AFHRL-TR-90-33(I)

AIR FORCE

HUMAN RESOURCES

INSTRUCTIONAL SUPPORT SYSTEM (ISS): UPGRADING THE VAX PROTOTYPE AND DEVELOPMENT OF THE ZENITH-248 MICRO-BASED ISS

Barbara J. Eaton

TRAINING SYSTEMS DIVISION
Brooks Air Force Base, Texas 78235-5601

Douglas Aircraft Company
2450 South Peoria, Suite 400
Aurora, Colorado 80014

Mei Associates, Incorporated
1050 Waltham Street
Lexington, Massachusetts 02173

September 1990

Final Technical Report for Period October 1985 - March 1990

LABORATORY

AIR FORCE SYSTEMS COMMAND
BROOKS AIR FORCE BASE, TEXAS 78235-5601

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the Government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

The Public Affairs Office has reviewed this report, and it is releasable to the National Technical Information Service, where it will be available to the general public, including foreign nationals.

This report has been reviewed and is approved for publication.

HENDRICK W. RUCK, Technical Advisor
Training Systems Division

HAROLD G. JENSEN, Colonel, USAF
Commander

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1990	Final - October 1985 - March 1990

4. TITLE AND SUBTITLE

Instructional Support System (ISS): Upgrading the VAX Prototype and Development of the Zenith-248 Micro-Based ISS

5. FUNDING NUMBERS

C - F33615-85-C-0011
F33615-88-C-0003
PE - 62205F
PR - 1121
TA - 10
WU - 34, 43

6. AUTHOR(S)

Barbara J. Eaton

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Douglas Aircraft Company (DAC)
2450 South Peoria, Suite 400
Aurora, Colorado 80014

Mei Associates, Inc.
1050 Waltham Street
Lexington, Massachusetts 02173

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Training Systems Division
Air Force Human Resources Laboratory
Brooks Air Force Base, Texas 78235-5601

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

AFHRL-TR-90-33(I)

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This report describes several efforts that led to the development of the Instructional Support System (ISS) from prototype research and development (R&D) software toward a production computer-based training (CBT) system that supports both computer-assisted instruction (CAI) and computer-managed instruction (CMI). These efforts include the operational test and evaluation of the software, rehosting the software to a validated Ada compiler, development of MicroCMI and interactive videodisc capabilities, development of critical user and system documentation, and design and development of a micro-based version of the software to run on IBM-compatible hardware. ISS is a Government-owned product, written in Ada in a modular format, enabling it to run on machines ranging from micros to mainframes, and is available through the National Technical Information Service (NTIS).

This is the first of three volumes on the Instructional Support System (ISS). Volume I contains a detailed description of the development efforts.

14. SUBJECT TERMS		15. NUMBER OF PAGES
Ada computer-assisted instruction computer-managed instruction	Instructional Support System transportable instruction system	120 16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

SUMMARY

From September 1985 through February 1987, personnel from several agencies performed operational test and evaluation on the Instructional Support System (ISS). These agencies include: the Air Force Human Resources Laboratory (AFHRL), Douglas Aircraft Company, Hq Strategic Air Command's 338 Combat Crew Training Squadron (CCTS) and Communication and Training Innovations. The contract used to fund the operational test and evaluation was Contract F33615-85-C-0011. As a result of the operational test and evaluation, Douglas Aircraft Co. incorporated several enhancements into Baseline Version 1.0. In addition, they made several advancements to the ISS software during this period. The first involved conversion of the ISS to a validated Ada * compiler. The second resulted in the development of critical system documentation. The third involved development of a microcomputer-managed instruction (MicroCMI) subsystem. The fourth resulted in the addition of interactive videodisc capability. The fifth involved development of training materials to teach ISS users to use the full computer-managed instruction (CMI) subcomponent.

From March 1987 until April 1988, Mei Associates, Inc. rehosted the ISS computer-assisted instruction (CAI) subsystem from the Vax to the Zenith 248 (Z-248). The Electronic Systems Division (ESD) funded the effort under a Department of Transportation contract. Later, in 1989, Mei Associates, Inc. rehosted the MicroCMI subsystem and interactive videodisc capability from the Vax to the Z-248. ESD funded this effort under Contract F33615-88-C-0003. This rehosting made both versions of the software functionally equivalent.

In the summer of 1989, Dr. J. Michael Spector attempted to rehost the Z-248 version of the ISS, MicroISS, to the Meridian Ada Compiler. The contract used to fund this effort was Contract F41622-89-M-5240. Although Dr. Spector's efforts were unsuccessful, his discoveries are useful to future modifications and upgrades of the ISS.

*Ada is a trademark of the U.S. Government (Ada Joint Program Office).

PREFACE

As an Air Force-owned training system, the ISS contributed substantially toward the development of three major training programs during the period covered by this report. The Strategic Air Command's 338 Combat Crew Training Squadron (CCTS) at Dyess AFB, Texas, is responsible for B-1B aircrew training. The 338 CCTS trained and evaluated over 350 pilots and navigators with the ISS. At the same time, the 338 CCTS performed operational test and evaluation of the ISS. This provided a rich testbed for developing and enhancing the software. The General Imagery Intelligence Training System (GIITS), which was developed by Air Training Command's 3480 Technical Training Wing (TCHTW) at Goodfellow AFB, Texas, is the second training program that benefited from the ISS. They used the ISS as the core software for their training system. The GIITS will provide training to over 1200 students in over 100 different intelligence courses at any given time. The Advanced On-the-Job Training System (AOTS) is the third training program that profited by using ISS as the foundation for their system. The AOTS development was directed by Air Staff and has the potential to support on-the-job training throughout the Air Force.

A number of individuals contributed significantly to the upgrade of the prototype ISS on the Vax and the development of the MicroISS on the Z-248. Alphabetically, they are: Mr. Leonid Altshul (Mei Assoc, Inc.), Mr. Jeff Benjamin (Mei Assoc, Inc.), Mr. Ben Bernar (Douglas Aircraft Co.), Capt Clay Blankenship (AFHRL/IDC), Mr. Dave Blossart (Douglas Aircraft Co.), Mr. Patrick Brennan (Mei Assoc, Inc.), Mr. Bill Char (Douglas Aircraft Co.), Ms. Vida Chase (Douglas Aircraft Co.), Ms. Patricia Cwynar (Communication and Training Innovations), Mr. Richard Dieterich (Mei Assoc, Inc.), Ms. Donna DeMaria (Communication and Training Innovations), Lt Barbara Edwards (338 CCTS/IS), Mr. Ed Fornier (Mei Assoc, Inc.), Ms. Denise Geller (Douglas Aircraft Co.), Mr. William Hawks (AFHRL/IDC), Capt John Herman (ESD/AVS), Lt Walter Hodge (AFHRL/IDC), Ms. Debbie Horan (Communication and Training Innovations), Mr. Frank Lhota (Mei Assoc, Inc.), Maj Michael Matthews (338 CCTS/IS), Mr. Ron Medo (Douglas Aircraft Co.), Dr. Peng Mei (Mei Assoc, Inc.), Mr. Andreas Moreira (Douglas Aircraft Co.), Mr. Dave Pflasterer (Douglas Aircraft Co.), Mr. Thanh Pham (Douglas Aircraft Co.), Ms. Diane Poston (Douglas Aircraft Co.), Mr. Bruce Pyles (Douglas Aircraft Co.), Dr. J. Michael Spector (Spector and Assoc.) and Mr. Richard Vigue (AFHRL/IDC).

TABLE OF CONTENTS

INSTRUCTIONAL SUPPORT SYSTEM (ISS): UPGRADING THE VAX PROTOTYPE AND DEVELOPMENT OF THE ZENITH 248 MICRO-BASED ISS

I. INTRODUCTION

From the early 1970s until 1983, under contracts with the Air Force Human Resources Laboratory (AFHRL), McDonnell Douglas Astronautics Company developed the prototype Advanced Instructional System (AIS). The AIS was developed to serve as a research and development (R&D) testbed for technical training. It demonstrated that individualized computer-assisted instruction (CAI) and computer-managed instruction (CMI) are directly applicable to an Air Force training environment.

Although demonstrated as feasible, the AIS was hardware-dependent and not transportable to other computers. This factor severely limited its exploitation to the training community (McDonnell-Douglas Astronautics Company, 1986). To correct this problem, the Technical Training Division of the AFHRL at Lowry AFB, Colorado, awarded a contract to Douglas Aircraft Co. to create a transportable system and expand its usage to a variety of Department of Defense (DOD) training environments. Under Contract F33615-81-C-0021, Douglas Aircraft Co. developed and alpha-tested the Instructional Support System (ISS) as a proof-of-concept prototype which operated on low-cost minicomputers and microcomputers (McDonnell Douglas Astronautics Co., 1986).

The efforts covered in the present report encompassed several goals under three contracts. The first goal was to perform operational test and evaluation on the ISS prototype and develop it into a production system. A second goal was to convert the ISS software from the nonvalidated Irvine Science Ada Compiler to the validated Digital Equipment Corporation (DEC) Ada Compiler. A third goal was to create ISS system documentation to enable long-term system maintainability. A fourth goal was to develop a MicroCMI capability for small, microcomputer-based training applications. The fifth goal was to integrate an interactive videodisc capability into the software. A sixth goal was to develop courses to teach users to operate the ISS CMI subcomponent. A seventh goal was to rehost the software to the Air Force's standard microcomputer, the Z-248, under the MS-DOS operating system. The final goal was to convert the microcomputer version of the ISS, MicroISS, from the ALSYS Ada Compiler to the Meridian Ada Compiler. This conversion would eliminate the need for a math coprocessor and Tektronix terminal emulator.

The ISS can be ordered through the National Technical Information Service (NTIS) Sales Department at (703) 487-4650. A complete listing of software and documentation available through NTIS is provided in Appendix H.

II. PROJECT DESCRIPTION

As the operational test and evaluation site for the ISS, the 338 Combat Crew Training Squadron (CCTS) discovered over 300 software problems and recommended several software enhancements. By the end of the operational test and evaluation, Douglas Aircraft Co. had fixed over 275 of the reported problems. They had also incorporated most of the recommended enhancements into the ISS. Mei Associates, Inc. fixed the remaining problems under a follow-on contract. The 338 CCTS's involvement in the operational test and evaluation was significant to the development of the ISS toward a production system.

Ada is the primary implementation language for the ISS. Conversion to a validated Ada compiler during this project was important to the long-range success of the software. Two major Air Force projects use the ISS as their core software. They are the Advanced On-the-job

Training System (AOTS) and the General Imagery Intelligence Training System (GIITS). These two systems solidified the requirement to convert ISS to a validated compiler. Additionally, validated Ada source code was necessary to enable use of validated Ada compilers in the future.

Douglas Aircraft Co. improved maintainability of the ISS software by creating in-depth application software, database and data structure documentation. The documentation includes inline comments and detailed descriptions of the ISS databases and data structures. That documentation details the functions of the software and Data Management System at a high level. It is available through the National Technical Information Service (NTIS). See Appendix H for a complete listing of ISS software and documentation available through the NTIS.

An interactive videodisc capability integrates video information from a laser disc with text and graphics created with the ISS authoring editor, CAI Authoring Support Software (CASS). The ISS delivery program, CAI Presentation (CAIPres) Program, then presents the integrated video/text/graphic frames. Access to video frames is random and controlled by the ISS software.

Douglas Aircraft Co. developed online and offline training materials to teach users to use the ISS CMI subsystem. Training personnel use the CMI subsystem to develop, track and evaluate students. Consequently, the courseware provides training for course developers, instructors and evaluators.

Mei Associates, Inc. rehosted the ISS to the Z-248 environment to make it accessible to Government users with access to IBM-compatible microcomputers. Because the ISS is a Government-owned computer-based training (CBT) system, this conversion made it standard software available on Air Force standard architecture.

Finally, Dr. J. Michael Spector attempted to make several upgrades to the MicroISS. Although the upgrades were unsuccessful, Dr. Spector derived useful information that will aid future enhancement efforts.

III. MAJOR ACCOMPLISHMENTS

The ISS evolved from a prototype toward a production system during this effort under an operational test and evaluation with the 338 Combat Crew Training Squadron (CCTS). Douglas Aircraft Co. converted the system to a validated Ada compiler and created critical system documentation. This made the software more maintainable. Douglas Aircraft Co. also implemented a minimum CMI subsystem for small training environments. In addition, they added interactive videodisc capability to the software and developed online and offline instructional materials to train CMI subcomponent users. Mei Associates, Inc. rehosted the CAI and MicroCMI subcomponents, including the videodisc capability, to the Air Force's standard microcomputer. Dr. J. Michael Spector attempted to upgrade and rehost the MicroISS software to another Ada compiler.

ISS Operational Test and Evaluation

Early in this effort, Douglas Aircraft Co. determined the optimum system configuration for the 338 CCTS's VAX minicomputer environment. This configuration supported a load of 24 online students and developers. Douglas Aircraft Co. and AFHRL personnel gathered sample data on the 338 CCTS system for a 3-day period during fully loaded training sessions. Douglas Aircraft Co. then recommended several upgrades to the 338 CCTS system. The first recommendation was to upgrade from the VAX 11/780 to the VAX 11/785 processor. The second recommendation was to increase central memory from 6 megabytes to 16 megabytes. The third recommendation

was to add a disk controller. The fourth recommendation was to increase the performance of the ISS terminal communications software to the Tektronix 4107 terminal. The fifth recommendation was to move the ISS database and executables onto separate disk drives to increase disk throughput. The sixth recommendation was to increase the system swap file from 23000 to 60000 blocks to prevent excessive central processing unit and disk activity with system and user paging. The final recommendation involved setting the Tektronix 4107 terminals to Pass Through Mode to increase terminal throughput. The 338 CCTS implemented all of the recommended upgrades. Douglas Aircraft Co. continued to develop software performance improvements throughout the project to support the desired load more efficiently.

During the operational test and evaluation, Douglas Aircraft Co. corrected over 275 software problems. They also incorporated several enhancements into the software. The 338 CCTS created 170 hours of online courseware on B-1B aircrew training operations during this period. They also trained, managed and evaluated the training progress of more than 350 students. Successfully supporting an operational environment of this magnitude with prototype software is possibly the most significant accomplishment of this effort. Appendix A provides a listing of the ISS functional capabilities established as a result of the operational test and evaluation. Appendix B provides a listing of the executables that provide these capabilities. Appendix C is a functional description of the VAX Version ISS.

Conversion to a Validated Ada Compiler

In 1982, Douglas Aircraft Co. developed the ISS on the Irvine Science Ada Compiler. There were two significant drawbacks that hampered a long-range development effort. The first was the inability to obtain the degree of maintenance support needed. The second was the compiler's nonvalidation by the Ada Joint Program Office. Conversion to a validated Ada compiler was essential to the long-range success of the ISS and two major training systems. The AOTS and GIITS used the ISS as a baseline for their training systems. Both of these systems required validated ISS source code for their program development efforts. In addition, validated source code fulfilled the portability objective of the ISS and enabled the use of future microcomputer-based validated Ada compilers. The following process converted ISS to the validated DEC Ada compiler:

1. The software was "frozen" on the nonvalidated version of the source code.

2. The nonvalidated version of the source code was moved to a new directory. Necessary changes were made to create a validated version of the source code.

3. Constant limits for appropriate data structures were changed to bring the limits to the correct levels.

4. Variant record declarations were deleted to eliminate invalid Ada source code.

5. Package specifications and package bodies were divided to enable proper configuration management.

6. Compilations were performed and compilation errors removed from the new version of the source code.

7. Identified Indexed Sequential Access Method (ISAM) files were changed to contain compound keys. This was necessary due to the use of data packing for validated Ada source code.

8. The CAI databases were converted to coincide with the validated Ada source code.

9. Test procedures were run for the following software:

   a. User Editor (UserEd)
   b. CAI Authoring Support Software (CASS)
   c. CAI Presentation Program (CAIPres)
   d. Graphics Editor (GrEdt)
   e. Simulation Dialogue Editor (SID)
   f. Simulation Presentation Program (SIDPres)
   g. Course Structure Editor (CSE)
   h. Lesson Definition Editor (LDE)
   i. Test Editor (Test)
   j. Curriculum Definition Editor (CDE)
   k. Student Registration Program (StuReg)
   l. CMI Operations
   m. Test Item Evaluation (TIE)
   n. Course Evaluation Summary (CES)
   o. Data Extraction Program (DEP)

Douglas Aircraft Co. completed the conversion in November 1986. They also converted hundreds of hours of B-1B and AFHRL courseware.

Creation of System Documentation

Douglas Aircraft Co. enhanced the maintainability of the ISS by developing application software, data structure and database documentation. They developed within code or inline documentation of the ISS application software source code as follows:

1. A document specification was written for each procedure/function that describes:

   a. The overall purpose of the procedure/function,
   b. The purpose of each parameter used in the procedure/function,
   c. Initial conditions and settings upon entry into the procedure/function,
   d. Assumptions of the procedure/function,
   e. Host dependencies of the procedure/function, and
   f. Potential side effects of the procedure/function.

2. Comments, entered at the beginning of each software package, specify the overall purpose of the package. Meaningful comments, entered into all procedures/functions, clarify algorithms and or complex obscure sequences of code.

A detailed description of each data structure identifies each field within the structure and its purpose. The documents created are the Software Detailed Design Document (SDDD) and the Data Base Design Document (DBDD). Together these documents describe the high-level, detailed functioning of the software and the Data Management System. The SDDD describes application software functionality and software global and local data. The DBDD describes the Data Management System and data-base structure. These documents are available through the NTIS. See Appendix H for a complete listing of ISS software and documentation available through the NTIS.

Development of a MicroCMI Capability

A requirement to develop a computer-based training (CBT) system to support stand-alone, microcomputer-based users surfaced with this effort. This need came with the emergence of microcomputer technology in the Air Force. Douglas Aircraft Co. developed a configuration for these users that includes the CAI subsystem and a MicroCMI subsystem. The MicroCMI subsystem is a scaled-down version of the VAX CMI subsystem.

Douglas Aircraft Co. deleted the Adaptive Model, the ISS Management Editor and the Resource Availability Editor from the MicroCMI subsystem. The simplified MicroCMI course management subsystem provides registration and tracking of students on a limited basis. It contains two functions, development (MicroCMI Instructor) and operation (MicroCMI Student). The basic capabilities of MicroCMI Instructor are:

1. Generation of prerequisite course lists or the capability to select courses in any order.

2. Generation of lesson lists containing either linearly ordered lessons or lessons accessible in any order.

3. The capability to resequence and reorganize lesson order within courses.

4. The capability to input both online (embedded) questions and mastery test items. The ISS provides the latter capability through tests developed in the Test Editor and presented with the Test Presentation Program.

The basic capabilities of MicroCMI Student include:

1. Registration of students into multiple courses. Students registered in more than one course have the option to choose which course to begin first.

2. Generation of student rosters for individual courses. Course rosters contain the number of lessons the students will take, student names and identification numbers. They also list current lessons for each student.

3. Generation of student assignments.

4. Lesson override capability.

5. Certify pass capability.

6. Initiation and presentation of CAI lessons.

7. Initiation and presentation of mastery tests.

8. Recording and tracking of student performance on a limited basis. This includes lessons passed, student performance on individual lesson questions and lesson completion times.

Integration of Interactive Videodisc Capability

Integration of interactive videodisc capability involved interfacing the ISS authoring and presentation programs, CASS and CAIPres, to the Sony videodisc player. The implementation

required software to control the player. It also required video/graphics controller card hardware. This hardware displays the images that control and stabilize the displayed video, computer-generated graphics and computer-generated text combinations. The Z-248 microcomputer was chosen to control the display of the image combinations because of its use on the AOTS project. It was also chosen due to its selection as the Air Force standard microcomputer.

The implementation approach involved the use of the Z-248 as a dumb terminal, via a terminal emulation capability attached to the VAX 11/785 or VAX 8600. The Z-248 contains the installed controller card. An RS-232 interface from the Z-248 to the videodisc player controls the videodisc player. The CASS and CAIPres software, used with the Z-248, accomplishes the videodisc implementation.

This approach allows future development of the Z-248 videodisc capability as a stand-alone system. Douglas Aircraft Co. accomplished this by coding generic videodisc commands into the CASS and CAIPres programs. The videodisc driver receives commands, such as "load disc," "unload disc," and "erase screen." Changing the driver command table allows easy integration of additional players in the future.

After selecting a controller card, Douglas Aircraft Co. chose a monitor to display videodisc images combined with computer-generated graphics/text. The implementation required a red-green-blue/analog (RGB/analog) device, which prohibited the use of commonly used Tektronix 4105 and 4107 terminals for displaying videodisc images. Douglas Aircraft Co. conducted a study at the beginning of the videodisc task which determined the controller card, RGB/analog monitor and associated equipment for use with the Sony videodisc player. They selected the Matrox VGO-AT card because it is compatible with the Z-248 and offers higher resolution than other candidate cards. They chose the Electrohome ECM 1311 monitor because it is compatible with the VGO-AT card and offers the highest display quality when used with the Matrox VGO-AT card. Lastly, they chose Sony SRS 150 speakers because they offer good audio quality at low cost. Other monitors may be used as long as they are compatible with the Matrox VGO-AT card. The complete configuration for the interactive videodisc implementation includes:

1. Matrox VGO-AT controller card,
2. Electrohome ECM 1311 monitor,
3. Elographics E274 touch screen,
4. Elographics E271-141 touch screen controller card, and
5. Sony SRS 150 speakers or Telex 600-1 headphones for audio capability.

The interactive videodisc implementation includes the following capabilities:

1. Disc loading and unloading;
2. Single video frame display;
3. Video sequence display;
4. Screen image erase;
5. Audio sequence play;

6. Compressed audio play;

7. Frame numbers display;

8. Definition of the video window and transparent video color to support overlay;

9. Video played as fast, normal, or slow;

10. Pause, step through, jump ahead, restart or continue video sequences;

11. Support of sequences that contain still frames, full motion sequences, and still frames with compressed audio and digital information;

12. Random access of still and motion sequence frames;

13. Forward or reverse display; and

14. Playback of video images and computer-generated text and graphics.

As an aside, Douglas Aircraft Co. examined the feasibility of installing the selected controller card into the VAX computers. They examined the feasibility of a VAX system acting as the sole controlling computer for the videodisc implementation without using a Z-248. This configuration would have allowed the use of an existing AFHRL interactive videodisc capability, the IEV-60 Graphics Controller. To be usable, the IEV-60 device had to support an Enhanced Graphics Adaptor (EGA) mode. The EGA is the display mode used by the AOTS project and other Air Force organizations slated to use the ISS interactive videodisc capability. Douglas Aircraft Co. found the IEV-60 Graphics Controller incompatible with the EGA mode and did not pursue the approach further.

Development of CMI Courseware

Douglas Aircraft Co. developed online courseware and offline user manuals to train CMI users on how to use the CMI subsystem. The CMI subsystem contains development, management and evaluation editors. The courseware and manuals provide training to course developers, instructors and evaluators. Courses developed include CMI Overview (Course 1), CMI Design (Course 2), Evaluation Design (Course 3), CMI Development (Course 4), Training Management (Course 5), Training Evaluation (Course 6) and System Management (Course 7). A high-level description of each course follows:

1. CMI Overview (Course 1) - This course introduces students to CAI, CMI, CBT and the ISS system. It describes the ISS structures and hierarchies and how to use each to develop a CMI curriculum. It also provides simple graphic representation for CMI structures, provides an overview of training management and shows how it affects a CMI curriculum. This course also provides students an overview of the ISS CMI curriculum and relates it to CMI tasks so students can choose training paths that meet their needs. Lastly, it provides students a context for later learning by presenting an overview of the three CMI phases and subcomponents. It then introduces them to the ISS editors in each.

2. CMI Design (Course 2) - This course teaches design and problem-solving skills needed to design CMI and determine policies for use of the ISS system.

3. Evaluation Design (Course 3) - This course teaches design and problem-solving skills needed to develop an evaluation strategy for use with the ISS CMI subsystem.

4. CMI Development (Course 4) - This course gives students hands-on experience with the CMI development editors. Prerequisite courses are Courses 1 and 2. Also, students must know the purpose and characteristics of CMI and the three subcomponents of the ISS CMI subsystem. In addition, students must be familiar with the hierarchy levels in the ISS system and the management functions available in the ISS CMI. Students must also be familiar with the choices required during CMI structuring, management and test design.

5. Training Management (Course 5) - This course gives students hands-on experience with the ISS CMI implementation editors to enroll and manage students. Prerequisite courses are Courses 1 and 2. Prerequisite knowledge is the same as that required for Course 4.

6. Training Evaluation (Course 6) - Students learn how to use the evaluation editors to gather data specified in an evaluation design. Instruction includes a discussion of data elements in reports and practice with report interpretation and training revision recommendations. Prerequisite courses are Courses 1, 2 and 3. Prerequisite knowledge is the same as that required for Course 4 plus knowledge provided by Course 3. Course 3 covers the types of evaluation data available from the ISS evaluation editors.

7. System Management (Course 7) - This course gives students hands-on experience controlling the ISS system with the system management editors. The students should have a working knowledge of the computer operating system and an understanding of directories and environments before taking this course. The students must have completed Course 1 and know the purpose and characteristics of CMI. They must also know the three subcomponents of the ISS CMI subsystem. In addition, they must know the hierarchy levels in the ISS system and the management functions available in the ISS CMI subsystem.

The online courseware is provided in the form of a database to users requesting the VAX ISS CMI subsystem. Both the courseware and offline user manuals are available through the NTIS.

Online training courses for the CAI subsystem editors CASS and Graphics Editor (GrEdt) are provided with requests for either version of the software. These courses, along with accompanying offline materials, are also available through the NTIS. See Appendix H for a complete listing of ISS software and documentation available through the NTIS.

Volume II of this technical report provides documentation to assist users in determining if CBT, and the ISS in particular, is the correct medium for a specific training application.

Rehosting ISS to the Z-248

One of the fundamental objectives of the ISS effort was to implement the software on microcomputers. This made the system accessible to any organization with access to an IBM-compatible microcomputer. The Z-248 was the prime candidate for this implementation. Mei Associates, Inc. performed the task under a Department of Transportation Contract. The Electronic Systems Division (ESD) at Hanscom AFB, Massachusetts, funded the effort.

The rehosting effort began in early 1987. The ISS VAX Version 3.0 was the baseline software for the rehosting effort. This version upgraded the software from the Irvine Science Ada Compiler to the validated DEC Ada Compiler.

The ISS's evolution through a succession of compiler and operating system conversions magnified the complexity of the rehosting effort. Douglas Aircraft Co. translated most of the code from CAMIL under the Cyber NOS operating system to non-standard Ada under the VAX VMS operating system. Next, they translated it to DEC Ada under the VAX VMS operating system. Finally, Mei Associates, Inc. translated it to ALSYS Ada under the Zenith MS-DOS operating system. The code still retains features of automatic translation from CAMIL to Ada, notably repetitive code generation. In addition, the code has typing features that relate to hardware instead of specific applications. It is also poorly modularized and does not make use of features in Ada that make code portable.

Possibly the most significant hurdle Mei Associates, Inc. overcame in the rehosting process was an integer size problem. Moving from the VAX to the Z-248 required moving from a 32-bit architecture to a 16-bit architecture. In VAX Ada, the standard integer type is 32 bits. Consequently, all the integers in the baseline software were 32 bits. The result was that many type integer objects were too short to hold the quantities they represented. The following systematic process resolved the long integer dilemma:

1. Identified all integer-dependent program units.

2. Edited VAX-specific pragmas in the dependent units. Pragmas are directives to the compiler that allow programs to take advantage of a variety of implementation-dependent features of a compiler. Examples are packing text information to save space and setting priority levels in a multi-tasking environment.

3. Compiled units to identify the differences between VAX integer types and Z-248 integer types. Also identified data structures whose size exceeded the ALSYS Version 3.0 limit and compilation units whose code exceeded the MS-DOS segment size limit.

4. Modified integer sizes to resolve differences.

5. Examined remaining integer declarations to ensure they were appropriate for the values they represented.

Following the rehosting process, Mei Associates, Inc. wrote a program to allow courseware created on the VAX to run on the Z-248. To accomplish this task, they developed a program to read the file definitions and convert the data to an intermediate form. Next, they ran a program to write binary ISS data to VAX output files. Lastly, Mei Associates, Inc. transferred the VAX output files to Zenith input files with an off-the-shelf file transfer program.

The initial rehosting, completed in April 1988, was a resounding success. A complete description of this effort is at Appendix D.

Later, Mei Associates, Inc. rehosted the MicroCMI subsystem and interactive videodisc capability to the Z-248. They completed this task, also funded by the ESD, in June 1989. Appendix E provides a listing of the MicroISS functional capabilities established as a result of the rehosting effort. Appendix F lists the executables that provide these capabilities.

Upgrading the ISS Testbed

In the Summer of 1988, Dr. J. Michael Spector examined the MicroISS source code and recommended three upgrades. The upgrades would have made it a more widely accessible testbed CBT delivery system. The first upgrade was to convert from the ALSYS Ada Compiler

to the Meridian Ada Compiler. This would have eliminated a potential licensing fee to ALSYS for each run-time version of the ISS. The second upgrade was to eliminate the need for a math coprocessor. This modification would have broadened the potential base of testbed developers. The third upgrade was to eliminate the Tektronix terminal emulation, a dated capability in the Z-248 environment.

Dr. Spector attempted to convert the ISS from the ALSYS Ada Compiler to the Meridian AdaVantage 2.01 Ada Compiler. The conversion was impossible due to code size restrictions. The AdaVantage 2.01 Compiler restricted programs to 10 user-defined packages and 200 statements per compilation unit. Maximum individual data objects was 64K.

AdaVantage 2.01 did not implement a needed pragma and a specific representation attribute. The AdaVantage 2.01 Compiler did not recognize the Pragma Controlled, which restricts the storage reclamation for dereferenced access types. Attributes operate much like functions, normally returning a single value when evaluated. AdaVantage 2.01 did not implement the Size Attribute, which returns the number of bits in memory allocated to a particular object.

Dr. Spector attempted the conversion with two other compilers, the Meridian AdaVantage Version 3.0 and AETECH's IntegrAda Version 4.01, to no avail. AdaVantage 3.0 overcame both the size restriction and the difficulty with the Size Attribute. It also has an extended mode version, which allowed it to accommodate the extended memory requirements of the ISS on the Z-248. However, it did not implement the Pragma Controlled; so, it was still not possible to convert the ISS to the Meridian Ada Compiler.

AETECH's IntegrAda Version 4.01 also proved inadequate to handle the ISS. The size restrictions were more serious than for the Meridian Compiler. The maximum code size per compilation unit was 32 kilobytes -- half the Meridian restriction. In addition, the maximum allowed source code size with IntegrAda 4.01 is 144 kilobytes. At least one ISS compilation unit exceeds this limit at 155 kilobytes. IntegrAda 4.01 allowed a maximum of 300 compilation units and 80 WITHs per program. It did not implement two essential Pragmas, Controlled and Interface. The ISS uses these pragmas extensively to define the interface to the assembly language driver routine.

The second upgrade issue involved the ISS requirement for a math coprocessor. Both the ISS code and ALSYS Ada Compiler require floating point capability. Floating point numbers are, in general, real numbers (i.e., 3.145, 1/3, 1.45e+12, etc). Since ALSYS does not perform floating point emulation, a math coprocessor handles arithmetic operations on floating point data types. Both AdaVantage and IntegrAda Compilers emulate floating point operations.The requirement for a math coprocessor still exists since conversion to these compilers was not possible. A possible future solution is to change floating point data types to fixed point data types.

The third upgrade recommended was eliminating the need to emulate the Tektronix 4105 and 4107 terminals on the Z-248 EGA system. It is possible to eliminate the Z-248 terminal emulation altogether on the Z-248; however, it requires rewriting the driver that provides interactive videodisc capability. Dr. Spector decided to leave the terminal emulation software intact since it functions very well. For a detailed description of this effort, see Appendix G of this report.

IV. CONCLUSIONS AND RECOMMENDATIONS

The ISS made significant advances under the efforts covered by this report. Contractor personnel corrected over 275 problems and added several enhancements to the software. The enhancements include a MicroCMI subsystem and interactive videodisc capability. The ISS became a more stable and maintainable product through several upgrades. The first upgrade

involved the rehosting of the VAX software to a validated Ada compiler. The second upgrade resulted in development of critical system documentation. The third upgrade involved development of the ISS CMI subsystem training materials. Mei Associates, Inc. rehosted the ISS to the Z-248 to provide Air Force users a standardized, microcomputer-based training system. Finally, Dr. Spector proposed three upgrades to the MicroISS to further standardize the software.

From its inception in the early 1970s until the present, the ISS has made notable strides in the area of CBT. As a research and development prototype, the ISS has provided a rich knowledge base for future CBT system developments.

However, the ISS needs numerous enhancements to advance it technologically and make it comparable to other CBT systems. Streamlining the code would make it more efficient. It would also eliminate duplicate code generated by automatic translations. Improving program access would make it faster and less cumbersome. Rewriting the Graphics Editor to employ a bit-mapping strategy would upgrade the graphics capability. Adding different fonts to the Graphics Editor would also enhance the graphics capability. Rewriting the user interface would update the software to incorporate pull-down windows or icons. Lastly, incorporating scheduling, word processing and automatic documentation capabilities would improve the ISS. However, it is neither wise nor cost-effective to incorporate these capabilities into 20-year-old software. Instead, future CBT system developers should use the knowledge gained from the development of this software as a foundation to develop future CBT systems. In addition, if feasible, they should extract and use the numerous algorithms that provide the ISS's extensive functional capabilities.

The ISS's strengths, when compared to other Government and commercial CBT systems, are its extensive branching and CMI capabilities. The ISS is highly recommended for organizations that must develop courseware with extensive individualization strategies. The ISS is also recommended for organizations that require scheduling, tracking, management and evaluation of a sizable number of students.

Both versions of the software, the ISS and MicroISS, are available through the NTIS.

REFERENCES

McDonnell Douglas Astronautics Company. (1986). *Instructional Support Software System.* (AFHRL-TR-85-53, AD-A166 776). Lowry AFB, CO: Training Systems Division, Air Force Human Resources Laboratory.

Mei Associates, Inc. (1988). *Computer Resource Management Technology Program - changes made porting ISS to the Z-248.*

Spector and Associates. (1989). *Upgrading the ISS test bed.*

APPENDIX A: ISS FUNCTIONAL CAPABILITIES

COURSEWARE DELIVERY
- Upper- and lowercase text
- Special characters
- Multiple colors
- Static, dynamic and interactive graphics
- Student interaction via keyboard or pointing device
- Initiation and presentation of CAI lessons
- Initiation and presentation of interactive videodisc sequences
- Initiation and presentation of simulations
- Initiation and presentation of mastery tests
- Review mode
- Course glossaries
- Student comments

COURSEWARE AUTHORING
- Menu-driven user interface
- Create, change, display, store and delete modules
- Insert, create, copy, reorder, store and delete segments
- Insert, create, copy, reorder, store and delete frames
  - Expository:
    - Information, Elaboration, Help, Title, Overview, Objective, Resource, Documentation
  - Interactive:
    - Touch, Multiple-Choice, True/False, Matching, Constructed Response (short answer)
  - Special Purpose:
    - Menu, Ada Programming Language, Simulation, Branch, Adjunct Material (Videodisc)
- Support for instructional strategies
  - Factual
  - Drill and practice
  - Tutorial
  - Simulation
  - Individualized tutorial
  - Problem solving
- Text development
- Individualization (branching)
  - Unconditionally
  - On number of frames presented/not presented
  - On number of questions answered correctly/incorrectly
  - On evaluation of author-supplied variable
    - System-defined
    - User-defined
- Overlay
- Partial screen erase/windowing
- Feedback and prompt creation
- Access, modify and display graphics created in the Graphics Editor
- Access simulations created in the Simulation Editor

- Glossary development
- Videodisc sequence development

GRAPHICS DEVELOPMENT

- Create, change, display, copy, store and delete graphics
- Keyboard, pointing device and bit pad/data pad user interface
- High-fidelity 2D graphics
- Static and dynamic
- Graphics primitives, selectable from menu
- Line drawings or filled objects, both regular and irregular-shaped
- Line color
- Colored fills
- Symbol library development
- Scale, rotate and repositioning
- Scaled text
- List existing graphics/symbol libraries
  - For a specific user
  - For all users
  - Starting with a specified name
  - Before or after a specified creation date

SIMULATION DEVELOPMENT

- Menu-driven user interface
- Create, change, display, copy, store and delete simulations
- Create, change, display, insert, copy, store and delete actions
- Create, display and delete objects (text or graphic)
- List actions/objects
- Branching logic (conditional/unconditional)
- Access, modify and display graphics created in the Graphics Editor
- Overlay
- Partial screen erase/windowing
- Feedback and prompt creation
- Expository actions
- Interactive actions
  - Touch, Multiple-Choice, True/False, Matching, Constructed
  - Response (short answer)
- Create and insert complex author-defined equations
- Random number generation

TEST DEVELOPMENT

- Menu-driven user interface
- Create, change, copy, display, store and delete mastery test questions
- Block, lesson, group and mission tests
- Online/offline
- Item randomization
- Criterion-referenced
  - Percentage
  - Number of subscales (objectives) passed
- Five Question types
  - Multiple-choice, Touch, Matching, True/False, Constructed
  - Response (short answer)

- Alternative weighing
- Definable scoring rules
  - Pass/fail by total test score
  - Pass/fail by objective
  - Critical items and objectives

TEST PRESENTATION
- Presentation of mastery test questions
- Review test items prior to scoring
- Score test items and report results
- Provide recap, detailing answer selected and correct answer

TRAINING INTERFACE
- Select/continue an assignment
- Take a test
- Receive messages
- Review training records

TRAINEE ASSIGNMENT
- Registration
- Course rosters
- Absence/disenrollment
- Certify pass
- Assignment override
- Automatic assignment processing
- Resource allocation
- Progress management
  - Estimated target completion date/computation of target date using actual rate of progress (regression equation)
  - Logon-to-logoff or shift open/close
- Record and track student performance

COURSE MANAGEMENT
- Curriculum structure and course version
- Description of each lesson, supporting modules and tests
- Definition of learning centers/shifts
- Definition of training resources managed by the system

CURRICULUM INTERFACE
- Manage and control curriculum
- Prerequisite relationships

COURSE STRUCTURING
- Course structure and management information down to lesson and test ID level
- Definition of shift hours
- Definition of learning centers
- Definition of training resources
- Specification of course management level (master course or system)

LESSON DEFINITION
- Lesson characteristics

- Module numbers of supporting lessons
- Test numbers used to evaluate mastery

TRAINING MANAGEMENT REPORTS
- Course (Current Assignment, Detailed Performance, Resources for Current Assignment, Graduate Performance History, Time Management)
- Individual (Lesson Completion Summary, Detailed Performance)
- User-defined

DATA COLLECTION AND ANALYSIS
- Response analysis
- Decision path analysis
- Student comments
- Statistical
  - Mean and standard deviation by item
  - Mean and standard deviation by objective
  - Correlation of item with objective
  - Correlation of item with test
  - Item analysis

MICRO COMPUTER-MANAGED INSTRUCTION (CMI)
- Prerequisite course lists or courses selectable in any order
- Lesson lists containing either lineraly ordered lessons or lessons selectable in any order
- Reordering lessons within courses
- Embedded and mastery test items
- Registering of students into multiple courses
- Student rosters
  - Specification of number of lessons that must be taken in a course
  - Student names and IDs
  - Students' current lesson
- Student assignment generation
- Lesson override
- Certify pass
- Recording and tracking of student performance on a limited basis
  - Lessons passed
  - Performance on individual lesson questions
  - Lesson completion times

HARD COPY PRINT
- Graphics database
  - Graphic names
  - Graphic descriptions
  - Number of times graphic accessed in authoring and simulation editors
- Courseware database
  - Lesson names
  - Text (all or specified segments)
  - Feedback and prompts
  - Graphic names
  - Branching logic
  - Number of attempts for question frames

- Question stem and alternatives, including correct answer
- Glossary database
  - Words specified for a course
  - Word definitions
- Simulation database
  - Simulation names
  - Text (all or specified actions)
  - Feedback and prompts
  - Objects
    - Text and graphic names
    - Rotation, scaling and x,y positioning coordinates
  - Branching logic
  - Number of attempts for question frames
  - Events

ADDITIONAL CAPABILITIES

- Deferred message
- Instructor monitoring
- Dual screen
- System security (user access levels)
- Offline test scoring for block, group, lesson and mission tests
- Homework completion outside of trainee's shift
- Student record keeping (archiving)

APPENDIX B: VAX ISS EXECUTABLES

Executable	Source	Description
ACAIREP	CAIREP	CAI Data Analysis Report - Prints a Response Analysis Report, Decision Point Analysis Report, or a Student Comment Report on any or all segments of a CAI course. Executed as a batch job.
ACASS	CASS	CAI Authoring Support System (CASS) Editor - Allows users to create, change, delete, or copy lessons.
ADACRF	EDTPAC	Cross Reference File Editor - Process data of Cross Reference (CRF) records produced by the Curriculum Definition Editor (CDE) and the Course Structure Editor (CSE). Allows displaying, updating, creating, or deleting CRF records.
ADAHRK	EDTPAC	Hierarchy File Editor - Processes the hierarchy (HRK) records established by the Curriculum Definition Editor (CDE) and the Course Structure Editor (CSE). The HRK records are used by the Adaptive Model (AM) to determine student assignment information. Allows displaying, updating, creating, or deleting HRK records.
ADALC	EDTPAC	Learning Center Editor - User can edit a learning center record or list learning center records.
ADAPERM	EDTPAC	Student Permanent File Editor - Allows displaying, updating, creating, deleting, or listing student data records in the database student permanent file VPERM.
ADAROS	EDTPAC	Roster File Editor - Allows displaying, updating, creating, deleting, or listing the roster records created by the Curriculum Definition Editor (CDE) in either the WROS or VROS database files.
ADATEST	TESTED	Test Development Editor - Allows users to create, change, delete, or copy tests.
ADAUTL	UTILITY	Student Absence and Disenrollment Program - User can report a student absent or present, change a student's absence reason and disenroll a student from a current course or all courses. Activated from FORMS.
ADEP	DAPPAC	Student Data Extraction Program (DEP) - The user interface to RUNDEP. Used for inputing variables to generate reports for evaluation.
AGLOSS	CASS	Glossary Editor - Allows user to display, add, change, or delete glossary entries.

AM	AMPAC	Adaptive Model - Processes a student's current assignment and determines the most appropriate assignment for the student to take next.
AMCAI	CASS	CAI Presentation Program - Displays the lessons authored by using ACASS.
AMSID	SID	Simulation Presentation - Presents simulations created by using ASID.
ASID	SID	Simulation Authoring - Editor for creating or changing simulations.
ASTL	STLPAC	Student Logon - Allows student to select or continue assignments, review training records, and read mail messages.
BGMONITOR	BGMON	Background Monitor - Performs various requests for ISS including: print requests, run program requests, suspend and resume requests, and status requests.
BGQSTATUS	BGMON	Background Monitor Queue Status - Shows background monitor processes. Also, allows suspending, killing, resuming, or changing background processes.
BLCCHK	DCP	Block Completion Check - Reads the database Block Completion File (VBLC), checks a flag to verify that the record was written to tape, and then deletes and rewrites the record with the flag set to false.
BLCVRSONE	DCP	Background program initiated by FIXVRS to set all the course version fields in the CSE and BLC record keys to 1.
BLKELPS	CSEPAC	Emergency Student Timing Update - Adjusts upward or downward the current block elapsed time of a student after a system crash.
BOOTDBD	UTIL	Boot Database - Creates the database file ISSFILES and makes ISSFILES and TDEFILE entries to it.
BP	GREDIT	Bit Pad - Test program for the Tektronix bitpad.
BUTIL	UTIL	Program to display the contents of the cache memory for buffered IO.
CAIREPREQ	CAIREP	CAI Data Analysis Report Request Program - Used to select the type of report and the options, and to execute ACAIREP as a batch job.
CAISIZES	TESTED	Displays sizes of CAI database records and keys.

CAR	CES	Batch processing portion of Course Evaluation Summary (CSE). User input and initiation provided by RUNCAR.
CDMAIN	CDEPAC	Curriculum Definition Editor (CDE) - Used to display, create, edit, copy or delete a curriculum.
CMIREPORT	FRMPAC	Submits reports to the background. Activated from the Forms Editor.
CMISIZES	OTHPAC	Displays sizes of CMI database records and keys.
CREATEDB	UTIL	Create Database - Creates the database files from definitions read from a file definition file (.FDF).
CREATEFDF	UTIL	Create FDF - Creates the file definition file (.FDF) from the file attributes read from the database file ISSFILES.
CRMAIN	CRSPAC	Course Description Program - Used for setting up general course data and documentation and Student Progress Management (SPM) data. Activated from the Course Structure Editor (CSE).
CRMAPFILE	UTIL	Create Map File - Creates the disk mapping file for any shared memory section.
CRSMAIN	EDTPAC	Course File Editor - Allows displaying, updating, creating, or deleting course and/or management record information in either the WCRS or VCRS database files.
CSMAIN	CSEPAC	Course Structure Editor (CSE) - Used to display, create, edit, copy or delete courses, blocks or groups.
DCP1	DCP	Data Collection Program. Collects data while running in the background. Activated from RUNDCP.
DISPUSER	ZZLOGON	Display Users - Used to display users currently logged onto ISS. Accessed by Pad-4 key from LOGON.
FIXFILE	OTHPAC	Update, view, or delete CMI database record keys of CMI files.
FIXLOCK	UTIL	Fix Lock - Reads the lock table, checks for and displays hung locks, and allows the user to unlock them. System aborts can cause hung locks.
FIXVRS	DCP	Fix Versions - Starts up the background process BLCVRSONE to change all course versions in the Course Structure (CSE) and Block Completion (BLC) records to 1.
FOMAIN	FRMPAC	ISS Administrative Management Editor (Forms Editor) - Used for CMI student operations such as requesting reports and performing administration management functions including absences, disenrollments, assignment overrides, and student completion updates.

FPATCH	UTIL	File Patch - Patch or examine database files or shared memory.
FPOUND	UTIL	File Pound - A database utility that exercises create, open, write, read, delete and close on the database.
FRECOVER	UTIL	File Recovery - Used to recover lost data blocks in a database file uncovered by the FUTIL Data Base Validation feature.
FUTIL	UTIL	File Utilities - Utilities for managing the ISS database files such as analyzing, creating, deleting, and validating.
GLIST	CAIUTIL	Glossary List - Background program initiated by TESTREQC to print a listing of glossary words and their definitions.
GRAFEDIT	GREDIT	Graphics Editor - Allows the user to create, modify, list, copy or delete graphics.
GRLIST	GREDIT	Graphics List - Displays a list of graphics and allows the user to select graphics to be deleted.
GRUPDCOM	GREDIT	Update Graphics Commands - Used for changing graphics editor menus and the commands associated with them.
GRVERIFY	GREDIT	Graphics Verify - Used to verify GRLIST deleted the selected graphics.
HELPED	HLP	Help Editor - ISS utility to create and edit help screens for other ISS programs. Uses the database file MHELP.
HELPGEN	MSG	Generates the Help records for the HELP function. Reads an ASCII file MAILHELP.DAT and writes the records to the database file MAIL to be used by the ISS MAIL message utility.
IMON	MONTOR	Instructor Monitoring - An author or instructor can monitor a student working within ISS.
INITPGM	ZZLOGON	Initialize PGMNAME - Creates the database file PGMNAME if it does not exist from information in AUSRPGM.
INITSHMEM	UTIL	Initialize Shared Memory - Initializes the shared memory file SHMEM.DAT by writing null blocks.
INITTERM	UTIL	Initialize Terminal - Initializes a particular ISS terminal/process entry. The user will be reprompted for terminal type the next time he/she runs a program.
ISSBGBOOT	UTIL	Background Boot - Starts the background monitor and the adaptive model (AM).

ISSUSERS	UTIL	ISS Users - Displays the current users/processes of ISS.
LDMAIN	LDEPAC	Lesson Definition Editor - Allows user to display, edit, copy, create and list lessons.
LGCOMMAND	ZZLOGON	Logon Commands - Allows user to update the system banner, start/stop program activity, start/stop background processor, and other ISS system operations. Accessed by Pad-3 key from LOGON.
LISTLOG	ZZLOGON	List Log - Displays the ISS log file. Accessed by Pad-5 key from LOGON.
LOGON	ZZLOGON	ISS Logon - Controls entry into ISS.
MAIL	MSG	ISS Mail Utility - Used to send messages to other ISS users and to read messages from other users.
MGCAI	CASS	CAI Data Mover (Get) - Reads a system file created by MPCAI containing lessons and writes the contents to the database CAI files ACAICAI, ACAIOBJ, ACAIBRN, ACAIVAR, ACAIEXP, ACAIALT, and ACAITXT. Use TMGCAI if the transfer file was made from a pre-version 4 database and you wish to move the lessons into a version 4.x database. Otherwise use this program. If unsure which to use, try MGCAI first.
MGGRMENU	GREDIT	Graphics Menus Mover (Get) - Reads a text file created by MPGRMENU containing graphic editor menus and writes the contents to the database graphics menu file GRMENUS.
MGMAIN	MGTPAC	Management Control - Management of courses for the Course Structure Editor (CSE). Sets up management levels for the management functions student, learning center, classroom hours, and resources. Activated from CSE.
MGSID	SID	SID Data Mover (Get) - Reads a system file created by MPSID containing simulations and writes the contents to the database simulation files ASIMSIM, ASIMOBJ, ASIMVAR, ASIMTXT, ASIMALT, ASIMEV, ASIMEXP, and ASIMSTR.
MGTEST	TESTED	Test Data Mover (Get) - Reads a system file created by MPTEST and writes the contents to the database test files VTKF, WTKF, WITM, PXWITM, WITMPTR, and WITMALT. Use TMGTEST if the transfer file was made from a pre-version 4 database and you wish to move the tests into a version 4.x database. Otherwise, use program MGTEST. If unsure which to use, try MGTEST first.
MICROMGR	MCMI	MicroCMI Management Editor - Used to create a course, define its structure, and enroll students in it.

MOVGRAF	GREDIT	Move Graphics - Moves graphics from the CYBER (CAMIL) to the VAX database. (This is an old program and should probably be deleted.)
MPCAI	CASS	CAI Data Mover (Put) - Creates a system file containing lesson contents from the database CAI files ACAICAI, ACAIOBJ, ACAIBRN, ACAIVAR, ACAIEXP, ACAIALT, and ACAITXT.
MPGRMENU	GREDIT	Graphics Menus Mover (Put) - Creates a text file containing graphics editor menus from the database graphics menu file GRMENUS.
MPMAIL	MSG	Mail Mover (Put) - Used to update the database file MAIL when the size changes.
MPSID	SID	SID Data Mover (Put) - Creates a system file containing simulations from the database simulation files ASIMSIM, ASIMOBJ, ASIMVAR, ASIMTXT, ASIMALT, ASIMEV, ASIMEXP, and ASIMSTR.
MPTEST	TESTED	Test Data Mover (Put) - Creates a system file containing test information from the database test files VTKF, WTKF, WITM, PXWITM, WITMPTR, and WITMALT.
MVGGRAF	GREDIT	Graphics Data Mover (Get) - Reads a system file created by MVPGRAF containing graphics and writes them to the database graphics file ADAGRAPH.
MVPGRAF	GREDIT	Graphics Data Mover (Put) - Creates a system file containing graphics from the database graphics file ADAGRAPH.
NEWIMP	OTHPAC	Curriculum and Course Implementation - Implements a curriculum, course, or system course by copying from the work to the production database files. Activated from Course Structure Editor (CSE) and Curriculum Definition Editor (CDE).
NEWIMR	EDTPAC	Instructor-Managed Resources Editor - Processes resources defined as instructor-managed by the Course Structure Editor (CSE).
NEWPTMGR	CAIUTIL	Print Manager - Used for running listing of CAI, SID, and graphics material. Includes procedures SIDREQC and TESTREQC.
NEWVAL	OTHPAC	Curriculum and Course Validation - Validates a curriculum, course, or system course by attempting to read all record types to determine if they exist. Activated from the Curriculum Definition Editor (CDE) and the Course Structure Editor (CSE).

PRFDAT	CASS	Student Test Response Report - Puts out a performance report based on students' responses for a given course.
RESAV	EDTPAC	Resource Availability Editor - Processes resources defined as system-managed resources by the Course Structure Editor (CSE).
RNGT	REGTEST	Student Registration - Allows user to perform registration functions such as initial registration, display or correcting registration data, and listing courses in a curriculum.
RUNATA3	ATAPAC	Automated Task Analysis - Main program for the Automated Task Analysis Author Aiding System (ATA3). Allows subject-matter experts to perform breakdown or decomposition of a task.
RUNCAR	CES	Course Evaluation Summary (CES) - The user input portion of CES. Actual processing is done by CAR. CES generates reports to evaluate student performance within a course.
RUNDCP	DCP	Data Collection Program (DCP) - The user interface to DCP which collects and creates data to be used for generating curriculum, course and student evaluations.
RUNDEP	DAPPAC	Data Extraction Program (DEP) - The batch job processing portion of DEP. The user input and initiation are provided by ADEP.
RUNSRK	DCP	Student Record Keeping (SRK) - Examines tape index files to locate student records and can retrieve student records from tape.
RUNTAR	TIE	Test Item Evaluation (TIE) - User interface to TIE which reports student performance for online and offline tests.
SCANTRON	SCAN	Optical Mark Reader - Reads and processes the SCANTRON forms.
SCEDITOR	GREDIT	Stroked Character Editor - Allows the user to edit a stroked character set.
SCRMOVER	UTIL	Script Mover Utility - Converts script definition files (.SDL) into script files (.SCR) or decodes script files (.SCR) into script definition files (.SDL).
SDPEDT	SDPPAC	Student Data Profile (SDP) Editor - Allows displaying, changing, creating, copying, or deleting student data profile records containing student/course status information located in the database file VSDP.

SDPMCR	SDPPAC	Student Data Profile Module Completion Record Editor - Allows displaying, changing, creating, copying, or deleting Module Completion Records (MCR), Course Structure Records (CSE), and Block Completion Records (BLC) located in the database files VSDP, VCSE, and VBLC.
SDPOMR	SDPPAC	Student Data Profile (SDP) Optical Mark Reader (OMR) Editor - Allows creating and editing OMR student data profile records located in the database file OSDP.
SEQUENCER	MCMI	MicroCMI Lesson Sequencer - Used by the student to get his/her lesson assignments from the course(s) he/she is registered in, and to manage his/her progress throughout the course.
SETCASSREF	CASS	Set CASS References - Sets the CASS reference count for all graphics in every lesson to the number of times they are referenced. Used after ZEROALLCASSREF.
SETSIDREF	SID	Set SID References - Set the SID reference count for all graphics in every lesson to the number of times they are referenced. Used after ZEROALLSIDREF.
SIDPRTC	CAIUTIL	Simulation Print - Prints simulation data in the background. Initiated from SIDREQC where the necessary options have been set.
SIDREP	SID	SID Reports - Processes and prints a SID report in the background.
SIDSIZES	SID	Displays SID record sizes.
SIREPREQ	SID	SID Reports Requests - Used to submit requests to SIDREP to print SID reports.
STARTAM	EXE	Start Adaptive Model - The command file that is executed as a batch job to run the adaptive model program AM in the background.
STARTBG	EXE	Start Background Monitor - The command file that is executed as a batch job to run the background monitor BGMONITOR in the background.
STARTUPBG	BGMON	Startup Background Monitor - Executes STARTBG.COM as a batch job.
STOPBG	BGMON	Stop Background Monitor - Sends a message to the background monitor BGMONITOR telling it to quit.
TAPE2DISK	DCP	Program to retrieve student data from tape and write it back to the database disk files.

TAR	TIE	Test Item Evaluation (TIE) - The batch processing portion of TIE. User input and initiation are provided by RUNTAR.
TDEFBLD	UTIL	Terminal Definition Build - Reads the .TDL, .DDL, and .NAM files and stores the device definitions in the terminal definition file (TDEFILE).
TESTPRES	TESTED	Test Presentation - Presents a test to its author or to a student. Statistics are also taken if requested by the author within ADATEST.
TESTPRT	TESTED	Test Print - Generates a report detailing the test information for a selected test.
TESTPRTC	CAIUTIL	Background program to print requested CAI modules. Initiated from TESTREQC where the necessary options have been set.
TGT_MAIN	CSEPAC	Student Progress Management Target Editor - The user may redefine the student's course speed, days to completion, course days total, or days credited.
TMGCAI	CASS	CAI Data Mover (Get) - Reads a system file created by MPCAI containing lessons and writes the contents to the CAI database files ACAICAI, ACAIOBJ, ACAIBRN, ACAIVAR, ACAIEXP, ACAIALT, and ACAITXT. Use TMGCAI if the transfer file was made from a pre-version 4 database and you wish to move the lessons into a version 4.x database. Otherwise use program MGCAI. If unsure which to use, try MGCAI first.
TMGTEST	TESTED	Test Data Mover (Get) - Reads a system file created by MPTEST and writes the contents to the database test files VTKF, WTKF, WITM, PXWITM, WITMPTR, and WITMALT. Use TMGTEST if the transfer file was made from a pre-version 4 database and you wish to move the tests into a version 4.x database. Otherwise, use program MGTEST. If unsure which to use, try MGTEST first.
TREEBEARD	BGMON	Creates a listing file of an Ada program along with an index to the procedures.
VDFEDT	DAPPAC	Variable Definition Editor - Used to create variables to be used for storing data during CMI operation. These data are then used for data analysis reporting through the Data Extraction Program (DEP).
ZALLCSSSDRF	SID	Zero All CASS and SID References - Zeros out the CASS and SID reference counts for every graphic in the database.
ZEROALLCASSREF	CASS	Zero All CASS References - Zeros out the CASS references to all graphics.

ZEROALLSIDREF	SID	Zero All SID References - Zercs out the SIC references to all graphics.
ZEROCASSREF	CASS	Zero CASS References - Zeros out the CASS references for a given graphic.
ZEROSIDREF	SID	Zero SID References - Zeros out the SID references for a given graphic.
ZZUSRED	ZZLOGON	User Editor - Used to define user access to ISS and establish permissions for those users.

APPENDIX C: ISS FUNCTIONAL DESCRIPTION (VAX VERSION)

12 FEBRUARY 1988

Last revised: 7 FEBRUARY 1990

FUNCTIONAL DESCRIPTION

FOR THE

INSTRUCTIONAL SUPPORT SYSTEM

Prepared Under Contract

F33615-85-C-0011

FOR

AIR FORCE HUMAN RESOURCES LABORATORY

BROOKS AIR FORCE BASE, TEXAS

BY

DOUGLAS AIRCRAFT COMPANY

1.0 SCOPE

This functional description describes an operational Computer Managed Instruction (CMI)/Computer Assisted Instruction (CAI) system called the Instructional Support System (ISS). The description includes those CMI and CAI functions identified as necessary in order to support appropriate training environments within the DOD. To determine what functional capabilities the ISS should contain, a portion of the effort has been to determine the requirements of various key DOD training environments. Trips have been made to key DOD installations in order to determine training requirements and to determine the DOD instructional environments that can be supported by the ISS. Additionally, existing Computer Based Instructional (CBI) systems have been analyzed to identify potential enhancements to the ISS. Appendix B contains a discussion of potential ISS enhancements. The ISS is transportable from one computer system to another, uses cost effective mini and micro computers, and is comprised of modular software to allow individual execution of the modular components.

A transportable ISS has been produced by (1) generating source code in Ada, a standard DOD High Order Language (HOL); and (2) developing a set of generalized interfaces indirectly linking the ISS application programs to the operating systems of the host machines and the terminal set supported by the ISS. Ada is the appropriate language in which to implement transportable ISS software, given its mission as a standard HOL that is available on many machines. The generalized interfaces have been produced by isolating the terminal and operating system dependencies into relatively few procedures within the interface code. In accomplishing this, the tools of the host operating system are used to the extent practicable but the interfaces are not designed for a particular machine.

In order to make the ISS an economically feasible system, low cost mini and micro computers have been utilized. It is assumed that the systems chosen for ISS implementation shall have acceptable support for terminal communication and data base input/output and shall be capable of exercising individual modular components of the application software. For terminal communications, the host operating system shall provide sufficient data manipulation capabilities so that ISS communications software can provide screen input/output, screen positioning, graphics display, color control, display synchronization, function key processing, and terminal mode control. For data base input/output, the host system shall provide sufficient functions for sequential, direct access, and indexed sequential files including creating, deletion, positioning, clearing, reading, writing, and sharing of files.

It is possible to transport data among the systems supporting the ISS via communication lines, disk, or tape. Real time distributed processing is not supported in the current version of

the system, however. The design allows for implementation of real time distribution of processes and data bases in the future.

In order to facilitate improvement, maintenance, and efficient execution of ISS software, the functional components of the ISS are modularized. Modules are available for the following components: Graphics, Materials Development, CAI Presentation, Simulation Development/Presentation, CMI, Testing, and Data Analysis. Where appropriate, a help function can be made available on a module basis to assist in the use of the module. Execution of each ISS modular component is supported so that individual portions of the system can be separately invoked and successfully executed.

Appendix A describes the process utilized in the development of this Functional Description.

2.0 APPLICABLE DOCUMENTS

The following documents form a part of this specification to the extent that they have been useful and available for reference during the development of the ISS.

SS1017F100	AIS System Specification
DP1017F005	AIS Computer Hardware Prime Item Specification
DC1017F024	AIS Computer Mainframe Critical Item Specification
DC1017F026	AIS Terminals Component Critical Item Specification
DD1017F023	Application Program Component Critical Item Specification
DD1017F004	AIS Software Subsystem Prime Item Specification
DD1017F010	Information Management Critical Item Specification
DD1017F021	Programming System Critical Item Specification
DD1017F022	Time Sharing Operating Systems Critical Item Specification
DP1017F014	Adaptive Models Component Critical Item Specification
MIL-STD-483 (USAF)	Configuration Management Practices for systems, Equipment, Munitions, and Computer Programs
AFHRL-TR-85-53	Instructional Support System Technical Report, March 1986
Software Detailed Design Document	12 February 1988
Data Base Design Document	12 February 1988

Computer Program Product Specification for the Instructional Support System 12 February 1988

ISS Reference Manuals, February 1988

2.1 Documents and information attained and used in the survey of DOD installations are as follows:

Graphics Simulation for Technical training, a panel presentation 1982 ADCIS Conference Van Courses, B.C., June 1982

Aviation Training Support System (ATSS) Functional Description (FD) for Naval Aviation Activities. REG 31408-98-76, 17 May 82, Naval Weapons Center, China Lake, CA.

Bunderson, C. Victor. Computer Support for Army Training. Final Report, 20 December 1977

Micheli, Gene S., Morris, Charles C., & Swope, William M. Computer Based Instruction Systems -- 1985 to 1995 TAEG Report No. 89, August 1980

Van Matre, Nick & Johnson, Kirk. Upgraded Navy Computer-Managed Instruction: Analysis of requirements for, and Preliminary Instructional System Specifications. NPRDC Special Report 81-26, September 1981.

System Specification for the Maintenance Information Authoring System (MIAS) N-712-354, 3 May 1982, Task 2790.

Computer-Managed Instruction in the Navy: I. Research Background and Status, NPRDC SR 80-33, September 1980.

Computer-Managed Instruction in the Navy: III. Automated Performance Testing in the Radioman "A" School, NPRDC TR-81-7, March 1981.

Computer-Managed Instruction in the Navy: IV. The Effects of Test Item Format on Learning and Knowledge Retention, NPRDC TR 81-8, March 1981.

Computer-Managed Instruction in the Navy: V. The Effects of Charter Feedback on Rate of Progress Through a CMI Course, NPRDC TR81-26, November 1981.

Computer-Based Education and Training Functions: A Summary, NPRDC TN 82-17, May 1982.

SNAP Outline (single page).

Low-Cost Microcomputer Training Systems, status report, 9/82.

3.0 FUNCTIONAL CAPABILITIES

This section describes the functional capabilities of the ISS.

3.1 Computer-Assisted Instruction (CAI)

Lesson materials are entered into, stored in, retrieved, and delivered by the computer system. When assigned by the system and requested by the trainees, they are displayed at an interactive terminal. Interactions are principally via a terminal keyboard and, if appropriate, via a touch panel, light pen, etc. (hereafter referred to as a pointing device).

3.1.1 CAI Applications. The CAI provided by this system supports the following instructional strategies:

- Factual - Linear presentation of material to convey information for later recall or recognition;
- Drill and Practice - Instruction characterized by systematic repetition of concepts, examples and problems;
- Tutorial - High level of interaction between presented materials and learning responses;
- Simulation - Presentation of a set of relationships or sequence of events of a real world device or situation;
- Individualized Tutorial Instruction;
- Concept and Principle Learning; and
- Problem Solving

3.1.2 Courseware Delivery. The ISS is capable of delivering a wide variety of courseware. This courseware is made available to all trainee and authoring terminals. Provisions are made for:

- Display of text in upper and lower case;
- Display of a variety of special characters;
- Display of multiple colors;
- Static, dynamic and interactive graphics;
- Student interaction via keyboard or pointing device; and
- Interactive Videodisc .

3.1.3 Hard Copy. Provision is made for producing hard copy versions of CAI materials.

As described in Section 3.3, a wide variety of CMI summary reports are available. Users are given the option of displaying reports online or obtaining hard copy printouts.

3.1.4 Graphics Preparation. The ISS is capable of interfacing with a keyboard, pointing device or bit pad/data pad.

3.1.5 Dual Screen Capabilities. The ISS allows presentation of a lesson on two terminals, if desired.

3.1.6 Interactive Videodisc Capabilities. The CAI subsystem supports a variety of instructional strategies and interaction capabilities, including videodisc. The videodisc capability integrates video information retrieved from a laser disc with text and graphics created with the Authoring program (CASS). The integrated video/text/graphic frames are presented with the Presentation program (CAIPres).

3.1.7 Instructional Performance. The ISS is a totally integrated instructional delivery and management system. The instructional development capabilities are discussed in Section 3.1.8 and management requirements are presented in Section 3.2.

3.1.7.1 ISS Training Interface. Provision is made for trainees to interact with the ISS through three components: The Trainee Log-on, CAI Presentation and Online Test component.

3.1.7.1.1 Trainee Log-On Component. The Trainee Log-on component provides the basic point of entry to the ISS. Upon providing identification data, the system presents the trainee with various options, including the ability to:

- Select or continue an assignment;
- Take a test;
- Receive messages;
- Review training records; or
- Log-off.

Trainee selection is through an interface component which provides an explanation of available options.

The capability to receive messages allows the trainee to receive messages from the instructor.

The option of reviewing training records allows the trainee access to data regarding a complete summary of detailed performance reports.

3.1.7.1.2 CAI Presentation Component. When a trainee selects a CAI module through the Trainee Log-on component, storage of the current assignment I.D. and transfer of control to the CAI Presentation component is provided. The Presentation component then selects and presents the module selected. The CAI Presentation component is capable of displaying and processing all developed material. Emphasis is provided by upper and lower case and colored text. The capability is provided for presentation of high fidelity 2-D graphics. These graphics are line drawings or filled with color. The capability is provided for filling both regular and irregular shapes. Graphic displays are either static or dynamic. Trainee responses to these displays are made via keyboard or pointing device.

3.1.7.1.3 On-Line Testing Component. When a student completes a CAI module with a corresponding online test or selects the testing option from the Trainee Log-on component, he is routed directly to the Online Test component.

The question types supported by the Online Test component are identical to those supported within CAI modules: touch, multiple-choice, true/false, matching, and constructed-response. The same graphics and color capabilities available within a CAI module are also available in online tests.

3.1.8 The CAI Authoring Subsystem. The ISS contains an authoring subsystem with the following characteristics:

- A menu-driven interface component provides CAI authoring capabilities, negating the need to code instructional content and logic in a computer language;
- A minimum requirement for author training;
- A capability to jump to the Simulation component and return to the original jump point upon completion of the simulation;
- A capability to develop CAI materials where text and computer graphics are integrated with video frames from a laser disc.

The ISS is structured so that the components of a CAI module--text, graphics, instructional strategy, and data collection rules--are represented as textual and graphical data rather than as program code.

The capability to manipulate graphics and create prompting and guidance for the authoring process is provided.

With regard to structure, the authoring support provides flexibility in instructional strategy selection. The Authoring

component provides support for three subsystem components:

- A CAI Authoring component;
- A Simulation component; and
- A Graphics Generation component.

3.1.8.1 CAI Authoring Component. This is the primary tool for developing CAI modules. The Authoring component allows an author to create, revise, display and delete a module. A means is provided to allow an author to change or delete only those modules he/she has created. Organization within a module is based on segments, with presentation by frames of information from within each segment. Prompts are supplied to remind the author of all possible actions whether at the segment or frame level. The Authoring component also offers highly qualified users the option of using an "expert mode." This mode features menu suppression, where feasible. At the segment level the author is given the capability of:

- Entering a segment reference number to access the frame list for that segment;
- Deleting segments;
- Inserting new segments;
- Copying segments;
- Reordering the sequence of segment presentation; or
- Backing out without taking any action.

Within a segment, a frame list is available to the author. This list defines the type of frames which comprise a given segment and provides an overview of individualization logic and/or any special conditions on specific frames. The author is provided options to:

- Access any frame;
- Insert new frames;
- Copy frames;
- Delete or reorder frames;
- Define frame individualization logic (branching); and
- Direct the CAI Presentation program to take specific action.

These options are supplied to the author through online menu selection and prompting. The following instructional frames are provided:

- Information frames;
- Question frames; and
- Special Purpose frames.

An author has the capability to copy frames from any module to which the author has access. Various types of information frames are provided. These are similar in that they present text and/or graphic information and require no trainee response other than an indication that the trainee is ready to proceed. All information frames contain up to four pages (screen displays). A distinction is made between two types of frames which make additional information available to the learner. These are the elaboration frames, which are accessed through author-defined individualization logic and the help frames, which are accessed at the trainee's option.

The system supports five types of question frames intended to evaluate trainee knowledge:

- Touch;
- Multiple choice;
- True/False;
- Matching; and
- Constructed Response.

Detailed templates are provided for these question frames. In addition, author prompting is provided.

Response processing is designed to allow considerable latitude in student response to constructed response questions. The system allows the author to specify key words, spelling tolerance, synonyms and order of input. Provisions are made for the author to try out constructed response test items without having to back out of the authoring mode.

There are five special purpose frame types:

- Menu frames - allow trainee control of instructional events. The menu consists of a series of alternatives which allow the trainees to select lessons, segments, frames, tests, etc.
- Ada Programming Language frames - allow authors access

to programs written in Ada.

- Simulation frames - allow access to simulation sequences developed via the Simulation component (see paragraph 3.1.8.2).

- Branch - a frame containing only branching logic that facilitates branching between segments.

- Adjunct Material - a frame containing a single videodisc action.

The Authoring component supports an easy-to-use overlay, partial screen erase, and windowing capability which allows the author to develop a flowing sequence of instruction in which text, graphics and video sequences can be added to and removed from the student display as a function of timing or student input. Authoring support is provided in the form of templates and prompts such that relatively complex sequences of instruction can be composed on a terminal.

Graphics are author-selectable from a library. The author is provided a set of prompts which allow changing the physical characteristics of the graphic such as the size, position, rotational angle, and outline color. Modifications made via the Authoring component do not affect the original drawing.

Videodisc processing includes the following actions:

- Display randomly selected individual video frames or a sequence of video frames from a laser disc with or without audio.

- Display text and high-resolution graphics overlay on video.

- Define video windows.

The CAI Authoring component permits individualization. The author is given the capability of specifying that a branch be taken, a counter set or incremented or a variable defined or altered:

- Unconditionally;

- Based on a specified trainee response;

- If a specified number of frames have or have not been presented;

- If at least a specified number of author-defined frames have been answered correctly/incorrectly; and

- On the basis of the evaluation of an author-supplied equation.

The ISS is also capable of supplementing the Authoring component by providing hard copy prints ranging from summary information to listings of frame content and logic. Additionally, a CAI monitoring function is provided. This function permits the concurrent display of CAI material on a slave instructor terminal and a master student terminal.

3.1.8.2 Simulation Component. The ISS allows an author without programming experience to use the Simulation component to create simple instructional simulations by the selection of overlay, partial screen erase, and branching. ISS allows the author to create complex simulations by building a detailed model of the process being simulated, including both static and dynamic graphics. This capability includes the creation, display and manipulation of text and graphics as well as the insertion of complex author-defined equations. During development, the author is able to review and modify any part of a scenario being defined without backing out of the simulation. The resulting simulation is presentable by the CAI Presentation program.

3.1.8.3 Graphics Generation Component. This component supports the generation, storage, copying, and revision of drawings employed in CAI modules, simulation scenarios, and tests. The component employs a menu-driven dialogue approach that continually supports the user with prompting information as to what options are available at all times during graphic development. The user is able to select from a menu, a variety of graphic primitives (e.g., lines, circles, arcs, ellipses, boxes, and points). Line color is specifiable before or after the geometric input. A graphic can be modified any time during or after development. The system supports the capability of filling areas that are fully bounded by lines with solid colors or patterns. In addition, the user is able to scale, rotate, or reposition a complete or partially complete drawing. The system provides the capability to input graphic information through a keyboard or pointing device. These two modes are continuously available and the user is able to switch back and forth between them at will.

The creation of a symbol library is supported, which allows the user to define a collection of drawing elements that can be recalled and used as integral parts of another graphic. The symbol library allows quick and easy revision of graphic elements for use in other graphic drawings.

The Graphics component also provides placement of alphanumeric information within graphics. The user is able to enter text in any position and store it as required. The author is also able to scale text, color it and arrange it at any angle/slant.

3.2 CMI Functions

The ISS provides a flexible and comprehensive training management system. No knowledge of computer programming is required to establish, monitor or revise the management of the training curriculum. The CMI system manages trainees through a curriculum, generates assignments, scores tests and records completed assignments and the results of each. The management system provides data collection and analysis capabilities for evaluating the effectiveness of instruction and tests. A means is also provided for insuring the security of all data and programs. Additionally, the user has the option of having the CMI system manage learning centers, instructional resources and rate of student progress.

Two separate CMI subsystems are available depending on the training needs of a particular training installation. The "full-blown" CMI subsystem provides a variety of capabilities including curriculum management, course management, resource management, student tracking, and status reporting. The MicroCMI subsystem is a streamlined CMI system for use on microcomputers in training environments where small student loads are prevalent. It provides the capability to specify linear or random lesson orders within courses, generation of student rosters and assignments, and limited recording and tracking of student performance.

3.2.1 Trainee Assignment. Provision is made for tracking a trainee's progress through a curriculum. The trainee is automatically routed through courses while tracking the trainee's performance. An online capability is provided to allow instructional personnel to review trainee status. Assignments provide the following capabilities:

- A means of determining which instructional segment the trainee should begin next;
- A means of allocating instructional resources; and
- A method for constructing an ordered list of possible assignments by balancing individualization and resource requirement concerns.

3.2.2 Progress Management. A means is provided to monitor the progress of trainees through a course. A target completion date can be estimated or computed using the actual rate of progress.

3.2.2.1 Trainee Tracking and Status Reports. Trainees are tracked through a course by updating their records with the results of each ISS interaction. The results of this detailed tracking are available in a variety of individual or class reports. The same data is recorded whether the assignment was an online CAI module or offline work.

The progress management function allows instructors ready access to the student's complete training record.

3.2.3 ISS Management Data Base. The design of the CMI function allows personnel to define the characteristics of their courses. Software changes are required only when the basic operational philosophy of the system is altered. Modifications to the management data base are accomplished through a menu-driven interface by specifying:

- The structure of each curriculum and course version;
- A description of each lesson, its supporting modules and tests;
- The definition of each learning center managed by the system; and
- Definitions and descriptions of all training resources managed by the system.

3.2.4 ISS Curriculum Interface. In order to manage and control a curriculum, four basic functions are provided:

- Curriculum Definition;
- Course Structuring;
- Lesson Definition; and
- Test Definition.

This management is provided through menu-driven editors complete with appropriate prompts. Data integrity, change control and access to old data controlled in curriculum management is provided.

3.2.4.1 Curriculum Definition. This function defines curriculum management information, including the courses within the curriculum and the prerequisite relationships among courses.

3.2.4.2 Course Structuring. This function is designed to define course structure and management information down to the level of lesson and test identification. Management applicable to the complete course, such as the number of hours in the normal working day, learning centers, and training resources are definable.

A capability is also provided to allow definition of the characteristics of the learning center or centers in which the courses are taught and the instructional resources managed by the ISS. It is possible to specify learning center characteristics

such as the center's hours of operation, and the inventory of instructional resources of each type available in each center. Definition of the instructional resources to be managed includes the following resource types:

- Fixed facilities (The student goes to the facility to work on the assigned module.);
- Portable equipment and materials (The student draws the items from a library and returns them when the assignment is completed.); and,
- Consumable materials (The trainee draws the items from supply and does not return them.).

The calendar days on which the course is scheduled to be taught is definable.

3.2.4.3 Lesson Definition. This function is used to specify the characteristics of a lesson. This will include the number(s) of the module(s) supporting the lesson and the number(s) of the test by which lesson mastery is to be evaluated.

3.2.4.4 Test Definition. This function is designed to develop the test itself, not just specify its characteristics. It is used for criterion-referenced tests. The same types of questions are supported as are available for CAI: touch, multiple choice, true/false, matching, and constructed response. Items are arranged in subscales that normally correspond to objectives. Scoring rules are definable and may include pass/fail by total test score, pass/fail by objective, and critical (must pass) items and subscales.

The ISS includes the capability for online testing. This capability allows the author to enter test items and answer keys. This component also formats the questions for the author and allows the author to decide whether or not to randomize the presentation order of the test items and the alternatives within test items.

For offline testing, an optical mark reader can be used to record results in ISS so they may be included with online test results.

3.2.5 Resource Allocation Capabilities. This function allows identification and scheduling of various training resources such as:

- Simulators;
- Part task trainers; and
- Actual equipment trainers.

Automatic scheduling of potentially scarce training resources is provided to avoid "bottlenecks" in the flow of personnel through a course. Provision is made to allow for the maintenance of an accurate equipment inventory. A capability is provided to allow personnel to review and change the training equipment inventory.

3.3 Instructional Management Functions and Reports

3.3.1 Instructor Interface. The ISS provides a number of capabilities designed to manage and report on instruction. A security system is provided to limit instructor access to only those data and trainee records for which they are directly responsible.

Instructors are provided access to the following functions for use in managing instruction:

- Registration;
- Resource control;
- Assignment (e.g., lesson assignment, course selection);
- Testing; and
- Status (e.g., absence/presence, shift open/closed, instructional progress).

3.3.2 Training Management Reports. The CMI function is capable of monitoring and evaluating the effectiveness of the entire training program. A capability is provided for generating standard reports in areas such as:

- Course;
- Training;
- Test; and
- Lesson performance.

Additionally, the capability to generate user-defined reports is provided.

3.4 Offline Test Scoring Capabilities. An offline test scoring capability exists for block, group, and lesson tests. The offline tests are completed by trainees and the corresponding test forms are input to the ISS scoring software by instructors via an optical mark reader (OMR). The OMR reads the forms and sends the results to the Adaptive Model for processing. The Adaptive Model receives either a certified pass or an assignment override instruction.

3.5 Homework. Homework can be completed outside a trainee's shift if a course is designated as a homework course. If homework is not authorized for a course then a trainee can only log on during the open shift hours.

3.6. Student Record Keeping. A capability exists for offline (tape) storage of student records for graduated students. Online storage can be efficiently utilized for active students by using this capability. Student records for graduated students can be easily retrieved for necessary reports.

3.7 MicroCMI. A simplified CAI course management system, called MicroCMI, exists within ISS. The development portion of MicroCMI, which provides course structuring, includes capabilities for:

(a) Generation of prerequisite course lists or courses selectable in any order,

(b) Generation of lesson lists containing either linearly ordered lessons or lessons that can be taken in any order,

(c) Capability to (re)sequence and (re)organize lesson orders within courses,

(d) Capability to input both online lesson questions (embedded) and mastery test items. The latter provides the capability to develop tests in the Test Editor and present them using the Test Presentation program.

The operation portion of MicroCMI, which provides student flow, includes capabilities for:

(a) Registration of students in multiple courses. If registered in more than one course and the courses are not in a prerequisite order, the student can choose which course to get into,

(b) Generation of student rosters for individual courses containing

(1) The number of lessons the student will take in a course,
(2) Student names and identifications, and
(3) The student's current lesson,

(c) Generation of student assignments,

(d) Lesson Override capability,

(e) Certify Pass capability,

(f) Initiation and presentation of CAI lessons,

(g) Initiation and presentation of mastery tests,

(h) Recording and tracking of student performance on a limited basis, including lessons passed, performance on individual lesson questions, and lesson completion times.

3.8 Access Control. Access control within the ISS is provided at several levels, ranging from unlimited access to restricted access. This control is implemented within a log-on procedure and designates the capabilities a user is allowed. A trainee is able to view his own records and is able to access only tests, courseware, etc., assigned to him by the ISS. In addition, layers of protection are provided in each access group.

3.9. Capability to Send and Receive Messages. All registered ISS students have the capability to receive messages from their instructor. All instructors, courseware developers, curriculum developers, and other non-trainee type of personnel have the capability to send and receive messages to/from other users. The function provided is a deferred message capability. As a user enters the system, notification is given of any messages existing for that user.

4.0 GENERAL HARDWARE CAPABILITIES

Computer systems are provided to support the instructional, administrative, and management functions specified in Section 3.0. The systems include the processor, memory, communication interfaces, and peripheral devices needed to provide support for authoring and student presentation areas of the ISS, and provide the CMI processing power and record storage.

The systems provide acceptable support for terminal communication and data base input/output. For terminal communications, the host operating system provides sufficient data manipulation capabilities so that ISS communications software can provide input/output, screen positioning, graphics display, color control, display synchronization, function key processing, and terminal mode control. For data base input/output, the host system provides sufficient file functions for sequential, direct access, and indexed sequential files including creation, deletion, positioning, clearing, reading, writing, and sharing of files. With the large memory and execution capacities of a virtual machine, concurrent execution of the modular components is supported. The computer has less central memory than the programs require; therefore the programs and data are paged from disk when necessary, making the address space larger than the memory available.

5.0 MicroISS

MicroISS is the microcomputer version of ISS. MicroISS has been transported from the minicomputer to microcomputer environment and maintains all the functional capabilities addressed in CAI and MicroCMI, sections 3.1 and 3.7 respectively. MicroISS exists in two versions, standalone and networked. The only distinction between the two is that workstations within the networked version share a common database that resides on the server. Standalone versions utilize a database that resides on the users system.

APPENDIX A

FUNCTIONAL DESCRIPTION DEVELOPMENT PROCESS

I. DOD installations/projects visited

McDonnell Douglas personnel visited the three sites recommended in the Statement of Work. These sites were:

- Navy CMI System, Memphis, Tennessee;
- Navy Versatile Training System (VTS), China Lake, California;
- Army Instructional Management System (AIMS), Fort Sill, Oklahoma

In addition, data was received from the Denver Research Institute (DRI) on the following sites/systems:

- Army Research Institute
  - -- AMTESS
  - -- TASK
  - -- FAULT
  - -- PEAM
  - -- ACTS
  - -- SDMS
  - -- AREIS
  - -- Computerized Tutor
- NPRDC
  - -- AIM
  - -- EEMT
  - -- CBESS
  - -- low cost Microcomputer training systems

Functions identified by this effort were incorporated wherever possible into the functional capabilities described in Section 3.0.

II. Methodology

Information was gathered using a structured interview technique. In the interest of standardization, both DRI and McDonnell Douglas used an identical interview schedule. The schedule is quite extensive as it was necessary to acquire information about hardware and software requirements in addition to instructional concerns. Wherever possible, interviews were obtained from developer/operators as well as users. However, as many of these systems surveyed were not

operational, the obtained information reflects a developer/operator bias.

APPENDIX B

POTENTIAL ISS ENHANCEMENTS

Design of the ISS does not preclude or prevent the following potential enhancements:

I. CAI Applications

Intelligent CAI - Artificial intelligence techniques offer a way to model the student-tutor learning environment. The tutor is a subject matter expert, can determine what the student knows about the subject matter area, and can develop a learning strategy suitable for the student's needs. The ISS does not prevent the capability for an intelligent CAI system which consists of an expert module, student module, and a tutor module. The expert module would use its information base to generate and solve problems. The student module would model the level of understanding of a student, and the tutor module would devise tutorial strategies for an individual student.

II. Aids to Instructional System Design and Development

Software modules could be developed which support the conduct of task analysis, curriculum design, media selection, instructional materials development and instructional system evaluation.

III. Support for Counseling and Guidance

A potential enhancement would be the ability to provide computer generated reports to support trainee guidance and counseling by instructor and course managers.

IV. Interface Capability

The capability to directly pass data to and from external training devices (not computers) would be of value in supporting certain large-scale training activities. The major value would be in the elimination of a paper chain between devices such as simulators and part-task trainers and the ISS. A direct interface between ISS and other training devices would also ease problems in scheduling scarce resources.

V. Real Time Terminal-to-Terminal Interaction

Gaming would be supported in this mode of operation. For example, two students could interactively play a chess game.

APPENDIX D: CHANGES MADE PORTING ISS TO THE Z-248

APPENDIX D:

Computer Resource Management Technology Program - Changes Made Porting ISS to the Z-248

U.S. Department of Transportation
Research and Special Programs Administration

August 1988

Prepared by:
Mei Associates, Inc.
1050 Waltham Street
Lexington, MA 02173

Sponsored by:
U.S. Air Force
AFHRL/IDC
Tech. Development Branch
Brooks AFB, TX 78235-5000

Prepared for:
U.S. Department of Transportation
Transportation Systems Center
Kendall Square
Cambridge, MA 02142

1. INTRODUCTION

The purpose of this report is to document the changes made to ISS to bring it into the PC/DOS environment, and in the process, to explain what needs to be done to port the ISS PC version to other machines. The report is organized into four sections:

- The System Environment. This documents the host O/S services ISS requires outside of the code, including command (batch) files and environment strings.
- The Virtual Machine Layer. The Virtual Machine Layer (VML) is the portion of the ISS code where all machine dependencies are to be located. The Z-248 design of this section of code is sketched.
- Implicit Implementation Dependencies. During the rehosting effort, many subtle VAX-specific assumptions were discovered in the ISS code. These dependencies are described, as well as what portable alternatives replaced them.
- Recommendations for Future Changes. With some re-design, ISS can be made even more portable than the current Z-248 version. Some proposals for such an effort are made in this section.

This document assumes some familiarity with both Ada and the VAX version of ISS.

The Ada language is well known for its support of portable programming. Ada provides many facilities for specifying a solution in machine-independent terms. Generally, an Ada program need not incorporate machine-specific details unless the underlying algorithm requires it. Furthermore, machine-specific details can be separated out and centralized, so that only a small well-defined section of code needs to be modified to port the program to another machine.

It is often assumed that any program written in Ada is automatically portable. This is not the case. Ada programs can include machine-specific constructs which are not portable. Such constructs may be required for certain applications. A program could also include machine-specific details even if they really were not necessary; for example, a program might use the literal "2147483647" to represent the last INTEGER value, when the machine-independent (and clearer) form "INTEGER'LAST" could have been used instead.

ISS is written in Ada, but it often fails to make use of Ada features that promote portability. Given that much of the ISS code was generated by automatic translation from another language, this is not surprising. The rehosting of ISS to the Z-248 was, therefore, unusually difficult for an Ada program.

The results of the porting efforts, however, were more than satisfactory. Our goal from the start was not just to modify the code so that it would run on the Z-248, but to eliminate or at least centralize as many of the machine dependencies as possible within the time constraints. The resulting PC version of ISS is not as portable as it could be; nevertheless, it can now be rehosted to the VAX or to other machines with reasonable ease. Moreover, groundwork has been laid to make ISS even more portable.

1.1 THE SYSTEM ENVIRONMENT

On the VAX, ISS requires more of the system environment than just data files and executable images. ISS also uses a logical name table, symbols, and command files. The PC version required that DOS equivalents be found for these VAX/VMS facilities.

1.1.1 Logical Names

The VAX version of ISS uses logical names to identify certain environment-specific details. The directories ISS uses are specified by logicals; e.g., the logical name DBDDIR identifies the directory of database files, and EXEDIR identifies the directory with the ISS executable images. We used the obvious DOS equivalent for logical names -- environment variables.

The DOS default size for the environment space is only 160 bytes, which is too small to accommodate all the environment strings ISS needs. Fortunately, it is possible to create much larger environment spaces using the SHELL command in DOS Versions 3.0 and later. The Installation Guide for the Z-248 version ISS explains how to increase the environment space.

1.1.2 Symbols

On the VAX, ISS uses symbols to simplify the commands a user must enter. ISS itself is defined as a symbol; its translation is a call to a command file that runs EXEDIR:LOGON. Roughly the same effect is obtained in DOS using batch files located in a directory specified in the PATH variable. For the Z-248 system, the EXEDIR directory is included in the path; so the batch files corresponding to ISS symbols have been placed in this directory.

1.1.3 Command Files

The VAX version of ISS uses VMS command files for common tasks. A command file is a text file of VMS commands. VMS command files can be fairly sophisticated; they can use rudimentary control structures, do simple text file I/O, and call one another. ISS uses command files to set up the system (that is, to define or redefine the symbols and logical names ISS uses), to run processes (including background processes), and to do system generation.

Theoretically, The DOS equivalent of VMS command files are batch files. Unfortunately, VMS command files do not always translate easily into DOS batch files, for the DOS batch file facility is considerably less powerful than the corresponding VMS facility. Until DOS 3.3, there was no mechanism by which batch file A could call batch file B without either abandoning the execution of A or creating a separate environment for B. Because of the differences between command and batch files, it was often easier to develop a batch file that met our DOS needs than to adapt the corresponding VMS command file.

The VAX command files can support several different ISS environments. This hardly seemed necessary on the Z-248; so support for multiple Z-248 environments has not been developed. Instead, we developed the batch file BOOTISS.BAT, which assigns the appropriate values to all of the logicals (i.e., environment variables) that the ISS software refers to, and adds the EXEDIR directory to the path. This batch file has roughly the same functionality as the ENVDEFS.COM file on the VAX.

There was one tricky part about implementing BOOTISS; the environment string SYSBOOTTIME needs to be set to the machine's boot time, written in the form

```
" YYYY MM DD HH MM SS "
```

On the VAX, the file ENVDEFS.COM file accomplishes this task by calling another command file, MGRDIR:GETBOOTIM.COM. GETBOOTIM gets the boot time in string format from a system call, edits the time string into the desired format, and assigns it to SYSBOOTTIME.

Performing the same task under DOS poses several problems. The batch file facility does not support even the most rudimentary string processing operations. Instead of implementing GETBOOTIM as a batch file, it was implemented as an Ada procedure. There is no DOS equivalent of the system call that returns the boot time; so the Z-248 version of GETBOOTIM uses the current time (obtained from CALENDAR.CLOCK) as the boot time. Finally, a DOS program cannot update the environment; so our version of GETBOOTIM outputs a batch file SETBOOTI.BAT that assigns the "boot time" to SYSBOOTTIME. The last action of BOOTISS.BAT is to call SETBOOTI.BAT.

As inelegant as the Z-248 batch files are, they do demonstrate that ISS can be supported on any system with even the most minimal command file support.

2. THE VIRTUAL MACHINE LAYER

The Virtual Machine Layer (VML) portion of ISS is the interface between the Ada code and system facilities. Much of the VML code on the VAX is written in languages (mostly in VAX Fortran, with a little Assembly code). Some of the Fortran code is used to access the Fortran math library. The Ada portion of the VML layer are packages that declare interfaces to the non-Ada subprograms. Most Ada VML packages have names that begin with VM (e.g., VMCONFIG, VMRTS). In the development environment, the directory SE:[EXTLIB] contains the non-Ada source files for the VML, as well as the library EXTLIB of VML object code. Every ISS module is bound with the EXTLIB library.

The VML, by its very nature, is the least portable portion of the system. The visible part of the Ada VML packages required little modification for rehosting, but the non-Ada portion of the VML was so VAX/VMS specific that it had to be redeveloped for the DOS environment.

The Z-248 VML subprograms are written in Ada and Assembler. Although none of the VAX VML subprograms were written in Ada, Ada turned out to be an excellent language for developing the VML. Access to DOS facilities was obtained with two Ada packages, DOS and DOSE, that are provided with the Alsys AT compiler. Writing the Z-248 VML in Ada had these advantages:

- The Ada code is guaranteed to be compatible with the Alsys Ada runtime.
- The Ada portion of the VML can be debugged with an Ada source line debugger.
- The VML code written in Ada is easier to maintain.
- Some VML subprograms can be written using portable Ada code.

The Z-248 does not offer equivalents to all of the VAX/VMS system services used in the VAX version of the VML; so some services had to be simulated or avoided.

2.1 CONFIGURATION CONSTANTS

The package VMCONFIG declares constants that establish certain characteristics of the machine or the version of ISS. To configure ISS for another machine or version, one would modify the constants in VMCONFIG to reflect the new context.

The concept behind VMCONFIG is laudable in that it centralizes machine dependencies. The implementation of VMCONFIG, however, was not well done. The following anomalies were discovered in the porting effort.

- Many of the constants in VMCONFIG are not used anywhere else in current versions of ISS.

- Some of the constants (such as MAXINT) should not be used, since they duplicate results that can be obtained more readily by using attributes or the SYSTEM package.

- One of the objects, the string VERSION, was implemented as a variable (rather than a constant). VERSION is assigned a static value (e.g., "3.0") during the elaboration of VMCONFIG; the body of VMCONFIG serves no other purpose other than to assign VERSION a value, which is not changed.

For the Z-248 version of ISS, the body was eliminated and VERSION is declared as a constant. Unused or ill-advised constants were commented out, leaving only five constants. These five constants were assigned appropriate values for the AT. Some constants specific to the Z-248 version are declared here.

2.2 THE ADDR PACKAGE

The ISS package ADDR declares operations on the type ADDRESS, declared in the pre-defined package SYSTEM. The ADDR operations fall into these categories:

- The functions TO_INTEGER and TO_ADDRESS, for INTEGER/ADDRESS conversions.

- Address arithmetic functions, in the form of overloading the "+" and "-" operators; the declared operations are:

```
function "+" (LEFT  : in ADDRESS;
              RIGHT : in INTEGER) return ADDRESS;

function "+" (LEFT  : in INTEGER;
              RIGHT : in ADDRESS) return ADDRESS;

function "-" (LEFT  : in ADDRESS;
              RIGHT : in INTEGER) return ADDRESS;

function "-" (LEFT  : in ADDRESS;
              RIGHT : in ADDRESS) return INTEGER;
```

- ADDRESS comparison operations; that is, versions of "<", "<=", ">=", and ">" for the type ADDRESS.

Developing a Z-248 version of this package was particularly difficult, due to the nature of the 80286 addressing scheme. The 80286 form of addressing uses two unsigned 16-bit integers: the segment and the offset. Moreover, the interpretation of the segment portion depends on the mode in which the program is running. Many of the ADDR operations do not always have reasonable interpretations in the 80286 segmented memory model. The following problems were readily apparent:

- The INTEGER type on the Z-248 is only 16 bits long; it is therefore too short to hold ADDRESS values, or to hold the displacement between two arbitrary addresses.

- If we have two ADDRESS values from distinct segments, there is no simple, reliable way to order them, or to subtract them.

- An integer can only be added to or subtracted from an ADDRESS value reliably if the result is an address in the same segment. If the address offset plus or minus the integer cannot be represented as an offset (a 16-bit unsigned integer), then there is no mode-independent method to add or subtract the address and integer.

Under no circumstances should the ADDR package return incorrect ADDRESS values. The potential damage caused by this package could be enormous. We therefore added an exception to ADDR called ADDRESS_ERROR, which is raised by an ADDR operation when it cannot return a meaningful ADDRESS result.

The use of the ADDR package was examined, and it was found that the address arithmetic operations were often unnecessary. For example, address subtraction was often used to find the offset of a record component from the beginning of the record. This offset can be found using the standard 'POSITION attribute. An expression such as

```
STR'ADDRESS + (N - 1)
```

would be used to find the address of the N^{th} component of the string STR, when the simpler, portable and more dependable expression

```
STR (N)'ADDRESS
```

could be used instead. Many unnecessary uses of the ADDR package were eliminated.

On the Z-248, the type INTEGER is not appropriate for ADDRESS conversions or arithmetic. It is rather poor style to use the pre-defined INTEGER type here, since its range is implementation dependent; a user-defined type or subtype should be used instead. We therefore added the subtype BYTE_COUNT, which is used in place of INTEGER in all the ADDR conversion/arithmetic operations. On the Z-248, BYTE_COUNT is declared as

```
subtype BYTE_COUNT is LONG_INTEGER;
```

On the VAX, it would be declared as a subtype of INTEGER. To port this package to another machine, a reasonable choice for BYTE_COUNT must be made, but at least the decision process has been centralized to this one declaration.

In addition to changes required for rehosting ADDR, other additions were made to ADDR. The subtype BIT_COUNT, used to represent the size of arbitrary objects in bits, was added, as well as the in-lined functions IN_BYTES and IN_BITS which perform the somewhat machine-dependent conversions between size measurements in bits and in bytes.

2.3 LOW-LEVEL MEMORY OPERATIONS

The VML offers six low-level memory subprograms that operate on objects specified by starting address and size. The subprograms are distinguished by having names that end with "MEM."

MOVEMEM Given a source address, a target address, and the number of bits to move, this procedure will copy bits from the source address to the target address.

NEWMEM This function allocates a block of memory of a given size, and returns the address of this block. The value returned by NEWMEM is of type ADDRESS; in most applications, the return value of NEWMEM is converted to an appropriate access value, usually by MOVEMEM.

FREEMEM Given the address of an access or address value, FREEMEM will deallocate the storage that the access/address value references, and sets the reference to null. If the access/address value is null to begin with, FREEMEM has no effect.

FILLMEM This procedure is used to fill in a section of memory with a given fill character. The default fill character is ASCII.NUL. FILLMEM is often used to zero out every component in a composite object as an initialization.

COMPAREMEM The COMPAREMEM function compares two equal-sized sections of memory. The parameters are two addresses and a common size. If the sections are not equal, the function returns the index of the first byte where these sections differ. If the sections are equal, 0 is returned.

SEARCH_MEM Searches for the occurrence of one string inside the other, where both strings are specified as address - length pairs.

The experienced Ada programmer may be puzzled by these operations, since standard Ada operations can produce the same effects without resorting to low-level memory manipulations. In-line documentation indicates that at least some of these operations were used because ISS was first translated to a subset of Ada that did not have features such as UNCHECKED_DEALLOCATION that could take the place of memory subprograms.

These subprograms are completely flexible in that they can be used on virtually any object, independent of its type. They are also completely unprotected; if the parameter specifying the size exceeds the size of the object specified with the address parameter, some section of memory can get overwritten. The size parameters are typically specified with the 'SIZE attribute to avoid this problem.

It should be noted that the low-level memory facility implicitly assumes that an access value consists solely of an address. This assumption is true for most Ada implementations, including the VAX and Z-248. An Ada implementation is not, however, required to represent access values this way, and without this assumption, some of these low-level memory operations simply will not work.

Curiously, in the VAX version of the VML, the low-level memory operations are declared in two separate packages: VMRTS and UT. In both packages, the memory subprograms were interfaced to exactly the same Fortran code. This duplicity was eliminated in the Z-248 version by dropping the UT package and implementing the VMRTS package. Looking back, it would have been more efficient to drop these operations from VMRTS and implement UT, for two reasons. More units depended on UT than on VMRTS; so, fewer context clauses would require modification. Moreover, the UT package could help decompose VMRTS, which is currently a rather large package of very loosely related entities.

The low-level memory operations were not difficult to implement in Ada. The general implementation strategy was to convert the address parameters to string access values. The required operations can then be easily and efficiently performed using the Ada array manipulation facilities. This approach was simple, and with the possible exceptions of NEWMEM and FREEMEM, very portable.

2.4 MATH FUNCTIONS

The package VMRTS (acronym for Virtual Machine Run Time Services) declares interfaced functions used to implement the MATH package. The MATH functions can be broken down into these categories:

- Functions that perform bit-level operations on integers, such as "and", "or", and circular shifts.
- Transcendental functions such as EXP, LOG, SIN, and COS.
- A random number generator.

The Z-248 implementation of the math functions was relatively straightforward. The bit-level operations were implemented using either the UNSIGNED package or assembly code. The transcendental functions were written in Assembler so that we could take advantage of the 80287 instruction set. The random number generator was written in Ada.

All of these functions should be easy to implement on other machines.

2.5 PROGRAM MANAGEMENT

An ISS session involves the execution of one or more ISS programs. The VMRTS package declares program management routines used by the PC package to coordinate the execution of the programs in an ISS session and maintain communication between the programs. The program management subprograms are:

GET_PGM_NAME Returns the primary file name of the currently executing program.

PGM_EXISTS Determines whether a program with a given name exists.

RUNPGM Terminates the current program and either starts the execution of another program or exits to the host O/S, depending on its parameter.

The program management subprograms were implemented on the Z-248 in Ada using the DOS package. The coding of GET_PGM_NAME and PGM_EXISTS made obvious uses of DOS services, but devising a Z-248 version of RUNPGM turned out to be a challenge. There is a DOS function that terminates the program that invokes it, but this DOS function does not start the execution of another program. There is also a DOS function that will dynamically load and execute a program, but this function does not terminate the program that invokes it.

RUNPGM was implemented on the Z-248 by writing an auxiliary program called CONTROL, which serves as a shell for ISS sessions. CONTROL dynamically loads and executes the various ISS programs. The CONTROL program communicates with other ISS processes via the file PROGRAM.NAM, a text file in the MGRDIR directory that defines the next ISS program to be executed. This is the basic algorithm of CONTROL.

```
loop

   Try to open PROGRAM.NAM;
   exit when (PROGRAM.NAM does not exist);

   Read the name and options of a program from the
      PROGRAM.NAM file;

   Delete the PROGRAM.NAM file;

   Load and execute the program whose name was read in;

end loop;
```

An ISS session is started by creating a PROGRAM.NAM file with the desired program name, and executing CONTROL. The batch file ISS.BAT creates a PROGRAM.NAM file for the execution of LOGON, then calls CONTROL.

The procedure RUNPGM can transfer control to another program by creating the appropriate PROGRAM.NAM file. To indicate that control should return to the host operating system, no PROGRAM.NAM file is created. In either case, a DOS procedure is called to terminate the current program, which presumably is being executed by the CONTROL procedure. The CONTROL procedure will then take the action indicated by the PROGRAM.NAM file (or its absence).

2.6 TIME MANAGEMENT

VMRTS has the following time management subprograms:

GET_DATETIME Returns the current date and time in the form of year, month, day, julian, hour, minute, and second.

GET_TIMERS Returns the elapsed time and the CPU time for the current user; both are measured in centi-seconds from the time the user logged in.

WAIT Delays the program for the number of centi-seconds indicated by its parameter.

The Z-248 version of GET_DATETIME was written in standard, fully portable Ada, using the CALENDAR package. The WAIT procedure could be implemented using the Ada delay statement, but instead this procedure was removed from the Z-248 version, and the few calls that were made to WAIT were replaced with delay statements. The only unusual time procedure was GET_TIMERS, because DOS does not maintain the information that this procedure should return.

To implement GET_TIMERS, CONTROL was modified so that its first action is to write the current time (obtained from the CALENDAR function CLOCK) to a file named LOGTIME.DAT, in the MGRDIR directory. The time stored in this file is assumed to be the user's logon time for the rest of the session. The elapsed time returned by GET_TIMERS is calculated by subtracting the current time from the "logon time" stored by CONTROL.

GET_TIMERS can return a reasonable value for the elapsed time on the Z-248. There is no way to return a reasonable value for the CPU time, however, so the Z-248 GET_TIMERS always sets the CPU time equal to the elapsed time.

One of the main uses of GET_TIMERS is in the LOGON procedure. The timer information is used to calculate the CPU utilization by the previous process, which is displayed at the top of the LOGON main page menu. Usually, the Z-248 LOGON will indicate 100% CPU utilization, which is a consequence of the fact that elapsed time

equals CPU time. Because of round-off errors, however, LOGON will sometimes indicate CPU utilization in excess of 100%.

2.7 MULTIPROCESS SUPPORT

The VMRTS package provides support for managing the current executing processes, in the form of these subprograms:

GET_PID	Returns the Process ID number for the currently executing process.
GET_TID	Returns the Terminal ID number for the current terminal.
GET_TT_NAME	Returns the type of the current terminal.
PRIMITIVE_LOCK	Obtains a lock for a given resource for the current process. If another process has locked the resource, this procedure will wait until the resource becomes unlocked.
PRIMITIVE_UNLOCK	Unlocks a resource previously locked with PRIMITIVE_LOCK, making it available to other processes.

In the absence of networking, none of these subprograms have any relevance, since there is only one process and one terminal. The current Z-248 versions of these programs are trivial. GET_PID, GET_TID, and GET_TT_NAME return constants, and the primitive locking subprograms do nothing.

When ISS is rehosted to a networked PC environment, these subprograms will require non-trivial bodies.

2.8 LOGICAL NAME TRANSLATION

The VMRTS procedure TRANLOG is used to translate logical names. For example, TRANLOG is called to translate the logical DBDDIR to find the directory that contains the database.

The Z-248 equivalent of logical names are environment strings. The DOS package provides a function for translating environment strings, which is used to implement TRANLOG. Presumably, a similar solution can be used on any machine whose host O/S has some form of the environment string facility.

2.9 EXIT HANDLING SUPPORT

The EXH package furnishes an exit handling facility, by which a package can declare an action (in the form of a local procedure) that is to be performed when the current program completes execution. Two VMRTS procedures support the EXH package, CALL and TRAPMACHINEEXCEPTIONS.

The CALL procedure has two parameters: an address and an integer. The address should be that of an Ada procedure with a single parameter of mode "in" and type integer. The CALL procedure calls the procedure at the address, passing the integer parameter to this procedure. The CALL procedure was implemented in Assembler. The AlSys Ada compiler was used to determine the Ada procedure calling conventions.

There was one minor complication with the implementation of CALL. The CALL procedure can only work with procedures that can be called by a "far" call; that is, a call from a different program segment. On the VAX, every procedure whose address is passed to CALL is local to the body of a package; such a procedure would only have "near" calls in normal Ada usage, since it would not be visible outside the package. Hence, an Ada implementation need not implement these procedures as far-callable programs.

The VAX version of ISS uses an implementation-dependent pragma, EXPORT_PROCEDURE, to make sure that the procedures whose addresses are passed to the CALL procedure are far-callable. The EXPORT_PROCEDURE pragma is not supported on the Z-248, but the desired effect of insuring that these procedures were far-callable was obtained by declaring them in their package specifications. This approach is both portable and more straightforward than the EXPORT_PROCEDURE approach.

It is hard to determine what TRAPMACHINEEXCEPTIONS is supposed to do. Currently, it is not functional on the VAX; so a null body was written for it on the AT. Its original purpose was probably to overcome some deficiency in the non-standard version of Ada that ISS was first translated to.

2.10 COMPOUND KEY CONVERSION

The ISS database (DM) management subsystem indexes records according to their keys. A key is a section at the beginning of a database record, consisting of one or more fields. There are two types of fields: string and integer. When two keys are compared, the leftmost field is considered most significant, followed by the second field from the left, and so on. The records in a database file are ordered by their keys.

It would be very convenient if two keys could be compared by simply comparing their bytes from left to right. On some machines, however, it is not that simple. On the VAX and on the Z-248, integer types are stored with their least significant byte appearing first. For example, the integer 258 (16#0102#) would be stored as

```
02 01
```

The integer 3 (16#0003#) would be stored as

```
03 00
```

A left-to-right comparison of the bytes that make up the integers 258 and 3 would therefore order 258 before 3.

The package VMCONFIG has a constant PACKS_LEFT_TO_RIGHT that indicates whether the host machine stores the most significant byte of an integer as the left-most byte. If PACKS_LEFT_TO_RIGHT is FALSE (as it is on the VAX and Z-248), keys can still be compared easily if they first go through a conversion process, by which each integer field is reversed. For example, if a key consists of the single integer 258 (16#0102#), stored as

```
02 01
```

then the converted version of the key would contain

```
01 02
```

The VMRTS procedure CONVERTCOMPOUNDKEY performs this conversion, given the address of a key and a description of its fields. The UNCONVERTCOMPOUNDKEY procedure reverses the process. These procedures were coded in Z-248 assembler. If bit-level packing were available on the Z-248, they could also be coded in Ada.

It should be noted that on machines that store the most significant byte of an integer on the left, these procedures are not called. The key conversion procedures can therefore be given null bodies on such machines.

2.11 LOW-LEVEL FILE I/O SUPPORT

The package VMIO declares two types of low-level I/O operations: file I/O and unit I/O. The file I/O subprograms in VMIO all begin with the letter F, for example FOPEN, FCLOSE, and FREAD.

The VMIO package assigns each open file a file descriptor in the form of an integer. The file descriptor is used to identify the file in the other operations. A location in the file is specified by a byte address, which is one more than the offset of the byte from the beginning of the file.

One change that had to be made in the specification of VMIO for the Z-248 is that the type INTEGER was used for byte addresses. On the Z-248, the LONG_INTEGER format is generally required to represent byte counts in files. To avoid this sort of portability problem, the subtype BYTE_ADDRESS was added to VMIO.

```
subtype BYTE_ADDRESS is LONG_INTEGER;
```

When VMIO is rehosted, BYTE_ADDRESS should be defined as an integer type or subtype that can represent arbitrary byte addresses in files.

The implementation of the VMIO file operations was done in Ada, using the DOS package. The implementation was simplified by the fact that DOS, like VMIO, assigns integers (file handles) to open files, and uses these integers to identify open files in file operations.

2.12 LOW-LEVEL UNIT I/O SUPPORT

The VMIO unit I/O operations all have names beginning with "UNIT_": UNIT_INIT, UNIT_READ, and UNIT_WRITE. These operations are at the heart of the ISS user interface. The terminal communications package (TC) uses these operations to carry out keyboard and screen I/O.

The Z-248 version supports two terminals: a Tektronix 4105 connected to the Z-248 COM1 port, and the Z-248 itself, using an EGA monitor. The Z-248 support includes the emulation of the 4105, so that courseware developed on the 4105 could be used on the Z-248. The Tektronix 4105 was chosen because the 4105 was the most commonly used terminal for ISS on the VAX at the time. The environment variable ISSSTDOUT indicates which terminal is to be used.

Both terminals require installable drivers. The EGA driver was the more complex of the two, because of the code to emulate the 4105. Both drivers were written in Assembler. To run ISS, the driver for the desired terminal must be installed via the DEVICE command in the CONFIG.SYS file, and ISSSTDOUT must be set to the correct value.

The VMIO unit I/O subprograms themselves were written in Ada, using the DOS package. These subprograms interface with the appropriate driver, as determined by ISSSTDOUT.

2.13 BACKGROUND PROCESSING

The package VMBG supplies background processing services. On the VAX, VMBG can be used to start executing either a command file or a program in the background. VMBG also has subprograms which can suspend, resume, or stop a background process, and check whether a background process is still active.

There are two VAX ISS programs that are run as background processes. These programs process requests from other ISS processes, and have no interactive I/O. An ISS background program communicates with the requesting program through a section of shared memory.

BGMONITOR Processes requests for background services, and submit print jobs.

AM Manages full CMI resources; AM is used for student registration, student logon, and course and test presentation when management is done through full CMI.

The AM and BGMONITOR processes can be started and controlled using the LGCOMMAND procedure. The LGCOMMAND can be invoked from the LOGON main page menu by pressing Pad 3.

The whole background processing facility is a major hurdle for the 80286-based machines. Without network support, there can be no real background processing, except for printing files.

Our approach was to implement the one VMBG program, PRINTFILE, that actually could be performed on the Z-248. Other VMBG operations were simply avoided. BGMONITOR and AM were not rehosted, and all BGMONITOR and AM commands were exorcised from the rehosted version of LGCOMMAND.

The full CMI system requires background processing; hence it cannot be ported to the Z-248 at this point in time. There is another course management system called MicroCMI which does not require background processing. The MicroCMI system was therefore rehosted to supply instructional management for the Z-248.

The VMBG package will no longer be a problem on either LAN's, or on 80386-based machines. When ISS is brought into these environments, the other VMBG functions can be implemented.

3. IMPLICIT IMPLEMENTATION DEPENDENCIES

Ideally, all machine dependencies should be consolidated in the Virtual Machine Layer, so that the entire system could be rehosted by simply revising the VML. The VAX version of ISS is far removed from this ideal. Many VAX-specific assumptions were discovered in the non-VML portion of ISS during the Z-248 rehost.

To merely replace all VAX-specific code with Z-248 specific code would not be acceptable for a production system. The machine dependencies found in the non-VML code were therefore replaced with machine-independent alternatives whenever possible. If a host dependency could not be removed without substantial rewriting, then we tried to centralize the dependency as much as possible, preferably by moving it to the VML.

Not all host dependencies in the current Z-248-based version of ISS are confined to the VML. The changes required to rehost the ISS code outside the VML, however, are minimal; the changes are mostly confined to a handful of package specifications in the SE and APPLIB layers. The vast majority of ISS units can now be ported without modification.

3.1 INTEGER SIZE CONFLICTS

Throughout ISS, all integer quantities are of the pre-defined type INTEGER. The type INTEGER is host dependent, and typically denotes the most common signed integer format for the given machine. Various pre-defined entities use the type INTEGER; for example, objects of the type STRING are indexed by INTEGER values.

On the VAX, the INTEGER type is 32 bits long, and can represent any value in the range

```
        - (2 ** 31) .. 2 ** 31 - 1
or
        - 2_147_483_648 .. 2_147_483_647
```

This range is large enough to represent any integer quantity encountered in ISS.

On 80286-based machines such as the Z-248, INTEGER is implemented using only 16 bits; its range is

```
        - (2 ** 15) .. 2 ** 15 - 1
or
        - 32_768 .. 32_767
```

Some ISS integer quantities cannot be represented by the Z-248 INTEGER type; for example, social security numbers can range from 0 to 999_999_999. There is 32-bit pre-defined integer type, LONG_INTEGER, available on the Z-248. The problem is that where one integer type was used on the VAX version, more than one integer type is required on the Z-248. Contrary to what the ISS

in-line documentation indicates, no real preparation was done for the possibility of moving ISS to a 16-bit machine.

This problem was by far the worst portability problem encountered in ISS. The majority of ISS compilation units required at least some modification to resolve problems related to integer types.

- Some objects that were declared as INTEGER had to be re-declared to have a larger integer type;
- ISS integer utilities such as TH.IN_ST and CAGEN.ACPTINT had to be overloaded to accommodate more than one integer type;
- Explicit type conversions were required in integer expressions that mixed integer types.

It should be noted that there is a standard Ada solution to this problem. The Ada programmer can define his or her own numeric types in terms of requirements of the application. To illustrate this, consider the following type definition (used to represent social security numbers):

```
type SOC_SEC_NUM is range 0 .. 999_999_999;
```

The type SOC_SEC_NUM will be derived from a pre-defined integer type that can represent the numbers in the given range. On the Z-248, SOC_SEC_NUM would be derived from the type LONG_INTEGER, since that is the only Z-248 pre-defined integer type that can represent all numbers in the range 0 to 999999999. On the VAX, SOC_SEC_NUM would be derived from the INTEGER type, for the same reason.

The virtue of user-defined types such as SOC_SEC_NUM is their portability. The range of values is defined in terms of the requirements of the application, rather than in terms of the numeric formats of a particular machine. The user-defined type definitions can therefore usually be ported to another machine without modification.

User-defined types is one of the many Ada portability features that, lamentably, were not incorporated into ISS. The VAX version of ISS has no user-defined numeric types. If ISS declared integer types such as SOC_SEC_NUM for every integer quantity that might exceed the 16-bit format, the integer size conflicts would not have occurred.

Eventually, ISS should be re-written to employ user-defined integer types. For the first rehosting effort, however, integer type declarations were not incorporated into the Z-248 version of ISS, because it was not feasible to make all the changes this would entail within the given time limits. Instead, we chose an alternate strategy to the integer size problem that was relatively quick to implement, and is nearly as portable as the common Ada approach.

Our overall strategy was to define application-specific subtypes of the pre-defined types. For example, we can define

```
subtype SOC_SEC_NUM is LONG_INTEGER range 0 .. 999_999_999;
```

and use this subtype to declare all objects that represent social security numbers. Examples of this can be found in the VML code; e.g. the subtype BYTE_COUNT in the package ADDR.

Application-specific subtypes of the pre-defined subtypes are not as portable as user-defined types, in that the subtype declarations themselves may require revision when rehosted. The objects declared using these subtypes, however, will not require modification; so at least we have reduced the number of changes required.

The advantage of application-specific subtypes is that only the two types INTEGER and LONG_INTEGER are used to represent almost all of the integers in ISS; therefore, we need only to come up with two versions of the integer utilities such as TH.IN_ST and CAGEN.ACPTINT. In addition, these subtypes can be turned into portable user-defined types when we have more time to improve the software.

The implementation of this strategy was time-consuming. The Ada compiler was tremendously helpful in this effort, for the type mismatches caught by the compiler often indicated an object declaration that needed to be changed. But even with the help of the compiler, lengthy analysis was often necessary to determine the range of values a given object could represent. The number of units requiring modification was also rather large.

Here are the steps taken to implement this strategy.

3.1.1 Modify the Package A

The package A defines useful subtypes for ISS. All but a handful of the ISS units depend on A. The package A declares INT as a shorter name for INTEGER. It also declares these 31 subtypes of INT:

```
subtype I1B  is INT range 0 .. 2 **  1 - 1;
subtype I2B  is INT range 0 .. 2 **  2 - 1;
subtype I3B  is INT range 0 .. 2 **  3 - 1;
...
subtype I31B is INT range 0 .. 2 ** 31 - 1;
```

In general, an INTEGER in the subtype InB can be represented using only **n** unsigned bits. These subtypes are used in component declarations to insure that when composite types are packed, the bare minimum number of bits will be used to represent each component. The author grants that this method of declaring components is a poor coding convention. Nonetheless, the A package turned out to be quite helpful in terms of fixing integer type problems.

The Z-248 version of A declares subtypes I16B through I31B to be subtypes of LONG_INTEGER instead of INTEGER. This change assigns the correct types to almost all record and array components. It also provides a pool of portable (but uninformative) subtype names for large unsigned quantities.

The subtype S32B was added to the package A. The purpose of this subtype is to rename the 32-bit signed integer type available on the host machine.

3.1.2 Add Application-Specific Subtypes

Subtypes were introduced to represent quantities too large for the 16-bit format. For example, the package PC declares

```
subtype SOC_SEC_NUM is A.I31B range 0 .. 999_999_999;
```

which is then used to declare social security numbers throughout the system.

3.1.3 Replace Constants with Named Numbers

A named number is a special kind of numeric constant whose type is universal, rather than a named type. An integer named number can therefore be implicitly converted to any integer type. The VAX version of ISS uses no named numbers, but the vast majority of the integer constants can be redefined as named numbers. To illustrate, the constant BLACK from the package TC was declared as follows on the VAX.

```
BLACK : constant INT := 1;
```

Using this declaration, BLACK has the type INT. Consequently, BLACK would require explicit conversion to be used in an expression requiring another integer type. For the Z-248 version, BLACK was redefined as a named number, as shown below:

```
BLACK : constant := 1;
```

By defining BLACK as a named number, BLACK can now be implicitly converted to any integer type. (Note: It is considered good Ada style to use named numbers in place of constants whenever possible.)

3.1.4 Create Generic Versions of Integer Utilities

To create overloaded versions of integer utilities, generic versions were written. The overloaded versions were created by replacing the original declarations with generic instantiations for the pre-defined types used on the host machine. (For the Z-248, the types are INTEGER and LONG_INTEGER.) For example, the VAX version of TH declared this integer utility:

```
function IN_ST (SOURCE : in INT;
                FORMAT : in INT := 0) return STRING;
```

The Z-248 required a version of IN_ST where SOURCE was type INTEGER, and where SOURCE was type LONG_INTEGER. The package NUM_CONVERSIONS was created with generic versions of the string-integer conversion functions (including IN_ST). Here is the specification of NUM_CONVERSIONS:

```
generic

   type NUM is range <>;

package NUM_CONVERSIONS is

   ...

   function IN_ST (SOURCE : in NUM;
                   FORMAT : in INT := 0) return STRING;

   ...

end NUM_CONVERSIONS;
```

The declaration of IN_ST in the specification was replaced with two instantiations of NUM_CONVERSIONS, one for the INTEGER and one for the LONG_INTEGER type. This provides the overloaded versions of IN_ST for the INTEGER and LONG_INTEGER types. The outline of the Z-248 version of TH is given below:

```
with NUM_CONVERSIONS;

package TH is

   ...

   package INT_CONVERSIONS is
      new NUM_CONVERSIONS (NUM => INTEGER);

   package LONG_INTEGER_CONVERSIONS is
      new NUM_CONVERSIONS (NUM => LONG_INTEGER);


   function IN_ST (SOURCE : in INT;
                   FORMAT : in INT := 0) return STRING
      renames INT_CONVERSIONS.IN_ST;


   function IN_ST (SOURCE : in LONG_INTEGER;
                   FORMAT : in INT := 0) return STRING
      renames LONG_INTEGER_CONVERSIONS.IN_ST;
   ...

end TH;
```

Note: When this version of TH is ported to another machine, the instantiations of NUM_CONVERSIONS may have to be revised to reflect the pre-defined integer types used on the new host for ISS.

3.1.5 Insert Needed Type Conversion

Even with the overloaded versions of the integer operations, numerous explicit conversions are still required. For example, the procedure CAGEN.JULIDAT has the statement:

```
JULIAN_DATE := JUL_YEAR * 1_000 + JUL_DAY;
```

where JULIAN_DATE is declared as A.I17B (a subtype of LONG_INTEGER) and JUL_YEAR, JUL_DAY are both declared integer. The expression on the right is of type INTEGER, whereas the variable on the left is of type LONG_INTEGER.

This type conflict is remedied by converting JUL_YEAR, JUL_DAY to the subtype A.I17B, as shown below.

```
JULIAN_DATE := A.I17B (JUL_YEAR) * 1_000 + A.I17B (JUL_DAY);
```

The types no longer conflict, since all calculations are now done in the A.I17B format.

Note: Every integer type conversion in ISS is a conversion to an application-specific subtype or a subtype defined in A. This was done to assure that all type conversions are portable.

3.1.6 Use Non-Integer Data Types When More Appropriate

ISS sometimes uses the type INTEGER to represent objects that can be expressed more naturally -- and portably -- using other data types. A prime example of this can be found in the package PC. The PC package defines eighteen events; many of its subprograms operate either on a single event, or on a set of events. The VAX version of PC represents each event as an INTEGER constant that is a distinct power of 2, for example:

```
F1          : constant INT := 2 **  0;
F2          : constant INT := 2 **  1;
...
CONDITIOND : constant INT := 2 ** 18;
```

A set of events is represented as the integer sum of the events. The members of the set can be derived from this sum by examining its binary notation.

The problems caused by the VAX representation of sets of events could have been solved using the other techniques discussed in this section, but the use of an integer type to represent sets of events seemed so unnatural and inconvenient that a different implementation was chosen instead. The Z-248 version of PC

represents an event as an enumeration type; a set of events is represented as a BOOLEAN array.

```
type EVENT_NAME is
   (F1,
    F2,
    ...
    CONDITIOND);

type EVENT_SET is array (EVENT_NAME) of BOOLEAN;
```

This representation is more natural and portable. The in-line documentation for the VAX version of PC recommends this change of representation for yet another reason: The number of events would not be needlessly bound by the word size.

The EVENT_SET type is the only type implemented so far as a replacement for a non-intuitive use of an integer type. There are other instances where this should be done, the most prominent being the representation of display attributes defined in the TC package.

3.1.7 Qualify Expressions That Have Become Ambiguous

When overloaded versions of integer utilities were created, some expressions that were not previously ambiguous became ambiguous. One such case is in MPCAI, where the statement

```
PUTINT (1);
```

became ambiguous once PUTINT was overloaded for the types INTEGER and LONG_INTEGER; the statement can be interpreted as a call to either version.

This was not a serious problem because it did not occur often. When this situation arose, the ambiguity could be removed by qualification, as illustrated below.

```
PUTINT (A.INT'(1));
```

3.2 PROGRAM UNIT SIZE

The size of each program segment in a Z-248 is limited to 64K. Some of the largest compilation units in ISS, such as AMCAI, could not be compiled because the size of the resulting code exceeded the limit for a single program segment.

The compilation units that exceeded the 64K limit were procedures that represent main programs. The bulk of these large procedures mostly consist of a plethora of local subprograms. The procedure AMCAI is typical; in VAX Version 3.0, the declarative part of AMCAI makes up over 4,100 of the 4,317 source lines.

Keep in mind that writing a single subprogram that is thousands of lines long is definitely not standard Ada practice. The recommended style is to use subprograms which are only a few pages long -- preferably one page. Units as long as AMCAI are difficult to understand, debug, and maintain; in addition, they require longer recompilations on the machines where they can be compiled.

The only solution to the program unit size problem is to decompose the large units into several units. Given that large units encumber program maintenance, this decomposition will be beneficial even on machines such as the VAX where larger program units are permitted.

The units that exceeded the 64K limit had all of their local subprograms, and many of their local declarations, divided among two newly created packages. Whenever possible, declarations were moved to the body of the new packages, so that the advantages of information hiding could be accrued.

The process of decomposing a large unit into packages was somewhat time-consuming. To determine where a certain entity should be declared, one must determine the other entities which depend on the given entity. In the future, however, the decomposition of large units can be done much faster by exploiting the cross-referencing tool from the AlSys Ada toolkit.

3.3 INITIALIZATION AND FILLMEM

The procedure FILLMEM is frequently used to initialize composite objects in ISS. The call

```
FILLMEM (CAI'ADDRESS, CAI'SIZE);
```

will fill the section of memory occupied by the object CAI with zeros. The desired effect is to set every scalar subcomponent to 0 or 0.0 or ASCII.NUL, which presumably is an acceptable initial value.

On the VAX, the effect of running FREEMEM can quite often be obtained automatically. Our debugging experience indicates that when fresh memory space is allocated on the VAX, it usually consists of zero bytes. Because of this, the failure to properly initial objects that should be initialized has gone undetected on the VAX.

On the Z-248, the bytes in uninitialized memory usually have the value 16#01#, so the failure to initialize objects cropped up immediately. One such problem occurred in the porting of the DM package. The local subprogram OPEN_ISSFILES reads the object ISSFILESFB, in spite of the fact that not all of the component of ISSFILESFB were initialized. The uninitialized components happen to have the correct values on the VAX, but not on the Z-248, where this procedure failed when it was first tested.

The way to avoid these problems is, of course, to initialize the uninitialized objects. The initializations added to the Z-248 version took one of two forms:

- Objects created by allocators were initialized by using a qualified expression in the allocator itself. Thus, statements such as

```
FILEID := new ASL_FILEBLOCK;
```

  were replaced with

```
FILEID := new ASL_FILEBLOCK'
   (FILE_NAME => "ISSFILES",
    FILE_TYPE => ISAM_FILE,
    ...);
```

  in which every subcomponent is initialized with a suitable value.

- Other objects were initialized by the assignment of an expression (usually an aggregate).

No additional initializations were performed using FILLMEM, for reasons that will be discussed shortly.

More difficulties arose from objects initialized with FILLMEM than from uninitialized objects. The trouble with FILLMEM initializations is that the bit pattern representation of every subcomponent is set to zero, regardless of whether this results in a desired or even a legal value for the variable. Two categories of major problems were caused by FILLMEM when used on the Z-248.

When FILLMEM is applied to an access value, the result is not the null value (as it is on the VAX). Instead, one obtains an address in segment 0, which no Ada program should be using. When FILLMEM was used on access values, the programs caused a general protection violation that would hang the system.

The AlSys Ada compiler generates implementation-dependent components in variant record types to store information useful in memory management, such as the current size of the record. The Ada language standard permits the addition of such components (see LRM 13.4, Paragraph 6). The FILLMEM procedure overwrites these components with zeros. This caused the Z-248 runtime to crash.

Besides these major problems, FILLMEM caused some minor problems in that the zero values it creates were sometimes unsuitable. All of these problems were solved by eliminating the call to FILLMEM and replacing them with standard Ada code, in the form of the assignment of an aggregate. The replacement is portable and more reliable than the corresponding FILLMEM call.

As the previous discussion indicates, the FILLMEM procedure is a poorly conceived programming construct that should be eliminated from some future version of ISS. Further arguments against the continued use of FILLMEM are presented in Section 4 of this document.

3.4 PACKED DATA TYPES

A composite data type can usually be represented in several different ways. Unless instructed otherwise, the criteria used to determine the representation of a data type is the ease of access of its components. The standard pragma PACK is used to specify that minimizing the size should be the major criteria for determining the representation of the type, even if this slows down the reading or updating of its subcomponents.

In VAX Ada, the PACK pragma is implemented as fully as possible. In a VAX packed type, each discreet subcomponent is represented by the bare minimum number of bits required to represent all its possible values.

The AlSys AT compiler does not implement the PACK pragma. In fact, the first version of the compiler (Version 1.3) did not even support more than one representation of a data type. The version of the AT compiler that we are currently using, Version 3.2, does allow representation clauses, but still uses one or more bytes to represent each subcomponent.

For the most part, the lack of packing does not affect the code, since in a high level language, representation details are abstracted out. The low-level memory operations, however, sometimes assume the bit-level packing done on the VAX. Consider the following code:

```
type BITMAP is array (INT range <>) of BOOLEAN;
pragma PACK (BITMAP);

A, B, C : BITMAP (1 .. 1024);

...

FILLMEM (A'ADDRESS, N);
MOVEMEM (C'ADDRESS, B'ADDRESS, 1024);
```

The size parameter in the calls to FILLMEM and MOVEMEM assume that each component in a BITMAP object is represented by a single bit. This assumption is not true on the Z-248; so these calls will not perform the desired function.

The simplest portable alternative to representation-dependent statements like these is to use standard, high-level alternatives. For example, the calls to FILLMEM and MOVEMEM presented above can be replaced by the following:

```
A (1 .. N) := (others => FALSE);  -- Replaces FILLMEM
B := C;                           -- Replaces MOVEMEM
```

Note that the high-level alternatives to the low-level memory operations are not only more portable, but also more readable than the code they replace. The high-level operations may also be more efficient, since they do not require a procedure call.

The package A defines two subtypes that depend on the level of packing, UNSIGNED_BYTE and UNSIGNED_BYTE_ARRAY, declared as follows:

```
subtype UNSIGNED_BYTE is INT range 0 .. 255;
pragma PACK (UNSIGNED_BYTE);

type UNSIGNED_BYTE_ARRAY is (INT range <>) of UNSIGNED_BYTE;
pragma PACK (UNSIGNED_BYTE_ARRAY);
```

When we were using Version 1.3 of the AlSys compiler, these types posed a problem, since a single byte representation of the INTEGER type was not supported. The problem was solved by investigating how UNSIGNED_BYTE_ARRAY was used. There are two general areas of the ISS code that use UNSIGNED_BYTE_ARRAY.

- In the Data Management (DM) subsystem, UNSIGNED_BYTE_ARRAY is used to store arbitrary data types and keys. For this purpose, the only properties required of UNSIGNED_BYTE is that it take up exactly one byte, that it can represent any byte, and that it have the appropriate ordering. The DM subsystem does not require that UNSIGNED_BYTE be a subtype of INT.

- In the Terminal control subsystem, the UNSIGNED_BYTE_ARRAY type is used to represent tab stops and command memory types. In the TC subsystem, it is very important that the components of these arrays be subtypes of INT, since they are used in many INT expressions. However, it is not necessary to have the components of these arrays take up one byte.

This investigation made obvious a solution. Array types were added to the TC package for the representation of tab stops and command memory types. The components of these newly added array types were of type INT. In the package A, the UNSIGNED_BYTE type was implemented using the AlSys provided package UNSIGNED. UNSIGNED_BYTE is not declared as a subtype of INT, but this is irrelevant to the DM subsystem, which is now the only place where UNSIGNED_BYTE is used.

The declaration of A.UNSIGNED_BYTE was revised again when Version 3.2 of the compiler became available. This version supports representation clauses, permitting this more portable declaration:

```
type UNSIGNED_BYTE is range 0 .. 2 ** SYSTEM.STORAGE_UNIT - 1;
for UNSIGNED_BYTE'SIZE use SYSTEM.STORAGE_UNIT;
```

3.5 MEMORY ALLOCATION AND FREEMEM

The VAX version of ISS uses the VML procedure FREEMEM for all deallocation operations. The standard Ada deallocation facility, UNCHECKED_DEALLOCATION, is not used in the VAX version of ISS.

The Z-248 version of FREEMEM will successfully free space allocated by NEWMEM. Space created by standard Ada allocators was not always completely deallocated by FREEMEM, however, due to the way the Z-248 runtime manages memory. Some Z-248 objects are preceded by a descriptor for management purposes. For example, the statement

```
S := new STRING (1 .. 10);
```

might not only allocate 10 characters, but also a descriptor located before S.all'ADDRESS for the string. If we now execute the statement

```
FREEMEM (S'ADDRESS);
```

the space taken up by the 10 characters in S.all will definitely be reclaimed. The corresponding descriptor, however, might not be reclaimed.

This problem was uncovered only recently. It was discovered that when AMCAI was presenting a particularly long lesson, it would raise STORAGE_ERROR and terminate. Our analysis showed that the reason for this was not because the storage demands of AMCAI actually exceeded what the Z-248 could supply. It was then determined that the problem was caused by leftover descriptors fragmenting the available memory.

The STORAGE_ERROR problem was alleviated by replacing nearly all calls to the FREEMEM procedure with calls to instantiations of UNCHECKED_DEALLOCATION. Since UNCHECKED_DEALLOCATION is provided by the implementation, it will deallocate any associated descriptor along with the object. At the same time, most of the calls to NEWMEM were replaced with Ada allocators.

There are two major categories of objects in ISS where NEWMEM and FREEMEM could not be readily replaced with standard Ada constructs. The storage management for these objects is still performed by NEWMEM and FREEMEM in ISS.

- In the DM subsystem, objects of the type DMTYPES.BLOCK might have, as its last component, a very large array. In many cases, the end of the array is not needed. NEWMEM is used to allocate just enough space for the part of the array component that is actually required; if Ada allocator was used, space for the whole array would be allocated.

- In the TC subsystem, NEWMEM is used to generate space for the packing and unpacking of device descriptors.

In the revised version of ISS, FREEMEM is used exclusively to free space created by NEWMEM. When this change was made, the STORAGE_ERROR problems ceased to occur.

In all future versions of ISS, the use of FREEMEM should be restricted to areas created by NEWMEM. Besides the problem encountered on the Z-248, there is another portability problem that can be caused by not adhering to this restriction. An Ada implementation may create separate storage areas for each access type. In such an implementation, the indiscriminate mixing of the VML and standard Ada memory management functions would quite likely confound the runtime system.

3.6 CODE DUPLICATION AND DEAD CODE

ISS is riddled with multiple copies of types and subprograms. There are at least four packages that have a copy of the procedure DRAW_CIRCLE. Types declarations from the GRTYPES package appear again in some of the units in the GRLIB directory. Text handling facilities are repeated (under a different name) in the package CAGEN, and the package TX contains only a subset of the TH entities with different names.

This duplication creates an added burden for any type of maintenance work, for all corrections must be done in duplicate or triplicate. During the rehosting effort, some of this duplication was eliminated. The CAGEN function I2S performs the same function as TH.IN_ST; in the Z-248 version, I2S renames TH.IN_ST, and the body of I2S was eliminated.

A related problem is dead code; that is, entities which are declared but not used in a meaningful way in the current version of the software. We discovered and eliminated a considerable amount of dead code during our investigation of the ISS code.

4. RECOMMENDATIONS FOR FUTURE CHANGES

The rehosting effort has produced a version of ISS that runs on the Z-248 and is substantially more portable than previous versions. The capabilities exist, however, to produce an even more portable version of ISS. ISS can be reworked so that all host dependencies are confined to the VML layer. This section presents a strategy for obtaining this level of portability.

This strategy should not, of course, be carried out before a truly stable version of ISS is produced, which can then serve as the basis for all future versions. The succeeding discussion assumes such a version of ISS, so that the capability to "roll back" to previous version need not be supported.

4.1 IMPLEMENT USER-DEFINED NUMERIC TYPES

The non-VML portion of ISS can be made completely independent of the pre-defined numeric formats by the careful use of user-defined types, as discussed in Section 3.1. Our work on the Z-248 has set the groundwork for such a change -- many of the application-specific subtypes can be turned into user-defined types. This change may require some additional explicit type conversions over those performed in the Z-248 version, but not many.

In moving from the current Z-248 version to the user-defined types model, the most difficult issue may well be what to do with the integer utilities such as TH.IN_ST and CAGEN.ACPTINT. As noted before, generic versions of these units have been developed. When additional integer types are added, additional instantiations of these utilities may be required. We need to determine what additional instantiations are required and where these instantiations should be performed.

Along with this effort, it may be desirable to phase out the use of the subtypes such as A.I1B, A.I2B, and A.I3B. Subtype names should reflect the objects being represented, not the space they will consume. If these subtypes were replaced with something more application-specific, the code would be easier to read, and it would be much easier to make changes.

4.2 A CRITIQUE OF LOW-LEVEL MEMORY OPERATIONS

A great number of the portability problems discussed in Section 3 involved calls to the low-level memory operations defined in the packages VMRTS and UT. Because of differences between the Z-248 and the VAX, many of the low-level memory operations had to be eliminated in the Z-248 version. There could still be machine-dependent calls to these operations that have gone undiscovered due to similarities between the Z-248 and the VAX.

The portability problems caused by these operations are due to their very nature. One of the reasons why programs written in Ada (or any other high-order language) can be portable is that

solutions are formulated in terms that are abstract enough to be independent of a particular machine's implementation. The low-level memory operations destroy this level of abstraction by introducing implementation details into the source code.

What is troublesome is that none of these operations performs a service that cannot be done with standard high level constructs. Simple, portable Ada alternatives exist for over 95% of the calls to low-level memory operations. With some more extensive changes, all calls to these operations can be eliminated.

In a program that is to be maintained on several machines, there are two generally accepted criteria for the inclusion of non-portable code:

- The non-portable code should perform some service that either cannot be done, or cannot be done as efficiently, with portable code.
- The non-portable code should be restricted to a few units, so that only a small, well defined portion of the software needs to be changed to rehost the system.

As the succeeding discussion will demonstrate, the low-level memory operations fail to meet either of these criteria, much less both of them. Their continued use cannot be justified.

4.3 ELIMINATE NEWMEM AND FREEMEM

As stated in Section 3.5, FREEMEM should only be used in conjunction with NEWMEM. If NEWMEM is discarded, then FREEMEM must be discarded as well. The question, then, is whether an access object should ever be created with NEWMEM, instead of an Ada allocator.

For the straightforward creation of objects, the Ada allocator is always preferable to using NEWMEM. The object created by an Ada allocator will fit certain requirements for the representation of the type, in particular:

- The object will conform to any alignment need.
- In the absence of explicit initializations, any applicable default expressions will be used to initialize subcomponents.
- Implementation dependent components (if any) will be given meaningful values.

There can be no assurance that objects created by NEWMEM will meet these requirements.

The only conceivable justification for NEWMEM would be for performing special functions, such as allocating only part of the space required for an object. An example of this can be found in

the procedure DMBLKMGR.NEWBLOCK. NEWBLOCK is used to create values of type DMTYPES.BLOCK_PTR, as declared below.

```
MAX_RECORD_SIZE : constant := 20_000; -- bytes

type BLOCK (BLK_KIND : BLOCK_TYPE) is
   record
      ...
      case BLK_KIND is
         ...
         when INDEX_BLOCK   =>
            ...
            IDATA: UNSIGNED_BYTE_ARRAY (1 .. MAX_RECORD_SIZE);
         when DATA_BLOCK    =>
            ...
            DDATA: UNSIGNED_BYTE_ARRAY (1 .. MAX_RECORD_SIZE);
      end case;
   end record;
pragma PACK (BLOCK);

type BLOCK_PTR is access BLOCK;
```

As these declarations indicate, a BLOCK object can end with a rather large array component. Assume that a block BLK_KIND => INDEX_BLOCK is needed, but only the first 1_000 components of the IDATA component will be used. It would be inefficient to allocate an entire BLOCK object, for 19_000 bytes of that object will not be used. The procedure NEWBLOCK uses NEWMEM to allocate space only up to the last array component used (in this case up to IDATA (1_000)).

Given the declaration of DMTYPES.BLOCK_PTR, an Ada allocator cannot achieve the same effect as NEWBLOCK. The allocator

```
new BLOCK (INDEX_BLOCK)
```

will create space for all 20_000 components of IDATA.

This is not to say, however, that the same functionality cannot be obtained using high-level constructs. The very existence of the NEWBLOCK procedure points out a flaw in the definition of the type BLOCK. The components IDATA and DDATA are declared as arrays of fixed length. In usage, IDATA and DDATA are treated as variable length arrays. If IDATA and DDATA are to be used as variable length arrays, why not declare them that way? Consider the following replacement for the previous declarations:

```
subtype RECORD_LENGTH is INT range 0 .. 20_000; -- bytes

type BLOCK (BLK_KIND : BLOCK_TYPE;
            LENGTH   : RECORD_LENGTH) is
   record
      ...
      case BLK_KIND is
         ...
         when INDEX_BLOCK   =>
            ...
            IDATA: UNSIGNED_BYTE_ARRAY (1 .. LENGTH);
         when DATA_BLOCK    =>
            ...
            DDATA: UNSIGNED_BYTE_ARRAY (1 .. LENGTH);
      end case;
   end record;
pragma PACK (BLOCK);

type BLOCK_PTR is access BLOCK;
```

With these declarations, we could use allocators such as the following to create BLOCK objects with just enough array space:

```
new BLOCK (INDEX_BLOCK, LENGTH => 1_000);
```

In short, even the special allocations that NEWMEM performs can be done using Ada allocators. Ada allocators do not have the portability problems that NEWMEM does. Code using allocators is simpler and cleaner than code using NEWMEM. There is no good reason, therefore, to use NEWMEM in place of allocators.

4.4 FILLMEM AND INITIALIZATION

Some of the problems caused by FILLMEM on the Z-248 were discussed in Section 3.3. In spite of these problems, one might be tempted to support the continued use of FILLMEM on the basis that it is a "convenient" method of initializing a composite object. If the variable CAI were initialized by an aggregate, then an expression would be needed for each component of CAI. This would require that the programmer find out what the components of CAI are, and what their possible values could be. Using FILLMEM, all of an object like CAI can be initialized using the statement

```
FILLMEM (CAI'ADDRESS, CAI'SIZE);
```

It can be argued that when FILLMEM is used, we are not required to consider the components of CAI.

This argument is easily seen to be fallacious when one considers the rationale for initializations. The danger of an uninitialized variable is that a misleading value may be read from it that is unsuitable for the application. An initialization can avoid this danger by providing the variable with a value that is meaningful in the context of the application. The initialization

may not serve this purpose, however, if the initial value is chosen arbitrarily. An arbitrarily chosen initial value could be just as meaningless, in the context of the use of the variable, as a misleading value read from an uninitialized variable.

This is precisely the problem with FILLMEM. It zeros out everything, regardless of whether this is an appropriate or even legal value for the components. In ISS, there are many components for which FILLMEM will generate inappropriate values.

- The text handling package TH uses ASCII.DEL as a terminating character; most STRING components, therefore, should be initialized with ASCII.DEL, not ASCII.NUL.
- Most of the FLT values in the graphic subsystem represent scale factors or ratios; these should be initialized as 1.0, not 0.0.
- Some components, such as DMTYPES.IND_HIER_TYPE.LEVEL, have range constraints that exclude 0. These components should obviously not be set to 0.
- As noted in Section 3.3, FILLMEM should not initialize access values or implementation-dependent components.

With all these problems, how did the ISS programmers get the FILLMEM initializations to work at all? The answer is that a lot of additional code has been added to correct the deficiencies of the FILLMEM calls. In ISS, a call to FILLMEM is frequently succeeded by a sequence of assignments to components of the object just initialized by FILLMEM, to give them reasonable values. We found one example where three-fourths of the components of a record had to be re-initialized after the FILLMEM initialization. Cases like this demonstrate rather conclusively that FILLMEM cannot relieve the programmer of the responsibility of providing a correct initial value for variable components.

Problems can occur if FILLMEM gives an improper value to a component, and the component is not re-initialized immediately. Such problems have been corrected by adding additional code to correctly interpret FILLMEM values; e.g. the test

```
(SCR.ASPECTRATIO = 0.0) or (SCR.ASPECTRATIO = 1.0)
```

is used in case SCR was initialized with FILLMEM, and code such as this

```
if (IBLK.LEVEL = 0) then
   IBLK.LEVEL := 1;
end if;
```

is used to bring an index into range.

There was another problem with FILLMEM that was corrected by ISS Version 3.0. If FILLMEM is applied to a variant record, all record variants will be zeroed out, since they are stored as record components. This undesirable result cannot be corrected easily by assignment. The problem was alleviated by Version 3.0, but not by the removal of FILLMEM. Instead, almost all variant record types were removed from ISS. This was a poor design decision. Record variants clarify the structure of a record type, and allow a much more efficient use of space.

There are two high-level alternatives to FILLMEM. An aggregate can be used to assign initial values, although admittedly some ISS types would require long aggregates. Another initialization technique would not require lengthy aggregates: Default expressions can be declared for record components. For example

```
type CAI_KEY_TYPE is
   record
      COURSE_NO      : A.I10B := 0;
      COURSE_VERSION : A.I5B  := 0;
      ...
   end record;
```

Component default expressions can be used to specify sensible default values for all objects of a type that are not explicitly initialized. This technique is rarely used in ISS.

In summary, the supposed convenience of FILLMEM is illusory. The use of FILLMEM has resulted in programs that are needlessly complicated, unreliable, and unportable. Portable alternatives to FILLMEM exist that perform its function more reliably, without any portability problems. For these reasons, FILLMEM should be phased out of ISS.

4.5 MOVEMEM AND ITS ALTERNATIVES

MOVEMEM has not caused as many problems as FILLMEM and FREEMEM. Nonetheless, it has introduced portability problems, and it can be replaced by high-level operations.

ISS generally uses MOVEMEM to perform one of two jobs:

- To assign one array slice to another, or
- To perform an unchecked conversion between two different data types.

Both of these jobs have obvious standard Ada replacements that are clearer, easier to use, and pose less portability problems. Array slices can be assigned by simply using a slice assignment statement. Unchecked conversions can be done using instantiations of UNCHECKED_CONVERSION.

MOVEMEM is another example of where the ISS programmers have done things the hard way. There is no good reason for its continued use.

4.6 THE COMPAREMEM AND SEARCH_MEM INTERFACE

The COMPAREMEM and SEARCH_MEM functions have caused the fewest problems of the low-level memory operations. This can attributed to the restricted usage of these functions in ISS. COMPAREMEM is always used to compare arrays of type UNSIGNED_BYTE_ARRAY, and SEARCH_MEM is always used to find either a CHARACTER or a STRING within a STRING.

The use of these functions suggests that they should not be implemented as low-level memory operations. A cleaner interface is provided by these proposed specifications:

```
function COMPAREMEM (LEFT, RIGHT : in UNSIGNED_BYTE_ARRAY)
   return NATURAL;

function SEARCH_MEM (BASE  : in STRING;
                     MATCH : in STRING := (1 => ASCII.DEL))
   return NATURAL;
```

These interfaces have several advantages over the current interfaces for these functions. Passing objects is easier and more natural than passing address-length pairs. None of the problems inherent to low-level operations can occur if the above interface is used.

Another advantage of the proposed interface is that it makes it very simple to write Ada subprogram bodies for these functions. This is not to say that these functions should necessarily be implemented in Ada; on a particular machine, there may be efficient assembly implementations that should be used. For this reason, these functions should remain part of the VML. The proposed specifications, however, can easily be interfaced with efficient code written in another language.

4.7 USE MORE PRE-DEFINED UNITS

As previously noted, ISS was originally written in another language (CAMIL), then automatically translated to a non-standard variant of Ada, then translated to Ada. Given this history, it is not surprising that ISS does not take full advantage of the pre-defined units available to the Ada programmer. Now that ISS is to be maintained in full standard Ada, the standard pre-defined units can be used to simplify the ISS code, and the effort required to rehost ISS.

- The VMIO subprograms FGETS and FPUTS supply a limited text I/O facility; their usage can be replaced with TEXT_IO operations.

- A variety of formats for time stamps are used in ISS. There is an obvious advantage to using one format; the type TIME defined in the CALENDAR package could be used as a standard.

- The package FIO supports direct file I/O; instantiations of DIRECT_IO can be used in place of FIO.

By using the pre-defined units, we can reduce the size of the ISS code, and give the code a look that would be more familiar to Ada programmers.

APPENDIX E: MICROISS FUNCTIONAL CAPABILITIES (Z-248 VERSION)

COURSEWARE DELIVERY
- Upper- and lowercase text
- Special characters
- Multiple colors
- Static, dynamic and interactive graphics
- Student interaction via keyboard or pointing device
- Initiation and presentation of CAI lessons
- Initiation and presentation of interactive videodisc sequences
- Initiation and presentation of simulations
- Initiation and presentation of mastery tests
- Review mode
- Course glossaries
- Student comments

COURSEWARE AUTHORING
- Menu-driven user interface
- Create, change, display, store and delete modules
- Insert, create, copy, reorder, store and delete segments
- Insert, create, copy, reorder, store and delete frames
  - Expository:
    - Information, Elaboration, Help, Title, Overview, Objective, Resource, Documentation
  - Interactive:
    - Touch, Multiple-Choice, True/False, Matching, Constructed Response (short answer)
  - Special Purpose:
    - Menu, Ada Programming Language, Simulation, Branch, Adjunct Material (Videodisc)
- Support for instructional strategies
  - Factual
  - Drill and practice
  - Tutorial
  - Simulation
  - Individualized tutorial
  - Problem solving
- Text development
- Individualization (branching)
  - Unconditionally
  - On number of frames presented/not presented
  - On number of questions answered correctly/incorrectly
  - On evaluation of author-supplied variable
    - System-defined
    - User-defined
- Overlay
- Partial screen erase/windowing
- Feedback and prompt creation
- Access, modify and display graphics created in the Graphics Editor
- Access simulations created in the Simulation Editor

- Glossary development
- Videodisc sequence development

GRAPHICS DEVELOPMENT
- Create, change, display, copy, store and delete graphics
- Keyboard, pointing device and bit pad/data pad user interface
- High-fidelity 2D graphics
- Static and dynamic
- Graphics primitives, selectable from menu
- Line drawings or filled objects, both regular and irregular-shaped
- Line color
- Colored fills
- Symbol library development
- Scale, rotate and repositioning
- Scaled text
- List existing graphics/symbol libraries
  - For a specific user
  - For all users
  - Starting with a specified name
  - Before or after a specified creation date

SIMULATION DEVELOPMENT
- Menu-driven user interface
- Create, change, display, copy, store and delete simulations
- Create, change, display, insert, copy, store and delete actions
- Create, display and delete objects (text or graphic)
- List actions/objects
- Branching logic (conditional/unconditional)
- Access, modify and display graphics created in the Graphics Editor
- Overlay
- Partial screen erase/windowing
- Feedback and prompt creation
- Expository actions
- Interactive actions
  - Touch, Multiple-Choice, True/False, Matching, Constructed Response (short answer)
- Create and insert complex author-defined equations
- Random number generation

TEST DEVELOPMENT
- Menu-driven user interface
- Create, change, copy, display, store and delete mastery test questions
- Block, lesson, group and mission tests
- Online/offline
- Item randomization
- Criterion-referenced
  - Percentage
  - Number of subscales (objectives) passed
- Five Question types
  - Multiple-choice, Touch, Matching, True/False, Constructed Response (short answer)

- Alternative weighing
- Definable scoring rules
  - Pass/fail by total test score
  - Pass/fail by objective
  - Critical items and objectives

TEST PRESENTATION
- Presentation of mastery test questions
- Review test items prior to scoring
- Score test items and report results
- Provide recap, detailing answer selected and correct answer

MICRO COMPUTER-MANAGED INSTRUCTION (CMI)
- Prerequisite course lists or courses selectable in any order
- Lesson lists containing either lineraly ordered lessons or lessons selectable in any order
- Reordering lessons within courses
- Embedded and mastery test items
- Registering of students into multiple courses
- Student rosters
  - Specification of number of lessons that must be taken in a course
  - Student names and IDs
  - Students' current lesson
- Student assignment generation
- Lesson override
- Certify pass
- Recording and tracking of student performance on a limited basis
  - Lessons passed
  - Performance on individual lesson questions
  - Lesson completion times

ADDITIONAL CAPABILITIES
- Dual screen
- System security (user access levels)

APPENDIX F: MICROISS EXECUTABLES

Executable	Source	Description
ACASS	CASS	CAI Authoring Support System (CASS) Editor - Allows users to create, change, delete, or copy lessons.
ADATEST	TESTED	Test Development Editor - Allows users to create, change, delete, or copy tests.
AGLOSS	CASS	Glossary Editor - Allows user to display, add, change, or delete glossary entries.
AMCAI	CASS	CAI Presentation Program - Displays the lessons authored by using ACASS.
AMSID	SID	Simulation Presentation - Presents simulations created by using ASID.
ASID	SID	Simulation Authoring - Editor for creating or changing simulations.
CONTROL	SE	Program Control - If the file MGRDIR:PROGRAM.NAM exists, this program 1) executes MGRDIR:LOADPGM.BAT to set up the virtual disk if necessary, 2) executes the program, and 3) executes MGRDIR:UNLDPGM.BAT to remove the .EXT file from the virtual disk. It will continue to loop until MGRDIR:PROGRAM.NAM does not exist.
CREATEDB	UTIL	Create Database - Creates the database files from definitions read from a file definition file (.FDF).
CREATEFD	UTIL	Create FDF - Creates the file definition file (.FDF) from the file attributes read from the database file ISSFILES.
CVDISK	(Alsys)	Contiguous VDISK - Insures that the heap file on the VDISK is contiguous. Execute after HEAP.
DISPUSER	ZZLOGON	Display Users - Used to display users currently logged onto ISS. Accessed by Pad-4 key from LOGON.
DRVCOM	DRIVERS	Communications Driver - An installable device driver which will drive ISS on the COM1 port. This should be installed (by adding a DEVICE=record in the CONFIG.SYS file) if the user wants to use a Tektronix 4105 terminal or videodisc.
EGADRV	DRIVERS	EGA Driver - An installable device driver which drives ISS on the Z248 EGA screen. It is installed by including a DEVICE=record in the CONFIG.DAT file.
FIXLOCK	UTIL	Fix Lock - Reads the lock table, checks for and displays hung locks, and allows the user to unlock them.

FRECOVER	UTIL	File Recovery - Used to recover lost data blocks in a database file.
FUTIL	UTIL	File Utilities - Utilities for managing the ISS database files.
GETBOOTI	SE	Get Boot Time - Captures the time ISS was booted and builds the ENVDIR:SETBOOTI.BAT file which is then executed to record the boot time in the environment string SYSBOOTTIME.
GRAFEDIT	GREDIT	Graphics Editor - Allows the user to create, modify, list, copy or delete graphics.
GRLIST	GREDIT	Graphics List - Displays a list of graphics and allows the user to select graphics to be deleted.
GRUPDCOM	GREDIT	Update Graphics Commands - Used for changing graphics editor menus and the commands associated with them.
GRVERIFY	GREDIT	Graphics Verify - Used to verify GRLIST deleted the selected graphics.
HEAP	(Alsys)	Creates the memory storage heap file ADA_HEAP.DTA using all available space remaining on the VDISK. Execute after copying the .EXT file to the VDISK.
INITPGM	ZZLOGON	Initialize PGMNAME - Creates the database file PGMNAME if it does not exist from information in AUSRPGM.
INITSHME	UTIL	Initialize Shared Memory - Initializes the shared memory file SHMEM.DAT by writing null blocks.
INITTERM	UTIL	Initialize Terminal - Initializes a particular ISS terminal/process entry. The user will be reprompted for terminal type the next time he/she runs a program.
ISSUSERS	UTIL	ISS Users - Displays the current users/processes of ISS.
LGCOMMAN	ZZLOGON	Logon Commands - Allows user to update the system banner, start/stop program activity, start/stop background processor, and other ISS system operations. Accessed by Pad-3 key from LOGON.
LISTLOG	ZZLOGON	List Log - Displays the ISS log file. Accessed by Pad-5 key from LOGON.
LOGON	ZZLOGON	ISS Logon - Controls entry into ISS.

MGCAI	CASS	CAI Data Mover (Get) - Reads a system file created by MPCAI containing lessons and writes the contents to the database CAI files ACAICAI, ACAIOBJ, ACAIBRN, ACAIVAR, ACAIEXP, ACAIALT, and ACAITXT. Use TMGCAI if the transfer file was made from a pre-version 4 database and you wish to move the lessons into a version 4.x database. Otherwise use this program. If unsure which to use, try MGCAI first.
MGCRS	MCMI	MicroCMI Course Mover (Get) - Reads a system file created by MPCRS containing MicroCMI course data records and writes the contents to the database MicroCMI course data file MCMICRS.
MGGRMENU	GREDIT	Graphics Menus Mover (Get) - Reads a text file created by MPGRMENU containing graphic editor menus and writes the contents to the database graphics menu file GRMENUS.
MGSID	SID	SID Data Mover (Get) - Reads a system file created by MPSID containing simulations and writes the contents to the database simulation files ASIMSIM, ASIMOBJ, ASIMVAR, ASIMTXT, ASIMALT, ASIMEV, ASIMEXP, and ASIMSTR.
MGSTU	MCMI	MicroCMI Student Mover (Get) - Reads a system file created by MPSTU and writes the contents to the database MicroCMI student files MCMISTU and MCMISDP.
MGTEST	TESTED	Test Data Mover (Get) - Reads a system file created by MPTEST and writes the contents to the database test files VTKF, WTKF, WITM, PXWITM, WITMPTR, and WITMALT. Use TMGTEST if the transfer file was made from a pre-version 4 database and you wish to move the tests into a version 4.x database. Otherwise, use program MGTEST. If unsure which to use, try MGTEST first.
MICROMGR	MCMI	MicroCMI Management Editor - Used to create a course, define its structure, and enroll students in it.
MPCAI	CASS	CAI Data Mover (Put) - Creates a system file containing lesson contents from the database CAI files ACAICAI, ACAIOBJ, ACAIBRN, ACAIVAR, ACAIEXP, ACAIALT, and ACAITXT.
MPCRS	MCMI	MicroCMI Course Mover (Put) - Creates a system file containing course data records from the MicroCMI course data file MCMICRS.
MPGRMENU	GREDIT	Graphics Menus Mover (Put) - Creates a text file containing graphics editor menus from the database graphics menu file GRMENUS.

MPSID	SID	SID Data Mover (Put) - Creates a system file containing simulations from the database simulation files ASIMSIM, ASIMOBJ, ASIMVAR, ASIMTXT, ASIMALT, ASIMEV, ASIMEXP, and ASIMSTR.
MPSTU	MCMI	MicroCMI Student Mover (Put) - Creates a system file containing student records from the database MicroCMI student files MCMISTU and MCMISDP.
MPTEST	TESTED	Test Data Mover (Put) - Creates a system file containing test information from the database test files VTKF, WTKF, WITM, PXWITM, WITMPTR, and WITMALT.
MVGGRAF	GREDIT	Graphics Data Mover (Get) - Reads a system file created by MVPGRAF containing graphics and writes them to the database graphics file ADAGRAPH.
MVPGRAF	GREDIT	Graphics Data Mover (Put) - Creates a system file containing graphics from the database graphics file ADAGRAPH.
PRFDAT	CASS	Student Test Response Report - Puts out a performance report based on students' responses for a given course.
SCEDITOR	GREDIT	Stroked Character Editor - Allows the user to edit a stroked character set.
SCRMOVER	UTIL	Script Mover Utility - Converts script definition files (.SDL) into script files (.SCR) or decodes script files (.SCR) into script definition files (.SDL).
SEQUENCE	MCMI	MicroCMI Lesson Sequencer - Used by the student to get his/her lesson assignments from the course(s) he/she is registered in, and to manage his/her progress throughout the course.
SETCASSR	CASS	Set CASS References - Sets the CASS reference count for all graphics in every lesson to the number of times they are referenced. Used after ZEROALLC.
SETSIDRE	SID	Set SID References - Sets the SID reference count for all graphics in every lesson to the number of times they are referenced. Used after ZEROALLS.
SET_LOGT	SE	Set Log Time - Captures the time of the beginning of an ISS session and records it in the file MGRDIR:LOGTIME.DAT to be used by other processes in the session.
SIDREP	SID	SID Reports - Runs in the background and processes and prints out SID reports.
SIREPREQ	SID	SID Reports Requests - Used to submit requests to SIDREP to print SID reports.

TDEFBLD	UTIL	Terminal Definition Build - Reads the .TDL, .DDL, and .NAM files and stores the device definitions in the terminal definition file (TDEFILE).
TESTPRES	TESTED	Test Presentation - Presents a test to its author or to a student. Statistics are also taken if requested by the author within ADATEST.
TESTPRT	TESTED	Test Print - Generates a report detailing the test information for a selected test.
TMGCAI	CASS	CAI Data Mover (Get) - Reads a system file created by MPCAI containing lessons and writes the contents to the CAI database files ACAICAI, ACAIOBJ, ACAIBRN, ACAIVAR, ACAIEXP, ACAIALT, and ACAITXT. Use TMGCAI if the transfer file was made from a pre-version 4 database and you wish to move the lessons into a version 4.x database. Otherwise use program MGCAI. If unsure which to use, try MGCAI first.
TMGTEST	TESTED	Test Data Mover (Get) - Reads a system file created by MPTEST and writes the contents to the database test files VTKF, WTKF, WITM, PXWITM, WITMPTR, and WITMALT. Use TMGTEST if the transfer file was made from a pre-version 4 database and you wish to move the tests into a version 4.x database. Otherwise, use program MGTEST. If unsure which to use, try MGTEST first.
VDISK	(DOS)	Virtual DISK - A DOS installable device driver used to create the virtual disk that is required for the .EXT and ADA_HEAP.DTA files. It is installed by including a DEVICE=VDISK.SYS record in the CONFIG.SYS file.
ZALLCSSS	SID	Zero All CASS and SID References - Zeros out the CASS and SID reference counts for every graphic in the database.
ZEROALLC	CASS	Zero All CASS References - Zeros out the CASS references to all graphics.
ZEROALLS	SID	Zero All SID References - Zeros out the SID references to all graphics.
ZEROCASS	CASS	Zero CASS References - Zeros out the CASS references for a given graphic.
ZEROSIDR	SID	Zero SID References - Zeros out the SID references for a given graphic.
ZZUSRED	ZZLOGON	User Editor - Used to define user access to ISS and establish permissions for those users.

APPENDIX G: UPGRADING THE ISS TESTBED (Z-248 VERSION)

ABSTRACT

This is a report of the results of the three proposed upgrades to the Instructional Support System (ISS). The purpose of the three proposed upgrades was to make ISS a more widely accessible testbed computer-based instructional delivery system, especially for the advanced instructional design systems now being developed by AFHRL/IDC. The three proposed upgrades were: 1) convert to the Meridian compiler to eliminate a potential licensing fee for each run-time version of ISS, 2) eliminate the need for a math co-processor in an attempt to broaden the potential base of testbed developers, and 3) eliminate the Tektronix terminal emulation as it was not needed in the Z-248 environment.

Deliverables were to consist of program modifications or reports and analyses of obstacles. While some modifications were made to ISS Version 2.2 as part of this effort, those modifications did not result in a completely modified working version of ISS. Therefore, the deliverables associated with this contract are in the form of this final report. The conversion to Version 2.01 of the Meridian compiler was not possible. Since the conversion to an alternate compiler was the basic effort, two other compilers were tried: Meridian AdaVantage Version 3.0 (released during the period of this effort) and AETECH's IntegrAda Version 4.0.1. This report represents an in-depth analysis of each of these Ada compilers and the obstacles involved in re-hosting ISS. The need for a math co-processor can be eliminated using any compiler; this report will indicate how this can be accomplished and why doing so is not desirable. It was decided that the emulation of the Tektronix terminal should be retained, given the inability to convert to another compiler, the current lack of interest in ISS as a testbed delivery system, and the need to provide support for IVD.

While these results might seem somewhat negative, the information about the other compilers and the specific nature of the ISS code is important enough to justify recommending that this report be made an appendix to the existing ISS documentation.

1.0 Introduction

The unsolicited proposal upon which this work is based was originally submitted to AFHRL on September 26, 1988. However, due to funding delays, the original timeframe for the work was adjusted to begin on May 4, 1989, with the deliverables due within 6 months of contract funding.

During the period of the delay of this project, several operating assumptions changed. First, as the Advanced Instructional Design Advisor project had progressed in AFHRL/IDC, it was becoming questionable as to whether ISS would be used as a testbed delivery system. The primary obstacle to doing so was the cost of software interfaces to ISS. A secondary concern was that using ISS to test next-generation computer-based authoring concepts and environments would bias the outcome of those tests in a negative manner.

Second, it was determined that AFHRL/IDC would no longer be the support agency for ISS. This meant that it was very unlikely that new ISS users would emerge. As a consequence, there was no longer a problem with run-time licensing.

Third, a new version of the Z-248 version of ISS was released. This new version was not evaluated for the three potential upgrades. However, a cursory review of ISS Version 4.29 indicates that the issues are basically the same.

Last, a new version of the Meridian AdaVantage compiler was released. Because the conversion to Meridian AdaVantage 2.0 was not possible, converting to Meridian AdaVantage 3.0 was explored. As this conversion also met obstacles and another vendor's compiler was validated in the delay period, yet a third conversion was attempted to AETECH's IntegrAda 4.0.1. This conversion also proved impossible.

2.0 Converting ISS to the Meridian AdaVantage Compiler

The initial conversion attempt was made using Meridian's AdaVantage Version 2.0 compiler. Several obstacles were immediately encountered. In reviewing the compiler documentation, it became obvious that this version could not accommodate ISS due to two source code size restrictions: (a) Programs were restricted to 10 user-defined packages, and (b) compilation units were restricted to 200 statements and individual data objects were restricted to 64 kilobytes. ISS contained many more than 10 packages and many compilation units were quite large. Meridian did have a developer's version of AdaVantage that was not restricted; so, re-compilation was attempted anyway to discover any other problem areas.

Two additional obstacles were encountered, both involving the fact that certain PRAGMAs and ATTRIBUTEs were not implemented in the Meridian compiler. PRAGMAs are basically directives to the compiler that can be embedded in Ada source code units. They allow programmers to take advantage of various implementation-dependent features of a compiler, such as packing text information to save space and setting priority levels in a multi-tasking environment. PRAGMAs are part of the standard Ada programming environment. To be validated, compilers must at least recognize the PRAGMA statement, regardless of whether the PRAGMA happens to be implemented. Although extensive use of PRAGMAs is regarded by some academic computer scientists as violating the generality and portability intended to be part of Ada programs, this complaint is valid only in the situation where vendors are required to recognize the PRAGMAs but not implement them. As requirements for Ada compilers become more strict and all PRAGMAs must be implemented, this criticism will vanish. However, it is exactly appropriate with regard to ISS and the several compilers tested. Only the Alsys compiler, which was used to develop ISS on the Zenith Z-248, implemented all of the PRAGMAs in the ISS code. The

particular PRAGMA that was not recognized by Meridian AdaVantage 2.0 was CONTROLLED. This PRAGMA is used to restrict the storage reclamation for de-referenced access types.

Also, Meridian AdaVantage 2.0 had not implemented certain representation ATTRIBUTEs, which were used in the Alsys version of ISS. ATTRIBUTEs, like PRAGMAs, are part of the pre-defined and/or implementation-defined Ada language environment. ATTRIBUTEs can be used in the midst of executable Ada code and operate much like a function, generally returning a single value when evaluated. For example, the programmer may want to know whether a particular Ada implementation rounds or truncates when performing an arithmetic operation in order to select a particular path in the code; in this case, the ATTRIBUTE name MACHINE_ROUNDS can be referenced and will return a value of TRUE or FALSE, as appropriate. The particular ATTRIBUTE used by ISS that was not implemented in AdaVantage 2.0 was the SIZE attribute, which returns the number of bits in memory allocated to a particular object.

In addition to these problems, the Alsys compiler had added some entities to the pre-defined package SYSTEM which were not recognized by other compilers. Specifically, the identifier name SYSTEM.NULL_ADDRESS is recognized by Alsys but not other Ada compilers. A certain portion of package SYSTEM must be exactly as specified in ANSI-MIL-STD-1815-A; other system-dependent declarations are allowed. Apparently, NULL_ADDRESS falls into this latter category. Again, the purists would argue against such variation in the so-called standard Ada language environment, but the fact is that this variation is in comformity with existing AJPO (Ada Joint Program Office) requirements. The existence of such variations has allowed non-portable Ada systems to be developed. ISS is one such example. References to both SYSTEM, NULL_ADDRESS and the SIZE ATTRIBUTE can be found in ISS\SE\A.ADS, which is included in Appendix A. (See page 105.)

In an attempt to make the conversion to another Ada compiler, this contractor acquired a new release of AdaVantage (Version 3.0), which did in fact overcome some of the size restrictions as well as the difficulty with ATTRIBUTE SIZE. In addition, AdaVantage 3.0 has an extended mode version which allows it to accommodate the extended memory requirements of ISS on the Z-248. Using the extended mode version, up to 16-megabyte MS-DOS programs can be accommodated. This is more than sufficient to accommodate ISS. The extended mode version comes with an extended mode kernel to relieve developers of writing their own non-standard extended operating system kernels. Compilation units and individual data objects (such as arrays) are still restricted to 64 kilobytes; so, some additional modularization of the ISS source code would have been required, had conversion been possible. It should be noted that this kind of modularization is highly desirable anyway, and it should have been part of the original ISS code.

Meridian AdaVantage 3.0 also has several other standard features which are relevant to ISS conversion. Specifically, there are provisions for interfaces to C and Assembly Language routines. ISS contains several Assembly Language driver routines; so, an interface to Assembly Language routines is necessary and is available in Meridian AdaVantage Version 3.0. This compiler no longer requires the use of a math co-processor. If the system contains a math co-processor, it will be utilized for floating point operations; if no math co-processor is found, then floating point operations are emulated in the software, transparent to the programmer. Had conversion been possible, this feature would have resolved the second upgrade issue.

Conversion to Meridian AdaVantage 3.0 was not possible, however. Again PRAGMA CONTROLLED was not implemented. Rewriting ISS without reference to this PRAGMA is probably desirable in the interest of portability, but it requires a re-thinking of all ISS storage management routines and references to this PRAGMA. Such an effort was beyond the scope of this contract.

Because the conversion to an alternate compiler without restrictive run-time licensing requirements was fundamental to this project, a third compiler that was released during the period of this contract was acquired and evaluated. AETECH's IntegrAda Version 4.0.1, validated under ACVC 1.10 as was AdaVantage, also proved unable to handle ISS without substantial revision of the existing ISS code. For this compiler, the maximum code size per compilation unit was 32 kilobytes (half the Meridian restriction). The maximum source code size allowed with IntegrAda is 144 kilobytes. ISS contains at least one compilation unit that is much larger than this limit; ISS\SE\APPLIB\CAILIB\CASS\MCMI\MGR_SEGM.ADB is about 155 kilobytes in size. In addition, a maximum of 300 compilation units and 80 WITHs were allowed per program -- these restrictions would require further modularization of ISS. There were 318 compilation units in Version 2.2 of ISS, one containing 11 WITHs.

Another restriction was that IntegrAda had not implemented the INTERFACE PRAGMA, which is used extensively to define the interface to the Assembly Language driver routines in ISS. IntegrAda has its own Ada Assembler, so that this difficulty could probably be overcome. Also, IntegrAda had not implemented the CONTROLLED PRAGMA; so, the same basic problem existed with regard to this compiler that was found to exist with AdaVantage. Another potential problem was that IntegrAda performed no form of automatic storage reclamation or garbage collection. The non-standard reference to NULL_ADDRESS in package SYSTEM was also foreign to IntegrAda. It should be added, however, that IntegrAda, like AdaVantage, does floating point emulation in the absence of a math co-processor.

An aside about the comparative worth of these two compilers is perhaps appropriate at this point. It is well known that the Alsys compiler for MS-DOS machines is highly regarded. These two compilers are now offering serious competition, which eventually might reduce the cost and restrictive licensing requirements of the Alsys compiler. IntegrAda includes an integrated package of Ada development tools in addition to the compiler. IntegrAda is a serious attempt to provide an APSE (Ada Programming Standard Environment). While this effort is to be commended, it should be noted that the documentation supplied with IntegrAda needs improvement. This, coupled with the more significant size restrictions, make this compiler less desirable for ISS than AdaVantage. Meridian's AdaVantage Version 3.0 also provides a full range of development tools, plus the extended mode features. Among the development tools is a complete text handling package, which would eliminate the need for the package\ISS\SE\TH.ADS. AdaVantage is simple to install. It comes with excellent documentation, and in this reviewer's opinion, it comes the closest to competing with the recognized leader in PC Ada compilers, Alsys, Inc. The primary area in which IntegrAda might have a clear advantage is that it provides a complete MOUSE driver package. Because ISS has recently had a MOUSE driver capability added, this is of no benefit in the current ISS environment.

3.0 Eliminating the Need for a Math Co-Processor in ISS

The second upgrade issue involved the ISS requirement for a math co-processor. It should be noted that both the ISS code and the Alsys Ada compiler require floating point capability. Floating point numbers are in general the real numbers, such as 3.141592865, 1/3, 1.555e+12, etc. Alsys does not perform floating point emulation; thus, a math co-processor is required to handle arithmetic operations on data types of pre-defined type FLOAT and of any user-defined types involving subsets of FLOAT or specified floating point types using the DIGITS declaration. Had conversion to either AdaVantage or IntegrAda been possible, the requirement for a math co-processor would have vanished, since both systems emulate floating point operations in the absence of a math co-processor.

The only way to eliminate the need for a math co-processor, short of re-hosting ISS on a compiler that performs floating point emulation, is to modify all references to floating point types

to fixed point types. Ada is one of the very few languages that allows the programmer to define fixed point types, which are those real numbers with a restricted and defined interval between model (machine-representable) numbers. ISS contains a package specification named\ISS\SE\A.ADS wherein ISS's floating point type FLT is defined. This package is included in this report as Appendix A. (See page 105.) ISS defines FLT as DIGITS 9--that is, floating point numbers with 9-digit accuracy. To change this definition of FLT to a fixed point type, the following line could be substituted: TYPE FLT IS DELTA 0.001 RANGE -1.0e6 .. +1.0e6. This indicates that only 3-digit accuracy to the right of the decimal point is required and the range of numbers allowed falls within a 16-bit representation scheme, which is required for systems without a math co-processor or floating point emulation scheme. This approach was in fact tried with AdaVantage 2.0 and worked well in systems without math co-processors.

Should there still be an interest in eliminating the math co-processor requirement, then the above solution can be implemented. There has been no evaluation of the entire ISS system to determine whether or not it makes sense to restrict real numbers to 3-digit accuracy within the indicated range. With regard to grade management, this restriction is certainly acceptable. However, math co-processors have become even more affordable (about $300), and all current users already have math co-processors. Because no new users are expected, it no longer makes sense to perform this modification. Should AFHRL/IDC decide to use ISS as a testbed delivery system, it already possesses the correct hardware to perform Alsys floating point operations.

As a final note with regard to fixed point types, the lack of a user-defined fixed point type in package\ISS\SE\A.ADS to parallel the floating point definition of FLT probably represents a serious planning flaw. A review of this package reveals several user-defined integer types in addition to the user-defined floating point type; future ISS versions might well use fixed point types.

4.0 Eliminating the Tektronix Terminal Emulation in ISS

The third proposed upgrade involved eliminating the need to emulate the Tektronix 4105 and 4107 terminals on Z-248 EGA systems. The ISS source code associated with terminal emulation is primarily in the form of Assembly Language drivers named\ISS\SE\EGADRV.ASM and \ISS\SE\DRVCOM.ASM. When the executable version of ISS is installed, it is possible to select EGADRV.SYS if the system contains an EGA terminal or DRVCOM.SYS if the system contains either a Tektronix terminal or an IVD terminal.

It is theoretically possible to eliminate the terminal emulation altogether in the Z-248 system. However, doing so would then introduce a new requirement to re-write the driver for an IVD terminal. Although it is not likely that AFHRL/IDC will have need of Tektronix terminal emulation in a testbed delivery system, it is highly likely that IVD capability would be required in a testbed delivery system, as IVD capability is a hallmark of state-of-the-science authoring systems. Because most of the terminal emulation software has been written at the Assembly Language level and is working well, and because there is still a potential need to support IVD, it is not recommended that any of this software be changed.

5.0 Conclusions

None of the three proposed upgrades was implemented. The second can be implemented at any time per the directions provided in section 3.0. Conversion to the Meridian AdaVantage Version 3.0 compiler is possible if it could be determined how ISS is using the Alsys compiler identifier SYSTEM.NULL_ADDRESS and how critical storage management is being handled in ISS. Making these determinations would require interviewing the appropriate Mei Associates

software engineers, an effort that was well beyond the budget and scope of this project. If the conversion to another compiler is ever performed, it is recommended that Meridian's AdaVantage Version 3.0 be given first consideration; the second upgrade would occur automatically at that time. Eliminating the terminal emulation software is completely unwise, because doing so would require a substantial Assembly Language effort to provide some support of IVD, which is now supplied by way of the Tektronix emulation software.

Appendix A: ISS\SE\A.ADS

```
with SYSTEM;
use  SYSTEM;

package A is

   subtype INT is INTEGER;
   type FLT is digits 9; --fjl
--
-- The following is the suggested FIXED POINT DEFINITION:
--
-- type FXT is delta 0.001 range -1.0e6 .. +1.0e6;  -- jms 9/5/89
--
   subtype ADDRESS is SYSTEM.ADDRESS;

   subtype  I1B is A.INT range 0 .. (2 **  1) - 1;
   subtype  I2B is A.INT range 0 .. (2 **  2) - 1;
   subtype  I3B is A.INT range 0 .. (2 **  3) - 1;
   subtype  I4B is A.INT range 0 .. (2 **  4) - 1;
   subtype  I5B is A.INT range 0 .. (2 **  5) - 1;
   subtype  I6B is A.INT range 0 .. (2 **  6) - 1;
   subtype  I7B is A.INT range 0 .. (2 **  7) - 1;
   subtype  I8B is A.INT range 0 .. (2 **  8) - 1;
   subtype  I9B is A.INT range 0 .. (2 **  9) - 1;
   subtype I10B is A.INT range 0 .. (2 ** 10) - 1;
   subtype I11B is A.INT range 0 .. (2 ** 11) - 1;
   subtype I12B is A.INT range 0 .. (2 ** 12) - 1;
   subtype I13B is A.INT range 0 .. (2 ** 13) - 1;
   subtype I14B is A.INT range 0 .. (2 ** 14) - 1;
   subtype I15B is A.INT range 0 .. (2 ** 15) - 1;
   subtype I16B is LONG_INTEGER range 0 .. (2 ** 16) - 1; --fjl
   subtype I17B is LONG_INTEGER range 0 .. (2 ** 17) - 1; --fjl
   subtype I18B is LONG_INTEGER range 0 .. (2 ** 18) - 1; --fjl
   subtype I19B is LONG_INTEGER range 0 .. (2 ** 19) - 1; --fjl
   subtype I20B is LONG_INTEGER range 0 .. (2 ** 20) - 1; --fjl
   subtype I21B is LONG_INTEGER range 0 .. (2 ** 21) - 1; --fjl
   subtype I22B is LONG_INTEGER range 0 .. (2 ** 22) - 1; --fjl
   subtype I23B is LONG_INTEGER range 0 .. (2 ** 23) - 1; --fjl
   subtype I24B is LONG_INTEGER range 0 .. (2 ** 24) - 1; --fjl
   subtype I25B is LONG_INTEGER range 0 .. (2 ** 25) - 1; --fjl
   subtype I26B is LONG_INTEGER range 0 .. (2 ** 26) - 1; --fjl
   subtype I27B is LONG_INTEGER range 0 .. (2 ** 27) - 1; --fjl
   subtype I28B is LONG_INTEGER range 0 .. (2 ** 28) - 1; --fjl
   subtype I29B is LONG_INTEGER range 0 .. (2 ** 29) - 1; --fjl
   subtype I30B is LONG_INTEGER range 0 .. (2 ** 30) - 1; --fjl
   subtype I31B is LONG_INTEGER range 0 .. (2 ** 31) - 1; --fjl

   subtype S32B is LONG_INTEGER; --fjl

   --fjl  subtype UNSIGNED_BYTE is INT range 0 .. (2 ** 8) - 1;
```

```
   type UNSIGNED_BYTE is range 0 .. (2 ** STORAGE_UNIT) - 1;
   for UNSIGNED_BYTE'SIZE use STORAGE_UNIT;

   ----------------------
   type  UNSIGNED_BYTE_ARRAY  is  array  (INT  range  <>)  of
UNSIGNED_BYTE;
   pragma PACK (UNSIGNED_BYTE_ARRAY);

   subtype WORD is INT range - (2 ** 15) .. (2 ** 15) - 1;

   ----------------------
   type WORD_ARRAY is array (INT range <>) of WORD;
   pragma PACK (WORD_ARRAY);


   --fjl ADDRESS_ZERO : CONSTANT ADDRESS := SYSTEM.ADDRESS_ZERO;
  ADDRESS_ZERO : SYSTEM.ADDRESS renames SYSTEM.NULL_ADDRESS;--fjl

   PRESENT      : constant BOOLEAN := FALSE;
   OMITTED      : constant BOOLEAN := TRUE;

end A;
```

Appendix B: General Purpose Search Routine

What follows is a general purpose search routine to search the entire set of source code files that comprise ISS Version 2.2 for specified input words (all of the compilers mentioned in this report with the exception of AdaVantage 2.0 have utilities to perform this function):

```
with text_io; use text_io;
procedure project is

--   Author     : Michael Spector
--   Date       : May 30, 1989

--   This Ada program searches the ISS files in the external file
--   ISS.DIR for the source code lines that contain the
--   specified input and then prints those lines to an external
--   file named PROJECT.DAT.

subtype file1 is string(1..80);
subtype file2 is string(1..14);
subtype file3 is string(1..10);

n1          : file1;       -- Ada files used from external file
n2          : file2 := "b:\project.dat"; -- output file
n3          : file3 := "b:\iss.dir"; -- ISS file names
last        : natural;                  -- length of line
line        : string (1..100);             -- line being searched
f1, f2, f3  : file_type;                  -- files used internally
found       : boolean;                    -- true or false
total_files,                              -- total files processed
lines_found,                             -- in each file
total_lines,                              -- line number
total_lines_found : integer;             -- in all files
key_word    : string (1..100);            -- item to search for
length      : positive;                   -- length of key_word
blanks100 : string (1..100) := (others => ' '); -- 100 blanks
blanks80  : string (1..80) := (others => ' ');  --  80 blanks

package my_int_io is new integer_io (natural);
use my_int_io;



-------------- This function searches for the KEY_WORD.

   function search (line : in string;
                    last : in natural) return boolean is
```

```
      result : boolean := false;
      match  : boolean;

   begin

      for i in 1 .. last loop

         exit when result = true;
         if line (i) = key_word (1) then
             match := true;
             for letter in 2..length loop
                 exit when match = false;
                 if line (i+letter-1) /= key_word (letter) then
                    match := false;
                 end if;
             end loop;
             result := match;
          end if;

      end loop;

      return result;
   end search;

--

begin     ------- MAIN PROGRAM -------

--------------  get external file iss.dir for
--------------  ISS files to be searched

open (f3, in_file, n3);
create (f2, out_file, n2);
total_files := 0;
total_lines_found := 0;
--
new_line (5);
put (" Enter the word that you wish to search for in ISS: ");
get_line (key_word, length);
put (key_word(1..length)); put ("  "); put (length);
new_line (2);
put_line (" Search beginning -- output goes to PROJECT.DAT ");
new_line (2);
--
put_line (f2, "-------------------------------------------------
-                -----------");
put_line(f2, " ");
put_line(f2, " ");
put(f2, "--- Searching for lines of code with specified input ");
put_line (f2, key_word(1..length));
put_line (f2, " ");
```

```
put_line (f2, " ");
put_line (f2, "-------------------------------------------------
-                ----------");

while not end_of_file(f3) loop

 begin

   n1 := blanks80;
   get_line (f3,n1,last);
   if (n1(1..last)) = n1 then
      put_line(n1);
   end if;
   total_files := total_files + 1;
   put (total_files);
   put_line (" files processed");
   open (f1, in_file, n1);

--------------    headings for each file printed in
--------------    external file PROJECT.DAT

   put_line(f2,"_____________________________________________________");
   put_line(f2, " ");
   put (f2, "searching file  -  ");
   put_line (f2, n1(1..last));
   put_line (f2, " ");
   total_lines := 0;
   lines_found := 0;

--------------   process each line of the file and print only the
--------------   lines containing the specified input in the
--------------   output file named PROJECT.DAT.

   while not end_of_file (f1) loop

      line := blanks100;
      get_line (f1,line,last);
      total_lines := total_lines + 1;
      if search (line,last) then
         put_line (f2,line(1..last));
         lines_found := lines_found + 1;
      end if;
      line := blanks100;
   end loop;  -- file f1 loop
   close (f1);
   total_lines_found := total_lines_found + lines_found;

   if lines_found = 0 then
      put_line (f2, "  --  None found  --");
   end if;
```

```
--    put_line (f2, " ");
--    put (f2, "  There are");
--    put (f2, total_lines);
--    put_line (f2, " lines in this file.");
--
-- exception handler for simple problems with input file
--
  exception
     when others =>
        close (f1);
        put_line (" PROBLEM WITH INPUT FILE ");
 end;

end loop;  --  file f3 loop

-- put_line (f2, " ");
-- put (f2, "There were");
-- put (f2, total_files);
-- put_line (f2, " files processed.");
-- put_line (f2, " ");
-- put (f2, "There were");
-- put (f2, total_lines_found);
-- put_line (f2," lines of code found containing the specified
input");
-- put_line (f2, "in all the files processed.");

   close (f2);
   close (f3);

--
-- more elaborate exception handler for debugging purposes
--
exception

   when name_error | use_error =>
     if is_open (f1) then
      close (f1);
   end if;
   if is_open (f2) then
      close (f2);
   end if;
   put_line ("** error copying " & n1 & " to " & n2);

when mode_error =>
   put_line ( " mode error ");
   put_line (line);

when status_error =>
   put_line ( " status error ");
   put_line (line);
```

```
when device_error =>
   put_line ( "device error ");
   put_line (line);

when end_error =>
   put_line ( "end error ");
   put_line (line);

when data_error =>
   put_line ( "data error " );
   put_line (line);

when layout_error =>
   put_line ( " layout error ");
   put_line (line);

end project;
```

APPENDIX H: LISTING OF ISS SOFTWARE AND DOCUMENTATION AVAILABLE THROUGH THE NATIONAL TECHNICAL INFORMATION SERVICE (NTIS)

The following ISS software and documentation are available through the NTIS Sales Department at (703) 487-4650.

DOCUMENT TITLE	NTIS ORDER NUMBER
COMPLETE SOFTWARE AND DOCUMENTATION:	
1. ISS Vax Version (Contains 3 tapes and 47 documents)	PB90-501529
2. MicroISS Zenith 248 Version (Contains 30 diskettes and 17 documents)	PB90-501537
SYSTEM DOCUMENTATION:	
1. ISS Vax Installation Package (Vax Only)	PB90-177338
2. MicroISS Installation Package (Zenith 248 Only)	PB90-177155
3. Functional Description Package (Both Vax and Zenith 248)	PB90-177197
4. Computer Program Product Specification (Vax Only)	PB90-177320
5. Data Base Design Document (Vax Only)	PB90-177361
6. Software Detailed Design Document (Vax Only)	PB90-177627
7. Terminal Emulator Installation and Use Instructions (Both Vax and Zenith 248) (For use with video disc capability)	PB90-177205
CAI REFERENCE MANUALS:	
1. CAI Authoring Support System (CASS) (Both Vax and Zenith 248)	PB90-177247
2. Glossary Editor (GlosEdt) (Both Vax and Zenith 248)	PB90-177239
3. Graphics Editor (GrEdt) (Both Vax and	PB90-177270

Zenith 248)

4.	Simulation Dialogue Editor (SID) (Both Vax and Zenith 248)	PB90-177304
5.	Video Disc (Both Vax and Zenith 248) (Operates as a CASS capability)	PB90-177312

CAI USER'S MANUALS:

1.	MicroISS User's Manual for the CAI Authoring Support System (CASS) (Zenith 248 Only)	PB90-177163
2.	MicroISS Beginner's Manual (Zenith 248 Only)	In Development

CAI TRAINING MANUALS:

1.	CAI Authoring Support System (CASS) Training Manual and Appendices (Both Vax and Zenith 248)*	PB90-177189
2.	Graphics Editor (GrEdt) Training Projects (Both Vax and Zenith 248)*	PB90-177171

* Accompanies online training databases provided with the software

MICROCMI AND CMI REFERENCE MANUALS:

1.	MicroCMI (Instructor) (Both Vax and Zenith 248)	PB90-177262
2.	MicroCMI (Student) (Both Vax and Zenith 248)	PB90-177254
3.	Test Editor (TestEd) (Both Vax and Zenith 248)	PB90-177288
4.	User Editor (UserEd) (Both Vax and Zenith 248)	PB90-177296
5.	CAI Reports (CAIRep) (Vax Only)	PB90-177510
6.	Curriculum Definition Editor (CDE) (Vax Only)	PB90-177544
7.	Course Evaluation Summary (CES) (Vax Only)	PB90-177528
8.	Course Structure Editor (CSE) (Vax Only)	PB90-177536

9.	Data Extraction Program (DEP) (Vax Only)	PB90-177551
10.	ISS Administrative Management Editor (FORMS) (Vax Only)	PB90-177569
11.	Instructor-Managed Resources Editor (IMR) (Vax Only)	PB90-177577
12.	Instructor Monitor (InstMON) (Vax Only)	PB90-181496
13.	Lesson Definition Editor (LDE) (Vax Only)	PB90-177619
14.	Resource Availability Editor (ResAvail) (Vax Only)	PB90-177601
15.	Student Logon Editor (StuLog) (Vax Only)	PB90-177593
16.	Student Registration Editor (StuReg) (Vax Only)	PB90-177585
17.	Test Item Evaluation (TIE) (Vax Only)	PB90-177502

MICROCMI AND CMI USER'S MANUALS:

1.	Test Editor (TestEd) (Both Vax and Zenith 248)	PB90-177221
2.	User Editor (UserEd) (Both Vax and Zenith 248)	PB90-177213
3.	Course File Editor (Vax Only)	PB90-177486
4.	Course Structure Editor (CSE) (Vax Only)	PB90-177478
5.	Curriculum Definition Editor (CDE) (Vax Only)	PB90-177460
6.	ISS Administrative Management Editor (FORMS) (Vax Only)	PB90-177494
7.	ISS Message Utility (MAIL) (Vax Only)	PB90-177411
8.	Hierarchy File Editor (Vax Only)	PB90-177452
9.	Instructor Monitoring (InstMON) (Vax Only)	PB90-177445

10.	Learning Center Editor (Vax Only)	PB90-177437
11.	Lesson Definition Editor (LDE) (Vax Only)	PB90-177429
12.	Permanent File Editor (Perm) (Vax Only)	PB90-177379
13.	Student Data Profile Editor (SDP) (Vax Only)	PB90-177395
14.	Student Logon (StuLog) (Vax Only)	PB90-177387
15.	Student Registration (StuReg) (Vax Only)	PB90-177403
16.	Variable Definition File Editor (VDF) (Vax Only)	PB90-177346
17.	ISS CMI Error Messages (Vax Only)	PB90-177353

* U. S. GOVERNMENT PRINTING OFFICE: 1990--761-051/20114