one examination of hexadecimal constants in DATA statements for validity and modification where necessary.

APPENDIX B - Benchmark Tests and Their Purpose

Benchmark Name (number)	Purpose
TSTAPF (1.1, 1.3)	Test order of formatted character transfers during I/O.
TSTAPU (1.2)	Test order of unformatted character transfers during I/O.
LOGICALS (1.4)	Test if logical unit assignments must be the same for each process in the group (JCL sequence).
TSTFILS (2.1, 2.3)	Test sequential disk file access.
TSTAPF and TSTAPU (2.2)	Test effect of end-of-file on tape.
TIMEOPEN (2.4)	Time OPEN/CLOSE sequence overhead for disk files. (314 mls)
JCL sequence (2.5)	Test need for tapes to be MOUNTed from code.
TSTFILU (3.1, 3.2, 3.3)	Test random disk file access.
TSTFILF and TSTFILU (3.4)	Test blocking and deblocking of disk file records.

ORIGINAL PAGE IS
OF POOR QUALITY

Benchmark Name (number)	Purpose
EQUIV (5.1)	Test effect of EQUIVALENCE statement on I*2 and I*4 arrays in FORTRAN.
TWODIM (5.2)	Test memory layout of 2 dimensional arrays in FORTRAN.
BYTESIGN (5.3)	Test if VAX treats bytes as signed integers.
STOL (5.4)	Test if FORTRAN functions recognize different data types.
DATCOM (5.6, 5.7)	Test effect of DATA statements in FORTRAN for HEX, ASCII & decimal data.
BUFFIO (5.8)	Test if BUFFIO routine can be written for the VAX.
LLSQ (5.9)	Test results of least squares polynomial fit on both the ModComp and the VAX.
STOL etc. (6.1)	Test compatibility of system services.
STOL etc. (6.2)	Test use of shared regions.
ASIT (6.3)	Test spooling of event messages in chronological order.

ORIGINAL PAGE IS
OF POOR QUALITY

Benchmark Name (Number)	Purpose
TIMEOVER (6.4)	Time system service call overhead. (0.07 mls)
SNDMSG and RECVMS (6.6)	Test how mailboxes work on the VAX.
TSTAPF (6.7)	Test I/O error handling on the VAX.
TIMEMBOX	Time mailbox read/write sequence. (0.4 mls)
TIMEFORT	Time FORTRAN subroutine call overhead. (0.02 mls)
TIMEIO (4.0)	Time I/O to disk file (read = 3.3 mls, write = 4.0 mls).
TIMEPOLY	Time 5th order polynomial evaluation (Horner's method = 0.11 mls, POLY instruction = 0.02 mls).

BIBLIOGRAPHY OF SEL LITERATURE

Anderson, L., "SEL Library Software User's Guide," Computer Sciences-Technicolor Associates, Technical Memorandum, June 1980

Bailey, J. W., and V. R. Basili, "A Meta-Model for Software Development for Resource Expenditures," Proceedings of the Fifth International Conference on Software Engineering. New York: Computer Societies Press, 1981

Banks, F. K., "Configuration Analysis Tool (CAT) Design," Computer Sciences Corporation, Technical Memorandum, March 1980

Basili, V. R., "The Software Engineering Laboratory: Objectives," Proceedings of the Fifteenth Annual Conference on Computer Personnel Research, August 1977

Basili, V. R., "Models and Metrics for Software Management and Engineering," ASME Advances in Computer Technology, January 1980, vol. 1

Basili, V. R., "SEL Relationships for Programming Measurement and Estimation," University of Maryland, Technical Memorandum, October 1980

Basili, V. R., Tutorial on Models and Metrics for Software Management and Engineering. New York: Computer Societies Press, 1980 (also designated SEL-80-008)

Basili, V. R., and J. Beane, "Can the Parr Curve Help with the Manpower Distribution and Resource Estimation Problems?", Journal of Systems and Software, February 1981, vol. 2, no. 1

Basili, V. R., and K. Freburger, "Programming Measurement and Estimation in the Software Engineering Laboratory," Journal of Systems and Software, February 1981, vol. 2, no. 1

Basili, V. R., and T. Phillips, "Evaluating and Comparing Software Metrics in the Software Engineering Laboratory," Proceedings of the ACM SIGMETRICS Symposium/Workshop: Quality Metrics, March 1981

Basili, V. R., and T. Phillips, "Validating Metrics on Project Data," University of Maryland, Technical Memorandum, December 1981

Basili, V. R., and R. Reiter, "Evaluating Automatable Measures for Software Development," Proceedings of the Workshop on Quantitative Software Models for Reliability, Complexity and Cost, October 1979

Basili, V. R., and M. V. Zelkowitz, "Designing a Software Measurement Experiment," Proceedings of the Software Life Cycle Management Workshop, September 1977

Basili, V. R., and M. V. Zelkowitz, "Operation of the Software Engineering Laboratory," Proceedings of the Second Software Life Cycle Management Workshop, August 1978

Basili, V. R., and M. V. Zelkowitz, "Measuring Software Development Characteristics in the Local Environment," Computers and Structures, August 1978, vol. 10

Basili, V. R., and M. V. Zelkowitz, "Analyzing Medium Scale Software Development," Proceedings of the Third International Conference on Software Engineering. New York: Computer Societies Press, 1978

Chen, E., and M. V. Zelkowitz, "Use of Cluster Analysis To Evaluate Software Engineering Methodologies," Proceedings of the Fifth International Conference on Software Engineering. New York: Computer Societies Press, 1981

Church, V. E., "User's Guides for SEL PDP-11/70 Programs," Computer Sciences Corporation, Technical Memorandum, March 1980

Freburger, K., "A Model of the Software Life Cycle" (paper prepared for the University of Maryland, December 1978)

Higher Order Software, Inc., TR-9, A Demonstration of AXES for NAVPAK, M. Hamilton and S. Zeldin, September 1977 (also designated SEL-77-005)

Hislop, G., "Some Tests of Halstead Measures" (paper prepared for the University of Maryland, December 1978)

Lange, S. F., "A Child's Garden of Complexity Measures" (paper prepared for the University of Maryland, December 1978)

Miller, A. M., "A Survey of Several Reliability Models" (paper prepared for the University of Maryland, December 1978)

National Aeronautics and Space Administration (NASA), NASA Software Research Technology Workshop (proceedings), March 1980

Page, G., "Software Engineering Course Evaluation," Computer Sciences Corporation, Technical Memorandum, December 1977

Parr, F., and D. Weiss, "Concepts Used in the Change Report Form," NASA, Goddard Space Flight Center, Technical Memorandum, May 1978

Perricone, B. T., "Relationships Between Computer Software and Associated Errors: Empirical Investigation" (paper prepared for the University of Maryland, December 1981)

Reiter, R. W., "The Nature, Organization, Measurement, and Management of Software Complexity" (paper prepared for the University of Maryland, December 1976)

Scheffer, P. A., and C. E. Velez, "GSFC NAVPAK Design Higher Order Languages Study: Addendum," Martin Marietta Corporation, Technical Memorandum, September 1977

Software Engineering Laboratory, SEL-76-001, Proceedings From the First Summer Software Engineering Workshop, August 1976

--, SEL-77-001, The Software Engineering Laboratory, V. R. Basili, M. V. Zelkowitz, F. E. McGarry, et al., May 1977

--, SEL-77-002, Proceedings From the Second Summer Software Engineering Workshop, September 1977

--, SEL-77-003, Structured FORTRAN Preprocessor (SFORT), B. Chu, D. S. Wilson, and R. Beard, September 1977

--, SEL-77-004, GSFC NAVPAK Design Specifications Languages Study, P. A. Scheffer and C. E. Velez, October 1977

--, SEL-78-001, FORTRAN Static Source Code Analyzer (SAP) Design and Module Descriptions, E. M. O'Neill, S. R. Waligora, and C. E. Goorevich, January 1978

--, SEL-78-002, FORTRAN Static Source Code Analyzer (SAP) User's Guide, E. M. O'Neill, S. R. Waligora, and C. E. Goorevich, February 1978

--, SEL-78-003, Evaluation of Draper NAVPAK Software Design, K. Tasaki and F. E. McGarry, June 1978

--, SEL-78-004, Structured FORTRAN Preprocessor (SFORT) PDP-11/70 User's Guide, D. S. Wilson, B. Chu, and G. Page, September 1978

--, SEL-78-005, Proceedings From the Third Summer Software Engineering Workshop, September 1978

--, SEL-78-006, GSFC Software Engineering Research Requirements Analysis Study, P. A. Scheffer, November 1978

--, SEL-78-007, Applicability of the Rayleigh Curve to the SEL Environment, T. E. Mapp, December 1978

--, SEL-79-001, SIMPL-D Data Base Reference Manual, M. V. Zelkowitz, July 1979

--, SEL-79-002, The Software Engineering Laboratory: Relationship Equations, K. Freburger and V. R. Basili, May 1979

--, SEL-79-003, Common Software Module Repository (CSMR) System Description and User's Guide, C. E. Goorevich, S. R. Waligora, and A. L. Green, August 1979

--, SEL-79-004, Evaluation of the Caine, Farber, and Gordon Program Design Language (PDL) in the Goddard Space Flight Center (GSFC) Code 580 Software Design Environment, C. E. Goorevich, A. L. Green, and F. E. McGarry, September 1979

--, SEL-79-005, Proceedings From the Fourth Summer Software Engineering Workshop, November 1979

--, SEL-80-001, Functional Requirements/Specifications for Code 580 Configuration Analysis Tool (CAT), F. K. Banks, C. E. Goorevich, and A. L. Green, February 1980

--, SEL-80-002, Multi-Level Expression Design Language-Requirement Level (MEDL-R) System Evaluation, W. J. Decker, C. E. Goorevich, and A. L. Green, May 1980

--, SEL-80-003, Multimission Modular Spacecraft Ground Support Software System (MMS/GSSS) State-of-the-Art Computer Systems/Compatibility Study, T. Welden, M. McClellan, P. Liebertz, et al., May 1980

--, SEL-80-004, System Description and User's Guide for Code 580 Configuration Analysis Tool (CAT), F. K. Banks, W. J. Decker, J. G. Garrahan, et al., October 1980

--, SEL-80-005, A Study of the Musa Reliability Model, A. M. Miller, November 1980

--, SEL-80-006, Proceedings From the Fifth Annual Software Engineering Workshop, November 1980

--, SEL-80-007, An Appraisal of Selected Cost/Resource Estimation Models for Software Systems, J. F. Cook and F. E. McGarry, December 1980

--, SEL-81-001, Guide to Data Collection, V. E. Church, D. N. Card, F. E. McGarry, et al., September 1981

--, SEL-81-002, Software Engineering Laboratory (SEL) Data Base Organization and User's Guide, D. C. Wyckoff, G. Page, F. E. McGarry, et al., September 1981

--, SEL-81-003, Software Engineering Laboratory (SEL) Data Base Maintenance System (DBAM) User's Guide and System Description, D. N. Card, D. C. Wyckoff, G. Page, et al., September 1981

--, SEL-81-004, The Software Engineering Laboratory, D. N. Card, F. E. McGarry, G. Page, et al., September 1981

--, SEL-81-005, Standard Approach to Software Development, V. E. Church, F. E. McGarry, G. Page, et al., September 1981

--, SEL-81-006, Software Engineering Laboratory (SEL) Document Library (DOCLIB) System Description and User's Guide, W. Taylor and W. J. Decker, December 1981

--, SEL-81-007, Software Engineering Laboratory (SEL) Compendium of Tools, W. J. Decker, E. J. Smith, A. L. Green, et al., February 1981

--, SEL-81-008, Cost and Reliability Estimation Models (CAREM) User's Guide, J. F. Cook and E. Edwards, February 1981

--, SEL-81-009, Software Engineering Laboratory Programmer Workbench Phase 1 Evaluation, W. J. Decker, A. L. Green, and F. E. McGarry, March 1981

--, SEL-81-010, Performance and Evaluation of an Independent Software Verification and Integration Process, G. Page and F. E. McGarry, May 1981

--, SEL-81-011, Evaluating Software Development by Analysis of Change Data, D. M. Weiss, November 1981

--, SEL-81-012, Software Engineering Laboratory, G. O. Picasso, December 1981

--, SEL-81-013, Proceedings From the Sixth Annual Software Engineering Workshop, December 1981

--, SEL-81-014, Automated Collection of Software Engineering Data in the Software Engineering Laboratory (SEL), A. L. Green, W. J. Decker, and F. E. McGarry, September 1981

Turner, C., G. Caron, and G. Brement, "NASA/SEL Data Compendium," Data and Analysis Center for Software, Special Publication, April 1981

Turner, C., and G. Caron, "A Comparison of RADC and NASA/SEL Software Development Data," Data and Analysis Center for Software, Special Publication, May 1981

Weiss, D. M., "Error and Change Analysis," Naval Research Laboratory, Technical Memorandum, December 1977

Williamson, I. M., "Resource Model Testing and Information," Naval Research Laboratory, Technical Memorandum, July 1979

Zelkowitz, M. V., "Resource Estimation for Medium Scale Software Projects," Proceedings of the Twelfth Conference on the Interface of Statistics and Computer Science. New York: Computer Societies Press, 1979

Zelkowitz, M. V., and V. R. Basili, "Operational Aspects of a Software Measurement Facility," Proceedings of the Software Life Cycle Management Workshop, September 1977