CR-159,152

NASA-CR-159152

1982 0007941

NASA Contractor Report 159152

(NASA-CR-159152) FUNCTIONAL DESCRIPTION OF THE ISIS SYSTEM (Virginia Univ.) 28 p HC A03/MF A01 CSCL 09B

N82-15814

Unclas
G3/61 08730

FUNCTIONAL DESCRIPTION OF THE ISIS SYSTEM

W. Joseph Berman

UNIVERSITY OF VIRGINIA
Charlottesville, Virginia 22904

NASA Contract NAS1-14862
October 1979

NASA
National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

Functional Description of the ISIS System

1. Introduction

The development of software for avionic and aerospace applications (flight software) is influenced by a unique combination of factors. This combination of factors includes the length of the life-cycle of each project, the necessity of cooperation between the aerospace industry and NASA, the need for flight software to be highly reliable, the increasing complexity and size of flight software, the high quality of the programmers and the tightening of project budgets. The Interactive Software Invocation System (ISIS) is designed to overcome the problems created by this combination of factors.

The life-cycle of flight software is usually several years. During this period, there is a need for a stable software development environment. Furthermore, flight software for NASA is usually a cooperative effort between one or more aerospace contractors and one or more NASA centers. This cooperation would be significantly simplified if the software development environment were transportable. ISIS achieves stability and transportability by having a minimal dependence upon its host operating system and by being written in the higher-order language PASCAL.

Flight software must be highly reliable. Consequently, large quantities of information are required during the specification, design, integration and testing of flight software. This information includes data which is organized into tables (e.g., sensor characteristics) as well as textual data (e.g., source code). ISIS includes a powerful data editor to manipulate this diverse data, and a sophisticated file manager to facilitate the storage and retrieval of this data.

As the reliability and power of flight computer hardware have increased, the tasks assigned to the flight computer have significantly increased in complexity and scope. In order to generate the necessary software, assembly language is being dropped in favor of higher-order languages (HOLs). In addition, many new tools have been developed for assisting in the process of writing correct software. The ISIS text editor has been designed for the new capabilities of expression (e.g., mnemonic names of arbitrary length and non-rigid format) that are found in HOLs, and ISIS has a simple, yet sufficient, capability for "invoking" software tools.

The real-time aspect of flight software makes it difficult to write. While the introduction of HOLs and tools will help, it will remain true that flight software requires specialized programming skills. In order to attract and retain high quality programmers, high salaries are common. At the same time, however, flight software projects are under strict budget requirements. This suggests that the software development environment should facilitate programmer productivity as much as possible. ISIS increases programmer productivity by being highly interactive, and by having a consistent, easy-to-use unified command language.

1.1 Development of ISIS

A contract was awarded to Dr. W. Joseph Berman at the University of Virginia to develop an engineering prototype of ISIS. The design and implementation of ISIS was evolutionary in nature. Preliminary versions of ISIS were installed at Langley Research Center (LaRC) to allow testing of the "feel" and effectiveness of both the functional capabilities of the system and the command syntax. Once sufficient capabilities were available, ISIS was used to complete its own implementation.

ISIS was originally developed under the CDC NOS-BE operating system at the University of Virginia and the CDC NOS

1.2 operating system at LaRC. The transportability of ISIS is being tested by efforts to rehost it to an IBM 370 system and to a PDP-11 machine. Further research is planned into allowing ISIS to be distributed over several processors.

1.2 Organization of the Report

This report presents an overview of the capabilities of ISIS and the flavor of the language syntax. It contains actual examples of ISIS operation, but is not meant to be complete or to be used as a guide for system operation. The _ISIS User's Manual_ contains a detailed description of how to access ISIS and of the complete syntax for all system commands. The _ISIS User's Manual_, instead of this document, should be used to answer detailed questions regarding ISIS operation.

Chapter 2 of this report presents an overview of the capabilities of ISIS. It includes a discussion of the system's structure, the basic functions of each system component, and examples of command language syntax. Chapter 3 is an example of how ISIS can be used; it illustrates how ISIS was used to develop itself and includes a sample file management organization and a sample interactive session. In Chapter 4, the underlying stability and transportability of ISIS are investigated. Finally, Chapter 5 presents the current status and future plans for ISIS.

2. A Functional Overview of ISIS

ISIS is an interactive software development environment. It is powerful enough to handle almost all data management requirements of the software developer, yet it can easily be used by clerical personnel. This flexibility is achieved by having a uniform, integrated interface between ISIS and the user.

In order to present a uniform interface to the user, the ISIS command language is an interactive programming language. This language is patterned after PASCAL and gives the user such familiar constructs as declarations, assignments, conditionals, loops and input/output statements. Modifications to the PASCAL syntax are minimal and all are motivated by the demands of an interactive environment.

In addition to traditional programming constructs, ISIS includes statements for controlling the three special functional components of ISIS. These components are the text/data editor, the file manager and the tool invoker. Because these components are totally integrated within ISIS, the user can freely intermix statements controlling these components. This is a major advantage over the use of a set of independent tools to perform these same tasks.

2.1 Flow of Control and Data within ISIS

There are many interrelationships among the interactive programming language, the text/data editor, the file manager and the tool invoker. Figure 2.1a shows the flow of control in ISIS. This is a simple tree structure in which the user always communicates control information to ISIS via the interactive programming language.

Figure 2.1b shows the flow of data in ISIS. This diagram is complicated by the data transfers which can be initiated.

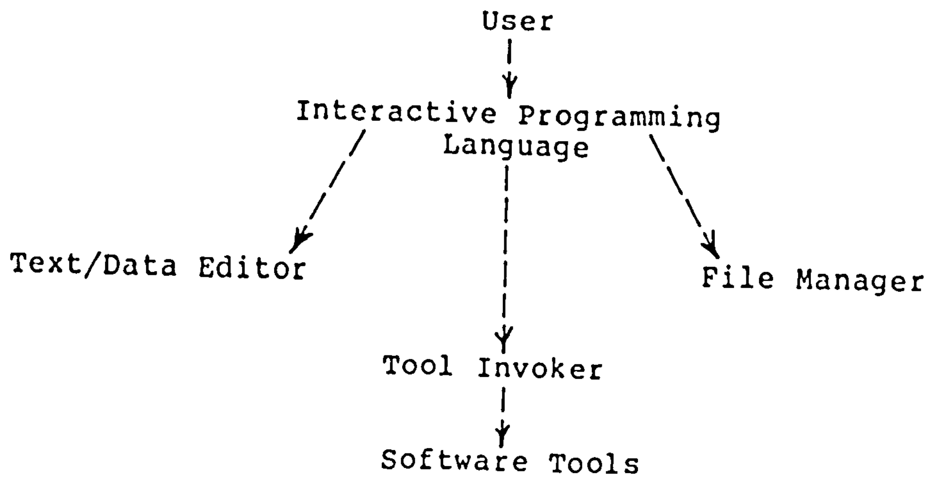

Figure 2.1a. Flow of Control in ISIS

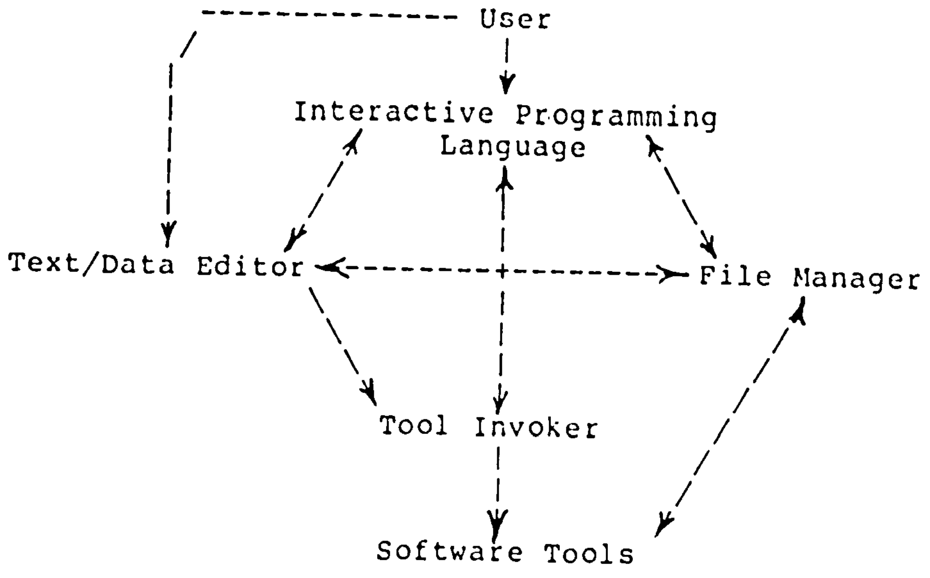

Figure 2.1b. Flow of Data in ISIS

The user supplies data for the INSERT, REPLACE and MODIFY statements of the text/data editor (see Section 2.3). Data is transferred between the text/data editor and the file manager in response to the USE and SAVE statements (see Section 2.4). Data from the text editor can also be moved to the tool invoker via the RUN statement (see Section 2.5). Finally, software tools can access and update an ISIS library by use of the ISISGET and ISISPUT utilities (see Section 2.6).

2.2 The ISIS Interactive Programming Language

The user communicates with ISIS via "commands", where a command is defined as a sequence of one or more "statements". Figure 2.2 is a list of the programming statements currently implemented in ISIS. These statements are in two distinct categories. The declarative statements (ABBREV, TYPE, VAR and ERASE) are executed as soon as they are parsed. For the other statements, intermediate-language code is generated as the statement is parsed. When the entire command has been parsed and no error has been encountered, the generated code is executed.

As indicated in Figure 2.2, there are some interesting differences between PASCAL and ISIS. These differences are discussed in the following paragraphs.

The ISIS interactive programming language supports the PASCAL concept of types. In ISIS, the built-in types include INT (PASCAL's INTEGER), REAL, BOOL (PASCAL's BOOLEAN) and STRING. STRINGs are arbitrarily-long character sequences since fixed-length strings are difficult to use in an interactive environment. As in PASCAL, these simple types may be combined to form complex data structures by use of the ARRAY and RECORD type constructors.

Because of the diversity of allowed data types and the desire to allow the use of mnemonic identifiers, all variables

Declarative Statements

```
ABBREV AB,ABR:ABBREV                 (* Statement verb abbreviations
TYPE   VECTOR:ARRAY [1..9] OF INT (* Type for future use, as in PASCAL
VAR    M1,M2 :VECTOR                 (* Variables of specified type
ERASE  ABR,VECTOR,M1                 (* Eliminate ABBREV, TYPE or VAR
```

Code-Generating Statements

Assignment Statement:

```
M1[TRUNC(LN(EXP(3)))] := M1[TRUNC(LN(EXP(3)))] + 1
```

Control Statements:

```
LOOP   EXITIF I>0;  I := I+1;  J := J-1;  EXITIF J<0;  END
REPEAT EXITIF I>0;  I := I+1;  J := J-1;  UNTIL J<0
WHILE  I<=0 DO      I := I+1;  J := J-1;  EXITIF J<0;  END
FOR    I := I TO 0 DO          J := J-1;  EXITIF J<0;  END

IF (I<0) AND (J>0) THEN
   I := I+1;  J := J-1;
ELSE
   I := I-1;  J := J+1;
END
```

Compile-and-Execute Statements:

```
XEQ 'I := 17;  PRINTLN I'   (* Do statements in STRING
EXEC                         (* Do statements in ACTIVE text frame
```

Terminal Input/Output Statements:

```
ASK     S,'VALUE TO BE ASSIGNED TO S = ' (* Prompt for terminal input
PRINT   'INT I=', I, '; REAL R=', R:12:4 (* Terminal output
PRINTLN '; BOOL B=', B, '; STRING S=', S (* Carriage Return at end
```

Figure 2.2. ISIS Programming Statements

must be declared. However, the user does not have to explicitly declare all of his variables: if an undeclared identifier is encountered during parsing, the user is prompted for the identifier's declaration. Another modification of the PASCAL use of identifiers is that identifiers must be explicitly ERASEd. This is because of the lack of block structure during an interactive session.

In PASCAL, each control structure allows only one statement. This restriction is circumvented by use of the BEGIN/END compound statement. In ISIS, each control structure allows multiple statements, has its own terminator, and may have "EXITIF condition" clauses among its controlled statements.

While user-defined procedures and functions similar to those of PASCAL are part of the ISIS design, they have not yet been implemented. There are, however, two statements which are similar to "parameterless procedures". The XEQ statement causes ISIS to treat the contents of a STRING variable as a command to be executed. The EXEC statement causes the contents of a text frame (see Section 2.3) to be treated as a command.

In contrast to PASCAL, there are two sets of input/output statements in ISIS. The ASK/PRINT statements are used to communicate with the user at the terminal, while the READ/WRITE statements are used to communicate with an editor frame (see Section 2.3).

Since PASCAL is a programming language which is compiled as a batch process, some information kept by the compiler is either inappropriate to display or is displayed in tabular form. In an interactive environment, the user often needs to gain access to this information. The SHOW statement is used in ISIS to display this information. Figure 2.3 lists several of the options of the SHOW statement.

System Information

```
    SHOW RESERVED       (* Displays the ISIS reserved words
    SHOW STATEMENTS     (* Displays the ISIS statement verbs

    SHOW SETS           (* Displays the options of the SET verb
    SHOW CLEARS         (* Displays the options of the CLEAR verb
    SHOW SHOWS          (* Displays the options of the SHOW verb
    SHOW OPTIONS        (* Displays the options for editor output

    SHOW ID id(s)       (* Displays the current use of the specified
                        (* abbreviation(s), type(s), variable(s) or frame(s)
```

Programming Information

```
    SHOW ABBREVS        (* Displays the current user-declared abbreviations
    SHOW TYPES          (* Displays the current user-declared types
    SHOW VARS           (* Displays the current user-declared variables
```

Figure 2.3. The SHOW Statement

2.3 The ISIS Text/Data Editor

The ISIS text/data editor is a line editor similar to the WYLBUR system developed at Stanford University. This editor was chosen as the best editor for the slow (300-1200 baud) terminals used at Langley Research Center. The ISIS editor has several improvements over the WYLBUR editor. The most important of these is the ability to edit multiple "frames" simultaneously.

A "frame" is a named unit of storage that consists of "items" that are always maintained in ascending order by their associated item-numbers (in the range 0.000 to 999.999). An item is either a STRING or a RECORD, and all of the items of a frame must be of the same type. If they are all STRINGs, the frame is a "text frame" and the data editor is called a text editor. If the items of a frame are RECORDs, the data editor resembles a simple version of a Relational Database system (this option has not yet been fully implemented).

The text editor allows the user to edit several frames simultaneously. Two frames, SHOWN and WORK, are predeclared by the system; the user may use the FRAME statement to create additional frames. Since a user will typically edit one frame at a time, ISIS allows the user to identify this frame as being ACTIVE. Unless a frame is referenced in a statement using a prefix of the form "frame-id/", the ACTIVE frame is used for all editing statements.

Figure 2.4 illustrates the operations which may be performed on a frame. Text to be stored in frames may be entered from the terminal (INSERT) or by use of the WRITE statement. When replacing, modifying, deleting or retrieving, the affected items may be specified by constraints on both the items' numbers and their contents (including column restrictions). A special retrieval statement is the FOREACH statement, an iterative statement similar to the WHILE statement. On each iteration, the next item satisfying the

Declarative Statements

```
  FRAME  DATA,AUX,BLD:STRING  (* Create editing frames of specified type
  ERASE  DATA,AUX             (* Eliminate frames
  ACTIVE BLD                  (* Specify the ACTIVE editing frame
```

Code-Generating Statements

Item Entry Statements:

```
  WRITE  'A NEW LINE',.L+1  (* Write a new item at end of frame
  INSERT 1.5, 2.1/2.4/.1    (* Prompt user for 1.5, 2.1, 2.2, 2.3, 2.4
```

Item Editing Statements:

```
  REPLACE 4                   (* Get replacement for item 4
  MODIFY  .F(4)               (* Get alterations to first 4 items
  ADD     '*' AT 1 IN ALL     (* Place a * in column 1 of every item
  CHANGE  '*' TO '+' IN .L :M (* All * in the last item become +
```

Item Removal Statements:

```
  DELETE NOT '*'  (* Remove all items which do not contain *
  VOID            (* An efficient form of 'DELETE ALL'
```

Item Retrieval Statements:

```
  COUNT '*' IN 10/100        (* Tally items in items 10 thru 100 with *
  LIST  '*' AND NOT '**'(1)  (* Print items with * but not ** in column 1
  READ  S,.F                 (* Read first item and place in variable S

  FOREACH S IN .F(20) DO  (* Iterate for the first 20 items
     PRINTLN .K,LEN(S)    (* Print item-number and item's length
  END
```

Item Manipulation Statement:

```
  MOVE  1/5 TO .L+1//1  (* Reposition items 1 thru 5 to end of frame
  COPY  2/4 TO 5.1//.1  (* Duplicate items 2 thru 4 at 5.1 by .1
  REKEY ALL TO 10//10   (* Resequenced item-numbers are 10, 20, 30, ...
```

Figure 2.4. Frame Editing Statements

retrieval criteria is assigned to an ISIS variable and can be processed using the ISIS programming language. Items can also be COPYed and MOVEd, even from one frame to another.

Frames are only used for temporary storage. In order to retain the editted data, it is necessary to SAVE a frame as a page in an ISIS library using the ISIS file manager.

2.4 The ISIS File Manager

The file manager controls access to a hierarchical file structure. In contrast to other file systems, the ISIS hierarchy is of fixed depth and each level has a name. The highest level of the hierarchy is the "library". Within the library are the "shelf", "book", "chapter" and "page" levels. A frame is stored as a page in this structure. Pages are identified by names of the form:

library.shelf.book.chapter.page

Figure 2.5 lists the statements which relate to an ISIS library. The SHOW PAGES statement displays the contents of an ISIS library. The output generated by this statement is controlled by the optional specification of shelf, book, chapter or page identifiers. The effect is to perform a pattern-match over the library's directory.

The USE statement is used to retrieve data from an ISIS library and place it in a frame for editing. In transferring the contents of the page to the frame, the pagename is associated with the frame. This name may be changed by using the SET NAME and CLEAR NAME statements, and its current value may be determined by SHOW NAME. This name is important because the SAVE and SAVE* statements use this name to determine the page in which a frame will be stored. All of these frame/page statements may be prefixed by a frame-id to access a frame other

Code-Generating Statements

Directory Display Statement:

```
  SHOW PAGES SYSLIB.ISIS.SOURCE. .   (* Display pages which match pattern
```

Frame/Page Interaction Statements:

```
  USE SYSLIB.ISIS.SOURCE.IPL.TAIL       (* Transfer from page to frame

  SET NAME SYSLIB.ISIS.SOURCE.IPL.HEAD  (* Change pagename of frame
  SHOW NAME                             (* Display pagename of frame
  CLEAR NAME                            (* Set pagename of frame to nil

  SAVE                                  (* Transfer from frame to page
  SAVE*                                 (* Overwrite page in library
```

Page Removal Statement:

```
  PURGE SYSLIB.ISIS.SOURCE.IPL.BAD   (* Eliminate page from library
```

Index Statements:

```
  SET INDEX HOLD ON SYSLIB.ISIS.SOURCE.IPL.TAIL  (* Make association

  SHOW INDEXES OVER SYSLIB                       (* Display indexes in use
  SHOW INDEX HOLD                                (* Display pages with index
  SHOW INDEXES ON SYSLIB.ISIS.SOURCE.IPL.TAIL    (* Display indexes on page

  CLEAR INDEX HOLD ON SYSLIB.ISIS.SOURCE.IPL.TAIL  (* Disassociate
  CLEAR INDEX HOLD                                 (* Eliminate an index
  CLEAR INDEXES ON SYSLIB.ISIS.SOURCE.IPL.TAIL     (* Eliminate indexing
```

Figure 2.5. File Managing Statements

than the currently ACTIVE frame.

The PURGE statement eliminates a page from a library. If this is the last page of a chapter, the chapter is also eliminated. This process is continued through the book and shelf levels.

In addition to a hierarchical storage organization, the ISIS file manager allows "indexing". An index term is a keyword that may be associated with one or more pages using the SET INDEX statement. There are no limits on the number of index terms over a library, on the number of pages which can be indexed by a single term, nor on the number of indexes which may be associated with a given page. The SHOW INDEX and SHOW INDEXES statements allow display of the index terms and their current associations. The CLEAR INDEX and CLEAR INDEXES statements can be used to disassociate index terms from pages.

2.5 The ISIS Tool Invoker

The tool invoker is used to send textual information to the host computer's internal reader. This is a primary mechanism for creating extensions to the standard ISIS system. Chapter 3 includes an example of the use of this mechanism.

There are three statements that form the basis of the tool invoker. The RUN statement causes textual information to be appended to a special sequential file. Before this file can be submitted to the host operating system, it must form an acceptable input to the host operating system. This typically means that it includes job control statements specifying the sequence of tasks to be performed.

The SEND statement causes the contents of the RUN file to be submitted to the host operating system's batch internal reader. This mechanism initiates a batch program which will be executed separately from the ISIS interactive session, possibly on a different computer. Because the user can continue his interaction with ISIS in parallel with the execution of his tool, this approach is very attractive if the tool does not involve user interaction. Another advantage is that existing tools can easily be made available to ISIS users. The only problem is that of accessing ISIS libraries from the tool. This is accomplished by use of the ISIS utilities ISISGET and ISISPUT (see Section 2.6).

If a tool requires user interaction, it could be implemented using the ISIS interactive programming language. This alternative will be more attractive when user-defined procedures and functions can be "compiled" into the ISIS intermediate language. However, existing interactive tools and large, complex interactive tools cannot use this alternative. Instead, they can use the STOP:SEND statement.

The STOP statement is used to terminate an ISIS session. If this statement has the option ":SEND", the termination of ISIS is accompanied by the submittal of the RUN file to the host operating system's <u>interactive</u> internal reader. This initiates an interactive program which temporarily replaces ISIS. When this tool reaches completion, it can return to ISIS via the host operating system. The advantage of this approach is that, as with batch tools, the tool is essentially independent of ISIS. The disadvantage is that the user will probably perceive a distinct change in the style of his dialogue with the system, contradicting the ISIS goal of uniformity. As with batch tools, interactive tools can gain access to ISIS libraries by use of the ISIS utilities ISISGET and ISISPUT (see Section 2.6).

2.6 The ISIS Utilities

Currently, there are three ISIS utilities. Of these, ISISGEN is used least frequently. It simply creates a new ISIS library. The other two utilities are ISISGET and ISISPUT.

ORIGINAL PAGE IS
OF POOR QUALITY

ISISGET is used to retrieve textual data stored in an ISIS library. The inputs to this program are the name of a local file into which the text is to be placed and the name of a page. ISISGET retrieves the specified page and writes it as a standard sequential file in the local file.

ISISPUT reverses the processing of ISISGET. It takes the textual contents of a local file and stores it in a named page. If the page already exists, it is overwritten.

3. Using the ISIS System

In order to show how ISIS might be used to solve some of the problems of software development, this chapter discusses how ISIS is being developed using ISIS. All of the data concerning ISIS is stored in the "SYSLIB" library. Since SYSLIB contains data on several software projects, the ISIS information is stored on the "ISIS" shelf of SYSLIB.

3.1 Storing the ISIS Source

The "SOURCE" book on the "ISIS" shelf contains all of the source code for ISIS. This book consists of 11 chapters and a total of 64 pages (see Figure 3.1). The chapters are based upon the functional organization of the ISIS software. Some of the pages contain part of a routine (e.g., IPL.HEAD), some contain a routine (e.g., PROGRAM.FATAL) and some contain several routines (e.g., ITEM.UTILITY). The pages containing only a part of a routine are used as part of a special technique that avoids having to recompile all of the ISIS source whenever only a few routines are modified. The rule used to create the remainder of this structure is that several short routines accessing a single data structure are stored together to simplify editing.

There are two important reasons why this structure is only two levels deep. First, this structure reflects the highly modular design of the ISIS software. Second, this structure makes it easier to remember where a particular routine is located since the "SYSLIB.ISIS.SOURCE." part of the page name is the same for all of these source pages.

3.2 Control of the ISIS Tool Invocations

The "CONTROL" book on the "ISIS" shelf contains several pages with job control statements. Each of these pages corresponds to a particular tool invocation used in creating

```
PROGRAM
    HEAD      (* CONST, TYPE, VAR AND FORWARD DEC[illegible]
    INIT      (* INITIALIZATION (EXCEPT PROGRAMMING LANGUAGE)
    FATAL     (* IF FATAL ERROR DETECTED, POST MORTEM DUMP
    TAIL      (* ISIS MAIN PROGRAM
SYSTEM
    VARS      (* READONLY VARIABLES (E.G., TIME, DATE)
    DATASET   (* DATASET INFO, ACCESS, STORE AND CLEAR
    SEND      (* RUN AND SEND CONTROL
    HASH      (* HASHCODE CALCULATIONS FOR EDITOR
    TERM      (* TERMINAL INTERACTION: SYSIN AND SYSLN
    TIME      (* ACCESS AND PACK TIME/DATE
    BLOCK     (* DISK BLOCK READ, WRITE AND ADD
    YESNO     (* ASK QUESTION AND ACCEPT YES/NO RESPONSE
    PASINTF   (* PASCAL INTERFACE TO OPERATING SYSTEM
BASE
    BUFFER    (* BUFFER CONTROL
    DUMP      (* PRINT BUFFER CONTENTS
    DATASET   (* DATASET CONTROL
    BLOCK     (* DATASET BLOCK READ, WRITE AND ADD
    AVSPUPD   (* UPDATE AVAILABLE SPACE IN A BUFFER
STRING
    TEMP      (* TEMPORARY-STRING CONTROL
    UTILITY   (* ALLOCATION, SETUP, COPY, ETC.
    SINGLE    (* PRINT, DUMP, LENGTH, ORD, ETC.
    DOUBLE    (* COMPARE, CONCATENATE, SUBSTRING, ETC.
    ITEM      (* CONVERT FROM EDITOR FORMAT TO STRING
ENTRY
    FREE      (* AVAILABLE SPACE CONTROL
    BUFFER    (* ACCESS, ADD, REPLACE, DELETE FROM BUFFER
    ACCESS    (* ACCESS AN ENTRY
    UPDATE    (* ADD, REPLACE AND REMOVE OF AN ENTRY
NODE
    NDBUF     (* NODE BUFFER PACK/UNPACK
    ACCESS    (* ACCESS A NODE
    UPDATE    (* ADD, REPLACE AND REMOVE OF A NODE
    REKEY     (* REKEY OF THE NODES
ITEM
    UTILITY   (* STRIP BLANKS, ETC.
    INPUT     (* READ AN ITEM FOR TERMINAL OR FILE
    ADD       (* ADD NEW ITEMS TO DATASET
    ACCESS    (* MASTER CONTROL FOR EDITOR STATEMENTS
    REMOVE    (* REMOVE ITEMS AFTER A MOVE STATEMENT
EDITOR
    TEXT      (* READ AND WRITE FOR TEXT FRAMES
    TAG       (* TAG PROCESSING
    VERSION   (* VERSION PROCESSING
LIBRARY
    BVD       (* BLOCK-VECTOR DIRECTORY PROCESSING
    PAGE      (* PAGENAME SEARCH AND OUTPUT
    LVL       (* PROCESSING OF PAGE DIRECTORY LEVELS
    DIRLOC    (* PAGE SEARCH USING PAGENAME
    DIR       (* PAGE DIRECTORY PROCESSING
    INDEX     (* INDEX PROCESSING
IPLBASE
    PUSHPOP   (* PUSH AND POP OF VIRTUAL MACHINE STACK
    ALLOC     (* ALLOCATE VARIOUS VIRTUAL RESOURCES
    STORE     (* STORE/RESTORE PROCESSING
    PRINT     (* PRINT VALUES (SIMPLE AND COMPLEX)
    ID        (* IDENTIFIER INFORMATION OUTPUT
    SET       (* SET STATEMENT PROCESSING
    CLEAR     (* CLEAR STATEMENT PROCESSING
    SHOW      (* PRINT INFORMATION OR PLACE IN SHOWN FRAME
IPL
    HEAD      (* DECLARATIONS FOR ISIS VIRTUAL MACHINE
    SHOW      (* ACCESS VIRTUAL MACHINE INFORMATION
    EXECUTE   (* VIRTUAL MACHINE PROCESSOR
    ERROR     (* SYNTAX ERROR MESSAGES
    TOKENIZ   (* TOKENIZATION OF COMMAND INPUT
    ID        (* IDENTIFIER PROCESSING (BINARY TREE)
    ERASE     (* ERASURE OF AN IDENTIFIER
    GENOP     (* CODE GENERATION DURING PARSING
    PARSE     (* PARSE A COMMAND
    INIT      (* INITIALIZE THE PROGRAMMING LANGUAGE PARSER
    TAIL      (* PARSE/EXECUTE CYCLE
```

Figure 3.1. Contents of SYSLIB.ISIS.SOURCE Book

and maintaining the ISIS system. Figure 3.2 is a listing of one of these CONTROL pages. Coding of these pages requires expertise in writing job control statements; however, once they are written, they should not have to be redone unless there is a change in the host operating system. These pages are under the control of the BUILD pages described in the following paragraph.

The "BUILD" book on the "ISIS" shelf contains several EXEC pages. When they are EXECed, they obtain the appropriate CONTROL page, ask the user for additional information, and create a RUN file. The user can then use either the SEND or STOP:SEND statements to invoke the tool. Figure 3.3 is a listing of the BUILD code associated with the CONTROL information of Figure 3.2. These pages require expertise in writing ISIS commands; however, once they are written, they are an easy-to-use mechanism for making standard tool invocations.

3.3 A Typical Interactive Session

Figure 3.4 shows how the SOURCE, CONTROL and BUILD pages might be used in a typical session. First, the WORK frame is used to hold the source code while it is being editted. Once this editing is completed, the appropriate BUILD page is loaded into the WORK frame and EXECed. The BUILD code then creates the CONTROL frame to hold the CONTROL page and asks the user for information which is added to the CONTROL frame. Once this frame is completed, its contents are transferred to the RUN file. In this case, a batch execution of the tool is invoked.

```
COMMENT.
COMMENT.  ACCESS THE PASCAL COMPILER AND COMPILE THE SOURCE
COMMENT.
ATTACH,PASCAL=PASCALR,PASLIB=PASLIBR/UN=LIBRARY,NA.
LIBRARY,PASLIB.
DEFINE,OUT=SYSOUT/M=W,NA.
ATTACH,OUT=SYSOUT/M=W,NA.
PASCAL,IN,OUT,F=120000.
COMMENT.
COMMENT.  STORE THE RELOCATABLE OBJECT DECKS IN A SYSTEM LIBRARY
COMMENT.
ATTACH,ISISOBJ/M=W,NA.
LIBEDIT,I=0,P=ISISOBJ,N=ISISLGO,L=1,B=LGO,C.
COMMENT.
COMMENT.  LOAD THE RELOCATABLE OBJECT DECKS TO FORM IXIX
COMMENT.  IXIX IS THE EXPERIMENTAL VERSION OF ISIS
COMMENT.
MAP,OFF.
GET,SEGIN,ISISIO/NA.
DEFINE,IXIX/M=W,NA.
ATTACH,IXIX/M=W,NA.
SEGLOAD(I=SEGIN,B=IXIX)
LOAD(ISISOBJ,ISISIO)
NOGO.
```

Figure 3.2. Listing of SYSLIB.ISIS.CONTROL.PASCAL.LOAD

```
ERASE S,K,CONTROL;  VAR S:STRING;  VAR K:KEY;  FRAME CONTROL:STRING
REPEAT
   ASK S,'FUNCTION (<?> FOR HELP) (<CR> TO ABORT): '
   IF S='?' THEN  (* WANTS HELP
      SHOW PAGES SYSLIB.ISIS.CONTROL.PASCAL.
   ELSE
      IF S<>'' THEN  (* GET CONTROL CARDS
         XEQ CAT('CONTROL/USE SYSLIB.ISIS.CONTROL.PASCAL.',S)
         (* ADD ACCOUNTING INFORMATION
         CONTROL/WRITE 'ISIS,T77,CM120000.  BERMAN/DAMACS',.01
         ASK S,'PASSWORD: ';  (* PASSWORD IS ONLY VOLATILE INFO
         CONTROL/WRITE CAT('USER,4600001,',CAT(S,'.')),.02
         CONTROL/WRITE 'CHARGE,M4600I,ISIS.',.03
         (* CREATE CONTROL CARDS TO FORM SOURCE FILE
         CONTROL/WRITE 'COMMENT.',.04
         CONTROL/WRITE 'COMMENT.  CREATE THE SOURCE FILE',.05
         CONTROL/WRITE 'COMMENT.',.06
         CONTROL/WRITE 'ATTACH,ISISGET/NA.',.07
         K := .100  (* INCREMENT BY .001
         LOOP
            ASK S,'SYSLIB.ISIS.SOURCE.: ';  EXITIF (S='') OR (S='?')
            S := CAT('SYSLIB.ISIS.SOURCE.',S)   (* CHECK IF EXISTS
            XEQ CAT('SHOW PAGES ',CAT(S,':KEEP'))
            SHOWN/IF .F<=.L THEN  (* PAGE DOES EXIST
               CONTROL/WRITE CAT('ISISGET,T. ',S),K
               CONTROL/WRITE 'REWIND,T.',   K+.0001
               CONTROL/WRITE 'COPYCF,T,IN.',K+.0002
               CONTROL/WRITE 'RETURN,T.',   K+.0003
               K := K+.001
            ELSE  (* REPORT FAILURE TO USER
               PRINTLN S,' DOES NOT EXIST.'
            END
         END
         IF S='' THEN  (* NORMAL TERMINATION
            CONTROL/WRITE 'PACK,IN,IN.',    K
            CONTROL/WRITE 'RETURN,ISISGET.',K+.001
            CLEAR RUN;  CONTROL/RUN:NK   (* COPY TO RFILE FOR SEND
            PRINTLN 'READY TO SEND .....'
         ELSE  (* USER WANTS TO TRY AGAIN
            PRINTLN 'XXX  RESTARTED  XXX'
         END
      ELSE  (* USER WANTS TO ABORT
         PRINTLN 'XXX  ABORTED  XXX'
      END
   END
UNTIL S<>'?'
```

Figure 3.3. Listing of SYSLIB.ISIS.BUILD.PASCAL.ENTER

```
USER NAME: 4600001,invoke
TERMINAL:       4, NAMIAF
RECOVER/ CHARGE: charge,m4600i,isis

TO GET NEWS AND THE TERMINAL DEFINITION CHARS, TYPE *INFORM,NEWS*.
TO MAKE SUGGESTIONS, *INFORM,SUGGEST*.
INFORMATION ABOUT THE SYSTEM MALFUNCTION ON 7/23/79 IS ON THE *NEWS*.

CHARGE.
/attach,isis.   notice that user input is in lower case
/isis.
 ISIS MONITOR   V 1.00     79/08/05. 13.10.40.
 13.10.40?use syslib.isis.source.item.access  (* fix a bug
SYSLIB.ISIS.SOURCE.ITEM.ACCESS USED AS WORK
 13.11.16?list 'ITMREPLC'  (* locate the area of interest
  70.    =  ITMCOUNT,ITMLISTF,ITMREPLC,
 458.    =   IF OP=ITMREPLC THEN
 533.    =  ITMREPLC,ITMCHANG,ITMADDST,
 13.11.48?list 533(4)
 533.    =  ITMREPLC,ITMCHANG,ITMADDST,
 534.    =  ITMMODIF: BEGIN OPCLASS := (OP=ITMCHANG) OR (OP=ITMADDST);
 535.    =              IF ECHO OR NOT OPCLASS THEN ITMOUT(1,KEYA);
 536.    =              IF ITMRPL THEN  (* POSSIBLE CHECK VETO FIRST *)
 13.12.06?delete 535  (* eliminate bad line
 535.    =              IF ECHO OR NOT OPCLASS THEN ITMOUT(1,KEYA);
 13.12.49?insert 535//.1  (* insert correction
 535.    =              if opclass then  (* check echo flag *)
 535.1   =              begin if echo then itmout(1,keya);
 535.2   =              end else itmout(options,keya);
 535.3   =  <<BREAK>>


INSERT TERMINATED.
 13.14.15?list 533/536  (* verify correct data entry
 533.    =  ITMREPLC,ITMCHANG,ITMADDST,
 534.    =  ITMMODIF: BEGIN OPCLASS := (OP=ITMCHANG) OR (OP=ITMADDST);
 535.    =              IF OPCLASS THEN  (* CHECK ECHO FLAG *)
 535.1   =              BEGIN IF ECHO THEN ITMOUT(1,KEYA);
 535.2   =              END ELSE ITMOUT(OPTIONS,KEYA);
 536.    =              IF ITMRPL THEN  (* POSSIBLE CHECK VETO FIRST *)
 13.16.06?save*  (* make changes permanent
SYSLIB.ISIS.SOURCE.ITEM.ACCESS SAVED.
 13.16.52?
```

Figure 3.4. Sample ISIS Session (Part 1 of 2)

```
 13.21.59?use build.pascal.enter; exec  (* SYSLIB.ISIS assumed by context
SYSLIB.ISIS.BUILD.PASCAL.ENTER USED AS WORK
FUNCTION (<?> FOR HELP) (<CR> TO ABORT):  ?
SYSLIB .ISIS   .CONTROL.PASCAL .ONLY
       .       .       .       .UPDATE
       .       .       .       .LOAD
       .       .       .       .SPECIAL
       .       .       .       .SIMPLE
       .       .       .       .COM
FUNCTION (<?> FOR HELP) (<CR> TO ABORT):  load
SYSLIB.ISIS.CONTROL.PASCAL.LOAD USED AS CONTROL
PASSWORD: invoke
SYSLIB.ISIS.SOURCE.:  program.head
1 ITEMS INSERTED. LAST ITEM INSERTED IN SHOWN     :   1.
SYSLIB.ISIS.SOURCE.:  item.acess
SYSLIB.ISIS.SOURCE.ITEM.ACESS DOES NOT EXIST.
SYSLIB.ISIS.SOURCE.:  item.access
1 ITEMS INSERTED. LAST ITEM INSERTED IN SHOWN     :   1.
SYSLIB.ISIS.SOURCE.:  program.tail
1 ITEMS INSERTED. LAST ITEM INSERTED IN SHOWN     :   1.
SYSLIB.ISIS.SOURCE.:
RUN CLEARED.
45 ITEMS IN SPECIFIED RANGE.
READY TO SEND .....
 13.24.38?send  (* submit to batch internal reader
'AAKAQZV' SENT TO BATCH EXECUTION.
 13.25.11?stop
ISIS TERMINATED.  (ADDRESS:    5)
- LOAD FL 042512     STACK FL 025266
/attach,ixix.  access experimental ISIS just created
/ixix.
 ISIS MONITOR    V 1.00     79/08/05. 13.27.26.
FRAME SHOWN        [SHOWN      ] RECOVERED: ....
FRAME WORK         [WORK       ] RECOVERED: SYSLIB.ISIS.BUILD.PASCAL.ENTER
FRAME CONTROL      [WORK1      ] RECOVERED: SYSLIB.ISIS.CONTROL.PASCAL.LOAD
 13.27.26?replace 533:nl  (* check operation of correction
NO ITEMS IN SPECIFIED RANGE.
 13.28.14?replace .f:nl  (* try again
   1.    =success.
1 ITEMS IN SPECIFIED RANGE.
 13.29.01?stop
ISIS TERMINATED.  (ADDRESS:    5)
- LOAD FL 042516     STACK FL 025262
/bye
```

Figure 3.4. Sample ISIS Session (Part 2 of 2)

4. Stability and Transportability of ISIS

A central goal of ISIS is that of stability. ISIS achieves this goal by having minimal dependence upon the host operating system. It is interesting that most of the tactics used to attain stability are applicable to a type of transportability. That is, ISIS can easily be transported to a new computer; however, certain details of its operation will not be the same from machine to machine.

4.1 PASCAL Coding

ISIS is coded almost entirely in PASCAL. Since no standard has yet been adopted, only a subset of the PASCAL language is used. There are no labels, GOTOs, SETs or NEWs. Packing is used sparingly and with consideration of how the structures might be implemented on a variety of machines.

The ISIS code has a very shallow structure. That is, there are few routines which are nested more than two levels deep. This is partially because most routines are used in more than one context. It is also because of the mechanism that was used during the development of ISIS to avoid having to completely recompile ISIS every time a change is made. The only exception to this shallow structure is in the recursive descent parsing.

4.2 Operating System Independence

Achieving total operating system independence is not possible; however, considerable care has been exercised in designing the interface between ISIS and its host operating system. This has resulted in an interface that is relatively insensitive to operating system changes and is implementable on a broad range of computers. This interface is used to access and control the host operating system's capabilities for file manipulation, disk input/output, multi-user scheduling, and remote terminal communications.

ISIS assumes that the host operating system provides a basic file system that allows for creating, accessing, extending and destroying files. Each ISIS library corresponds to a file in the host file system. In addition, the host file system must allow temporary files for each ISIS user. Since some file systems do not allow dynamic creation of these temporary files, ISIS has a limit on the number of frames a user can have at any point during a session.

ISIS uses only two disk data structures. The first is a sequential file. This structure is used for input, temporary storage and output. Frames and libraries are stored as "direct access files". That is, ISIS issues requests for randomly reading or writing fixed-length records by specifying their relative position within the file. These organizations are available on almost all operating systems.

ISIS is coded as a single-user system. The only interaction between users occurs as contention for a library. To prevent deadlock, each time a statement uses a library, ISIS gains exclusive control of the library, performs the statement and releases control of the library. Since no statement references more than one library, deadlock is impossible.

ISIS communicates to the terminal via the SYSIN and SYSLN routines. SYSIN is straight-forward in that a prompt is written to the terminal and input is accepted from the user. SYSLN is called at the termination of each line to be written to the terminal. The line to be transmitted is placed in the standard OUTPUT file. On most systems, SYSLN is merely a WRITELN of the OUTPUT file.

4.3 Operating System and Machine Dependencies

There are several dependencies upon the host operating system and the host machine for which no attempt has been made to gain independence. These dependencies include the idiosyncracies of the teleprocessing monitor, the character set, and the accuracy of integers and reals.

ISIS assumes that there exists a teleprocessing monitor for communicating with the terminal. The characteristics of available monitors varies significantly. Most monitors allow for a "backspace" key, an "input abort" key and an "output abort" key. However, the association of which key performs which function is not standardized.

ISIS uses the character set of the host PASCAL compiler. Similarly, the accuracy of integer and real values is precisely that provided by the PASCAL compiler.

5. Current Status and Conclusions

ISIS was specified during 1976-77, designed during 1977-78 and an engineering prototype was implemented during 1978-79. During the year that it has been available for testing, ISIS has proven itself to be a very well-behaved and powerful system. Interestingly, users with little previous interactive computing experience have found it easier to learn ISIS than experienced users. This is apparently due to the fact that experienced users have worked in environments where they are expected to remember a large number of details about how to use a system. They have some trouble adjusting to the ability of ISIS to perform most of these details, leaving them free to concentrate upon the more interesting problems of software development.

The current implementation of ISIS is an engineering prototype and, as such, it is still evolving. The interactive programming language and the text editor are now relatively stable, the file manager and system invoker are nearing completion and the data editor is being implemented. It is hoped that full production status will be reached by July 1980.

Efforts have already begun on rehosting ISIS to IBM and DEC equipment. In addition, a research effort has been initiated to allow ISIS to be distributed over several cooperating processors.