NCR CONTROL DATA CORPORATION

ADVANCED SYSTEMS LABORATORY

APPENDIX E ONLY

LIBRARY HAS ONLY
ONE COPY OF
APPENDIXES A-D
in REFERENCE SECTION.

INTERNAL REQUIREMENTS ON THE OPERATING SYSTEM

DOC. NO. ASL00282

Rev. 04

Copy No. 87

REV 06/13/75

TABLE OF CONTENTS

1.0 SOURCE

The requirements contained herein were drawn from two principal sources. The contents of Section 6.1.3 were drawn from the documents submitted by the various Product Set and internal subsystem design groups as requirements on the Operating System. The contents of Section 6.1.7 were drawn from the IPL RAS Features document, dated 3/21/74, submitted by V.O. Torres and J.A. Wilson.

1.1 NUMBERING CONVENTIONS

The numbering conventions for the requirements set forth herein conform to the numbering system established in the IPL Requirements and Goals document, for major headings (Sections 6.1.3.1 through 6.1.3.14 and Section 6.1.7). Minor headings are organized with the intent of indicating what area of the O.S. is affected by a particular requirement, and are uniform across all major headings. E.g., minor heading 4.3 under any major heading always indicates requirements on Record Management.

1.2 VOID HEADINGS

The numbered outline is intended to be complete, to allow for future expansion. Therefore, some major headings are listed as "To be supplied", indicating that no requirements have been submitted by the applicable design group as yet. Some minor headings are followed by the statement "None", indicating that although requlements have been received from the pertinent design group, none were identified as applying to this area of the O.S.

6.1.3 PRODUCT SET AND SUBSYSTEM REQUIREMENTS ON THE O.S.

6.1.3.1 SWL

6.1.3.1.1 GENERAL REQUIREMENTS

1. The object program makes the following assumptions when it receives control from the IPL environment.

   a. The stack segments and environment registers have been established.
   b. There is no support by the environment in case of a runtime abort.

2. As far as can be determined, the primary user of Release 1.0 will be the IPLOS project.

3. Time of day, date, and interval timer services will be required.

6.1.3.1.2 REQUIREMENTS ON SCL

None

6.1.3.1.3 REQUIREMENTS ON JOB MANAGEMENT

1. Standard Accounting services will be required.

2. Standard Spooling services will be required.

6.1.3.1.4 REQUIREMENTS ON DATA MANAGEMENT

1. The object program must be able to output character and binary data in some form by August, 1975.

2. There is no need to provide compiler support for SWL Input-Output for Release 1.0 as there will not be any IPLOS support for the I-O by the release date.

3. The ability to write sequential legible and binary files from the simulator is a requirement in order to be able to record the results of test case execution.

4. I/O interfaces for creating, opening, accessing, closing and deleting sequential and random text and binary files, and for supporting terminals are required.

6.1.3.1.4.1 Requirements on Volume Management

None

6.1.3.1.4.2 Requirements on File Management

None

6.1.3.1.4.3 Requirements on Record Management

None

6.1.3.1.4.4 Requirements on Block Management

None

6.1.3.1.4.5 Requirements on Device Drivers

None

6.1.3.1.5 REQUIREMENTS ON PROGRAM MANAGEMENT

1. There will be no special action taken to support the execution of SWL programs in multiple rings. The compiler will assume that the entire program will execute within a single ring.

2. The operating system will be responsible for the allocation of the stack segment(s) for the program. It will also be responsible for setting up the canonical address registers and executing the initial procedure call to the SWL program.

3. If coroutines are to be supported, the operating system must provide a mechanism for allocating and freeing the stack segment(s) required for the coroutine.

4. The operating system must take on the major responsibility for managing critical regions, shared-variable locks, events, event queues, the deactivation and reactivation of tasks, the stacking of soft-interrupts attached to event variables, and the activation of interrupt procedures.

5. Shared variables associated with critical regions are in the program's name space; however, their associated queues must be managed by the operating system. Locks on shared variables must also be managed by the operating system. The locks should be associated with descriptors established in system storage by the loader.

6. Some mechanism for determining the ownership of locks on shared variables ("signatured locks") is required to keep a process from stalling itself.

7. Event variables must be shared between processes (but should not be shared variables associated with critical regions). Event variables must be in the system name space, and be capable of being established at run time. However, it is not necessary that all event primitives be implemented by system calls. The object program could interrogate the status of event variables to determine whether or not a system call was necessary.

8. Tasks are characterized oy a procedure and an associated task variable. Asynch procedures can be executed asynchronously; critical procedures can exist only in one process at a time. Global variables are all shared; critical procedures may have local static variables that are not shared. The operating system is responsible for all synchronization and stack management.

9. Although task variables are in the program name space, they are associated with task-control-blocks (at least, for asynch procedures) some of whose elements are within the ken of spawner and spawned. References to these are "qualified" by the task variable, which requires the generation of an associated entry into the system name space at execution time.

10. Critical procedures require a signatured lock to ensure that they exist in, and only in, the calling process.

11. The stack frames associated with the spawning process and with the asynch or critical procedure are critical in that their associated blocks cannot be terminated until all processes depending on them have terminated (alternatively, termination attempts should result in the termination of subordinate processes).

12. The operating system is responsible for initializing and handling stack forks. Operating system support may be required to monitor returns, exits and go-tos across stack forks and critical frames in general.

13. The conventional mechanism for communication and synchronization between the simple kinds of asynchronous processes cited above is the conventional message buffer, which is the only variable that is shared. The exclusion of these should be reconsidered.

14. Soft interrupts and faults result in procedure calls. When an interrupt is caused or a fault is sensed, the state of the interrupted process must be saved in the process stack and a call to the handling procedure generated just as though the call had actually occurred in the interrupted process.

15. The operating system is responsible for: attaching and

detaching interrupts; queuing and handling of the associated event variables; determining when an interrupt procedure is enabled or disabled - and activating or queuing accordingly.

16. The operating system is responsible for fielding all faults, determining whether or not the fault is enabled, and activating the currently attached fault procedure. The system fault-handler, itself called as a procedure, must disengage itself and activate the currently attached fault procedure as though the fault procedure had itself been called from the interrupted process when the fault was sensed.

17. Interrupt and fault procedures may be parameterized: in general, interrupt procedures waiting on any and all events must be notified of which event triggered them; similarly for a fault procedure attached to any and all faults. Fault-specific parameters will probably be required, and interrupt procedures requiring more information may be needed.

18. Information about the existence and status of interrupt and fault procedures must be kept on the process stack. This implies that the operating system can be cognizant of stack structure and that all processes (whether SWL-compiled or not) use a stack.

19. Stack initialization and allocation is required.

20. Allocation of stack space on and after procedure calls will be handled by compiled-out code sequence.

21. Traps on references outside of allocated stack space are required.

22. Stack underflow and overflow require special handling; they are exceptions to the rule of handling fault procedures in the user's stack.

23. Coprocesses are synchronous processes with their own stack. The establishment and switching of stacks associated with coprocess control should not require excursions to the operating system.

24. Standard error and exception handling: set, reset, simulate traps and interrupts; attachment and detachment of exception-handling procedures are required.

6.1.3.1.6 REQUIREMENTS ON STORAGE MANAGEMENT

1. Standard segment creation, limit management, and deletion are required.

2. Standard storage and working-set management is required: get and free pages; specification of 'sticky' parts; overlay control.

3. Special handling of allocated pages to minimize page-fault interference on references to allocated but unaccessed pages would be desirable.

6.1.3.1.7 REQUIREMENTS ON SYSTEM MANAGEMENT

1. The project must be able to link, load, and execute object decks by June, 1975.

2. The use of some form of IPL linking loader is a requirement to link separate SWL compilation and runtime procedures together for execution.

3. We need such facilities as type checking across procedure calls. It seems that the Loader is the appropriate place to perform that task for all languages provided that it is possible to specify the severity of a type conformity error.

The following are all requirements on the loader.

4. Policing of xdcl-xref type matchings, shared type matchings, and parameter type matchings accross separately-compiled modules; these may be either data or program types.

5. Handling and policing of external variables.

6. Establishment and initialization of locks on shared variables and event variables.

7. Packaging of code sections, binding sections and, possibly, static sections for future linking.

8. Handling of context tables in such packagings.

9. Handling of full length SWL identifiers.

10. Establishment and initialization of static section(s) and system heap.

11. Establishment of both SWL-local and global segments.

12. Establishment, and possible allocation and initialization,

of stack segments.

13. General handling of object libraries.

14. Initialization, handling of local segments and establishment of context-table connective tissue for debugging could be handled by a capability for calling user-supplied procedures during loading. The cost for installing such a test on the loader might be preferable to burdening the loader with detailed knowledge of mapping functions, object structures and idiosyncrasies of all possible languages.

6.1.3.1.8 REQUIREMENTS ON OCS

1. Standard Operator Communications services will be required.

6.1.3.2 COBOL

6.1.3.2.1 GENERAL REQUIREMENTS

1. A Message Control System (MCS) is definitely needed.

2. The same general facilities as in the ATG proposal will be needed by COBOL by the time the product is released.

3. The IPL COBOL compiler group anticipates a symbolic dump will be needed by the COBOL programmer as a supplemental debugging aid. Object code is not to be presented to the user since a high level language user has no interest in such detail.

The COBOL compiler should be able to provide (on request, perhaps) the following dumping information as part of the object code file:

1. Symbolic data names

2. A description of each data area:

   a. memory address (PVA format)
   b. length
   c. data type
   d. decimal position (if applicable)
   e. number of occurrences of an item if subscripted

A dump is usually viewed as a system function, and so the IPLOS group should state its position on this matter.

6.1.3.2.2 REQUIREMENTS ON SCL

None

6.1.3.2.3 REQUIREMENTS ON JOB MANAGEMENT

1. The minimum O.S. support required for data recovery is a checkpoint/restart facility to support the RERUN statement. It is permissible to require that this function be specified outside of the source program. If a superior recovery facility is not specifiable outside of the source program, however, then the COBOL RERUN facility must be supported.

6.1.3.2.4 REQUIREMENTS ON DATA MANAGEMENT

None

6.1.3.2.4.1 Requirements on Volume Management

None

6.1.3.2.4.2 Requirements on File Management

1. Support of the 3 file organizations (sequential, relative and indexed) is absolutely required.

2. Indexed file organization must support the existence of several (alternate, not multiple level) indices.

3. The relative file organization "relative key" requires the same treatment as the indexed file organization "prime key".

4. It is required to allow program access to all labels, user and system labels (for security reasons, certain fields of the system labels might have to be blank filled before the label contents are passed to the program). Label processing is planned for all file organizations, not only for sequential files, at OPEN and CLOSE time (beginning and ending file labels) and at beginning and end of volumes.

5. An OPEN of an unavailable file should not automatically discontinue the program; it should put it in a WAIT status, if the OPTIONAL clause is not present, and output a message requesting the file from the operator or the terminal user; it should return an OK status if the OPTIONAL clause is present. The Operating System should also be able to recognize all labelled files and attach them automatically at OPEN time.

6. The Operating System should close all unclosed files attached to a given job when this job reaches end-of-job, whether this is due to a STOP RUN or a job termination by the operator or the Operating System.

7. The Operating System should keep track whether an End of File has occurred and return an error code if a subsequent READ NEXT is executed prior to the execution of a CLOSE followed by an OPEN statement, or the execution of a START or READ with KEY statement for relative and indexed files.

8. Nonpermanent files should be qualified by the job name in order to make them unique in case of multiple executions of the same program.

9. Four file OPEN statements must be supported: OPEN INPUT, OPEN OUTPUT, OPEN I/O, and OPEN EXTEND. The first three are self-explanatory. OPEN EXTEND opens the file in output mode, but positions the file so that the last record is now the preceding record.

10. Three CLOSE statements must be supported: CLOSE FILE, CLOSE REEL, and CLOSE UNIT. CLOSE FILE terminates processing on a file. CLOSE REEL and CLOSE UNIT terminate processing on the current volume and prepare the next volume of the same file for processing. CLOSE REEL/UNIT only apply to sequential files in the output mode.

11. An input file may be declared as optional. This means that the file may or may not be present when opened. If it is not present, then the first subsequent READ statement will give the "At End" condition.

12. Tape files may be labelled or unlabelled. Record formats F, V, D, U, and S must be supported on tapes, the blocking mechanisms defined in the label standards must be supported on tapes, and multi reel files and multi file reels must be supported. String consideration should also be given to support of IBM tape label conventions.

13. When the new label standard is defined in JOD COBOL, strong consideration should be given to including it in IPL COBOL. ASL should ensure that this situation is reviewed periodically to see if ay new requirements on the I/O system emerge.

14. Emerging Requirements

CODASYL currently has a task group (the File Processing Task Group) at work clarifying and extending the I/O facilities of JOD COBOL. ASL should periodically review the progress of the FPTG work to determine whether any of

their proposals should be incorporated in IPL COBOL and what additional requirements such inclusion might impose on the IPL I/O system.

6.1.3.2.4.3 Requirements on Record Management

1. Code conversion does not affect the placement of records for indexed sequential files.

2. Provision must be made for the use of a program specified I/O error routine to be called after completing the standard I/O error routine or upon recognition of an invalid key or end of file condition when an INVALID KEY pnrase or AT END phrase respectively has <u>not</u> been specified in the I/O statement.

3. Four types of record I/O statements must be supported: WRITE, READ, REWRITE, and DELETE. Each may be keyed or unkeyed. WRITE, READ, and DELETE are self-explanatory. REWRITE replaces an existing record. REWRITE and DELETE operate on the last record read, in a sequential organization.

4. A START statement exists in COBOL; its function is basically internal and consists of positioning a file by providing a new key value. Support of this statement by the O.S. (by initiating a SEEK operation) could enhance throughput.

5. The following chart summarizes the valid operations for a file in output mode.

Access	SEQUENTIAL			RANDOM		DYNAMIC	
Organization	SEQ	REL	INDX	REL	INDX	REL	INDX
WRITE (NO KEY)	YES	YES	YES	NO	NO	NO	NO
WRITE (KEY)	NO	NO	NO	YES	YES	YES*	YES*

* Buffering may be advantageous, since WRITE statements may be primarily in ascending order.

6. The following chart summarizes the valid operations for a file in input mode:

Access	SEQUENTIAL			RANDOM		DYNAMIC	
Organization	SEQ	REL	INDX	REL	INDX	REL	INDX
START	NO	YES	YES	NO	NO	YES	YES
READ (NO KEY)	YES	YES	YES	NO	NO	YES*	YES*
READ (KEY)	NO	NO	NO	YES	YES	YES*	YES*

* Buffering may be advantageous

7. The following chart summarizes the valid operations for a file in update mode.

Access	SEQUENTIAL			RANDOM		DYNAMIC	
Organization	SEQ	REL	INDX	REL	INDX	REL	INDX
START	NO	YES	YES	NO	NO	YES	YES
READ (NO KEY)	YES	YES	YES	NO	NO	YES	YES
READ (KEYED)	NO	NO	NO	YES	YES	YES	YES
REWRITE (NO KEY)	YES*	YES	YES	NO	NO	NO	NO
REWRITE (KEYED)	NO	NO	NO	YES**	YES**	YES	YES
DELETE (NO KEY)	NO	YES	YES	NO	NO	NO	NO
DELETE (KEYED)	NO	NO	NO	YES**	YES**	YES	YES
WRITE (NO KEY)	NO	NO	NO	NO	NO	NO	NO
WRITE (KEYED)	NO	NO	NO	YES***	YES***	Yes	YES

* Record size cannot be changed
** Must refer to an existing record
*** Must not refer to an existing record

8. Each file in a program may have associated with it a FILE STATUS data item. This two character item is updated with a status value during each executed reference to the file. It must be possible to uniquely identify these conditions from the status responses of the I/O system.

00 SUCCESSFUL COMPLETION

The usual case.

02 SUCCESSFUL Read of a record with a DUPLICATED KEY value.

For an INDEXED file, a READ NEXT operation, based on an Alternate Key for which Duplicates are Allowed, has retrieved a record which has the same "key of reference" value as that of the next record.

10 AT END (end of file condition) (Sequential Access)

A READ NEXT operation (Sequential or Dynamic Access) was unsuccessful; there are no more records available in the file.

21 INVALID KEY - OUT OF SEQUENCE

- A WRITE to an INDEXED file in SEQUENTIAL OUTPUT mode attempted to create a record with a Prime Key value which was not greater than the previous record written.
- A REWRITE to an INDEXED file in SEQUENTIAL I-O mode did not specify the same Prime Key value as the preceding READ.

22 INVALID KEY - DUPLICATE KEY VALUE

- A WRITE or REWRITE to an INDEXED file would have created 2 records with the same key value in the Prime index, or in one of the Alternate indexes which does not allow duplicates.
- A WRITE to a RELATIVE file addressed a relative record position which was already occupied.

23 INVALID KEY - NO RECORD FOUND

- A START operation did not find a record which satisfied the logical key condition expression.
- A format 2 READ operation (non-sequential access to a RELATIVE or INDEXED file) did not find a record with the key value specified.
- A REWRITE or DELETE statement to a Relative or Indexed file in non-sequential (Random or Dynamic) access mode did not find a record with the key value specified.

24 INVALID KEY - BOUNDARY OVERRUN

A WRITE statement to any file on a mass storage medium has addressed a location which is beyond the externally specified boundary of the file.

30 PERMANENT ERROR

A permanent error may occur at any time that the system attempts a physical I/O operation which results in an unrecoverable error (including OPEN, START (INDEXED files), CLOSE UNIT, and CLOSE, as well as READ, REWRITE, WRITE or DELETE).

6.1.3.2.4.4 Requirements on Block Management

None

6.1.3.2.4.5 Requirements on Device Drivers

None

6.1.3.2.5 REQUIREMENTS ON PROGRAM MANAGEMENT

None

6.1.3.2.6 REQUIREMENTS ON STORAGE MANAGEMENT

None

6.1.3.2.7 REQUIREMENTS ON SYSTEM MANAGEMENT

The following definitions, used by the COBOL design team, are necessary in order to lend absolute clarity to the intent of requirements stated in this section:

"Binding" is the combination of 2 or more object modules into one single object module, requiring offset adjustment and possibly a change in the OP code.

"Linking" is the resolution of external references from one module (either the result of a compilation or of a binding process) to another. It can be done either statically or dynamically at the time of the call or reference.

"Loading" is what the name implies: the loading of a program in the computer memory for execution.

1. Since the COBOL compiler will initialize all data entries declared in the WORKING-STORAGE SECTION, the loader should be capable of zero or space filling large areas. In addition, it should allow initialization of individual data items (VALUE clause).

2. The COBOL compiler requires a linking facility both in static and dynamic modes. There is no requirement for a binding facility. An efficient Linking Loader is all that is required.

3. All external references should be qualified by the name of the module where they are declared.

4. Unresolved references should not cause more than a warning message at linking time. At execution time, they should cause a trap to the Linking Loader to attempt to resolve them.

6.1.3.2.8 REQUIREMENTS ON OCS

None

6.1.3.3 FORTRAN

6.1.3.3.1 GENERAL REQUIREMENTS

1. A means for determining the current CPU time, time of day, and date must be provided.

2. If a digit, or a character, string follows the STOP or PAUSE statement this string must be displayed and must be availaple for examination.

3. Facilities which permit an executing program to display information in the dayfile and/or on a terminal are required for the DISPLA and REMARK sub-routines.

4. A program must be able to distinguish between batch and terminal usage.

5. The first piece of software to detect a condition which caused or will cause an error must flag the error.

6.1.3.3.2 REQUIREMENTS ON SCL

1. It is necessary for a programmer to be able to examine the digit, or character, string, which may accompany a FORTRAN STOP or PAUSE statement, with SCL commands.

6.1.3.3.3 REQUIREMENTS ON JOB MANAGEMENT

None

6.1.3.3.4 REQUIREMENTS ON DATA MANAGEMENT

1. Security. A program must be able to establish its right to access a file. For example it may be able to write on a file when that file is not associated with another program.

2. IPL FORTRAN provides five File and Record Manipulation Statements. These are: REWIND, BACKSPACE, ENDFILE,

BACKFILE, and SKIPFILE.

We suggest that a partitioned file structure should be used to facilitate the implementation of these statements. Partitions within the file can be named and/or numbered. Each partition is separated from its predecessor by an end-of-partition marker which is part of the preceding record. The last partition in the file need not be terminated by an end-of-partition. The file is terminated by an end-of-information marker.

The implementation of a partitioned file scheme should result in maximum flexibility. For example it should be possible to expand a given partition. From the FORTRAN point of view it is not necessary for the partitions to be contiguous on a physical device, so long as the logical structure appears contiguous.

3. REWIND positions the current partition at the beginning of its first record, but has no effect if the partition is at its initial point. ASL/C insist that this statement causes the first record ever written in the sequence of files to become the next record. It is not clear that this is the intention of IPL FORTRAN. This position must be clarified.

4. BACKSPACE positions the file at the beginning of the preceding record. If there is no preceding record BACKSPACE has no effect. This is easily implemented for U and F file organizations and is difficult for all other sequential file organizations. However, the most flexible sequential file organization is the Y type and this will be the IPL FORTRAN default for sequential files. IPL FORTRAN insists that BACKSPACE be available for records in a Y organized file.

   NOTE: An endfile record is counted as a record during execution of a BACKSPACE statement.

5. An ENDFILE statement causes an end-of-partition marker to be written and this may be considered as the FORTRAN endfile Record.

   IPLOS point out that any form of data delimiter involved in the implementation of ENDFILE is likely to cause incompatibility with other language processors.

6. Execution of a BACKFILE statement positions the file at the start of the preceding partition. If there is no preceding partition the statement has no effect.

7. SKIPFILE will position the file at the beginning of the next partition. If a file is positioned at the end of the last partition SKIPFILE will cause an error to be generated.

8. BACKFILE and SKIPFILE are applicable to sequences of sequential files. It is not clear whether or not files in a sequence can be updated, and/or extended. The general consensus of opinion is that the sequence of sequential files usurps the function of the operating system.

   IPLOS will insist that ENDFILE, BACKFILE, and SKIPFILE are restricted to Magnetic Tape files. It is not clear that this approach satisfies ANSI standards.

   A clearer definition of the requirements for these features must be generated.

9. Both the UNIT function and the EOF function need to be able to detect an end-of-information marker.

10. The EOF function must be able to detect an end-of-partition marker.

11. The UNIT function must be able to check for parity errors on a specified device.

12. The function IOCHEC issues a parity check request against a file and _not_ a device. It is understood that if the file connected to the specified unit is a mass storage file _any_ error in the device on whicn the file resides will be taken as a parity error. A single mass storage file is not necessarily mapped into a single mass storage device, and the device may hold more than one file.

13. The function LENGTH must return the number of bytes in the last physical record read by BUFFER IN. This I/O request may have requested more or less bytes than the physical record contained. LENGTH enables the user to determine if the buffer length is correct.

    With the LENGTH function lost data can be indicated but it is understood that it is absolutely impossible for IPLOS to say how much was lost.

14. The SWL I/O facilities were studied and were found to be not sufficiently comprehensive to allow us to implement FORTRAN I/O using SWL I/O. It would not be desirable to do so in any case because it moves the FORTRAN program at least one stage farther away from the OS and hence the external environment.

    The Data Manager will be available as part of IPLOS on the simulator. SWL would like some of the I/O requests to be directly available as part of the simulator, thereby avoiding IPLOS. At the moment IPL FORTRAN would prefer a single interface with the hardware; this interface will be

IPLOS.

15. Sensible default values are required for the Data Management macros where these are not currently supplied. These may be installation dependent.

16. A clear definition is needed of what happens when a Data Management I/O request cannot be satisfied.

17. Choice of suspension or continued execution of a program after issuing an I/O request is required.

18. Whilst requiring specific features in IPLOS to support FORTRAN I/O, it is desirable that files compatible with other language processors can be produced by FORTRAN programs.

19. The IPL FORTRAN ERS will contain a matrix which defines the permissible combination of IPL FORTRAN I/O statements with file organizations and record structures. This will help to clarify the FORTRAN requirements on the O.S.

20. Formatted records are assumed by the O.S. to contain ASCII characters, and conversion utilities may be required.

21. It is not clear whether an attempt to write on a unit which is not connected to a file should cause an error or not. FORTRAN could undertake to connect a scratch file during execution of the first write on the specified unit. The requirements here must be defined.

6.1.3.3.4.1 Requirements on Volume Management

None

6.1.3.3.4.2 Requirements on File Management

1. There is a need for a specific means of associating a FORTRAN unit number with a file name and for associating files with a program. In IPLOS terminology, this means FORTRAN unit numbers must be associated with files and unit numbers must be associated with jobs. A program must be able to determine which files have been associated with it.

   A method of resolving this requirement is suggested:

   The LFN should have standard form. The suggestion is that the LFN is FTN#<N>, where <N> is the FORTRAN unit number. For example, the FORTRAN statement OPEN(10) would generate FTN#10 for the LFN.

   The association between a FORTRAN unit number and a file outside a program can be achieved by job control consisting of a sequence of SCL commands. For example:

```
DCL FTN#10,TYPE=FILE
FTN#10.FN = 'ALPHA'
ATTACH FTN#10
FTN
```

2. One important aspect of FORTRAN I/O is that in general no file organization or specific devices are implied. For example, a program cannot specify that a unit number refers to a magnetic tape. The exception is BUFFER I/O.

3. In order that we may implement sequences of files, IPLOS must provide a partitioned file capability where a logical file (composed of FORTRAN logical records) corresponds to a partition in the file. Each partition should be accessible by name and/or number as a separate entity within the OS.

4. Information regarding file existence must be available to the program.

   A file may exist but not be associated in any way with the program.

   The FORTRAN definition of "file existence" requires clarification. At the moment FORTRAN defines existence with respect to a program. For example, if a FORTRAN unit is CLOSED with STATUS = 'DELETE', the file connected to that unit no longer exists for that program. The user is then at liberty to try to create another file with the same name. The problem is that we are not convinced that the first file should be deleted from the permanent file catalogue if it is a permanent file.

5. If the file is not connected to the program requiring the information, we must know if it is connected to another program, and in what mode.

6. INQUIRE by file name is not possible at the moment. IPLOS must support this feature.

7. Permanent files are known by their "Real IDs"; their names in the permanent file catalogue. FORTRAN may have to keep a table of LFNs and corresponding Real IDs in order to support the INQUIRE statement.

8. FORTRAN does not have to specify a file name and OS requires files to be named, so programs must be able to determine system supplied names. For example, a FORTRAN CLOSE statement can make a scratch file permanent. The file name is an optional parameter on the FORTRAN OPEN statement so

that a FORTRAN program could create a permanent file to which it did not give a name. FORTRAN provides no facilities for the user to identify such a file once the unit to which it was connected is closed.

9. When a FORTRAN OPEN statement does not specify a STATUS parameter, the OS should supply a default which can be made known to the program.

10. Programs must be able to distinguish between Direct and Sequential Access files.

11. If a direct access file was created with the Maximum Record Number property then the maximum number of records that the file can contain is fixed. The maximum length of each record is also fixed but shorter length recods can be employed so that the product of the maximum record number and current record length does not indicate the length of the file.

12. An executing program must have the ability to create a file if it does not exist. However, the program cannot supply information about devices and file organizations (other than sequential or direct) and the GDS does not define default values. It is not possible for IPL FORTRAN library I/O routines to specify the vsn, efnq, gen, ver, or expd parameters of the FILEID macro.

13. Whilst the file creation process is in progress ,the file organization may be U type, at the end of the process the user may wish to change the description of the file organization. Therefore, the ability to redefine the description of a file's organization at runtime is needed. This is not presently possible, as file organization, as well as access method, is fixed at the time the file is created.

14. The default file organization for BUFFER I/O will be the sequential U type file organization.

<u>6.1.3.3.4.3 Requirements on Record Management</u>

1. Definitions:
The basic repository of data in IPL FORTRAN I/O is the logical record and unqualifed use of record in the following sections means logical record. The IPLOS definition of logical record is acceptable to IPL FORTRAN. The four kinds of FORTRAN record are:

formatted - (ASCII)
unformatted - (binary, Variable length)
free-field, and

endfile

The endfile is a record without a length property.

A free-field record is essentially a record of unknown length: unknown, that is, until it is complete. Each free-field I/O request causes the transmission of part of the record.

2. FORTRAN only allows certain combinations of record types and a record must be marked as either formatted (character) or unformatted (binary).

3. Record lengths should be in bytes.

4. The last record of a file need not be an endfile record. This implies that the OS must provide some sort of file termination mark to terminate files and which is distinguishable from a FORTRAN endfile record.

5. IPLOS should flag an error if an attempt is made (on a direct access file) to read a record which has not been written.

6. Implementation of free field I/O will involve the use of discrete records for every free-field write. The FORTRAN library routines will unpack free-field records on input and only issue an input request when the last record read is exhausted. Thus every free-field write will cause an output request to be issued to the operating system, whereas a free-field read will not necessarily cause an input request.

7. The record length of a free-field record is not known until it is complete and we would hope that the entire contents of an incomplete free-field record would not be lost if a program terminated abnormally and the file was still open.

<u>6.1.3.3.4.4 Requirements on Block Management</u>

1. Buffer I/O represents a strict byte by byte transfer of data. No structure can be imposed on the records or file by the OS. The Sequential U file organization suggests itself in this case. However, BUFFER I/O can transmit records of varying lengths and it is not clear whether or not the records in a U organized file can be of varying lengths.

2. BUFFER I/O and block level access should be synonymous. At the moment data transfers can only occur in single blocks and unused space in a block is wasted.

3. BUFFER I/O may be incompatible with a paged environment.

With BUFFER I/O execution continues whilst the I/O request is being satisfied and the user must ascertain when it is complete. For a variety of reasons the Operating System may not choose to allow the program to continue until the request is satisfied. If IPL FORTRAN intends to provide BUFFER I/O then the FORTRAN ERS should make it clear that control may *not necessarily* be returned to the program before the I/O request is complete.

6.1.3.3.4.5 Requirements on Device Drivers

None

6.1.3.3.5 REQUIREMENTS ON PROGRAM MANAGEMENT

None

6.1.3.3.6 REQUIREMENTS ON STORAGE MANAGEMENT

None

6.1.3.3.7 REQUIREMENTS ON SYSTEM MANAGEMENT

None

6.1.3.3.8 REQUIREMENTS ON OCS

None

6.1.3.4 RPG

6.1.3.4.1 GENERAL REQUIREMENTS

1. Definitions

In this document we distinguish between 'mandatory services', 'desirable services', and 'exploitable services'.

Mandatory services are considered the minimal requirements for effective RPG support.

Desirable services will ease program conversion and encourage migration.

Exploitable services are not required by RPG but will be externalized to the IPL RPG user.

2. Telecommunications

IPL RPG will have a telecommunications capability. I would like to defer a detailed analysis of requirements until I have studied the 'standard terminal definition'.

6.1.3.4.2 REQUIREMENTS ON SCL

None

6.1.3.4.3 REQUIREMENTS ON JOB MANAGEMENT

None

6.1.3.4.4 REQUIREMENTS ON DATA MANAGEMENT

1. RPG requires the following *mandatory* interface:

RPG allows the programmer to specify his own procedure for I/O error conditions. Data management must look for such an error procedure on I/O error conditions.

2. An RPG implementation on IPL will only be effective if the compiler can accept EBCDIC files containing fields with any of the data types defined by the RPG de facto standard.

To accomplish this, the following services are *desirable*:

An intercept provided such that all records read from a tape may be translated under control of the RPG program. This includes all label records.

A link back to the data management routine after labels have been translated such that the labels are checked by the system label checking procedures.

An 'on the fly' utility provided that will accomplish translation of a record only after the RPG program has recognized its data type composition.

6.1.3.4.4.1 Requirements on Volume Management

None

6.1.3.4.4.2 Requirements on File Management

1. Support of the following file structures is *mandatory*:

Sequential F file structure	(SF)
Sequential D file structure	(SD)
Sequential S file structure	(SS)
Relative fixed length structure	(RF)
Indexed File Organization	(IS)

2. Support of the following interfaces is *mandatory*:

RPG allows the programmer to specify his own procedure for processing labels whether they be non-standard or ANSI standard user labels. Data management is required to allow the specification of two label processing procedures. One procedure for non-standard labels or ANSI UHL's; the other for ANSI UTL's.

3. Support of the following file structures is *desirable*:

Sequential U file structure	(SU)
Foreign file organization	
NCR variable length structure	(2 byte VLI)

4. Support of the following feature is *desirable*:

De facto standard RPG allows label procedures to be specified on mass sto~age devices irrespective of file organization.

5. Support of the following file structures is *exploitable*:

Sequential Y file structure	(SY)
Relative variable length structure	(RV)
User defined file organization	

6. The 'alternate' key feature of Indexed files is *not mandatory* though it is *exploitable*.

6.1.3.4.4.3 Requirements on Record Management

1. Support of the following record requests is *mandatory*:

REQUEST	USAGE	FILE ORGANIZATION
GET	I,IO	SF,SD,SS,IS,RF
GETKEY	I,IO	SF (see note),IS,RF
PUT	O,IO,E	SF,SD,SS,IS,RF
PUTKEY	O,IO	IS,RF
REPLACE	IO	SF,SD,SS,IS,RF
DELETE	IO	IS,RF
DELKEY	IO	IS,RF
FINDKEY	I,O,IO	IS,RF
FINDD	I,O,IO	SF,SD,SS,IS,RF

NOTE: A sequentially organized mass storage file, that has fixed length file structure, may have its records randomly accessed by relative record number in an RPG program.

2. RPG has the following *mandatory* requirements on record address values:

a) Record addresses are relative to the start address of a file (and would be valid for a copy of the file).

b) Record addresses are valid for the life of a file as long as the user does not update the file in such a way that record positions are altered. This means that data management must not reorganize records without the users acknowledgement.

c) Record addresses can be used to access records in any type of file organization.

3. Support of the following record requests is *desirable*:

REQUEST	USAGE	FILE ORGANIZATION
GET	I,IO	SU
PUT	O,IO,E	SU
REPLACE	IO	SU
FINDD	I,O,IO	SU

4. Support of the following interface is *desirable*:

De facto standard RPG allows signed packed as well as alphanumeric keys for indexed sequential files. We request that keys be communicated to the access method through the use of parameters giving address, length in bytes, and data type.

5. Support of the following record requests is *exploitable*:

REQUEST	USAGE	FILE ORGANIZATION
GET	I,IO	SY,RV
GETKEY	I,IO	RV
PUT	O,IO,E	SY,RV
PUTKEY	O,IO	RV
REPLACE	IO	SY,RV
REPKEY	O(see note),IO	IS,RF,RV
DELETE	IO	RV
DELKEY	IO	RV
FINDKEY	I,O,IO	RV
FINDD	I,O,IO	SY,RV

NOTE: RPG allows records to be added to an existing file which has output usage and indexed or relative file organization. Such addition of records is subject to a 'duplicate record' situation, i.e., his request to 'overwrite' the existing record would be serviced by REPKEY.

6.1.3.4.4.4 Requirements on Block Management

1. Support of the following feature is desirable:

   De facto standard RPG allows indexed file keys to be contained within a block prefix when the records are not blocked.

6.1.3.4.4.5 Requirements on Device Drivers

None

6.1.3.4.5 REQUIREMENTS ON PROGRAM MANAGEMENT

1. It is a high priority objective of the RPG project that the RPG user need never see a 'hex' dump.

   Support of the following interfaces is therefore mandatory:

   A hook provided between the OS program error routine and RPG's symbolic dump formatter.

   An interface provided whereby the RPG symbolic dump formatter may read the core image RPG program (that is in error) before the job is terminated. This interface should be generalized so that it is available to a 'dynamic' symbolic dump which returns control of an executing program.

6.1.3.4.6 REQUIREMENTS ON STORAGE MANAGEMENT

None

6.1.3.4.7 REQUIREMENTS ON SYSTEM MANAGEMENT

None

6.1.3.4.8 REQUIREMENTS ON OCS

None

6.1.3.5 PL/I

Requirements to be supplied

6.1.3.6 BASIC

Requirements to be supplied

6.1.3.7 APL

Requirements to be supplied

6.1.3.8 SORT/MERGE

6.1.3.8.1 GENERAL REQUIREMENTS

None

6.1.3.8.2 REQUIREMENTS ON SCL

None

6.1.3.8.3 REQUIREMENTS ON JOB MANAGEMENT

None

6.1.3.8.4 REQUIREMENTS ON DATA MANAGEMENT

None

6.1.3.8.4.1 Requirements on Volume Management

None

6.1.3.8.4.2 Requirements on File Management

1. A fast open function is needed for scratch/temporary files.

2. The capability to switch processing states on files must exist in the form of a RE-OPEN function (i.e., write a temporary file and then in the same program be able to read the file).

3. Because the SORT will be working in a shared media environment a requirement exists to uniquely identify the temporary work files associated with each sorting function.

6.1.3.8.4.3 Requirements on Record Management

None

6.1.3.8.4.4 Requirements on Block Management

None

6.1.3.8.4.5 Requirements on Device Drivers

1. Certain SORT techniques require the use of read backward tape. If the hardware is provided the device drivers need to provide the read backward function.

6.1.3.8.5 REQUIREMENTS ON PROGRAM MANAGEMENT

None

6.1.3.8.6 REQUIREMENTS ON STORAGE MANAGMENT

None

6.1.3.8.7 REQUIREMENTS ON SYSTEM MANAGEMENT

1. There exists a need to dynamically link/bind modules at run time.

2. The need also exists to be able to link/edit modules prior to execution/run time.

6.1.3.8.8 REQUIREMENTS ON OCS

None

6.1.3.9 DBMS and Data Utilities

6.1.3.9.1 GENERAL REQUIREMENTS

1. An explanation is required in the IPLOS Structure Overview of how the OS intends tapes to be used.

2. In the IPLOS some means of associating files into processing groups must exist (i.e., associating one or more user files to a common log file). The Data Recovery utility must have a means whereby it can ascertain the identity of the user in order to properly track the usage of monitored files.

6.1.3.9.2 REQUIREMENTS ON SCL

None

6.1.3.9.3 REQUIREMENTS ON JOB MANAGEMENT

1. Although the logging utility will not monitor entire Checkpoint files it must be able to uniquely identify the Checkpoint in order to properly recover data files to a predetermined point in time.

2. The Checkpoint function must call the Logging utility at the

beginning of each Checkpoint that is requested, if the owner of the file has indicated that tracking of Checkpoints is required.

3. Control on user abort sufficient for us to flush buffers, etc.

6.1.3.9.4 REQUIREMENTS ON DATA MANAGEMENT

1. Password checking for user and terminal ID's, and macros to retrieve the ID's.

2. Ability to rename the record access method processing a file to be our own method. Our method must be able to use standard record requests and open additional files.

3. Asynchronous I/O is required.

4. Wait option with time limit on data management requests.

5. Data streams are required.

6. Data streaming is required.

6.1.3.9.4.1 Requirements on Volume Management

None

6.1.3.9.4.2 Requirements on File Management

1. Rapid open-close sequences.

2. Multiple concurrent independent opens in a run, task, etc.

3. A method to relate our data description files with user files.

4. Concurrent update (multiple writers) on all disk file organizations supported by Cobol.

5. The Data Recovery utility must work in harmony with the File Manager. The Logging portion of the utility should be called by the File Manager whenever any file that is to be monitored is opened.

6. The Logging utility will need to access the Request Block of the user file being opened.

6.1.3.9.4.3 Requirements on Record Management

1. Locking via record requests, including Finds, locking by file address, and locking of all records on a file.

2. Option on Find for obtaining a file address without record retrieval.

3. Delete and Replace by file address and key so we can modify records in addition to the last one read.

4. Pointer mode of Get, to allow inspection of the record without transfer into a separate record buffer.

5. Identical options for major and minor index keys for a multiple-index file (duplicates permitted/restricted, key modification permitted/restricted, etc.).

6. The Logging utility must be attached in such a manner that all Record I/O requests for monitored files pass through the logging utility.

6.1.3.9.4.4 Requirements on Block Management

None

6.1.3.9.4.5 Requirements on Device Drivers

None.

6.1.3.9.5 REQUIREMENTS ON PROGRAM MANAGEMENT

1. Use of LNS by some modules is required.

2. A simple way to determine at run time a routine's program name, the date/time of compilation, and the compiler version used.

6.1.3.9.6 REQUIREMENTS ON STORAGE MANAGEMENT

1. Secure libraries to restrict user substitution of our major routines at run time.

2. Shared segments between runs, with serialization macros provided.

6.1.3.9.7 REQUIREMENTS ON SYSTEM MANAGEMENT

1. Dynamic link loading is required.

2. Common requests for all terminal types.

6.1.3.9.8 REQUIREMENTS ON OCS

None

6.1.3.10 Media Utilities

Requirements to be supplied

6.1.3.11 System Utilities

Requirements to be supplied

6.1.3.12 IOSS

6.1.3.12.1 GENERAL REQUIREMENTS

1. Mutual Exclusion to Shared Resources by serializing user access is required, bearing in mind that some "users" are on the hardware/firmware side of the IOSS interface.

6.1.3.12.2 REQUIREMENTS ON SCL

None

6.1.3.12.3 REQUIREMENTS ON JOB MANAGEMENT

None

6.1.3.12.4 REQUIREMENTS ON DATA MANAGEMENT

1. A requirement for data streaming exists. It is a requirement that the operating system provide the necessary close-coupling of the user's buffer condition with calls upon the device interface software in order to implement the required level of data streaming.

6.1.3.12.4.1 Requirements on Volume Management

None

6.1.3.12.4.2 Requirements on File Management

None

6.1.3.12.4.3 Requirements on Record Management

None

6.1.3.12.4.4 Requirements on Block Management

None

6.1.3.12.4.5 Requirements on Device Drivers

None

6.1.3.12.5 REQUIREMENTS ON PROGRAM MANAGEMENT

1. Event Creation, Posting and Wakeup services are required.

6.1.3.12.6 REQUIREMENTS ON STORAGE MANAGEMENT

1. Proolems must be solved in the OS design for relating the real memory address of tables to a similar virtual address.

6.1.3.12.7 REQUIREMENTS ON SYSTEM MANAGEMENT

1. At system initialization, and potentially whenever a processor is restarted, the location of tables used on both sides of the hardware/controlware/software interface must be established for all users.

2. The data structures of the interface must be initialized. The operating system must establish initialization and restart procedures in a general sense, and must include provisions for the IOSS tables and data structures.

3. Visibility to the mechanisms and capabilities for generating a system from miscellaneous modules is required.

6.1.3.12.8 REQUIREMENTS ON OCS

1. The operating system must provide a path by which the system operator can communicate with the device interface software on problems of mutual concern.

6.1.3.13 MSS

6.1.3.13.1 GENERAL REQUIREMENTS

1. The system (hardware and OS) must be designed so that the system down MTBF is a minimum of 168 hours of system power on time.

2. The IPLOS must be designed to function with a minimum number of critical hardware elements.

6.1.3.13.1.1 Error Detection

1. The general requirement is that IPLOS be capable of detecting all of the fault types which are inherent to a given system element. System elements which cannot cause traps/interrupts or otherwise signal a fault state must be

periodically polled.

2. Software timeouts must be provided for all channel communication (if not provided by hardware).

3. Software timeouts must be provided for processor activity in a multiprocessor environment.

4. Hardware status registers must be periodically examined for fault status (if no special signal is generated by one or more classes of faults).

5. Errors in system elements which are not "directly interfaced" to IPLOS must be reported back and detected by IPLOS via standard system protocol.

6. Recoverable system errors (hardware and OS) should be invisible to the customer.

7. IPL Errors detected by IPLOS must include:

   7.1 Memory

   Single error detected
   Uncorrectable error

   7.2 Processor

   Processor malfunction condition bit set and processor fault status register value
   Processor hung (timeout in multiprocessor system)

   7.3 Data/address paths

   Parity

   7.4 Peripherals

   Controller malfunction including timeout
   Device malfunction
   Media malfunction

   7.5 Networks

   Node, Line, Device, Media

   7.6 Other

   Power failure imminent

6.1.3.13.1.2 Damage Assessment

1. The requirement here is that IPLOS upon detection of a fault immediately attempt to assess the damage caused by the fault.

2. Damage assessment must differentiate between critical errors and noncritical errors.

3. IPLOS equipment/configuration/allocation/assignment tables describing IPL hardware elements must be designed to allow damage assessment to efficiently pinpoint impacted processes/tasks/jobs.

4. IPLOS damage assessment must not be externally interruptable.

6.1.3.13.1.3 Recovery

1. Error recovery procedures defined and approved for the IPL must be implemented.

2. Recovery action involving unrecoverable errors in noncritical elements will not result in system shutdown.

3. OS automatic recovery procedures must be provided, such as data transfer retry on parity error, retry on timeouts, reconfiguration when a solid fault is detected, etc.

4. Recovery action involving unrecoverable errors in critical system elements will be to attempt to initiate a system recovery.

5. Jobs utilizing noncritical system elements which have unrecoverable errors must be temporarily suspended, restarted, or rerun, but, in any case, not allowed further access to the element until repair is effected.

6. Error conditions should be recoverable after a repair has been made without having to rerun the entire job.

7. System restart must be capable of being initiated without operator intervention.

6.1.3.13.1.4 Reconfiguration

1. The OS must provide capabilities for system degradation and reconfiguration so that there are a minimum number of system critical elements.

2. System reconfiguration capabilities must exist so that equipment can be worked on concurrently with customer

operaton (PM and remedial maintenance).

3. Redundancy of all units in the system should be supported by the OS for the customer that requres a high degree of availability.

4. Reconfiguration must include:

   4.1 Utilization of alternate paths to an element

   4.2 Logical deletion of a noncritical system element.

   4.3 Full access to logically deleted system elements for a maintenance task through standard system drivers, etc.

   4.4 Reinstatement of logically deleted system elements as well as addition of "new" elements

   4.5 Logical deletion, maintenance access, and reinstatement of noncritical portions of system elements.

5. Reconfiguration of critical system elements must be supported at system restart subject to the following considerations:

   5.1 The system will restart without the services of the element

   5.2 The system will operate without the services of the element

   5.3 The system restart process must be able to accept configuration parameters from an external source

   5.4 The system must accept "new" elements introduced during system execution (elements which were configured out during restart).

6.1.3.13.1.5 Concurrent Repair

1. Diagnostic programmers must work with the OS programmers so that on-line diagnostic capability is built into the OS.

2. The OS must provide clear information and procedures to the customer's personnel when a nonrecoverable system fault is detected. Where possible, the OS should automatically call in the necessary diagnostic.

3. On-line diagnostic programs must operate concurrently with the customers operation, whether from a local or remote console.

4. Features must be provided for updating the maintenance software concurrent with customer operation.

5. Area on mass storage devices must be reserved by the OS so that diagnostic tests can test the storage device using the reserved area. The OS must provide protection from the diagnostics to the other areas of the storage device.

6.1.3.13.1.6 Remote Access

1. Phone line couplers and related software must be provided which will provide the same maintenance testing capabilities to a remote C.E. as is provided to a local C.E.

2. To satisfy some customer's security requirements, provision must be made so that the customer has control of when remote access to his system is allowed.

6.1.3.13.2 REQUIREMENTS ON SCL

1. Specific reconfiguration requests which can be issued by the MSS task will include the following:

   1.1 Memory

   Assign page frames (contiguous real memory locations) or memory banks to MSS task.

   1.2 Processor

   Idle processor - this will effectively take an IPL processor "off-line" and make it available exclusively for maintenance functions.

   Activate processor - Return processor to IPLOS activity. Assign a specific processor to a specific task.

   1.3 Peripheral

   Turn device "off" - Suspend normal access to the device. MSS will request reinstatement upon maintenance completion.

2. The IPLOS must be able to respond to MSS requests for "immediate" idle down and checkpoint system.

6.1.3.13.3 REQUIREMENTS ON JOB MANAGEMENT

1. Error/usage log data should be separate from customer logs, and not accessible by the customer.

2. All hardware errors detected on elements not in a maintenance state must be recorded. Multiple occurrences of correctable errors may be logged as single entries including occurrence counts. Error logging is optional for those elements in a maintenance state (being exercised by a CE through MSS).

3. The OS must log operating hours or events (lines printed, cards punched, etc.) for each unit in the system to allow preventive maintenance actions to be determined.

4. The OS must enforce maintenance action logging (i.e., the C.E. must log repair data before returning system to the customer).

5. Maintenance log information will include date, time, and element i.d. (where applicable) as well as a variable amount of data including type identification.

6. The maintenance log must be accessable/purgeable only by an operator of class CE#OP.

7. The maintenance log must be recoverable across system restarts.

8. The error/usage log should be periodically analyzed and the customer and/or C.E. notified if immediate maintenance action is required. The limits used to determine maintenance action should only be selectable by the C.E.

9. Space requirements for the error/usage logs should be minimized. Data compaction techniques should be used so that log overflows do not occur between maintenance periods.

10. MSS "tasks" must be schedulable on the folloing basis:

    10.1 Time (elapsed) - the MSS task should execute at fixed intervals of time to perform such functions as maintenance log analysis, confidence level testing, etc.

    10.2 Time (of day) - the MSS task should be executed at certain times of the day to perform "scheduled" testing, analysis, etc.

    10.3 System idle - the MSS task should be executed during idle system periods.

    10.4 Event driven - the MSS task should be called into execution based on certain system conditions occurring.

10.5 Explicit call - the MSS task should be executed upon a level CE#OP operator console request.

11. The MSS task must be able to execute in a "privileged" state which may include Monitor mode, specific ring numbers, segment numbers/descriptors, etc.

12. The MSS task must be able to create other asynchronous "diagnostic" tasks and communication between tasks must be supported in a convenient and efficient fashion.

6.1.3.13.4 REQUIREMENTS ON DATA MANAGEMENT

None

<u>6.1.3.13.4.1 Requirements on Volume Management</u>

None

<u>6.1.3.13.4.2 Requirements on File Management</u>

None

<u>6.1.3.13.4.3 Requirements on Record Management</u>

None

<u>6.1.3.13.4.4 Requirements on Block Management</u>

None

<u>6.1.3.13.4.5 Requirements on Device Drivers</u>

1. A service processor "IORP" mechanism must be supported to enable request processing between the service processor and the MSS task.

2. System I/O drivers must be capable of supporting all diagnostic/maintenance features, such as:

   - support of all H/W functions
   - activate/deactivate error checking logic
   - utilization of a H/W "echo" feature

3. Equipment/Device tables must contain certain fault history information such as fault counts, fault thresholds, time stamp of last fault, etc. and device drivers must update this information where applicable.

6.1.3.13.5 REQUIREMENTS ON PROGRAM MANAGEMENT

None

6.1.3.13.6 REQUIREMENTS ON STORAGE MANAGEMENT

1. The IPLOS must maintain a pool of page frames which may be utilized by maintenance/recovery function without impacting system recoverability. The page frames must be identifiable by service processor firmware referencing memory in a "dead OS" situation.

6.1.3.13.7 REQUIREMENTS ON SYSTEM MANAGEMENT

1. The OS should be capable of performing a controlled power down sequence when it has detected that the electrical or cooling system is going down (electrical power loss, chilled water loss, etc.)

2. A linker/loader must be provided.

3. Diagnostic/test libraries including IPL source and object code and firmware source and object must be maintainable on system mass storage utilizing "standard" library maintenance procedures. These libraries must be maintained utilizing checksums and/or other verification mechanisms.

6.1.3.13.8 REQUIREMENTS ON OCS

1. IPLOS must be able to accept a login of an operator of class CE#OP from any valid IPL supported terminal.

2. Security considerations and command syntax must be the same for all CE class operator consoles whether local or remote.

3. System command language interpreters must enable processing of "NCS format" maintenance commands.

<u>6.1.3.14 Compatibility Subsystem</u>

6.1.3.14.1 GENERAL REQUIREMENTS

1. Compatibility Subsystems will take specific advantage of the multiple monitor concept as outlined in 2.1.1 of the OS Structure document.

2. Each CSS user of a particular CSS type; C1, Cyber 3000, will be assigned to the singular monitor for that Subsystem type.

3. The listed Record, File, LNS, Program Communication and

Program Execution requests seem entirely adequate for our objectives.

6.1.3.14.2 REQUIREMENTS ON SCL

1. A command to invoke compatibility operation is required.

6.1.3.14.3 REQUIREMENTS ON JOB MANAGEMENT

1. Each logical target system will operate as a task.

2. The signalling mechanism must be both efficient in operation and general in nature.

3. The problem to be solved requires an IPL task (specifically the interface processor - CIP) to signal processes that are as diverse as:

   a. Century interpreter, an IPL task limited to P1.

   b. 3000 interpreter, an alternate P1 machine state on selected P1's.

   c. Cyber interpreter, an alternate P2 machine state on selected P2's.

4. The IPL emulator task, 3000L or Cyber must be assigned to the particular processor within the system with that interpreter capability.

6.1.3.14.4 REQUIREMENTS ON DATA MANAGEMENT

1. Access to old data base management software and files from IPL users is required for the life of the migration task.

2. The host IPL task must have knowledge of the structure of the target file in terms of the externalization of that file's address scheme (sequential, index sequential, or whatever else the parent corporations have supported).

3. OS services and access methods must be resident in two distinct virtual machine environments.

6.1.3.14.4.1 Requirements on Volume Management

None

6.1.3.14.4.2 Requirements on File Management

None

6.1.3.14.4.3 Requirements on Record Management

None

6.1.3.14.4.4 Requirements on Block Management

None

6.1.3.14.4.5 Requirements on Device Drivers

1. The requirement for device drivers for CDC or NCR non-IPL devices is an absolute requirement on the part of both parent companies.

2. All devices (alien or IPL) dedicated to CSS are controlled by IOCB's through the same RSM protocol.

3. Processor interpreters are also controlled by IOCB/IORP structures using the same RSM protocol.

4. Assignment of processes and processes in particular processors of IOCB'S and IORP's is required.

6.1.3.14.5 REQUIREMENTS ON PROGRAM MANAGEMENT

1. Restoration of alternate states to IPL processors upon return of control to those processes after interruption must be automatic and efficient.

2. Code sharing between CSS subsystems is a requirement.

6.1.3.14.6 REQUIREMENTS ON STORAGE MANAGEMENT

1. No instruction interpreter, either emulative or partial software will reserve real memory for its use.

2. Most compatibility subsystems will utilize only a single mapped memory segment, although Cyber may utilize an additional virtual segment as ECS.

3. All interpreters, firmware or software will utilize Map service in firmware for Map faults. A provision in the exchange packages for all processors must be made to allow interrupting to IPL state for page services and restoration of the interrupted processor state upon completion.

6.1.3.14.7 REQUIREMENTS ON SYSTEM MANAGEMENT

None

6.1.3.14.8 REQUIREMENTS ON OCS

None

6.1.7 RAS REQUIREMENTS ON THE OPERATING SYSTEM

6.1.7.1 General Requirements

1. Detection - Software is inherently no more reliable than hardware, and in practice, is frequently less reliable, thereby limiting the reliability of the total system. This fact is recognized and accepted in the IPL where software procedures will be incorporated to repair failing modules and ensure continued operation of the system. The key to this is detection. Checksums, parity indicators and other techniques will be used to validate the integrity of key system tables and parameters.

2. Hardware Fault Detection - The following percentage of hardware faults should be detectable using various combinations of techniques:

75%	detectable by hardware alone
80%	detectable by hardware and software combined
95%	detectable by hardware and diagnostics combined
99%	detectable by hardware, software, and diagnostics combined

3. Fault Isolation - When a fault occurs, fault isolation procedures will be invoked to determine the extent of the damage.

4. The OS should be able to isolate 80% of software faults to the product responsible.

5. The OS must record fault isolation data to support Log Analysis programs.

6. Reconstruction - A class of software errors manifests itself by destroying part of the environment. Procedures will be provided in the IPL to reconstruct this environment when the condition is detected. This reconstruction will include repeating portions of the preceding job steps, if necessary.

7. When tables and pointers have been corrupted, the IPL operating system will activate procedures to reconstruct them. If data in memory cannot be used for this purpose then back-up data carried on a particular device will be used. Three facilities are provided on mass storage files to accomplish this. These facilities are specified in 6.1.7.4.1 and 6.1.7.4.2.

8. Reconfiguration - When a permanent failure is detected, typically subsequent to retry, the system will be reconfigured to continue operation. A combination of hardware and software techniques will be used to achieve this, and the goals will be to make the process fully automatic.

9. Standardization - Techniques for designing, coding and documenting the system will be standardized. Implementation, in particular, will be by high level language using well defined structured programming methods. Modules will be broken down into procedures a maximum of two pages long with a single entry and exit, and will be limited to pre-specified structures such as DO-WHILE, IF-THEN-ELSE and simple statements. In addition, all code will be generated as pure procedures. Design will use a top-down approach which will establish the basic system framework and guarantee the modularization discussed below.

10. Modularization - The system will be separated into functional modules whch are then placed in water-tight compartments. That is the data bases on which each module operates will be clearly defined, as will interfaces with other modules.

11. The IPL operating system will be constructed such that if one module fails in a catastrophic manner then the remaining modules will not be destroyed or affected.

12. Both separation by function, and division into self-contained procedures help to isolate a problem to a small code segment. This code segment will then be tested in a simulated environment.

13. Control - Privileged operational modes will exist to allow the maintenance subsystem, under program control, to vary margins, to master clear, to set internal states, to override faults, etc.

14. Source Level Maintenance - The IPL will use a comprehensive source level maintenance system, which will permit concurrent fault repair and evolutionary development.

15. Diagnostics - The objective of the IPL software diagnostics is to isolate a fault to a particular failing procedure. To achieve this they will operate in a simulated environment, if necessary.

16. Integrated on-line diagnostics - To assure the successful development and implementation of on-line diagnostics, it is necessary to have the diagnostics and their supporting software designed and developed by the operating system design team. It will not serve the success of the IPL to have these functions split off into separate organizations. The success of this integration will be evident by the presence of diagnostic and MSS sections in the operating system GDS.

17. Simulation - An IPL environment simulator will exist to provide a mechanism for fault isolation and repair of development software concurrent with customer operations.

18. The O.S. must freeze features by DR time.

19. Testing - All paths must be tested during Unit Test, and all interfaces must be tested in a separate Interface Test.

6.1.7.2 Requirements on SCL

None

6.1.7.3 Requirements on Job Management

1. Error Log - All errors detected on all devices or media will be recorded in the system maintenance file.

2. All software errors will be recorded in the system maintenance file.

6.1.7.4 Requirements on Data Management

1. User Exits - Upon the initial detection of an error, or after standard system error procedures have been executed, the user will be able to take an exit to invoke his own error recovery algorithms.

2. Checksums - Tables controlling I/O transfers will be checksummed, or individual entries will carry a parity indicator.

3. Corroboration - Certain functions such as request issue will be validated by corroborating data contained in the request against data contained in a separate software table.

4. Permissions - Read, write and modify permissions are granted on an individual file basis. The software will ensure

thatthese permissions are not breached.

6.1.7.4.1 REQUIREMENTS ON VOLUME MANAGEMENT

1. Device Chaining - Each allocation unit of each file is related to its successor and its predecessor. This data enables the reconstruction of device labels when necessary.

6.1.7.4.2 REQUIREMENTS ON FILE MANAGEMENT

1. Device Labels - Device labels carry information on the allocation of all files on a given device.

2. File Labels - File labels contain sufficient data in themselves to permit reconstruction of permanent file directories in the event that they have been destroyed.

6.1.7.4.3 REQUIREMENTS ON RECORD MANAGEMENT

None

6.1.7.4.4 REQUIREMENTS ON BLOCK MANAGEMENT

None

6.1.7.4.5 REQUIREMENTS ON DEVICE DRIVERS

1. Standard Error Recovery Algorithms - Standards will be defined for the IPL governing the recovery from all device or medium errors.

2. Locks and Keys - Hardware locks will exist on all devices preventing a write on a device unless the correct software key has been issued to enable a write.

3. Write Certainty Checks - The IPL hardware logic at the recording head will perform a write only if separate signals from the controller and device driver indicate that a write was intended.

4. Position Certainty Checks - The hardware will ensure that a write will only take place where it was intended to occur. A validation check with the software address will be made to ensure this.

6.1.7.5 Requirements on Program Management

None

6.1.7.6 Requirements on Storage Management

None

6.1.7.7 Requirements on System Management

1. The ability is required to be able to load firmware from system devices via an O.S. interface.

2. All diagnostic development for the IPL will be of the on-line variety. The only off-line varieties are those which can be incorporated as a part of the system's loading procedure. In other words, if, because of a hardware malfunction, it is not possible to load the operating system, then the system loading procedure must contain the means by which it can determine (diagnose) the reasons for the failure to load. (Diagnostics should be to the plug-in board level.) Therefore, the system loader, before attempting to load or move on to the next process, must make a cursory examination of those facilities it is about to use and, if necessary, call in diagnostics to examine further questionable facilities.

6.1.7.8 Requirements on OCS

1. Consoles - The IPL will not have a dedicated maintenance console or oerators console. Normal remote terminals with keyboards and display facilities will be used for this purpose. A console will become either a maintenance or an operator console by software control.