2

NAVAL POSTGRADUATE SCHOOL
Monterey, California

UNITED STATES NAVY · NAVAL POSTGRADUATE SCHOOL ·

THESIS

A Functional Bar Code Inventory System
for
Marine Corps Systems Command
by

Richard M. Hancock

September, 1993

Thesis Advisor:	William Haga
Co-Advisor:	Shu S. Liao

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE Sep 1993	3. REPORT TYPE AND DATES COVERED Master's Thesis, Final

4. TITLE AND SUBTITLE A Functional Bar Code Inventory System for Marine Corps Systems Command	5. FUNDING NUMBERS
6. AUTHOR(S) Richard M. Hancock	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY *STATEMENT* Approved for public release; distribution unlimited	12b. DISTRIBUTION CODE A

13. ABSTRACT *(maximum 200 words)*

Marine Corps Systems Command, located in Quantico, Virginia, maintains a large amount of computer assets to support its vast and varied operations. This property requires accurate record keeping to assure accountability of each item throughout its lifetime, from initial acquisition through disposal. This thesis designs and implements a Bar Code Inventory System (BCIS) to support the management and accountability of the command's assets. The BCIS is a fully tested, menu driven system designed to increase the efficiency and effectiveness of the inventory process.

14. SUBJECT TERMS Bar Code Inventory, Intermec, Interscan, PCIRL, Ada, Interactive Reader Language (IRL).	15. NUMBER OF PAGES 127
	16. PRICE CODE

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

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

A Functional Bar Code Inventory System for Marine Corps Systems Command

by

Richard M. Hancock
Captain, United States Marine Corps
B.S., Randolph-Macon College, 1985

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
September 1993

Author: [illegible]
Richard M. Hancock

Approved by: [illegible]
William Haga, Thesis Advisor

[illegible]
Shu S. Liao, Thesis Co-Advisor

[illegible]
David R. Whipple, Chairman
Department of Administrative Sciences

ABSTRACT

Marine Corps Systems Command, located in Quantico, Virginia, maintains a large amount of computer assets to support its vast and varied operations. This property requires accurate record keeping to assure accountability of each item throughout its lifetime, form initial acquisition through disposal. This thesis designs and implements a Bar Code Inventory System (BCIS) to support the management and accountability of the command's assets. The BCIS is a fully tested, menu driven system designed to increase the efficiency and effectiveness of the inventory process.

TABLE OF CONTENTS

I. INTRODUCTION

A. BACKGROUND

Marine Corps Systems Command (MARCORSYSCOM), located in Quantico, Virginia, is responsible for the acquisition and accountability of computer assets. Their current Consolidated Memorandum Receipt (CMR)--a listing of all assets owned by the command--is in excess of thirty pages with approximately 3,000 computer assets. The size of the CMR is further complicated by the fact that many of these assets are portable, thus hampering the command's ability to maintain proper accountability of these assets.

The implementation of an automated inventory system will increase the efficiency and effectiveness of the inventory process and provide the command with the resources necessary to maintain proper accountability of its computer assets.

B. PROBLEMS WITH MANUAL INVENTORY SYSTEM

The manual system takes approximately six weeks to complete an inventory. Many of the items are portable, thus complicating the inventory process. Because of the length of time it takes to complete an inventory, many of the assets move from location to location during the inventory. This results in some assets being inventoried twice while other assets are never inventoried. This problem hampers the command's ability to maintain proper accountability of its computer assets.

C. GOALS AND OBJECTIVES

The primary goal of this project is to design, develop and implement a bar code inventory system which could assist MARCORSYSCOM in conducting an accurate and timely inventory of its computer assets. Automating the inventory process will provide substantial time savings and significantly increase the accuracy of the inventories.

The automated system was designed using the Object-Oriented Design approach and was implemented using Ada and Interative Reader Language (IRL) programming languages. The system accepts input from a bar code scanner, manual keypunch entries, and files downloaded from the local area network (LAN). The Bar Code Inventory System (BCIS) produces bar code labels and generates periodic reports on the status of the assets within the command. The BCIS is a fully integrated, menu driven system with established communication between the various hardware platforms--personal computer, bar code reader and bar code printer.

D. CHAPTER DESCRIPTIONS

Chapter II studies the old system and its problems, defines business needs and requirements, and evaluates alternative solutions. The step by step strategies employed during this analysis is covered in this chapter.

Chapter III will review the System Design methodology used in designing the Bar Code Inventory System (BCIS). The four phases associated with Object-Oriented Design, object identification, defining operations with each object, determination of relationships between objects, and developing interface specifications, will be covered.

Chapter IV covers the implementation of the BCIS. This chapter discuses how the four phases of the system were implemented--creating a serial number file to download to the bar code reader, conducting an inventory using the reader, processing the results of the inventory into reports, and printing bar code labels--and problems encountered while integrating these phases into the BCIS.

Chapter V, the conclusion, determines whether the system is operational and ready for implementation at MARCORSYSCOM.

Appendix A includes all the source code that was incorporated in the BCIS. This section includes the programs written in Ada, IRL, and the Batch File composed to integrate the system.

Appendix B is the user's manual. This chapter provides the documentation necessary to install and operate the Bar Code Inventory System.

II. SYSTEM ANALYSIS

The purpose of systems analysis is to study the current business system and its problems, define business needs and requirements, and evaluate alternative solutions. The three phases used in the analysis of the Bar Code Inventory System (BCIS) will be discussed in this section. The step-by-step strategies for completing each phase will be discussed followed by how that phase was applied in developing the BCIS.

A. SURVEY PHASE

1. Methodology

A project is initiated with a preliminary analysis of project scope and feasibility. The survey phase determines whether significant resources should be committed to the future phases of the life cycle. "During the Survey Phase, we define the scope of the project, perceived problems and opportunities, business and technical constraints, perceived project goals, and possible solutions" [Whitten, Bentley and Barlow, 1989, p.87]. Information gathered during this phase, although probably not very detailed--or even acurate--will be the starting point for a detailed Systems Analysis [Page-Jones, 1988, p.21].

2. Application

The goal of this project was to develop an automated inventory system which could assist Marine Corps Systems Command (MARCORSYSCOM) in conducting an accurate and timely inventory of existing assets. Captain Lang, the Resource Manager for the Information Systems Management Division at MARCORSYSCOM, wanted to limit the scope of the inventory system to include only serialized assets--such as computers, printers, monitors, etc. These types of assets have a high dollar value and represent the majority of items on Captain Lang's Consolidated Memorandum Receipt (CMR)-- a listing of all assets under his control. The current CMR is in excess of thirty pages with approximately 3,000 serialized items that must be inventoried quarterly. The size of the CMR is further complicated by the fact that many of these assets are portable, thus making the inventory process extremely difficult. The proper accountability of these assets is very important and requires a great deal of time and effort. The problems identified with the current system and the potential benefits of automating this process will be discussed in the following sections of this chapter.

It was decided that the scope of this work warranted development as an individual thesis project. The core of the work would be performed on a personal computer owned by the author. MARCORSYSCOM provided the hardware necessary to automate the current system to include: a bar code reader, scanner, bar code printer, and manuals. The cost of the additional hardware was considered negligible in comparison to the benefits expected with a fully automated inventory system. A time span of nine months, with commencement in December 1992 and system completion by August 1993, was considered feasible. The Survey Phase was accomplished during a single interview with Captain Lang that took approximately one hour.

B. STUDY PHASE

1. Methodology

During this phase facts are collected about how the current system functions. Problems with the current system are identified and potential opportunities are diagnosed in an attempt to improve the process. Data are collected using interviews and modeling techniques to learn about the system. "You need to understand the existing system, manual or computerized, before you can design and build a new system." [Whitten, Bentley, and Barlow, 1989, p.90]

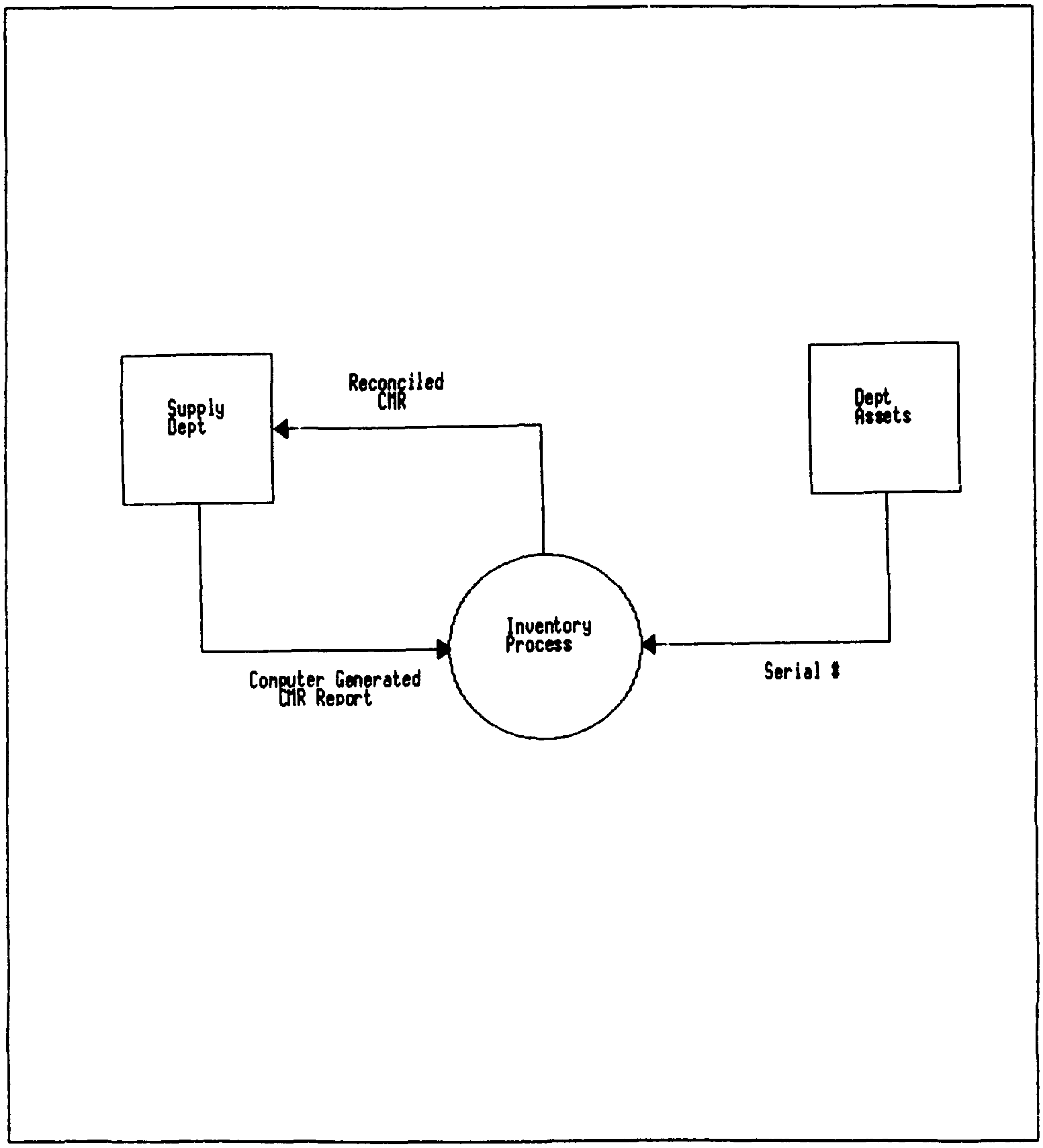
2. Application

On 14 December 1992, the author took a four day research trip to MARCORSYSCOM, which is located in Quantico, Virginia. Captain Lang was interviewed to acquire an understanding of the current inventory process. Captain Lang is accountable for all the assets assigned to his division. He is responsible for conducting quarterly inventories and possesses a thorough understanding of the process. During the interview the author developed a detailed understanding of the current system. Physical data flow diagrams (DFDs) were drafted to document the processes and the flow of data and information through the existing system (see Figures 2.1 through 2.3). These models show not only what a system does, but also how the system is physically implemented. The use of these models helped the author grasp the inputs, outputs, and processes, and the relationships among the different processes of the system.

After evaluating the models and data collected during the research trip, the following problems were identified:

1. Current system is inefficient resulting in a slow inventory process;
2. Poor accountability; and
3. Duplication of effort.

The process of manually searching the CMR for a serial number that matches the serial number on the asset was determined to be the most inefficient and time-consuming process in the system. The current CMR is in excess of thirty pages with approximately 3,000 serial numbers. The CMR is organized in such a way that each item, for example a monitor, can have many National Stock Numbers (NSNs). Under each NSN there may be many serial numbers. In order to efficiently match the serial number of the asset with the serial number on the CMR the person



<u>FIGURE 2.1</u> Physical Context Data Flow Diagram

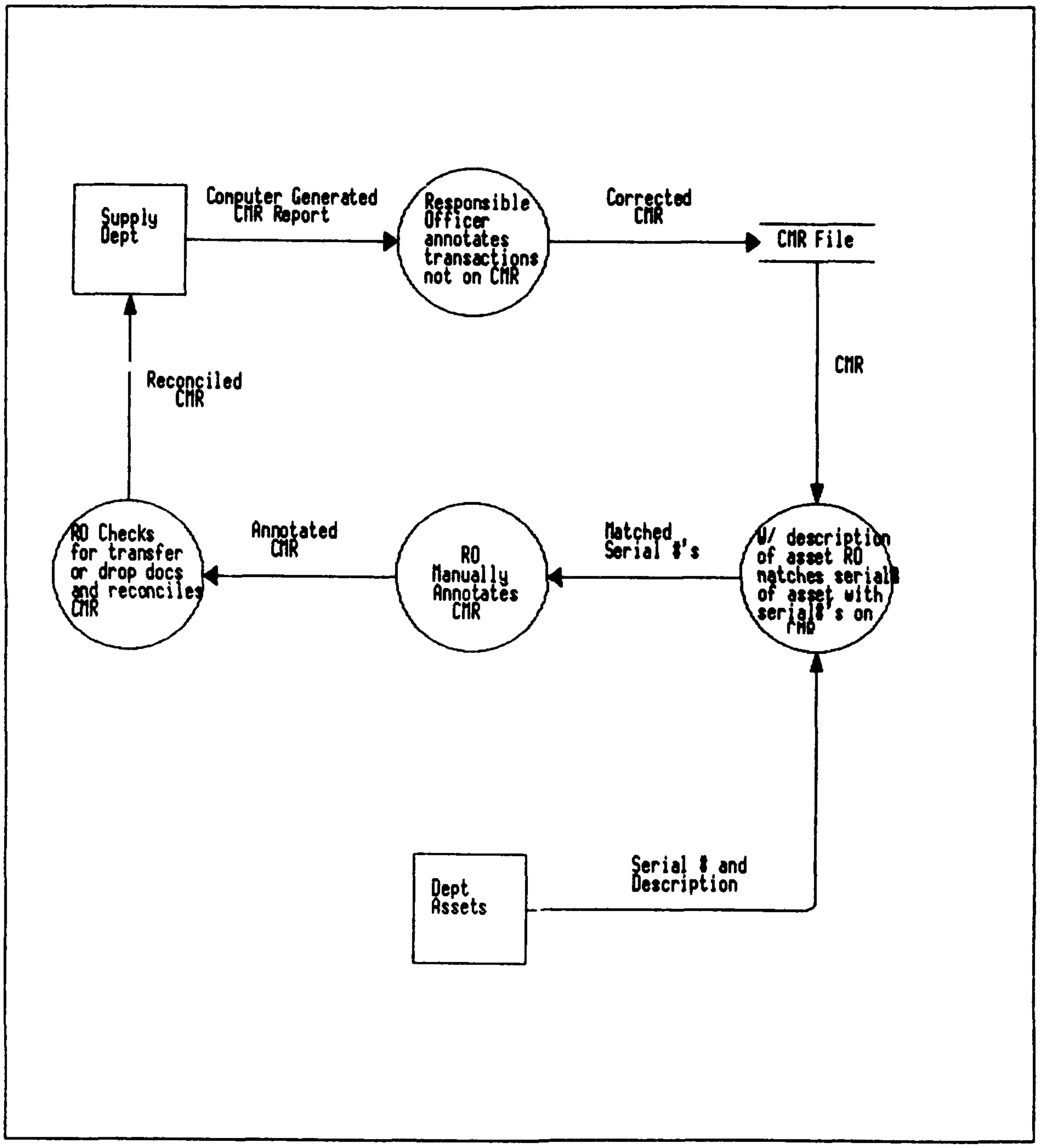

Supply Dept
Computer Generated CMR Report
Responsible Officer annotates transactions not on CMR
Corrected CMR
CMR File
CMR
W/ description of asset RO matches serial# of asset with serial#'s on CMR
Matched Serial #'s
RO Manually Annotates CMR
Annotated CMR
RO Checks for transfer or drop docs and reconciles CMR
Reconciled CMR
Dept Assets
Serial # and Description

FIGURE 2.2 **Physical Data Flow Diagram (Level 1)**

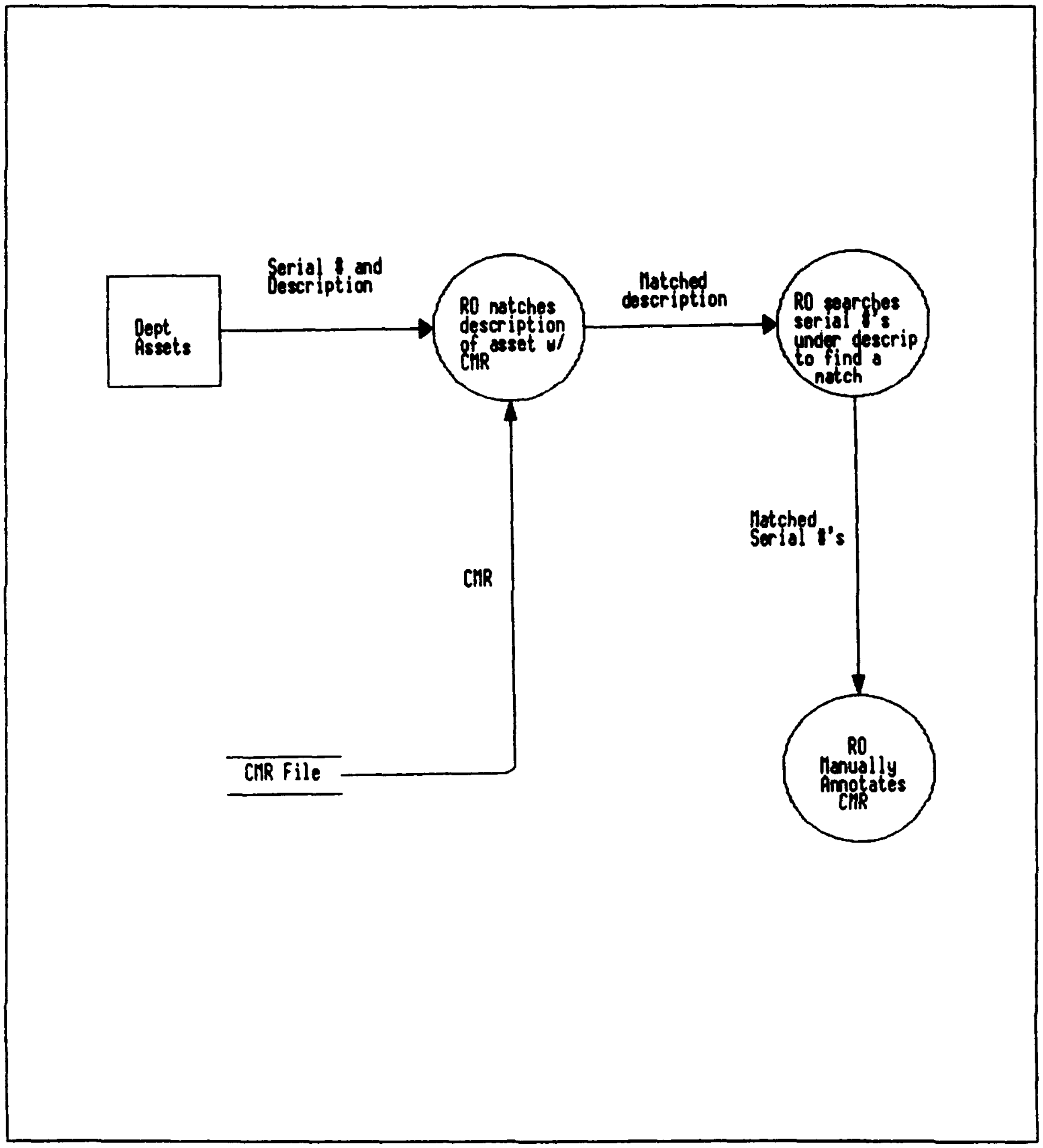

Dept Assets
Serial # and Description
RO matches description of asset w/ CMR
Matched description
RO searches serial #'s under descrip to find a match
Matched Serial #'s
CMR
CMR File
RO Manually Annotates CMR

FIGURE 2.3 **Physical Data Flow Diagram (Level 2)**

conducting the inventory must know the NSN of the asset. This information is not obtainable in the current system, forcing the user to search each NSN for a serial number that matches. For some items, such as monitors and printers, the user must search through several pages of serial numbers to find a match. This results in an inventory process that takes approximately six weeks to complete.

By automating the process of searching the CMR for a serial number, the problem of duplication of effort and poor accountability could also be reduced. For example, many of the items are portable, thus complicating the inventory process. Because of the length of time it takes to complete an inventory, many of the assets move from location to location during the inventory. This results in some assets being inventoried twice (duplication of effort) while other assets are never inventoried. This creates a problem in the accountability of the command's assets. By automating the current process the inventory could be completed in less than a week, lessening the problem of assets moving during the inventory. Automating the current process will increase the efficiency and accuracy of the current system.

C. DEFINITION PHASE

1. Methodology

The goal of the definition phase is to develop a detailed definition of the requirements and objectives of the proposed system. Requirements are the blueprint that will be used to design and implement the new system. Before moving on to development, the developer must know exactly what the system is supposed to do. "The purpose of the Definition Phase is to identify what the improved system must be able to do without specifying how the system could or will do it" [Whitten, Bentley and Barlow, 1989, p.156].

It is not only important that the system is built correctly, but vital that the **correct** system is built. The analyst should actively involve all of the end-users who were identified during the Study Phase. It is important to give end-users at every level of the organization the opportunity to define goals, objectives, and information system needs. Proper definition of the requirements is the first step toward preventing future maintenance nightmares.

2. Application

The research trip identified the end user's information needs and what functions the new system is expected to perform. A six-week turnaround time to complete an inventory was considered unsatisfactory. The major goal of the new system was to reduce the amount of time it takes to conduct an inventory and thus increase the accuracy of the inventory.

Marine Corps Systems Command wanted to use bar code technology to completely automate the inventory process. The system needed to read the serial number of the asset, search the

CMR for a match, and produce a report specifying which serial numbers were found. By using bar code technology, MARCORSYSCOM felt the inventory process could be condensed into one week.

These basic requirements were used to model the new system using logical DFDs (see Figures 2.4 and 2.5). Logical DFDs are implementation-independent models that display the essential requirements of the system--those requirements that must be fulfilled no matter how the system might be implemented. These models were presented to the end user and the following detailed system requirements were drafted:

1. System will be a fully automated bar code inventory system and will meet all specified requirements;
2. System will use bar code technology;
3. System will function on an IBM compatible desktop computer;
4. System will interface with the bar code reader and printer;
5. System scope will be limited to serialized assets: such as computers, laptops, monitors, etc.;
6. User will provide a CMR file;
7. CMR will be obtained from an ASCII file contained in an external file memory source;
8. The values of all serial numbers are of alpha-numeric type;
9. A serial number is a unique identifier--assets will not have the same serial number;
10. The system will produce bar code labels to be affixed to each asset. User will have the option of producing a batch of labels from the CMR or individual labels by manual keypunch entry;

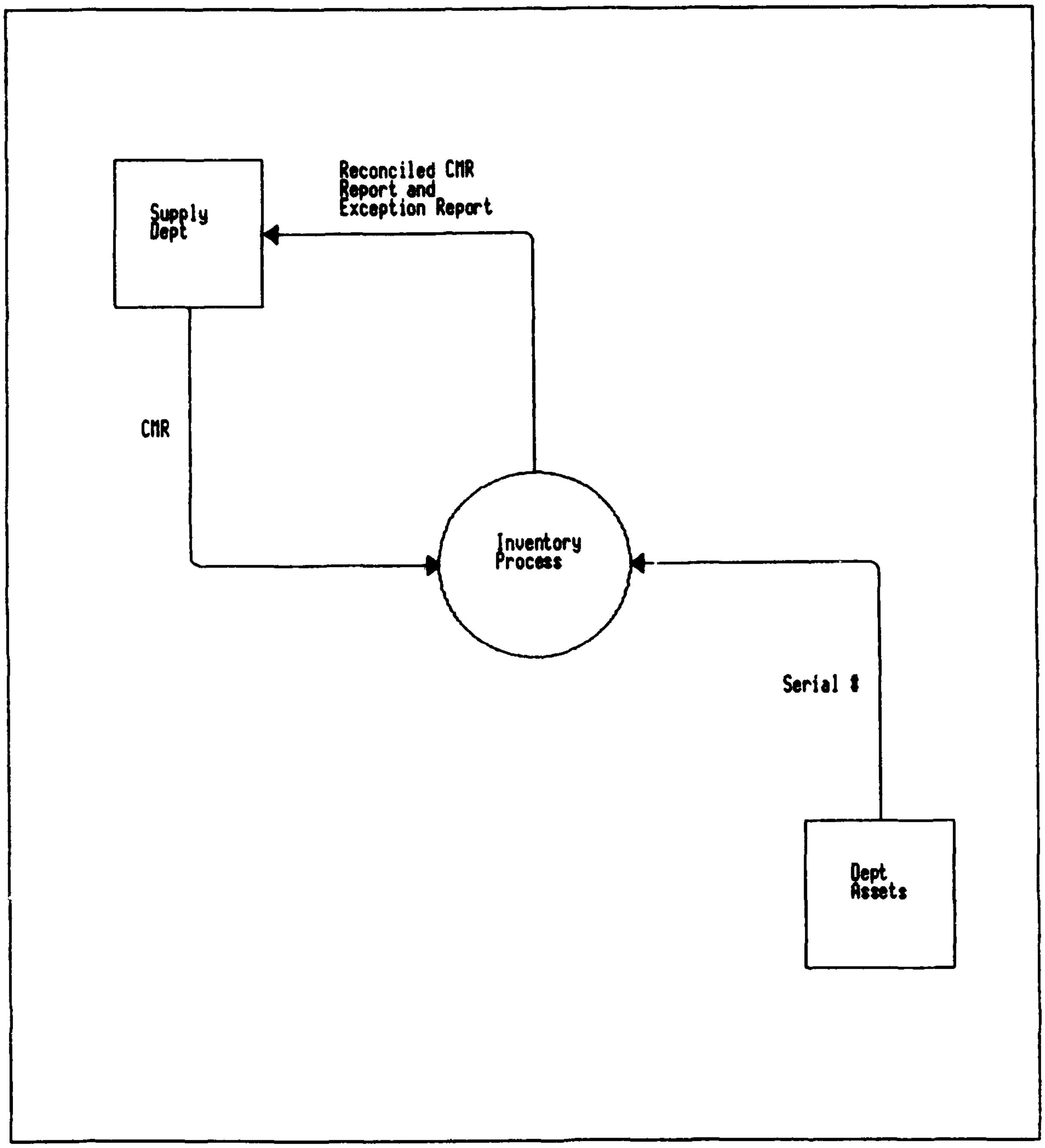


FIGURE 2.4 **Logical Context Data Flow Diagram**

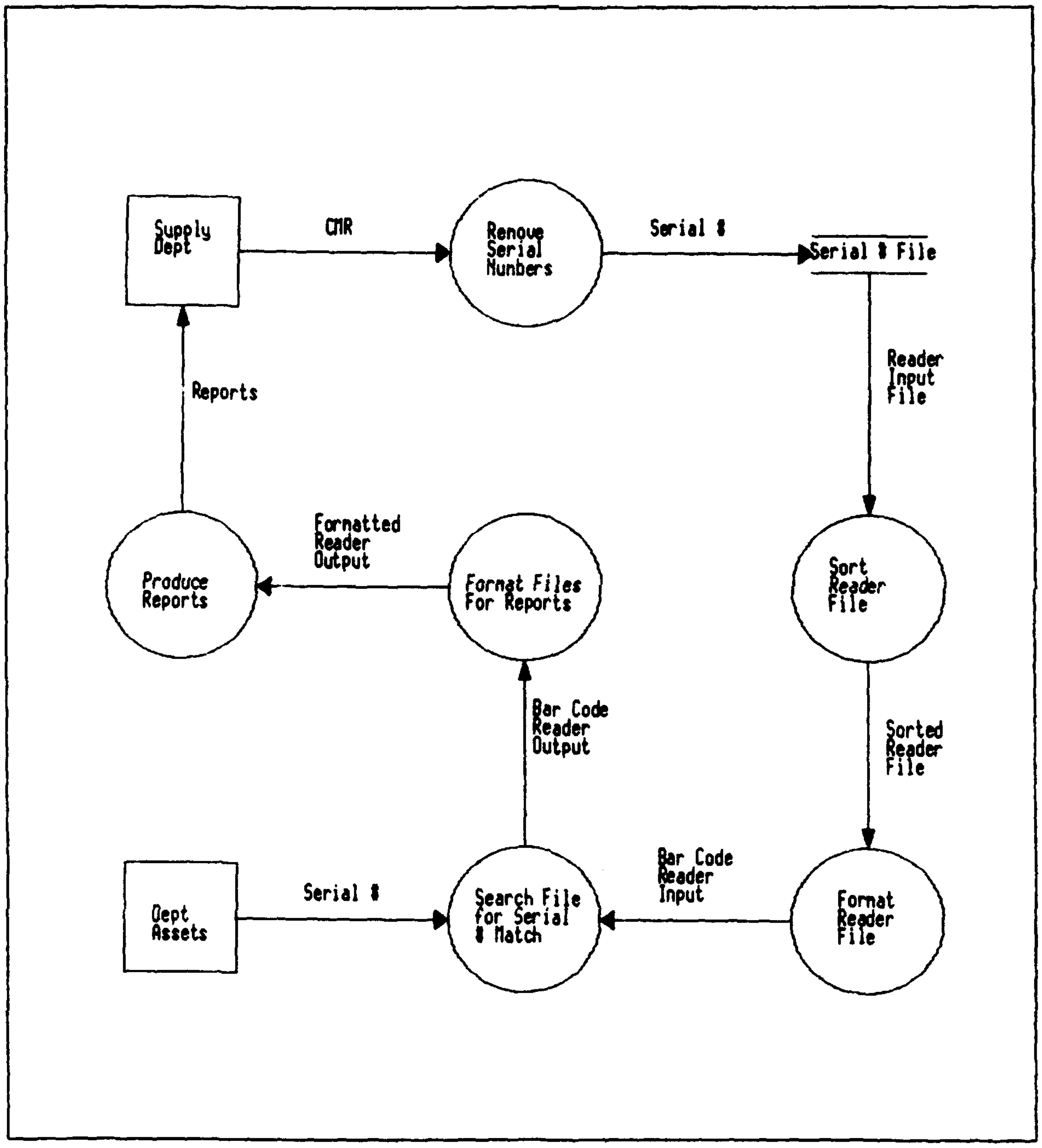


FIGURE 2.5 **Logical Data Flow Diagram (Level 1)**

11. System will prompt the user if the item scanned is not on the CMR. The user will be prompted for a description of the item and its location;

12. System output will be in CMR format with found serial numbers annotated;

13. System will produce an exception report, listing the serial numbers inventoried but not on the CMR;

14. System will be implemented in Ada or DBase III;

15. System will be menu driven and be user friendly (will not require computer experience to operate the system);

16. System will be fully documented and a users manual will accompany the system; and

17. System will be designed with ease of modification or upgrade in mind.

III. SYSTEM DESIGN

A. INTRODUCTION

The purpose of the design phase is to design a software solution for the new system, including a definition of the interfaces among units and a detailed procedural flow. During system design, the target system is organized into subsystems based on both the analysis structure and the proposed architecture. The system designer must decide what performance characteristics to optimize, choose a strategy to attack the problem, and make tentative resource allocations.

An Object-Oriented Design (OOD) strategy was employed to design the Bar Code Inventory System. This method of design results in a software solution that closely resembles the real-world problem. Object- Oriented Design incorporates three important software design concepts: abstraction, information hiding and modularity. "All design methods strive for software that exhibits these fundamental characteristics, but only OOD provides a mechanism that enables the designer to achieve all three without complexity or compromise" [Pressman, 1992, p.395].

The Design Phase uses the results of the System Analysis Phase to complete several sequential steps. These steps and their results are presented in the remainder of this chapter.

B. OBJECT AND ATTRIBUTE IDENTIFICATION

The system requirements and logical data flow diagrams were used to identify the objects judged critical to the production of the Bar Code Inventory System, (see Figure 3.1). The objects identified were derived from nouns used in describing the system. An object is a component which exists in the real world--a person, place, thing, occurrence, role, or event--that is mapped into the software domain. Objects are typically producers or

Objects	Attributes
CMR Listing	File Name Status
Serial Number	Value Position Type Found
Bar Code Reader Input	File Name Status
Bar Code Reader Output	File Name Status
Inventory	None
Report	File Name Type
File Processor	File Name Type
Label	Format Size Serial Number Quantity
Serial Number File	File Name Size

FIGURE 3.1 Objects and Attributes of BCIS

consumers of information or an information item. Only those objects that played a critical role in the system's purpose were included, as these are the items which will eventually be implemented as part of the solution.

Additionally, the defining characteristics or attributes of the objects were defined. Attributes are values or features of an object that distinguish one instance of an object from another. For example, the attribute *Value* helps distinguish one serial number from another. Identifying the attributes of an object help the programmer define the functional relationship between two instances of the same object. For example, the system will produce two different reports--a CMR report and an exception report. The attribute *File Name* and *Type* tell the programmer that the two reports can be distinguished by the file name and the type of report required.

C. OPERATIONS WITH OBJECTS

The major objects of interest and their attributes have been identified, but that is not sufficient to establish the design of the software solution. The next step is to determine the operations associated with each object, (see Figure 3.2). This was accomplished by considering which operations can be performed on or by a particular object. In identifying the operations associated with an object the designer can characterize the external behavior of that object. This external view captures

Object	Operation
CMR Listing	Open Reset Close Get Serial Number Get Line
Serial Number	Create Mark Return Value Compare
Bar Code Reader Input	Create Open Close Add Serial Number Get Serial Number
Bar Code Reader Output	Open Close Get Serial Number Get Description Get Location
Inventory	Get Serial Number Match Serial Numbers Get Location of Asset Get Description of Asset
Report	Create Close Add Line Format Exception Report
File Processor	Format Serial Number File Format CMR Report File Format Exception File Format Labels File
Label	Create Open Close Add Serial Number Get Serial Number Print Label
Serial Number File	Create Insert Serial Number Sort by Value Sort by Position Mark Serial Number Found Remove Serial Number

FIGURE 3.2 **Objects and Operations of BCIS**

the behavior of an object from the perspective of its clients without concern for how the object will be implemented. By separating the behavior of the object from its implementation the principles of abstraction and information hiding are applied.

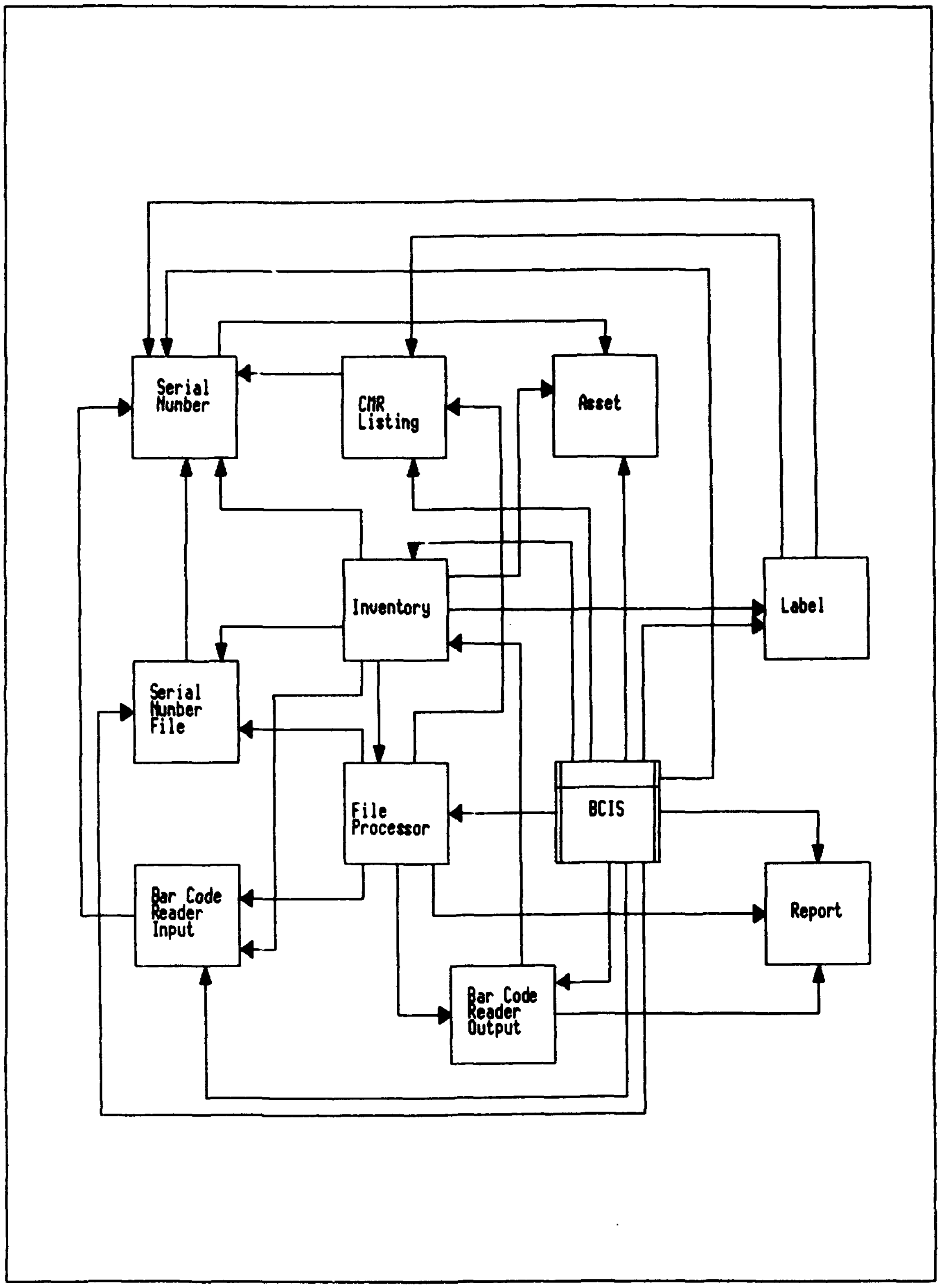
D. VISIBILITY RELATIONSHIPS

The next step in the Object-Oriented Design process is the determination of how these objects relate to one another. To establish the visibility relationship, each object is examined to determine which objects it depends on and what objects depend on it.

Figure 3.3 is a graphical representation of the relationships between the various objects in the system. An object that points to another signifies its dependence on that object. For example, the object *BCIS* depends on all the other objects in the system to perform its intended functions. Understanding the relationships between objects simplifies the implementation process and enforces the principles of modularity and cohesion. Taking advantage of these inherent relationships will result in a well-structured system that is easy to modify.

E. INTERFACE SPECIFICATIONS

The last step in the development process is to establish a detailed design of the BCIS. The results of the prior steps--Object and Attribute Identification, Operations



<u>FIGURE 3.3</u> BCIS Visibility Diagram

with Objects and Visibility Relationships--were used to organize the target system into subsystems.

First, a separate package was created for each object. A package is a separate compilation unit consisting of two parts: a package body and a package specification. During this step the package specification of each object was drafted. The package specification can be regarded as the package's "shop-window" that says what the package has to offer the potential user.

Next, the operations associated with each object were incorporated into the package specifications. This allows the designer to display what operations a package will perform, without writing any code.

Finally, how a package interfaces with the other parts of the program was specified. This was accomplished by declaring the Visibility Relationships between packages in the package specifications. The package specifications for each object are listed in alphabetical order in Appendix A.

IV. IMPLEMENTATION

The Implementation Phase used the results of the Analysis and Design Phases to create an automated inventory system. During implementation the system was divided into four phases--creating a serial number file to download to the bar code reader, conducting an inventory using the reader, processing the results of the inventory into reports, and printing the bar code labels that will be attached to the assets prior to an inventory.

Each phase was implemented separately and then integrated into the Bar Code Inventory System. This chapter will present how each phase was developed and the problems that were encountered during integration.

A. CREATING SERIAL NUMBER FILE PHASE

This portion of the system opens the CMR File--an ASCII file containing a list of assets that need to be inventoried--and searches the file for serial numbers. When a serial number is found it is copied to a file called Readin. During the transfer process the serial number is converted from the type string to a private type. Private type is an Ada convention which limits the operations that can be performed on an object outside of the object's package.

In this case, declaring serial number as a private type prevents the user from modifying a serial number outside the package *Serial Number*. This incorporates abstraction and information hiding and prevents a serial number from being inadvertently modified. The private type serial number is a record that stores the value of the serial number and its position in the CMR.

Once all the serial numbers are transferred to the Readin File, an array is created. The serial numbers, along with their original CMR position, are read into the array. Inside the array the serial numbers are formatted for the bar code reader. This consists of removing any blanks that might be in front of a serial number and sorting the serial numbers in alpha-numeric order. The formatted serial numbers are then read back into the Readin File for download to the bar code reader.

The Create Serial Number Phase of the system resides on a personal computer and incorporates the following packages (objects) identified during the System Design Phase--*CMR Listing*, *Serial Number*, *Serial Number File*, *Bar Code Reader Input* and *File Processor*. The operations necessary to extract and format the serial numbers from the CMR were implemented using Ada programming language. A program called DOWNLOAD acts as a driver and calls all the operations needed to create the serial number file.

B. INVENTORY PHASE

The Inventory Phase was implemented using Interactive Reader Language (IRL) and resides in the bar code reader. This program has two major functions: conducting an inventory of the command's serialized assets and transmitting the results of the inventory back to a file on a personal computer.

During the Inventory Function, the user is prompted to enter a serial number. The reader receives the serial number that is scanned by the user and searches File A for a match. File A contains the serial numbers extracted from the CMR during the Create Serial Number File Phase. If the serial number scanned matches a serial number on the CMR, the program searches File B, the Match File, to see if the asset was already inventoried. If the assets has not been inventoried, the serial number is saved in File B. If the asset has already been inventoried, the serial number is not saved and the user is asked to scan another asset.

If a serial number does not match the serial numbers extracted from the CMR, the reader emits a warning beep and prompts the user to enter a description and the location of the asset. This information along with the serial number is saved in File C, the Not Found File. The inventory portion of the program is terminated when the user enters 'F3' instead of a serial number.

During Transmit the user is asked which file to transmit--the Match or Not Found File. A file is then transmitted to a personal computer one serial number at a time. If the user attempts to transmit a file that does not contain any records an error message is displayed and the user is again prompted which file to transmit. The Transmit portion of the program is terminated when the user turns off the reader.

C. CREATE REPORTS PHASE

The Create Reports Phase consists of two programs--one for the CMR Report and another for the Exception Report.

The program that creates the CMR Report is called CMR_RPT. This program was implemented using Ada and resides on a personal computer. The packages (objects) used to create this portion of the system are: *CMR Listing*, *Serial Number*, *File Processor*, *Bar Code Reader Input*, *Serial Number File*, *Reader Output*, and *Report*. The program CMR_RPT acts as a driver and calls the operations that reside in these packages to perform the functions needed to convert the results of the inventory into a CMR Report.

This program opens the CMR File and copies the serial numbers along with their CMR position into an array. The Match File is then opened and each serial number is matched with the serial numbers in the array. When a match is found the serial number in the array is flagged as found--a blank space and an 'F' are appended to the end of the serial number. When all the serial numbers in the Found File have been matched, the file is closed and the array is sorted by original CMR position.

The CMR File is again opened and each line is copied to the CMR Report File. If a line contains serial numbers the program copies the serial numbers contained in the array (i.e., the flagged serial numbers) to the CMR Report File instead of the original serial numbers. The result is a report that is identical to the original CMR with the serial numbers found during the inventory marked as found.

The program that creates the Exception Report is called X_REPORT. This program creates an Exception Report File and formats the report for input. This consists of giving the report a title, setting the columns in the report and giving each column a header. The program then opens the Not Found File--a file uploaded from the reader containing assets found during the inventory that were not on the CMR. The serial number, description and location of an asset are extracted from the Not Found File and transferred to the correct position in the Exception Report.

The Exception Report program resides on a personal computer and incorporates the following packages (objects) identified during the System Design Phase--*Bar Code Reader Output*, *Report* and *File Processor*. The operations necessary to create the report were implemented using Ada programming language.

D. PRINT BAR CODE LABELS

The Print Bar Code Labels Phase consists of two programs--one for creating individual labels and another to produce a batch of labels from the CMR listing. Both of the programs were implemented using Ada and reside on a personal computer.

The program that generates individual labels is called PRN_INDIV. This program acts as a driver and calls the operations that reside in the package *Labels* to create bar code labels. The application generates a screen and queries the user to enter a serial number or "quit" to exit. If the user enters a serial number, the application appends the necessary control characters to the serial number and sends this data to the COM2 port of the personal computer. The Intermec Bar Code Printer accepts the control characters and serial number and produces a bar code label.

The program that generates a batch of bar code labels from a CMR listing is called PRN_BATCH. This program opens the CMR and extracts all the serial numbers into a file called Label. Once all the serial numbers are transferred to the Label File, an array is created. The serial numbers are then read into the array to be formatted for the bar code printer. This consists of removing any blanks from the front of a serial number and sorting the serial numbers in alpha-numeric order.

The application appends the necessary control characters to the serial numbers in the array. The COM2 port is then opened for communication with the bar code printer. The serial numbers with control characters are sent to the printer via the COM port.

This portion of the system incorporates the following packages (objects) identified during the System Design Phase--*CMR Listing*, *Labels*, *Serial Numbers*, *Serial Number File*, and *File Processor*. The operations contained in these packages were used to generate the batch bar code labels.

E. SYSTEM INTEGRATION

The four phases of the system were developed and tested separately and then integrated into a single bar code inventory system. The communication among the various hardware platforms--personal computer, bar code reader, and bar code printer--proved to be the most difficult portion of implementing the system.

The integrated system needed to pass various files across hardware platforms for the system to perform the functions specified in the System Requirements. Various approaches were tried in an attempt to establish a common protocol and initialize the COM port--set the baud rate, parity, number of data bits and stop bits--on the computer.

A communication driver was developed using *QBasic* to establish communication between a bar code reader and personal computer. This driver initialized the COM port on the computer to match the settings on the reader. The driver then appended control characters to the file being transmitted which would start and stop communication. This approach proved to be ineffective. The communication driver failed to establish communication between the personal computer and reader.

A commercial communication package called Crosstalk was used to establish communication between the personal computer and reader. This software package allows the user to establish a direct connection with other hardware platforms. The program initializes the COM port of the computer and allows the user to select a protocol to establish communication. The problem with this approach was the protocols provided in the software package did not match the protocols in the reader. Communication between the computer and reader was never established using this approach.

Intermec Corporation, the manufacturer of the bar code reader and printer, provided a software package called Interscan. This communications package was designed to establish communication between a personal computer and Intermec hardware. The software initializes the COM port in the computer and the reader. During the initial tests this program effectively established communication between the personal computer and reader. When a large Serial Number File, with three thousand serial numbers, was downloaded to the reader the program failed. The package would initiate communication but would "time out" before the entire file could be downloaded. After further testing, it was established that the problem originated from bugs in the Interscan software.

However, the Interscan software proved to be effective uploading the Match and Not Found Files from the reader to the personal computer. The software furnishes special IRL commands that allow the programmer to send the uploaded data files to specified files on the computer. This capability cannot be achieved with other software applications. The Bar Code Inventory System utilizes the Interscan application for uploading data files from a reader to a personal computer.

Intermec Corporation provided another software package called PCIRL. This software was designed as an IRL development system. It allows users to create and compile IRL programs on a personal computer. The application also provides the capability of downloading programs and files to the reader. This package was able to download the large Serial Number File to the reader. The PCIRL software is utilized in the Bar Code Inventory system to download files from the computer to a bar code reader.

When the communication problem was solved the system was integrated into the Bar Code Inventory System. The System Requirements specified the system must be menu driven and user friendly. A batch file was developed to create a menu system that would execute the necessary programs when the user selected an option. The batch file was compiled into a .COM file to increase the processing speed of the program. The .COM file, called BCIS, controls the Bar Code Inventory System and is the program that integrated the four phases of the system.

V. CONCLUSION

This thesis developed a Bar Code Inventory System (BCIS) for Marine Corps Systems Command (MARCORSYSCOM). Ada and Interactive Reader Language were used as the programming languages to implement this system.

The BCIS is a fully integrated, menu-driven system that will automate the inventory process at MARCORSYSCOM. This system extracts the serial numbers from the command's CMR--a listing of all assets owned by the command--and downloads them into a bar code reader. During the inventory, the system informs the user if the asset inventoried is not on the CMR. The system saves the information gathered during the inventory and produces two Reports--a CMR and an Exception Report. The system also produces the bar code labels, which are attached to the command's assets.

The BCIS is fully tested and ready for implementation at MARCORSYSCOM. The system meets all the requirements specified by Captain Lang, the Resource Manager for the Information Systems Management Division at MARCORSYSCOM.

APPENDIX A

SOURCE CODE

This appendix contains the source code used to implement the Bar Code Inventory System. The Programs, Package Definition Specifications, Package Definition Bodies, and batch files are included in this section and are listed in alphabetical order.

```
rem----------------------------------------------------------------------------------------
rem                                   BCIS.COM
rem----------------------------------------------------------------------------------------
rem - This program integrated the BCIS by creating a menu driven system that calls all the
rem - other programs.
rem----------------------------------------------------------------------------------------

@ECHO OFF

rem----------------------------------------------------------------------------------------
rem - This batch file automates the Bar Code Inventory System by creating a menu system and
rem - calling all the applications needed to perform the desired function.
 rem---------------------------------------------------------------------------------------
:
rem----------------------------------------------------------------------------------------
rem - The DRAWMAIN procedure creates the main menu for the system. The user is prompted
rem -  to enter the number in front of the wanted function
rem----------------------------------------------------------------------------------------
:
:DRAWMAIN                                          rem  Procedure DRAWMAIN
CLRSCR 07
CLRSCR 40 1 20 7 60                                rem  creates a red box
SETPOS 1 20                                        rem  sets postion of cursor
DRAWBOX 4E 41 7 2                                  rem  puts outline around box
 SETPOS 4 28
TEXTOUT 4F "BAR CODE INVENTORY SYSTEM"             rem  outputs text to screen
CLRSCR 1F 10 15 23 64                              rem  creates a blue box
SETPOS 12 27
TEXTOUT 1F "1. Download File to Reader"            rem  outputs text to screen
SETPOS 14 27
TEXTOUT 1F "2. Receive Inventory Results"
SETPOS 16 27
TEXTOUT 1F "3. Print Reports"
SETPOS 18 27
TEXTOUT 1F "4. Print Bar Code Labels"
SETPOS 20 27
TEXTOUT 1F "5. Exit to DOS"
SETPOS 22 40
TEXTOUT 1F "ENTER (1..5)  "

rem----------------------------------------------------------------------------------------
rem - Procedure GETKEY uses the program BATCHKEY.COM (must be in same dir) to get the
rem - users selection and assigns the users selection to an errorlevel which is used to branch to
rem -  the desired procedure
rem----------------------------------------------------------------------------------------
```

```
 :GETKEY
BATCHKEY "12345"
IF ERRORLEVEL 5 GOTO END               rem  if user enters 5 goto end procedure
IF ERRORLEVEL 4 GOTO LABELS            rem  if user enters 4 goto labels procedure
IF ERRORLEVEL 3 GOTO REPORTS           rem  if user enters 3 goto reports procedure
IF ERRORLEVEL 2 GOTO RECEIVE           rem  if user enters 2 goto receive procedure
IF ERRORLEVEL 1 GOTO DOWNLOAD          rem  if user enters 1 goto download procedure
GOTO GETKEY                            rem  if user enters wrong choice try again
 :
rem-----------------------------------------------------------------------------------
rem - Procedure DOWNLOAD changes to the Ada directory where the Ada program Download
rem - is located. Starts the program which will produce the file READIN which will be
rem - downloaded to the reader using PCIRL.  This procedure copys this file to C:\PCIRL then
rem - changes to that directory and starts the application. When the user is finished downloading
rem - the is returned to the main menu
rem-----------------------------------------------------------------------------------
:
:DOWNLOAD
CLRSCR 07
SETPOS 23 35
TEXTOUT 4F "EXTRATING SERIAL NUMBERS FROM CMR......"
CD\BCIS\ADA
CALL DOWNLOAD
COPY READIN C:\PCIRL >NUL
CD\PCIRL
CALL PCIRL
CD\BCIS
GOTO DRAWMAIN
 :
rem-----------------------------------------------------------------------------------
rem - The RECEIVE procedure creates the menu for the reveive portion of  the system.  The user
rem -  is prompted to enter the number in front of the wanted function
rem-----------------------------------------------------------------------------------
:
:RECEIVE
CLRSCR 07
CLRSCR 1F 1 20 7 60                              rem  creates a blue box
SETPOS 1 20                                      rem  sets postion of cursor
DRAWBOX 1D 41 7 2                                rem  puts outline around box
SETPOS 4 30
TEXTOUT 1F "RECEIVE FILE OPTIONS"
CLRSCR 3F 10 15 23 64                            rem  creates a cyan box
SETPOS 12 27
TEXTOUT 3F "1. Receive Found File"
SETPOS 15 27
TEXTOUT 3F "2. Receive Not Found File"
```

```
SETPOS 18 27
TEXTOUT 3F "3. Receive Both Files"
SETPOS 21 27
TEXTOUT 3F "4. Exit to Main Menu"
SETPOS 23 40
TEXTOUT 3F "ENTER (1..4)  "
:
rem-----------------------------------------------------------------------------------------
rem - Procedure RECEIVEGET uses the program BATCHKEY.COM (must be in same dir) to
rem -  get the users selection and assigns the users selection to an errorlevel which is used to
rem -  bra  h to the desired procedure
rem-----------------------------------------------------------------------------------------
:
:RECEIVEGET
BATCHKEY "1234"
IF ERRORLEVEL 4 GOTO DRAWMAIN      rem  if user enters 4 goto labels procedure
IF ERRORLEVEL 3 GOTO BOTH          rem  if user enters 3 goto reports procedure
IF ERRORLEVEL 2 GOTO NOTFOUND      rem  if user enters 2 goto receive procedure
IF ERRORLEVEL 1 GOTO FOUND         rem  if user enters 1 goto download procedure
GOTO RECEIVEGET                    rem  if wrong value entered waits for another try
:
rem-----------------------------------------------------------------------------------------
rem - The FOUND procedure changes to the INTRSCAN directory where the found will be
rem -  uploaded from the reader. This procedure copies an existing found file to found.bak then
rem -  deletes the found file.  The new found file is then uploaded from the bar code reader.  The
rem -  file is copied to the ada directory and the program CMR_RPT is executed. When
rem -  completed  the user is returned to the receive menu
rem-----------------------------------------------------------------------------------------
:
:FOUND
CD\INTRSCAN
COPY FOUND FOUND.BAK >NUL
DEL FOUND >NUL
CALL INTRSCAN                                  rem execute intrscan program
CD\BCIS
CLRSCR 07
SETPOS 23 35
TEXTOUT 4F "CREATING CMR REPORT......."
CD\INTRSCAN
COPY FOUND C:\BCIS\ADA >NUL
CD\BCIS\ADA
CALL CMR_RPT
CD\BCIS
GOTO RECEIVE                                   rem  returns to receive menu
```

```
rem-------------------------------------------------------------------------------------------------
rem - The NOTFOUND procedure changes to the INTRSCAN directory where the NOFIND file
rem -  will be uploaded from the reader.  This procedure copies an existing NOFIND file to
rem -  NOFIND.BAK then deletes the NOFIND file. The new NOFIND file is then uploaded
rem -  from the bar code reader. The file is copied to the ada directory and the program
rem -  X_REPORT is executed. When completed the user is returned to the receive menu
rem-------------------------------------------------------------------------------------------------

 :NOTFOUND
CD\INTRSCAN
COPY NOFIND NOFIND.BAK >NUL
DEL NOFIND >NUL
CALL INTRSCAN
CD\BCIS
CLRSCR 07
SETPOS 23 35
TEXTOUT 4F "CREATING EXCEPTION REPORT......"
CD\INTRSCAN
COPY NOFIND C:\BCIS\ADA >NUL
CD\BCIS\ADA
CALL X_REPORT
CD\BCIS
GOTO RECEIVE          rem  returns to receive menu

rem-------------------------------------------------------------------------------------------------
rem - The BOTH procedure performs the same functions as the FOUND and NOT FOUND
rem -  procedures.  It combines the operations of those two procedures into a single procedure
rem -  for the users convenience.
rem-------------------------------------------------------------------------------------------------

:BOTH
CD\INTRSCAN
COPY FOUND FOUND.BAK >NUL
DEL FOUND >NUL
COPY NOFIND NOFIND.BAK >NUL
DEL NOFIND >NUL
CALL INTRSCAN
CD\BCIS
CLRSCR 07
SETPOS 23 35
TEXTOUT 4F "CREATING CMR AND EXCEPTION REPORTS......"
CD\INTRSCAN
COPY FOUND C:\BCIS\ADA >NUL
COPY NOFIND C:\BCIS\ADA >NUL
CD\BCIS\ADA
CALL CMR_RPT
CALL X_REPORT
CD\BCIS
GOTO RECEIVE          rem  returns to receive menu

rem--------------------------------------------------------------------------------
rem - The REPORTS procedure creates the menu for the reports portion of the system.  The user
rem -  is prompted to enter the number in front of the wanted function
rem--------------------------------------------------------------------------------

:REPORTS
CLRSCR 07
CLRSCR 74 1 20 7 60               rem  creates a white box
SETPOS 1 20                    rem  sets postion of cursor
DRAWBOX 74 41 7 2                 rem  puts outline around box
SETPOS 4 30
TEXTOUT 74 "PRINT REPORT OPTIONS"
CLRSCR 3F 10 15 23 64             rem  creates a cyan box
SETPOS 12 27
TEXTOUT 3F "1. Print CMR Report"
SETPOS 15 27
TEXTOUT 3F "2. Print Exception Report"
SETPOS 18 27
TEXTOUT 3F "3. Print Both Reports"
SETPOS 21 27
TEXTOUT 3F "4. Exit to Main Menu"
SETPOS 23 40
TEXTOUT 3F "ENTER (1..4)  "

rem--------------------------------------------------------------------------------
rem - Procedure REPORTGET uses the program BATCHKEY.COM (must be in same dir) to
rem -  get the users selection and assigns the users selection to an errorlevel which is used to
rem -  branch to the desired procedure
rem--------------------------------------------------------------------------------
:REPORTGET
BATCHKEY "1234"
IF ERRORLEVEL 4 GOTO DRAWMAIN         rem  if user enters 4 goto main menu
IF ERRORLEVEL 3 GOTO BOTHRPTS         rem  if user enters 3 goto bothrpts procedure
IF ERRORLEVEL 2 GOTO XRPT             rem  if user enters 2 goto XRPT procedure
IF ERRORLEVEL 1 GOTO CMRRPT          rem  if user enters 1 goto CMRRPT procedure
GOTO REPORTGET                      rem  if wrong value entered waits for another try

rem--------------------------------------------------------------------------------
rem - The CMRRPT procedure displays a message on the screen that the CMR REPORT is
rem -  printing.  The procedure changes to the directory where REPORT is located and sends the
rem -  report to the printer. When finished the procedure returns the user to the reports menu.
rem--------------------------------------------------------------------------------
```

```
:CMRRPT
CLRSCR 07
SETPOS 22 38
TEXTOUT 4F "PRESS ENTER......"
SETPOS 23 35
TEXTOUT 4F "PRINING CMR REPORT......"
SETPOS 40 1
CD\BCIS\ADA
PRINT REPORT
CD\BCIS
GOTO REPORTS

rem-------------------------------------------------------------------------------------------
rem - The XRPT procedure displays a message on the screen that the XREPORT  is printing.
rem -  The procedure changes to the directory where XREPORT is located and send the report to
rem -  the printer. When finished the procedure returns the user to the reports menu.
rem-------------------------------------------------------------------------------------------

:XRPT
CLRSCR 07
SETPOS 22 42
TEXTOUT 4F "PRESS ENTER......"
SETPOS 23 35
TEXTOUT 4F "PRINTING EXCEPTION REPORT......"
SETPOS 40 1
CD\BCIS\ADA
PRINT XREPORT
CD\BCIS
GOTO REPORTS

rem-------------------------------------------------------------------------------------------
rem - The procedure BOTHRPTS prints both the CMR REPORT and the XREPORT. It
rem -  performs the same operations as the above two procedures and is included for user
rem -  convenience.
rem-------------------------------------------------------------------------------------------
 :BOTHRPTS
CLRSCR 07
SETPOS 22 46
TEXTOUT 4F "PRESS ENTER......"
SETPOS 23 35
TEXTOUT 4F "PRINTING CMR AND EXCEPTION REPORTS......"
SETPOS 40 1
CD\BCIS\ADA
PRINT REPORT XREPORT
CD\BCIS
GOTO REPORTS
```

```
rem-----------------------------------------------------------------------------------------------
rem - The LABELS procedure creates the menu for printing bar code labels.  The user is
rem -  prompted to enter the number in front of the wanted function
rem-----------------------------------------------------------------------------------------------

:LABELS
CLRSCR 07
CLRSCR 4F 1 20 7 60             rem  creates a red box
SETPOS 1 20                     rem  sets postion of cursor
DRAWBOX 40 41 7 2               rem  puts outline around box
SETPOS 4 26
TEXTOUT 4F "PRINT BAR CODE LABEL OPTIONS"
CLRSCR 3F 11 15 23 64           rem  creates a cyan box
SETPOS 14 27
TEXTOUT 3F "1. Print Batch Labels"
SETPOS 17 27
TEXTOUT 3F "2. Print Individual Labels"
SETPOS 20 27
TEXTOUT 3F "3. Exit to Main Menu"
SETPOS 23 40
TEXTOUT 3F "ENTER (1..3)  "

rem-----------------------------------------------------------------------------------------------
rem - Procedure LABELGET uses the program BATCHKEY.COM (must be in same dir) to get
rem -  the users selection and assigns the users selection to an errorlevel which is used to branch
rem -  to the desired procedure
rem-----------------------------------------------------------------------------------------------

:LABELGET
BATCHKEY "123"
IF ERRORLEVEL 3 GOTO DRAWMAIN        rem  if user enters 4 goto main menu
IF ERRORLEVEL 2 GOTO INDIV           rem  if user enters 2 goto INDIV procedure
IF ERRORLEVEL 1 GOTO BATCH           rem  if user enters 1 goto BATCH procedure
GOTO LABELGET                        rem  if wrong value entered waits for another try

rem-----------------------------------------------------------------------------------------------
rem - The procedure BATCH initializes COM2 for communication with the bar code printer. The
rem -  procedure moves to the directory where PRN_BATCH is located and executed the
rem -  program. The user is then returned to thelabels menu.
rem-----------------------------------------------------------------------------------------------

:BATCH
MODE COM2: BAUD=96 PARITY=E DATA=7 STOP=1 >NUL
CLRSCR 07
SETPOS 23 35
TEXTOUT 4F "PRINTING BAR CODE LABELS......"
```

```
SETPOS 40 1
CD\BCIS\ADA
CALL PRN_BATC
CD\BCIS
GOTO LABELS

rem-----------------------------------------------------------------------------------------
rem - The procedure INDIV initializes COM2 for communication with the bar code printer.  The
rem -  procedure moves to the directoru where PRN_INDI located and executes the program.
The
rem -  user can then print as many bar code labels as he needs. The user is then returned to the
rem -  labels menu.
rem-----------------------------------------------------------------------------------------

:INDIV
MODE COM2: BAUD=96 PARITY=E DATA=7 STOP=1 >NUL
CD\BCIS\ADA
CALL PRN_INDI
CD\BCIS
GOTO LABELS

rem-----------------------------------------------------------------------------------------
rem - The end procedure exits the user from the batch file and clears the screen. This is the end
rem -  of the program
rem-----------------------------------------------------------------------------------------

:END
CLRSCR 07
SETPOS 1 0
C:\
------------------------------------------------------------------------------------------

-- TITLE          : CMR LISTING package definition specifications
-- NAME           : Richard Hancock
-- DATE           : 28 July 1993
-- DESCRIPTION    : This package contains all operations associated with the object CMRL,
--                : which stands for CMR Listing.  This object: is an ASCII file containing a
--                : listing of all assets a responsible officer is accountable for.  The format of
--                : the CMR is fixed.  This package contains operations that will OPEN,
--                : CLOSE, and RESET the file.  It also contains operations that will search
--                : the CMR and pull out all the serial numbers or search the CMR and get
--                : each line.  A Detailed description of each of the operations is provided in
--                : the package body.
------------------------------------------------------------------------------------------

 with SERIAL_NUMBERS; use SERIAL_NUMBERS;

package CMRL is

   procedure  OPEN (NAME : in STRING);
   procedure RESET (NAME : in STRING);
   procedure CLOSE (NAME : in STRING);

   procedure GET_SN (SN : out SERIAL_NUMBER);
   procedure GET_LINE (LINE : out STRING; LAST : out NATURAL);

   function ENDOF_FILE return BOOLEAN;

   OPEN_ERROR  : exception;
   RESET_ERROR : exception;
   CLOSE_ERROR : exception;


end CMRL;
```

```
------------ ------------------------------------------------------------------------

-- TITLE          : CMR LISTING package definition body
-- NAME           : Richard Hancock
-- DATE           : 28 July 1993
-- DESCRIPTION    : This package contains all operations associated with the object CMRL,
--                : which stands for CMR Listing.  This object is an ASCII file containing a
--                : listing of all assets a responsible officer is accountable for.  The format of
--                : the CMR is fixed.  This package contains operations that will OPEN,
--                : CLOSE, and RESET the file.  It also contains operations that will search
--                : the CMR and pull out all theserial numbers or search the CMR and get
--                : each line.  A Detailed description of each of the operations is provided in
--                : the package body.
-------------------------------------------------------------------------------------

 with TEXT_IO; use TEXT_IO;

package body CMRL is


   CMRL_FILE                  :  FILE_TYPE;
   MORE_SERIAL_NUMBERS        :  BOOLEAN :=  false;
   LAST_LINE_HAD_SERNRS       :  BOOLEAN :=  false;
   SN_COUNT                   : INTEGER;
   END_OF_FILE                : BOOLEAN :=  false;



------------------------------------PROCEDURE OPEN------------------------------------
--  This procedure opens the CMRL file.  The name of the file to be opened is passed to this
--  procedure from the calling procedure.  This procedure sets the default input to be the CMRL
--  file.
-------------------------------------------------------------------------------------

 procedure OPEN (NAME : in STRING) is

begin
   TEXT_IO.OPEN (CMRL_FILE, IN_FILE, NAME);
   TEXT_IO.SET_INPUT (CMRL_FILE);
exception

  when others =>
     raise OPEN_ERROR;

end OPEN;
```

```
-----------------------------------PROCEDURE RESET-----------------------------------
-- This procedure assumes the same file name is passed as when the CMRL file was opened. It
-- resets the file so that reading from its elements can be restarted from the beginning (i.e., resets
-- the pointer to the beginning of the file). Sets the file as the current input file and resets the
-- End-of-File Flag.
-------------------------------------------------------------------------------------


procedure RESET (NAME : in STRING) is

begin
    TEXT_IO.RESET (CMRL_FILE, IN_FILE);
    TEXT_IO.SET_INPUT (CMRL_FILE);
    END_OF_FILE := false;

exception
  when others =>
     raise RESET_ERROR;

end RESET;



-----------------------------------PROCEDURE CLOSE-----------------------------------
--  This procedure closes the CMRL file and the default input is set to standard (i.e., keyboard).
-------------------------------------------------------------------------------------


procedure CLOSE (NAME : in STRING) is

begin
    CLOSE (CMRL_FILE);
    TEXT_IO.SET_INPUT (TEXT_IO.STANDARD_INPUT);

exception
  when others =>
      raise CLOSE_ERROR;

end CLOSE;
```

```
-------------------------------------PROCEDURE GET_SN-----------------------------------------
-- This procedure reads in the first 18 characters of each line of the CMRL file. If the line has a
-- serial number, it reads the serial number into a 22 character field and converts the type string
-- to type serial number (private type) without changing the value. This serial number type is
-- passed back to the calling procedure. This procedure continues to read in and process serial
-- numbers until the end of line. Then it skips to the next line and continues to search for serial
-- numbers until the end of file.
-----------------------------------------------------------------------------------------------

procedure GET_SN (SN : out SERIAL_NUMBER) is

    SER_NR       : STRING (1..22);
    HEADER       : STRING (1..18);
    JUNK_CHAR    : CHARACTER;
    COLUMN       : TEXT_IO.count;
    LINE         : TEXT_IO.count;

begin

  if not MORE_SERIAL_NUMBERS then              -- have a new line
    SN_COUNT := 0;                             -- set serial number count to 0.
    loop
       TEXT_IO.get (CMRL_FILE, HEADER);        -- get first 18 chars in line
       exit when HEADER = "         SER NRS:";
       exit when LAST_LINE_HAD_SERNRS AND HEADER = "                  ";
      LAST_LINE_HAD_SERNRS := false;           -- if line just read in had no serial
      TEXT_IO.skip_line (CMRL_FILE);             numbers then reset flag
   end loop;
    LAST_LINE_HAD_SERNRS := true;          -- now have a line with serial #'s
  end if;                                  -- so next time around this state
                                             -- will be true.
  if SN_COUNT > 1 then
    get (CMRL_FILE, JUNK_CHAR);            -- special case where serial #'s in
  end if;                                 -columns 3-5 have 23 vice 22 chars
  for CHAR in 1..22 loop                   -- read the sn in to a 22 char field
     get (CMRL_FILE, SER_NR(CHAR));
  end loop;
  SN_COUNT := SN_COUNT + 1;                    -- increment our count
  if not TEXT_IO.END_OF_LINE then              -- if we haven't reached the end of
       MORE_SERIAL_NUMBERS := TRUE;             -- the line then we have more sn's
  else                                         -- other wise we don't.
     MORE_SERIAL_NUMBERS := FALSE;
     skip_line (CMRL_FILE);
  end if;
  SERIAL_NUMBERS.CREATE (SER_NR, SN);  -- create a sn record(private type)
```

```
exception
  when TEXT_IO.END_ERROR =>
       END_OF_FILE := true;
  when CONSTRAINT_ERROR =>
       TEXT_IO.PUT_LINE (STANDARD_OUTPUT,  "-CONSTRAINT_ERROR raised
                               reading in from CMRL --");
  when others =>
     TEXT_IO.PUT_LINE (STANDARD_OUTPUT,  "-CONSTRAINT_ERROR raised
                               reading in from CMRL --");
end GET_SN;

-------------------------------PROCEDURE GET_LINE-----------------------------------
--   This procedure reads a full line from the CMRL into a 150 char string and returns the string
--   and the position of the last char in the line.
-------------------------------------------------------------------------------------

procedure GET_LINE (LINE : out STRING; LAST : out NATURAL) is

begin
    TEXT_IO.GET_LINE (CMRL_FILE, LINE, LAST);          -- get the full line
exception
    when TEXT_IO.END_ERROR =>
            END_OF_FILE := true;
   when CONSTRAINT_ERROR =>
         TEXT_IO.PUT_LINE (STANDARD_OUTPUT,  "-- CONSTRAINT_ERROR
                                 raised reading in from CMRL --");
   when others =>
            TEXT_IO.PUT_LINE (STANDARD_OUTPUT,  "--CONSTRAINT ERROR raised
                                  reading in from  CMRL --");
end GET_LINE;

---------------------------------FUNCTION ENDOF_FILE----------------------------------
--   This function returns the value of the End-of-File flag.  The flag is initially set to false but is
--   changed to true when an  END_OF_FILE exception is raised in either the GET_SN or
--   GET_LINE  procedures.
-------------------------------------------------------------------------------------

function ENDOF_FILE return BOOLEAN is

begin
    return END_OF_FILE;
end ENDOF_FILE;

-------------------------------------------------------------------------------------

 end CMRL;
```

```
---------------------------------------------------------------------------------------------------
-- TITLE          : CMR REPORT
-- NAME           : Richard M. Hancock
-- DATE           : 08 Aug 1993
-- DESCRIPTION    : This procedure is the main driver for the Report portion of the Bar Code
--                : Inventory System.  This Part of the system opens the found file which as
--                : was uploaded to the computer upon completion of the inventory.  This
--                : procedure formats the serial numbers (appends blanks to the front of each
--                : serial number to return them to 22 character fields). The serial numbers
--                : are flaged as found (ie a space and an F is appended to the serial number
--                : to identify which serial numbers were found during the inventory). The
--                : serial numbers are then sorted, ie returned to their original order when
--                : extracted from the CMRL. The CMRL is then opened and each line is
--                : copied to a File called REPORT.  When a line is extracted from the
--                : CMRL it is searched for serial numbers, if serial numbers are found they
--                : are replaced with the serial numbers that have been flaged, thus
--                : producing a final report in CMR format of which serial numbers were
--                : found during the inventory.  The system was designed using the object
--                : oriented approach and each package is an object that was identifed during
--                : the design phase.  All operations concerining a particular object will be
--                : found in that Package.  The Packages used in this portion of the system
--                : are CMRL, READER_INPUT, READER_OUTPUT,
--                : SERIAL_NUMBERS, SN_ARRAY, FILE_PROCESSOR and REPORT
---------------------------------------------------------------------------------------------------

 with CMRL; with READER_INPUT; with READER_OUTPUT; with REPORT;
 with SERIAL_NUMBERS; use SERIAL_NUMBERS;
 with SN_ARRAY; use SN_ARRAY;
 with FILE_PROCESSOR; use FILE_PROCESSOR;
 with TEXT_IO; use TEXT_IO;

procedure CMR_RPT is

   CMRL_FILE_NAME              : STRING (1..4) := "CMRL";
   REPORT_FILE_NAME            : STRING (1..6) := "REPORT";
   READER_INPUT_FILE_NAME      : STRING (1..6) := "READIN";
   READER_OUTPUT_FILE_NAME     : STRING (1..5) := "FOUND";

   SN          : SERIAL_NUMBER;
   SN_COUNT    : INTEGER := 0;
   ARR_PTR     : ARRAY_POINTER;

begin
   CMRL.OPEN (CMRL_FILE_NAME);
   READER_INPUT.CREATE (READER_INPUT_FILE_NAME);
```

```
    loop
       CMRL.GET_SN (SN);
       exit when CMRL.ENDOF_FILE;
       SN_COUNT := SN_COUNT + 1;
       READER_INPUT.ADD_SN (SN);
    end loop;
    CMRL.RESET (CMRL_FILE_NAME);
    READER_INPUT.CLOSE_OUTPUT (READER_INPUT_FILE_NAME);
    SN_ARRAY.CREATE_ARRAY (SN_COUNT, ARR_PTR);
    FORMAT_SERNR_FILE (SN_COUNT, READER_INPUT_FILE_NAME, ARR_PTR);
    FORMAT_CMR_REPORT_FILE (SN_COUNT, READER_OUTPUT_FILE_NAME,
                                      REPORT_FILE_NAME, ARR_PTR);
    CMRL.CLOSE (CMRL_FILE_NAME);


exception

    when CMRL.OPEN_ERROR =>
      TEXT_IO.PUT_LINE (STANDARD_OUTPUT, "Error OPENING file CMRL");
   when CMRL.RESET_ERROR =>
      TEXT_IO.PUT_LINE (STANDARD_OUTPUT, "Error RESETTING file CMRL");
   when CMRL.CLOSE_ERROR =>
      TEXT_IO.PUT_LINE (STANDARD_OUTPUT, "Error CLOSING file CMRL");

   when READER_INPUT.CREATE_ERROR =>
     TEXT_IO.PUT_LINE (STANDARD_OUTPUT, "Error CREATING file READIN");
   when READER_INPUT.OPEN_ERROR =>
      TEXT_IO.PUT_LINE (STANDARD_OUTPUT, "Error OPENING file READIN");
   when READER_INPUT.CLOSE_ERROR =>
      TEXT_IO.PUT_LINE (STANDARD_OUTPUT, "Error CLOSING file READIN");

   when READER_OUTPUT.OPEN_ERROR =>
      TEXT_IO.PUT_LINE (STANDARD_OUTPUT, "Error OPENING file FOUND.DAT");
   when READER_OUTPUT.CLOSE_ERROR =>
      TEXT_IO.PUT_LINE (STANDARD_OUTPUT, "Error CLOSING file FOUND.DAT");

   when REPORT.CREATE_ERROR =>
      TEXT_IO.PUT_LINE (STANDARD_OUTPUT, "Error CREATING file REPORT");
   when REPORT.CLOSE_ERROR =>
      TEXT_IO.PUT_LINE (STANDARD_OUTPUT, "Error CLOSING file REPORT");


end CMR_RPT;
```

```
-----------------------------------------------------------------------------------------
--TITLE             : Download
-- NAME             : Richard M. Hancock
-- DATE             : 05 Aug 1993
-- DESCRIPTION      : This procedure is the main driver the for download portion of the Bar
--                  : Code Inventory System.  This part of the system opens the CMRL file
--                  : (ASCII file) provided by the user and extracts all the serial numbers into
--                  : a Readin file.  These serial numbers are then formatted (any blanks at the
--                  : front of the serial numbers are removed) and the file is sorted into
--                  : alpha-numeric ascending order prior to download to the bar code reader.
--                  : This portion of the system is necessary before the actual inventory is
--                  : conducted.  The system was designed using the object oriented approach
--                  : and each package is an object that was identified during the design phase.
--                  :  All operations concerning any object may be found in that Package.  The
--                  : Packages used with this portion of the system are CMRL,
--                  : READER_INPUT,  SERIAL_NUMBERS, SN_ARRAY, and
--                  : FILE_PROCESSOR.
-----------------------------------------------------------------------------------------

with CMRL; with READER_INPUT;
with SERIAL_NUMBERS; use SERIAL_NUMBERS;
with SN_ARRAY; use SN_ARRAY;
with FILE_PROCESSOR; use FILE_PROCESSOR;
with TEXT_IO; use TEXT_IO;

procedure DOWNLOAD is

   CMRL_FILE_NAME               : STRING (1..4) := "CMRL";
   READER_INPUT_FILE_NAME       : STRING (1..6) := "READIN";

   SN            : SERIAL_NUMBER;
   SN_COUNT      : INTEGER := 0;
   ARR_PTR       : ARRAY_POINTER;

begin
   CMRL.OPEN (CMRL_FILE_NAME);
   READER_INPUT.CREATE (READER_INPUT_FILE_NAME);
   loop
      CMRL.GET_SN (SN);
      exit when CMRL.ENDOF_FILE;
      SN_COUNT := SN_COUNT + 1;
      READER_INPUT.ADD_SN (SN);
   end loop;
   CMRL.CLOSE(CMRL_FILE_NAME);
   READER_INPUT.CLOSE_OUTPUT (READER_INPUT_FILE_NAME);
   SN_ARRAY.CREATE_ARRAY (SN_COUNT, ARR_PTR);
   FORMAT_SERNR_FILE (SN_COUNT, READER_INPUT_FILE_NAME, ARR_PTR);

exception
   when CMRL.OPEN_ERROR =>
       TEXT_IO.PUT_LINE (STANDARD_OUTPUT, "Error OPENING file CMRL");
   when CMRL.CLOSE_ERROR =>
      TEXT_IO.PUT_LINE (STANDARD_OUTPUT, "Error CLOSING file CMRL");

   when READER_INPUT.CREATE_ERROR =>
      TEXT_IO.PUT_LINE (STANDARD_OUTPUT, "Error CREATING file READIN");
   when READER_INPUT.OPEN_ERROR =>
      TEXT_IO.PUT_LINE (STANDARD_OUTPUT, "Error OPENING file READIN");
    when READER_INPUT.CLOSE_ERROR =>
      TEXT_IO.PUT_LINE (STANDARD_OUTPUT, "Error CLOSING file READIN");


end DOWNLOAD;
```

```
-------------------------------------------------------------------------------------
-- TITLE            : Exception Report
-- NAME             : Richard M. Hancock
-- DATE             : 11 Aug 1993
-- DESCRIPTION      : This procedure is the main driver the for uploading the NoFind file from
--                  : the bar code reader. This procedure creates an Exception Report file and
--                  : and opens the NoFind file. Formats the report by giving the report a title
--                  : and column headings. The procedure then gets the serial number,
--                  : description and location of the asset not on the CMR and puts it in the
--                  : report, under the appropriate heading. This is continued until the end of
--                  : file flag is raised for the NoFind file. The Packages used with this portion
--                  : of the system are CMRL, READER_INPUT, SERIAL_NUMBERS,
--                  : SN_ARRAY, and FILE_PROCESSOR.
-------------------------------------------------------------------------------------

with READER_OUTPUT; with REPORT;
with FILE_PROCESSOR; use FILE_PROCESSOR;
with TEXT_IO; use TEXT_IO;

procedure X_REPORT is

   REPORT_FILE                   : FILE_TYPE;
   REPORT_FILE_NAME              : STRING (1..7) := "XREPORT";
   READER_OUTPUT_FILE_NAME       : STRING (1..6) := "NOFIND";

begin
  CREATE (REPORT_FILE, OUT_FILE, REPORT_FILE_NAME);
  SET_OUTPUT (REPORT_FILE);
  FORMAT_EXCEPTION_FILE (READER_OUTPUT_FILE_NAME, REPORT_FILE);
   CLOSE (REPORT_FILE);

exception

  when READER_OUTPUT.OPEN_ERROR =>
     TEXT_IO.PUT_LINE (STANDARD_OUTPUT, "Error OPENING file NOFIND");

  when READER_OUTPUT.CLOSE_ERROR =>
     TEXT_IO.PUT_LINE (STANDARD_OUTPUT, "Error CLOSING file NOFIND");


end X_REPORT;
```

```
-------------------------------------------------------------------------------------
--TITLE            : FILE PROCESSOR definition package specifications
-- NAME            : Richard Hancock
-- DATE            : 17 July 1993
-- DESCRIPTION     : This package contains all operations associated with the object FILE
--                 : PROCESSOR. The operations are necessary to format the files that will
--                 : be passed between different hardware platforms. The Package performs
--                 : operations on files such as removing any blank spaces from serial
--                 : numbers before they are passed to the bar code reader and after the
--                 : inventory is completed the serial numbers that found are flaged and
--                 : returned to their origninal length of 22 characters. A detailed description
--                 : of each of the operations is provided in the package body.
-------------------------------------------------------------------------------------

with SN_ARRAY; use SN_ARRAY;
with TEXT_IO; use TEXT_IO;

package FILE_PROCESSOR is


    REPORT_FILE : FILE_TYPE;


procedure FORMAT_SERNR_FILE (SN_COUNT    : in INTEGER;
                             SERNR_FILE  : in STRING;
                             ARR_PTR     : in ARRAY_POINTER);

procedure FORMAT_CMR_REPORT_FILE (SN_COUNT    : in INTEGER;
                                  SOURCE_FILE : in STRING;
                                  DEST_FILE   : in STRING;
                                  ARR_PTR     : in ARRAY_POINTER);

procedure FORMAT_EXCEPTION_FILE (SOURCE_FILE : in STRING;
                                 REPORT_FILE : in FILE_TYPE);

 procedure FORMAT_LABELS_FILE (SN_COUNT    : in INTEGER;
                               LABEL_FILE  : in STRING;
                               PRINT_FILE  : in STRING;
                               ARR_PTR     : in ARRAY_POINTER);


 end FILE_PROCESSOR;
```

```
-------------------------------------------------------------------------------------
--TITLE          : FILE PROCESSOR definition package body
-- NAME          : Richard Hancock
-- DATE          : 17 July 1993
-- DESCRIPTION   : This package contains all operations associated with the object FILE
--               : PROCESSOR. The operations are necessary to format the files that will
--               : be passed between different hardware platforms. The Package performs
--               : operations on files such as removing any blank spaces from serial
--               : numbers before they are passed to the bar code reader and after the
--               : inventory is completed the serial numbers that found are flagged and
--               : returned to their original length of 22 characters. A detailed description
--               : of each of the operations is provided in the package body.
-------------------------------------------------------------------------------------

 with SERIAL_NUMBERS; use SERIAL_NUMBERS;
with TEXT_IO; use TEXT_IO;
with READER_INPUT; with READER_OUTPUT;
with REPORT; with CMRL; with LABELS;

package body FILE_PROCESSOR is

-------------------------PROCEDURE FORMAT_SERNR_FILE-------------------------------
-- This procedure opens the file READIN, which contains the serial numbers extracted from
-- the CMR and formats this file for the bar code reader. The serial numbers in the file are 22
-- characters long, this procedure reads each serial number into an array and removes any blank
-- spaces in front of the serial number. The array is sorted in alpha-numeric order and then read
-- back to the original file called READIN. This file is then going to be downloaded to the bar
-- code reader.
-------------------------------------------------------------------------------------

 procedure FORMAT_SERNR_FILE (SN_COUNT    : in INTEGER;
                              SERNR_FILE  : in STRING;
                              ARR_PTR     : in ARRAY_POINTER) is

   VALUE   : STRING (1..22);
   INDEX   : INTEGER := 0;
   SN      : SERIAL_NUMBER;

begin
   READER_INPUT.OPEN_INPUT (SERNR_FILE);          -- open sn file
   for I in 1..SN_COUNT loop                      -- for the number
      READER_INPUT.GET_SN (SN);                   -of serial nums
      SN_ARRAY.INSERT_SN (I,SN,ARR_PTR);            -- get one and place
   end loop;                                      -- it in the sn array
  READER_INPUT.CLOSE_INPUT (SERNR_FILE);          -close the file to input
 READER_INPUT.OPEN_OUTPUT (SERNR_FILE);          -- reopen it for output
```

```
  SN_ARRAY.SORT_BY_VALUE (SN_COUNT,ARR_PTR);        -- sort sn's

  for I in 1..SN_COUNT loop
    VALUE := SERIAL_NUMBERS.VALUE (SN_ARRAY.REMOVE_SN(I,ARR_PTR));
      for J in 1..22 loop
        if VALUE(J..J) /= " " then                 -- for each sn in the
          INDEX := J;                              -- array, get its value
          exit;                                    -- remove any blanks
        end if;                                    -- and output it to the
      end loop;                                      -- bar code reader input file
    READER_INPUT.ADD_SN (VALUE (INDEX..22));
  end loop;
  READER_INPUT.ADD_SN ("*");                       -- tell reader EOF
 READER_INPUT.CLOSE_OUTPUT (SERNR_FILE);    -- close the file

end FORMAT_SERNR_FILE;


------------------------PROCEDURE FORMAT_CMR_REPORT_FILE----------------------
-- This procedure opens the file FOUND, which contains the serial numbers found during the
-- inventory that were on the CMR. The serial numbers are read from the FOUND file and the
-- array is searched for a match. The serial numbers in the array are the serial numbers
-- extracted from the CMR and were placed in the array when procedure
-- FORMAT_SERNR_FILE was called. When the serial number in the array is matched with a
-- serial number in the found file the serial number in the array is flagged as found. The serial
-- number is appended with a blank and an 'F'. The serial numbers in the array are then sorted
-- back to their original CMR position. A file called REPORT is created and the CMR file is
-- opened. Each line of the CMR is copied to the REPORT file. If a line in the CMR contains
-- any serial numbers the serial numbers in the array are written to the report file instead of the
-- serial numbers in the CMR. This creates a report that is the same format as the CMR except
-- the serial numbers that were found during the inventory are flagged as found.
------------------------------------------------------------------------------------------

procedure FORMAT_CMR_REPORT_FILE (SN_COUNT    : in INTEGER;
                                  SOURCE_FILE : in STRING;
                                   DEST_FILE  : in STRING;
                                    ARR_PTR   : in ARRAY_POINTER) is

   INDEX      : INTEGER := 0;
   SN         : SERIAL_NUMBER;
   SN_STR    : STRING (1..22);
   IN_LINE   : STRING (1..150);
   LAST       : NATURAL;
```

```
begin
  READER_OUTPUT.OPEN (SOURCE_FILE);              -- open the bar code reader
  REPORT.CREATE (DEST_FILE);                     -- output file, create the
  loop                                           -- cmrl report file
    SN_STR := "                    ";
    READER_OUTPUT.GET_SN (SN_STR);                    -- for every sn in the
    exit when READER_OUTPUT.ENDOF_FILE;               -- file, get it and mark
    SN_ARRAY.MARK_SN_FOUND (SN_STR,ARR_PTR);          -- it found
  end loop;
  SN_ARRAY.SORT_BY_KEY (SN_COUNT, ARR_PTR);              -- sort the array of sn's
                                                         -- back to original order
  INDEX := 1;
  loop                                                -- get each line in the original
    CMRL.GET_LINE (IN_LINE, LAST);                    -- cmrl
    exit when CMRL.ENDOF_FILE;
    if IN_LINE (1..18) /= "                  " then
      while IN_LINE (1..18) /= "         SER NRS:" loop
        REPORT.ADD_LINE (IN_LINE (1..LAST));          -- if the line doesn't
        CMRL.GET_LINE (IN_LINE, LAST);                -- have any sn's then get
      end loop;                                       -- the next line until EOF
   end if;
   SN := SN_ARRAY.REMOVE_SN (INDEX, ARR_PTR);

          -- want to take the line input from the original cmrl and reinsert serial numbers from
          -- the array and then output the reconstructed line to the cmrl report file

     IN_LINE (19..40) := SERIAL_NUMBERS.VALUE (SN);        -- insert 1st sn
     if LAST >= 62 then                                    -- insert 2nd sn
       INDEX := INDEX + 1;
       IN_LINE(41..62):=
SERIAL_NUMBERS.VALUE(SN_ARRAY.REMOVE_SN(INDEX,
                              ARR_PTR));
     end if;
     if LAST >= 85 then                                     -- insert 3rd sn
       INDEX := INDEX + 1;

IN_LINE(64..85):=SERIAL_NUMBERS.VALUE(SN_ARRAY.REMOVE_SN(INDEX,
                                             ARR_PTR));
     end if;
     if LAST >= 108 then                                    -- insert 4th sn
        INDEX := INDEX + 1;
        IN_LINE(87..108):=SERIAL_NUMBERS.VALUE(SN_ARRAY.REMOVE_SN
                                                   (INDEX, ARR_PTR));
     end if;
     if LAST = 131 then                                     -- insert 5th sn
        INDEX := INDEX + 1;
        IN_LINE(110..131):= SERIAL_NUMBERS.VALUE(SN_ARRAY.REMOVE_SN
                                                        (INDEX, ARR_PTR));
    end if;
    INDEX := INDEX + 1;                                    -- add reconstructed line
   REPORT.ADD_LINE (IN_LINE(1..LAST));                     to the cmrl report
  end loop;
  READER_OUTPUT.CLOSE (SOURCE_FILE);                       close the files
 REPORT.CLOSE (DEST_FILE);

end FORMAT_CMR_REPORT_FILE;

-----------------------PROCEDURE FORMAT_EXCEPTION_FILE--------------------------
-- This procedure formats the serial numbers that were not on the CMR but were found during
--  the inventory into an exception report. During the inventory the user is notified when an asset
--  is not on the CMR.  He is instructed to enter a description of the asset and its location . This
--  information will be recorded in the exception report.
-------------------------------------------------------------------------------------------
 procedure FORMAT_EXCEPTION_FILE (SOURCE_FILE : in STRING;
                                  REPORT_FILE : in FILE_TYPE) is

   SN_STR               : STRING (1..22);
   ASSET_DESCRIPTION    : STRING (1..30);
   ASSET_LOCATION       : STRING (1..18);
   LAST                 : NATURAL;

begin
   REPORT.FORMAT_EXCEPTION_REPORT;
   READER_OUTPUT.OPEN (SOURCE_FILE);
   loop
      exit when READER_OUTPUT.ENDOF_FILE;
      SN_STR := "                      ";
      ASSET_DESCRIPTION := "                              ";
      ASSET_LOCATION := "                  ";
      READER_OUTPUT.GET_SN (SN_STR);
      TEXT_IO.SET_COL(REPORT_FILE, 1);
      TEXT_IO.PUT(SN_STR);
      READER_OUTPUT.GET_DESCRIPTION(ASSET_DESCRIPTION);
      TEXT_IO.SET_COL(REPORT_FILE, 28);
      TEXT_IO.PUT(ASSET_DESCRIPTION);
      READER_OUTPUT.GET_LOCATION(ASSET_LOCATION);
     TEXT_IO.SET_COL([illegible]ORT_FILE, 62);
      TEXT_IO.PUT_LINE(ASSET_LOCATION);
   end loop;
   READER_OUTPUT.CLOSE (SOURCE_FILE);

end FORMAT_EXCEPTION_FILE;
```

```
----------------------------PROCEDURE FORMAT_LABELS_FILE--------------------------------
This procedure opens the file LABELS, which contains the serial numbers extracted from the
-- CMR and formats this file for printing bar code labels. The serial numbers in the file are 22
-- characters long, this procedure reads each serial number into an array and removes any blank
-- spaces in front of the serial number. The array is sorted in alpha-numeric order and then the
-- serial numbers are appended with the necessary control characters for communication with the
-- bar code printer. The serial numbers with the control characters are then written to COM2 port
-- for printing.
-------------------------------------------------------------------------------------------

 procedure FORMAT_LABELS_FILE (SN_COUNT    : in INTEGER;
                               LABEL_FILE  : in STRING;
                               PRINT_FILE  : in STRING;
                               ARR_PTR      : in ARRAY_POINTER) is

   VALUE    : STRING (1..22);
   INDEX    : INTEGER := 0;
   LENGTH   : INTEGER := 0;
   SN        : SERIAL_NUMBER;

begin
   LABELS.OPEN_INPUT (LABEL_FILE);                       -- open sn file
   for I in 1..SN_COUNT loop                              -- for the number
      LABELS.GET_SN (SN);                                -- of serial nums
     SN_ARRAY.INSERT_SN (I,SN,ARR_PTR);                   -- get one and place
   end loop;                                              -- it in the sn array
  LABELS.CLOSE_INPUT (LABEL_FILE);                       -- close the file to input
  LABELS.CREATE (PRINT_FILE);                             -- open com2 for output
   SN_ARRAY.SORT_BY_VALUE (SN_COUNT,ARR_PTR);          -- sort sn's
   for I in 1..SN_COUNT loop
      VALUE := SERIAL_NUMBERS.VALUE (SN_ARRAY.REMOVE_SN(I,ARR_PTR));
       for J in 1..22 loop
          if VALUE(J..J) /= " " then                          --  for each sn in the
             INDEX := J;                                     --  array, get its value
             LENGTH := 23 - INDEX;                            -- get length of each sn
             exit;                                           --  remove any blanks
         end if;                                             --  and output it to the
      end loop;                                              --  com2 port
     LABELS.PRINT_LABEL (VALUE (INDEX..22), LENGTH);
  end loop;
  LABELS.CLOSE_OUTPUT (PRINT_FILE);                      --  close com2 port

end FORMAT_LABELS_FILE;
---------------------------------------------------------------------

 end FILE_PROCESSOR;
```

```
               :INVENTORY PORTION OF SYSTEM

:*****************************************************************
: This program is designed to be used by Marine Corps Systems
: Command in conducting inventory of its serialized assets. This
: portion of the system is written using IRL and will reside in
: the bar code reader. This program has two major functions:
: conducting the inventory and transmitting the results back to a
: computer. During the inventory portion, the user is prompted to
: enter a serial number or F3 to transmit.  The user should
: continue to enter serial numbers until the inventory is
: complete. The inventory takes a serial number , and performs a
: search of the serial number file (FILE A).  This file contains
: all the serial numbers found on the CMR Listing. If the serial
: number scanned is is on the CMR (in FILE A), the program jumps
: to a procedure called CHKDUP.  This procedure then checks the
: found file to see if the asset just scanned has already been
: inventoried.  If the asset has not already been inventoried the
: program jumps to the FOUND procedure, where the serial number
: is recorded in the found file (FILE B).  If the asset has
: already been inventoried the program jumps back to the INV
: procedure and the user is asked to enter another serial number.
: If the serial number scanned does not match any of the serial
: numbers in FILE A, the user receives a warning beep and is
: instructed to enter the description and location of the asset.
: This data is then recorded in the not found file (FILE C). When
: the user selects transmit data he is promted whether he wants
: to transmit the found file or the not found file.  The reader
: then assigns the name the file will be transmitted to on
: the computer using IRL commands specially created to interact
: with intrscan software.  The files are then uploaded to the
: computer.
:*****************************************************************

:              SPECIFICATION DEFINITION SECTION

OA(4000,23)           : Open file A for 4000 records, 23 char per
                      : record. File will contain serial #'s
                      : downloaded from CMR.
OB(4000,23)           : Open file B for 4000 records, 23 char per
                      : record. File will contain the serial# of
                      : assets scanned that matched the CMR.
OC(100,128)           : Open file C for 100 records, 128 char per
                      : record. File contains the serial #,
                      : description, and location of assets that
                      : were not on the CMR.
P" MARCORSYSCOM"      : Displayed on screen when program is
P"  INVENTORY"        : initiated.
W3                    : Wait 3 seconds
```

```
:----------------------------------------------------------------
                 :INVENTORY PORTION OF PROGRAM
:----------------------------------------------------------------


:***************************************************************
: The INV routine commences the actual inventory portion of the
: program. It prompts the user to enter a serial # or F3 to
: transmit files after the inventory is completed.
:***************************************************************


.INV                    : INV routine
    D$0=""              : Clear input register 0
    P"\e[2J"            : Clear screen on reader
    P"ENTER SERIAL # OR"
    P"F3 TO TRANSMIT"
    P"\r"               : Carriage return
    A                   : Get ASCII input
    G$0="F3".XMIT       : Goto XMIT routine if F3 is entered


:****************************************************************
: SEARCH is a routine designed to perform a search of File A
: to see if the serial # of the asset inventoried is on the CMR.
: The routine calls two other routines (CHKDUP and NOFIND).
:****************************************************************


.SEARCH                 : SEARCH ROUTINE
    H#9=A               : Puts the location number of the last
                        : record in File A into register #9.
    LA$0#8              : Searches file A for a serial number that
                        : matches the serial number is string
                        : register $0. It then places the location
                        : of the match in register #8. If there is
                        : not a match it places the location number
                        : of the last serial number in the file
                        : plus one into register #8.
    G#8<=#9.CHKDUP      : Goto CHKDUP routine if the location
                        : number in register 8 is less than or
                        : equal to the location # in register 9.
    G#8>#9.NOFIND       : Goto routine NOTFOUND if the location
                        : number found in register 8 is greater
                        : than the number of records in the file.
```

```
;*****************************************************************
: The CHKDUP routine is called from the SEARCH routine.  This
: routine searches the found file to make sure the asset just
: scanned hasn't already been inventoried.  If the asset has not
: already been inventoried the program jumps to the FOUND
: routine.  If the asset has already been inventoried the program
: returns to the INV routine.
;*****************************************************************


.CHKDUP                  : CHKDUP ROUTINE
     H#7=B               : Puts the location number of the last
                         : record in File B into register #7.
     LB$0#6              : Searches file B for a serial number that
                         : matches the serial number is string
                         : register $0. It then places the location
                         : of the match in register #6. If there is
                         : not a match it places the location number
                         : of the last serial number in the file
                         : plus one into register #6.
     G#6<=#7.INV         : Goto INV routine if the location number
                         : in register 6 is less than or equal to
                         : the location # in register 7. (If the
                         : asset has already been inventoried).
     G#6>#7.FOUND        : Goto routine FOUND if the location
                         : number found in register 6 is greater
                         : than the number of records in the file.
                         : The asset has not already been
                         : inventoried.


;*****************************************************************
: FOUND routine is called from .chkdup and is initiated when
: the serial # scanned is found in FILE A. This routine saves the
: serial number in the found file (FILE B) and returns to the INV
: routine.
;*****************************************************************


.FOUND                   : FOUND ROUTINE
     RB                  : Puts the serial number scanned into file
                         : B which is the matched file and clears
                         : string register 0.
     G.INV               : Goto INV routine, this forms a loop
                         : taking you back to the beginning of
                         : the program an asks for another serial
                         : number
```

```
;*****************************************************************
: The NOFIND routine is called from the SEARCH routine. It is
: initiated when the entire file A has been searched and none of
: the serial numbers matched.  This routine querries the user
: to enter a description of the item and the location and
: stores the information in file C.
;*****************************************************************

.NOFIND                  : NOFIND ROUTINE
     B111111             : bar code reader will beep to warn user
     P"\e[2J"            : Clears the screen on the reader
     P"ITEM NOT ON CMR"  : Prompt user
     W3                  : Wait 3 sec before going to next
                         : command
     P"\e[2J"            : Clears screen on reader
     RC                  : Transfers the serial number to the Not
                         : Found file (file C) and clear the
                         : register
     P"ENTER DESCRIPTION": Prompt user
     P"OF THE ASSET "
     P"\r"               : Carriage return
     K                   : Get input from the keypad
     RC                  : Transfers the description to the Not
                         : Found file (file C) and clear \ ıe
                         : register
     P"\e[2J"            : Clears screen on reader
     P"ENTER LOCATION"   : Prompt user
     P"OF THE ASSET "
     P"\r"               : Carriage return
     K                   : Get input from the keypad
     RC                  : Transfers location to Not found file
                         : (file C) and clears the register
     G.INV               : Goto INV routine


;-----------------------------------------------------------------
                  :TRANSMIT PORTION OF PROGRAM
;-----------------------------------------------------------------


;*****************************************************************
: The XMIT routine prompts the user to choose which file to
: transmit: the found file (FILE B) or the not found file
: (FILE C). Depending on the users input, this routine calls
: either XFIND or XNOFIND routines.
;*****************************************************************
```

```
.XMIT
     D$0=""                      : Clears input register 0
     P"\e[2J"                    : Clears screen on reader
     P"  ENTER TO XMIT"          : Prompt user
     P"F1- FOUND FILE"
     P"F2- NOT FOUND FILE"
     P"\r"                       : Carriage return
     A2                          : Get ASCII input of 2 chars
     G$0="F1".XFIND              : Goto XFIND routine if F1
     G$0="F2".XNOFIND            : Goto XNOFIND routine if F2



:******************************************************************
: The XFIND routine is called from XMIT routine.  This routine
: finds how many records are in the file.  If there are no
: records in the file the routine calls NO_RECS routine.  This
: routine also prompts the user that the reader is transmitting.
:******************************************************************


.XFIND
     H#1=B                       : Puts the number of records in
                                 : FILE B into register 1
     G#1=0.NO_RECS               : Goto NO_RECS routine if FILE B
                                 : doesn't have any records
     D#2=0                       : Sets register 2 equal to 0 (the
                                 : first record in the file)
     P"\e[2J"                    : Clears screen on reader
     P"TRANSMITTING..."          : Tells user reader is
                                 : transmitting

:******************************************************************
: The TRANSFD routine transmits the found file (FILE B), one
: record at a time.  The file will be transmitted to the computer
: using intrscan software.  The file will be transmitted to a
: file called FOUND.DAT located in the intrscan directory on the
: computer.
:******************************************************************
```

```
.TRANSFD
     D$0=""                      : Clears input register 0
     D$0="~DATA_1_FOUND.DAT_"    : IRL command for intrsacan to put
                                 : the data appended to this
                                 : statement in a file called
                                 : FOUND.DAT
     D$0=$0+B(#2)                : #2 is a pointer that is pointing
                                 : to a record in FILE B.  That
                                 : record will be appended to the
                                 : statement above.
     XMP,$0                      : Transmit using protocol the
                                 : contents of register 0.
     D#2=#2+1                    : Increment the pointer
     G#2<#1.TRANFD               : Loop until the pointer equals the
                                 : number of records found in FILE B
     G.INV                       : Goto INV routine


:*****************************************************************
: The XNOFIND routine is called from XMIT routine.  This routine
: finds how many records are in the file.  If there are no
: records in the file the routine calls NO_RECS routine.  This
: routine also prompts the user that the reader is transmitting.
:*****************************************************************

.XNOFIND
     H#1=C                       : Puts the number of records in
                                 : FILE C into register 1
     G#1=0.NO_RECS               : Goto NO_RECS routine if FILE C
                                 : doesn't have any records
     D#2=0                       : Sets register 2 equal to 0 (the
                                 : first record in the file)
     P"\e[2J"                    : Clears screen on reader
     P"TRANSMITTING..."          : Tells user reader is
                                 : transmitting


:*****************************************************************
: The TRANSNO routine transmits the not found file (FILE C), one
: record at a time.  The file will be transmitted to the computer
: using intrscan software.  The file will be transmitted to a
: file called NOFIND.DAT located in the intrscan directory on the
: computer.
:*****************************************************************
```

```
.TRANSNO
     D$0=""                       : Clears input register 0
     D$0="~DATA_1_NOFIND.DAT_": IRL command for intrsacan to put
                                  : the data appended to this
                                  : statement in a file called
                                  : NOFIND.DAT
     D$0=$0+C(#2)                 : #2 is a pointer that is pointing
                                  : to a record in FILE C.  That
                                  : record will be appended to the
                                  : statement above.
     XMP,$0                       : Transmit using protocol the
                                  : contents of register 0.
     D#2=#2+1                     : Increment the pointer
     G#2<#1.TRANNO                : Loop until the pointer equals the
                                  : number of records found in FILE C
     G.INV                        : Goto INV routine


:****************************************************************
: NO_RECS is called from either XFIND or XNOFIND routines.  This
: routine is an exception handler.  If the user tries to transmit
: a file that doesn't have any records, the user will be prompted
: the file is empty and the program will return to the INV
: routine.
:****************************************************************

.NO_RECS
     B111111                      : Reader will beep a warning
     P"\e[2J"                     : Clears screen on reader
     P"NO RECORDS IN FILE"        : Prompts user
     W3                           : Waits 3 seconds
     G.INV                        : Goto INV routine
     E                            : End of program
```

```
--------------------------------------------------------------------------------------------
--TITLE          : LABELS package definition specifications
-- NAME          : Richard Hancock
-- DATE          : 20 Aug 1993
-- DESCRIPTION   : This package contains all operations associated with the object Labels.
--               :  Labels are created by extracting serial numbers from the CMR.  The
--               : serial numbers are placed in a file called LABEL.  These serial numbers
--               : are formatted the same as the reader input file (i.e. all the blanks are
--               : removed and the serial numbers are sorted in alpha numeric order.  The
--               : serial numbers are then appended with the necessary control characters
--               : for communication with the bar code printer .  This package contains the
--               : following operations: CREATE, OPEN_INPUT, CLOSE_INPUT,
--               : CLOSE_ OUTPUT, ADD_SN, PRINT LABEL and GET_SN .  A
--               : detailed description. of each of the operations is provided in the package
--               : body.
--------------------------------------------------------------------------------------------

 with SERIAL_NUMBERS; use SERIAL_NUMBERS;

package LABELS is

   procedure CREATE  (NAME : in STRING);

   procedure OPEN_INPUT   (NAME : in STRING);

   procedure CLOSE_INPUT  (NAME : in STRING);
   procedure CLOSE_OUTPUT (NAME : in STRING);

   procedure ADD_SN (SN : in SERIAL_NUMBER);
   procedure ADD_SN (SN : in STRING);

   procedure PRINT_LABEL (SN: in STRING; LENGTH: in INTEGER);

   procedure GET_SN (SN : out SERIAL_NUMBER);

   function ENDOF_FILE return BOOLEAN;

   CREATE_ERROR : exception;
   OPEN_ERROR   : exception;
   CLOSE_ERROR  : exception;


end LABELS;
```

```
-----------------------------------------------------------------------------------------------
-- TITLE            : LABELS package definition body
-- NAME             : Richard Hancock
-- DATE             : 20 Aug. 1993
-- DESCRIPTION      : This package contains all operations associated with the object Labels.
--                  :  Labels are created by extracting serial numbers from the CMR.  The
--                  :  serial numbers are placed in a file called LABEL.  These serial numbers
--                  :  are formatted the same as the reader input file (i.e. all the blanks are
--                  :  removed and the serial numbers are sorted in alpha numeric order.  The
--                  :  serial numbers are then appended with the necessary control characters
--                  :  for communication with the bar code printer .  This package contains the
--                  :  following operations:  CREATE,  OPEN_INPUT,  CLOSE_INPUT,
--                  :  CLOSE_ OUTPUT,  ADD_SN,  PRINT LABEL and GET_SN .  A
--                  :  detailed description of each of the operations is provided in the package
--                  :  body.
-----------------------------------------------------------------------------------------------

 with TEXT_IO; use TEXT_IO;

package body LABELS is

   INFILE            : FILE_TYPE;
   OUTFILE           : FILE_TYPE;
   END_OF_FILE       : BOOLEAN := false;

--------------------------------PROCEDURE CREATE----------------------------------------
-- This procedure creates a file to store the serial numbers extracted from the CMRL file.  Sets
--  the  default output to this file.
-----------------------------------------------------------------------------------------------

 procedure CREATE (NAME : in STRING) is

begin
   TEXT_IO.CREATE (OUTFILE, OUT_FILE, NAME);
   TEXT_IO.SET_OUTPUT (OUTFILE);

exception
   when others =>
      raise CREATE_ERROR;

end CREATE;
```

```
---------------------------------PROCEDURE OPEN_INPUT------------------------------------
-- This procedure opens the file storing serial numbers for input to format (ie remove blank
-- spaces) for printing bar code labels. Sets the default input to this file
------------------------------------------------------------------------------------------

procedure OPEN_INPUT (NAME : in STRING) is

begin
   TEXT_IO.OPEN (INFILE, IN_FILE, NAME);
   TEXT_IO.SET_INPUT (INFILE);
exception
   when others =>
      raise OPEN_ERROR;

end OPEN_INPUT;

----------------------------------PROCEDURE CLOSE_INPUT----------------------------------
-- This procedure closes the file, setting the default to standard input (ie keyboard)
------------------------------------------------------------------------------------------

procedure CLOSE_INPUT (NAME : in STRING) is

begin
   TEXT_IO.CLOSE (INFILE);
   TEXT_IO.SET_INPUT (TEXT_IO.STANDARD_INPUT);
exception
   when others =>
      raise CLOSE_ERROR;

end CLOSE_INPUT;

-------------------------------PROCEDURE CLOSE_OUTPUT------------------------------------
-- This procedure closes the file to output, setting the default to standard output (ie the
-- monitor).
------------------------------------------------------------------------------------------

procedure CLOSE_OUTPUT (NAME : in STRING) is

begin
   TEXT_IO.CLOSE (OUTFILE);
   TEXT_IO.SET_OUTPUT (TEXT_IO.STANDARD_OUTPUT);
exception
   when others =>
      raise CLOSE_ERROR;

end CLOSE_OUTPUT;
```

```
-----------------------------------PROCEDURE ADD_SN----------------------------------------
-- This procedure receives a serial number of private type serial number and calls the function
-- VALUE in package SERIAL NUMBERS which converts the serial number (to type string).
-- This procedure then writes the serial number to the file that is opened for current input .
-------------------------------------------------------------------------------------------

procedure ADD_SN (SN : in SERIAL_NUMBER) is

begin
   TEXT_IO.PUT_LINE (OUTFILE, SERIAL_NUMBERS.VALUE(SN));

end ADD_SN;

-----------------------------------PROCEDURE ADD_SN----------------------------------------
-- This procedure receives a serial number (type string) and writes the serial number to the
-- current file open for input
---------------------------------------------------------------------------------------------

 procedure ADD_SN (SN : in STRING) is

begin
   TEXT_IO.PUT_LINE (OUTFILE, SN);

end ADD_SN;

------------------------------------PROCEDURE PRINT_LABEL---------------------------------
-- This procedure receives a serial number of type string and appends the necessary control
-- characters for communication with the bar code printer to each serial number. The serial
-- number and control characters are then written to COM2 port for printing.
-------------------------------------------------------------------------------------------

procedure PRINT_LABEL (SN : in STRING; LENGTH : in INTEGER) is

    STX:  CHARACTER:= ASCII.STX;
    ESC:  CHARACTER:= ASCII.ESC;
    CAN:  CHARACTER:= ASCII.CAN;
    ETB:  CHARACTER:= ASCII.ETB;
    ETX:  CHARACTER:= ASCII.ETX;
    LABEL : STRING(1..LENGTH+7);

begin
   LABEL:=STX&ESC&"E1"&CAN&SN&ETB&ETX;
  TEXT_IO.PUT_LINE(OUTFILE,LABEL);

end PRINT_LABEL;
```

```
-----------------------------------PROCEDURE GET_SN-----------------------------------
-- This procedure gets a serial number from the current file open for output. Passes the serial
-- number (type string) to CREATE procedure in package SERIAL NUMBERS which returns a
-- serial number (private type serial number) with the same value. The procedure advances the
-- open file one line and returns the serial number(private type) to the calling procedure.
------------------------------------------------------------------------------------------

procedure GET_SN (SN : out SERIAL_NUMBER) is

    VALUE : STRING (1..22);
    LAST   : INTEGER;

begin
   TEXT_IO.GET (INFILE,VALUE);
   SERIAL_NUMBERS.CREATE (VALUE, SN);
   TEXT_IO.skip_line (INFILE);
exception
   when TEXT_IO.END_ERROR =>
      END_OF_FILE := true;

end GET_SN;

--------------------------------FUNCTION ENDOF_FILE--------------------------------------
-- This function returns the value of the END_OF_FILE flag
------------------------------------------------------------------------------------------

function ENDOF_FILE return BOOLEAN is

begin
    return TEXT_IO.END_OF_FILE;

end ENDOF_FILE;


------------------------------------------------------------------------------------------

 end LABELS;
```

```
-----------------------------------------------------------------------------------------
-- TITLE            : Print Batch Labels
-- NAME             : Richard M. Hancock
-- DATE             : 20 Aug 1993
-- DESCRIPTION      : This procedure is the main driver the for the print bar code label protion
--                  :  of the BCIS. This part of the system opens the CMRL file (ASCII file)
--                  :  provided by the user and extracts all the serial numbers into a Label file.
--                  :  These serial numbers are then formatted (any blanks at the front of the
--                  :  serial numbers are removed) and the file is sorted into alpha-numeric
--                  :  ascending order. This program uses the same code as the download
--                  :  portion of the system to ensure the serial numbers downloaded to the
--                  : reader match the serial numbers on the labels. This program then
--                  : appends the necessary control characters for communication with the bar
--                  : code printer to each serial number and sends this info to the bar code
--                    printer via COM port 2. The objects used in this portion ofthe system are
--                    CMRL, READER_INPUT, SERIAL_NUMBERS, SN_ARRAY, and
--                  : FILE_PROCESSOR.
-----------------------------------------------------------------------------------------
 with CMRL; with LABELS;
 with SERIAL_NUMBERS; use SERIAL_NUMBERS;
 with SN_ARRAY; use SN_ARRAY;
 with FILE_PROCESSOR; use FILE_PROCESSOR;
 with TEXT_IO; use TEXT_IO;

procedure PRN_BATCH is

  CMRL_FILE_NAME         : STRING (1..4) := "CMRL";
  LABEL_FILE_NAME        : STRING (1..6) := "LABELS";
  OUTPUT_FILE_NAME       : constant STRING := "COM2";

  SN            : SERIAL_NUMBER;
  SN_COUNT      : INTEGER := 0;
  ARR_PTR       : ARRAY_POINTER;

begin
  CMRL.OPEN (CMRL_FILE_NAME);
  LABELS.CREATE (LABEL_FILE_NAME);
  loop
     CMRL.GET_SN (SN);
     exit when CMRL.ENDOF_FILE;
     SN_COUNT := SN_COUNT + 1;
     LABELS.ADD_SN (SN);
  end loop;
```

```
   CMRL.CLOSE(CMRL_FILE_NAME);
   LABELS.CLOSE_OUTPUT(LABEL_FILE_NAME);
   SN_ARRAY.CREATE_ARRAY (SN_COUNT, ARR_PTR);
   FORMAT_LABELS_FILE (SN_COUNT, LABEL_FILE_NAME, OUTPUT_FILE_NAME,
                               ARR_PTR);

exception
   when CMRL.OPEN_ERROR =>
      TEXT_IO.PUT_LINE (STANDARD_OUTPUT, "Error OPENING file CMRL");

   when CMRL.CLOSE_ERROR =>
      TEXT_IO.PUT_LINE (STANDARD_OUTPUT, "Error CLOSING file CMRL");

   when LABELS.CREATE_ERROR =>
      TEXT_IO.PUT_LINE (STANDARD_OUTPUT, "Error CREATING file LABELS");

   when LABELS.OPEN_ERROR =>
      TEXT_IO.PUT_LINE (STANDARD_OUTPUT, "Error OPENING file LABELS");

   when LABELS.CLOSE_ERROR =>
      TEXT_IO.PUT_LINE (STANDARD_OUTPUT, "Error CLOSING file LABELS");


end PRN_BATCH;
```

```
------------------------------------------------------------------------------------------
--TITLE           : Print Individual Labels
-- NAME           : Richard M. Hancock
-- DATE           : 22 Aug 1993
-- DESCRIPTION    : This procedure is the main driver the for printing individual bar code
--                : labels. This program queries the user to enter a serial number. The serial
--                : number is then appended with the necessary control characters for
--                : communication with a bar code printer. The serial number and control
--                : characters are then sent to the printer via the COM2 port. The object
--                : used in this portion of the system is LABEL.
------------------------------------------------------------------------------------------

with LABELS; with TTY;
with CURSOR; use CURSOR;
with COMMON_DISPLAY_TYPES; use COMMON_DISPLAY_TYPES;
with TEXT_IO; use TEXT_IO;

procedure PRN_INDIV is

   OUTPUT_FILE_NAME       : constant STRING := "COM2";
   SN      : STRING(1..22);
   LENGTH  : INTEGER := 0;
   NO_BLINK: BOOLEAN := false;
   FINISHED: BOOLEAN := false;
   BLANK   : constant STRING:= "                                                ";
   ASK     : constant STRING:= "  ENTER A SERIAL NUMBER OR 'QUIT' TO EXIT  ";
   BACK    : COLOR   := blue;
   FORE    : COLOR   := bright_white;
   ESC     : CHARACTER := ASCII.ESC;

begin
  while not FINISHED loop
     TTY.CLEAR_SCREEN;
     TTY.PUT(10,15,BLANK,FORE,BACK,NO_BLINK);
     TTY.PUT(11,15,ASK,FORE,BACK,NO_BLINK);
     TTY.PUT(12,15,BLANK,FORE,BACK,NO_BLINK);
     TTY.PUT(13,15,BLANK,FORE,BACK,NO_BLINK);
     CURSOR.MOVE(13,30);
     TEXT_IO.GET_LINE(SN,LENGTH);
     if SN(1..LENGTH) = "QUIT" or SN(1..LENGTH) = "Quit"
              or SN(1..LENGTH) = "quit" then
          FINISHED := true;
     else
          LABELS.CREATE (OUTPUT_FILE_NAME);
          LABELS.PRINT_LABEL(SN(1..LENGTH),LENGTH);
          FINISHED := false;
```

```
            LABELS.CLOSE_OUTPUT (OUTPUT_FILE_NAME);
        end if;
    end loop;

exception
    when LABELS.CREATE_ERROR =>
        TEXT_IO.PUT_LINE (STANDARD_OUTPUT, "Error CREATING file LABELS");

    when LABELS.CLOSE_ERROR =>
        TEXT_IO.PUT_LINE (STANDARD_OUTPUT, "Error CLOSING file LABELS");


 end PRN_INDIV;
```

```
--------------------------------------------------------------------------------
-- TITLE         : READER INPUT package definition specifications
-- NAME          : Richard Hancock
-- DATE          : 28 July 1993
-- DESCRIPTION   : This package contains all operations associated with the object reader
--               :  input. READER INPUT is a file that contains only serial numbers that
--               :  were extracted from the CMRL. This file will be downloaded to the bar
--               :  code reader before an inventory is conducted. The operations contained
--               :  in this package are CREATE, OPEN INPUT, OPEN OUTPUT, CLOSE
--               :  INPUT, CLOSE OUTPUT, ADD SERIAL NUMBER, GET SERIAL
--               :  NUMBER, and END OF FILE. A detailed description of each of these
--               :  operations are contained in the Package Body.
--------------------------------------------------------------------------------

 with SERIAL_NUMBERS; use SERIAL_NUMBERS;

package READER_INPUT is

   procedure CREATE (NAME : in STRING);

   procedure OPEN_INPUT   (NAME : in STRING);
   procedure OPEN_OUTPUT  (NAME : in STRING);

   procedure CLOSE_OUTPUT (NAME : in STRING);
   procedure CLOSE_INPUT  (NAME : in STRING);

   procedure ADD_SN (SN : in SERIAL_NUMBER);
   procedure ADD_SN (SN : in STRING);

   procedure GET_SN (SN : out SERIAL_NUMBER);

   function ENDOF_FILE return BOOLEAN;

   CREATE_ERROR : exception;
   OPEN_ERROR   : exception;
   CLOSE_ERROR  : exception;


end READER_INPUT;
```

```
------------------------------------------------------------------------------------
-- TITLE          : READER INPUT package definition body
-- NAME           : Richard Hancock
-- DATE           : 28 July 1993
-- DESCRIPTION    : This package contains all operations associated with the object reader
--                :  input.  READER INPUT is a file that contains only serial numbers that
--                :  were extracted from the CMRL.  This file will be downloaded to the bar
--                :  code reader before an inventory is conducted.  The operations contained
--                :  in this package are CREATE,  OPEN INPUT,  OPEN OUTPUT,
--                :  CLOSE INPUT, CLOSE OUTPUT, ADD SERIAL NUMBER, GET
--                :  SERIAL NUMBER, and END OF FILE.  A detailed description of each
--                :  of these operations are contained in the Package Body.
------------------------------------------------------------------------------------

 with TEXT_IO; use TEXT_IO;

package body READER_INPUT is

   INFILE          : FILE_TYPE;
   OUTFILE         : FILE_TYPE;
   END_OF_FILE     : BOOLEAN := false;

-----------------------------PROCEDURE CREATE-----------------------------------
-- This procedure creates a file to store the serial numbers extracted from the CMRL file.  Sets
--  the default output to this file
------------------------------------------------------------------------------------

 procedure CREATE (NAME : in STRING) is

begin
   TEXT_IO.CREATE (OUTFILE, OUT_FILE, NAME);
   TEXT_IO.SET_OUTPUT (OUTFILE);
exception
   when others =>
      raise CREATE_ERROR;

end CREATE;


-----------------------------PROCEDURE OPEN_INPUT-------------------------------
-- This procedure opens the file storing serial numbers for input to format (ie remove blank
-- spaces) for bar code reader input.  Sets the default input to this file
------------------------------------------------------------------------------------
```

```
procedure OPEN_INPUT (NAME : in STRING) is

begin
   TEXT_IO.OPEN (INFILE, IN_FILE, NAME);
   TEXT_IO.SET_INPUT (INFILE);
exception
   when others =>
      raise OPEN_ERROR;

end OPEN_INPUT;

-------------------------------PROCEDURE OPEN_OUTPUT-----------------------------------
-- This procedure opens the file storing serial numbers for output to hold formatted serial
-- numbers for bar code reader input. Sets default output to this file.
-------------------------------------------------------------------------------------------

procedure OPEN_OUTPUT (NAME : in STRING) is

begin
   TEXT_IO.OPEN (OUTFILE, OUT_FILE, NAME);
   TEXT_IO.SET_OUTPUT (OUTFILE);
exception
   when others =>
      raise OPEN_ERROR;

end OPEN_OUTPUT;

------------------------------PROCEDURE CLOSE_OUTPUT-----------------------------------
-- This procedure closes the file to output, setting the default to standard output (ie the monitor).
-------------------------------------------------------------------------------------------

 procedure CLOSE_OUTPUT (NAME : in STRING) is

begin
   TEXT_IO.CLOSE (OUTFILE);
  TEXT_IO.SET_OUTPUT (TEXT_IO.STANDARD_OUTPUT);
exception
  when others =>
      raise CLOSE_ERROR;

end CLOSE_OUTPUT;
```

```
-------------------------------PROCEDURE CLOSE_INPUT-------------------------------
-- This procedure closes the file, setting the default to standard input (ie keyboard)
-----------------------------------------------------------------------------------

procedure CLOSE_INPUT (NAME : in STRING) is

begin
   TEXT_IO.CLOSE (INFILE);
   TEXT_IO.SET_INPUT (TEXT_IO.STANDARD_INPUT);
exception
   when others =>
      raise CLOSE_ERROR;

end CLOSE_INPUT;

-----------------------------------PROCEDURE ADD_SN-----------------------------------
-- This procedure receives a serial number of private type serial number and calls the function
-- VALUE in package SERIAL NUMBERS which converts the serial number (to type string).
-- This procedure then writes the serial number to the file that is opened for current input.
-----------------------------------------------------------------------------------

procedure ADD_SN (SN : in SERIAL_NUMBER) is

begin
   TEXT_IO.PUT_LINE (OUTFILE, SERIAL_NUMBERS.VALUE(SN));

end ADD_SN;

-----------------------------------PROCEDURE ADD_SN-----------------------------------
-- This procedures receives a serial number (type string) and writes the serial number to the
-- current file open for input.
-----------------------------------------------------------------------------------

 procedure ADD_SN (SN : in STRING) is

begin
   TEXT_IO.PUT_LINE (OUTFILE, SN);

end ADD_SN;

-----------------------------------PROCEDURE GET_SN-----------------------------------
-- This procedure gets a serial number from the current file open for output. Passes the serial
-- number (type string) to CREATE procedure in package SERIAL NUMBERS which returns a
-- serial number (private type serial number) with the same value. The procedure advances the
-- open file one line and returns the serial number(private type) to the calling procedure.
-----------------------------------------------------------------------------------
```

```
procedure GET_SN (SN : out SERIAL_NUMBER) is

    VALUE   : STRING (1..22);
    LAST    : INTEGER;

begin
   TEXT_IO.GET (INFILE,VALUE);
   SERIAL_NUMBERS.CREATE (VALUE, SN);
  TEXT_IO.SKIPLINE (INFILE);
exception
   when TEXT_IO.END_ERROR =>
      END_OF_FILE := true;

end GET_SN;

----------------------------------FUNCTION ENDOF_FILE----------------------------------
-- This function returns the value of the END_OF_FILE flag
---------------------------------------------------------------------------------------

function ENDOF_FILE return BOOLEAN is

begin
   return TEXT_IO.END_OF_FILE;

end ENDOF_FILE;



---------------------------------------------------------------------------------------
end READER_INPUT;
```

```
---------------------------------------------------------------------------------
-- TITLE          : READER OUTPUT package definition specifications
-- NAME           : Richard Hancock
-- DATE           : 28 July 1993
-- DESCRIPTION    : This package contains all operations associated with the object reader
--                : output. READER OUTPUT is a file that contains only serial numbers
--                : that were uploaded from the barcode reader. These files will be formated
--                : for processing into reports. The operations contained in this package are
--                : OPEN, CLOSE, GET SERIAL NUMBER AND END OF FILE. A
--                : detailed description of each of these operations are contained in the
--                : Package Body.
---------------------------------------------------------------------------------

package READER_OUTPUT is


   procedure  OPEN (NAME : in STRING);

   procedure CLOSE (NAME : in STRING);

   procedure GET_SN (SN : out STRING);

   procedure GET_DESCRIPTION (ASSET_DESCRIPTION : out STRING);

   procedure GET_LOCATION (ASSET_LOCATION : out STRING);

  function ENDOF_FILE return BOOLEAN;

   OPEN_ERROR  : exception;
   CLOSE_ERROR : exception;


end READER_OUTPUT;
```

```
--------------------------------------------------------------------------------
-- TITLE            : READER OUTPUT package definition body
-- NAME             : Richard Hancock
-- DATE             : 28 July 1993
-- DESCRIPTION      : This package contains all operations associated with the object reader
--                  :  output.  READER OUTPUT is a file that contains  serial numbers that
--                  :  were uploaded from the barcode reader.  These files will be formated
--                  :  for processing into reports.  The operations contained in this package are
--                  :  OPEN, CLOSE, GET SERIAL NUMBER, GET LOCATION, GET
--                  :  DESCRIPTION and END OF FILE.  A detailed description of each of
--                  :  these operations are contained in the Package Body.
--------------------------------------------------------------------------------

with TEXT_IO; use TEXT_IO;

package body READER_OUTPUT is

   READER_OUTFILE   : TEXT_IO.FILE_TYPE;
   END_OF_FILE      : BOOLEAN := false;



--------------------------------PROCEDURE OPEN----------------------------------
-- This procedure opens the file generated by the bar code reader and sets the default input to the
-- given file name.
--------------------------------------------------------------------------------

 procedure OPEN (NAME : in STRING) is

begin
   TEXT_IO.OPEN (READER_OUTFILE, IN_FILE, NAME);
   TEXT_IO.SET_INPUT (READER_OUTFILE);
exception
   when others =>
      raise OPEN_ERROR;

end OPEN;



---------------------------------PROCEDURE CLOSE---------------------------------
-- This procedure closes the output file produced by the bar code reader and sets the default
-- input to standard input (ie keyboard).
--------------------------------------------------------------------------------
```

```
procedure CLOSE (NAME : in STRING) is

begin
   CLOSE (READER_OUTFILE);
   TEXT_IO.SET_INPUT (TEXT_IO.STANDARD_INPUT);
exception
   when others =>
      raise CLOSE_ERROR;

end CLOSE;


-------------------------------PROCEDURE GET_SN------------------------------------
-- This procedure gets a serial number from the bar code reader output file. This procedure
-- counts whow many characters are in the serial number and pads the front of the serial number
-- with blanks to return the serial number to a 22 character field
-----------------------------------------------------------------------------------

procedure GET_SN (SN : out STRING) is

   VALUE              : STRING (1..22);
   LAST               : NATURAL;
   BLANK_COUNT        : NATURAL;

begin
   TEXT_IO.GET_LINE (READER_OUTFILE, VALUE, LAST);      -input a sn
  BLANK_COUNT := 22 - LAST;                            -- and then pad it
   for I in 1..BLANK_COUNT loop                          -- with blanks to
      SN (I..I) := " ";                                  -- get 22 chars.
   end loop;
  SN (BLANK_COUNT+1..22) := VALUE (1..LAST);
exception
   when TEXT_IO.END_ERROR =>
         END_OF_FILE := true;

end GET_SN;


-----------------------PROCEDURE GET_DESCRIPTION------------------------------
-- This procedure gets the description of the asset from the not found file that was created during
-- the inventory. The description is returned to the procedure FORMAT_EXCEPTION_FILE to
-- be processed into exception report.
-----------------------------------------------------------------------------------
```

```
procedure GET_DESCRIPTION (ASSET_DESCRIPTION : out STRING) is

   VALUE : STRING (1..30);
   LAST   : NATURAL;

begin
  TEXT_IO.GET_LINE (READER_OUTFILE, VALUE, LAST);
  for I in LAST+1..30 loop
     VALUE(I..I):= " ";
  end loop;
  ASSET_DESCRIPTION := VALUE;
exception
  when TEXT_IO.END_ERROR =>
     END_OF_FILE := true;

end GET_DESCRIPTION;


-------------------------PROCEDURE GET_LOCATION------------------------------------
-- This procedure gets the location of the asset from the not found file that was created during
-- the inventory. The location is returned to the procedure FORMAT_EXCEPTION_FILE to be
-- processed into the exception report.
------------------------------------------------------------------------------------


procedure GET_LOCATION (ASSET_LOCATION : out STRING) is

   VALUE : STRING (1..18);
   LAST   : NATURAL;

begin
  TEXT_IO.GET_LINE (READER_OUTFILE, VALUE, LAST);
  for I in LAST+1..18 loop
     VALUE(I..I):= " ";
  end loop;
  ASSET_LOCATION := VALUE;
exception
  when TEXT_IO.END_ERROR =>
     END_OF_FILE := true;

end GET_LOCATION;
```

```
---------------------------------FUNCTION ENDOF_FILE---------------------------------
--  This function returns the END_OF_FILE flag variable.
--------------------------------------------------------------------------------------

function ENDOF_FILE return BOOLEAN is

begin
   return END_OF_FILE;

end ENDOF_FILE;

--------------------------------------------------------------------------------------
end READER_OUTPUT;
```

```
--------------------------------------------------------------------------------------------
-- TITLE            : REPORT package definition specifications
-- NAME             : Richard Hancock
-- DATE             : 28 July 1993
-- DESCRIPTION      : This package contains all operations associated with the object report.
--                  :  REPORT is a file that replicates the orignial CMRL but the serial
--                  : numbers that were found during the inventory were annotated with a flag.
--                  :  XREPORT is a file that contains assets that were found during the
--                  : inventory but were not on the CMRL.  This report contains the serial
--                  : number, description and location of the assets. The operations contained
--                  : in this package are CREATE, CLOSE, ADD LINE and
--                  : FORMAT_EXCEPTION_REPORT.  A detailed description of each of
--                  : these operations are contained in the Package Body.
--------------------------------------------------------------------------------------------

 with TEXT_IO; use TEXT_IO;

package REPORT is


   procedure CREATE (NAME : in STRING);

   procedure  CLOSE (NAME : in STRING);

   procedure ADD_LINE (LINE : in STRING);

   procedure FORMAT_EXCEPTION_REPORT;

   CREATE_ERROR : exception;

   CLOSE_ERROR  : exception;


end REPORT;
```

```
--------------------------------------------------------------------------------------------------------
-- TITLE           : REPORT package definition body
-- NAME            : Richard Hancock
-- DATE            : 28 July 1993
-- DESCRIPTION     : This package contains all operations associated with the object report.
--                 : REPORT is a file that replicates the orignial CMRL but the serial
--                 : numbers that were found during the inventory were annotated with a flag.
--                 : XREPORT is a file that contains assets that were found during the
--                 : inventory but were not on the CMRL. This report contains the serial
--                 : number, description and location of the assets. The operations contained
--                 : in this package are CREATE, CLOSE, ADD LINE and
--                 : FORMAT_EXCEPTION_REPORT. A detailed description of each of
--                 : these operations are contained in the Package Body.
--------------------------------------------------------------------------------------------------------

with TEXT_IO; use TEXT_IO;

package body REPORT is

   REPORT_FILE               : FILE_TYPE;
   MORE_SERIAL_NUMBERS       : BOOLEAN := FALSE;
   SN_COUNT                  : INTEGER::= 0;


----------------------------------PROCEDURE CREATE------------------------------------------
-- This procedure creates the output report file with the given name and sets the default output to
-- that file.
--------------------------------------------------------------------------------------------------------

procedure CREATE (NAME : in STRING) is

begin
   TEXT_IO.CREATE (REPORT_FILE, OUT_FILE, NAME);
   TEXT_IO.SET_OUTPUT (REPORT_FILE);
exception
   when others =>
      raise CREATE_ERROR;

end CREATE;


------------------------------------PROCEDURE CLOSE---------------------------------------
-- This procedure closes the CMRL report file of the given name and sets the default output back
-- to standard output (ie monitor).
--------------------------------------------------------------------------------------------------------
```

```
procedure CLOSE (NAME : in STRING) is

begin
   CLOSE (REPORT_FILE);
   TEXT_IO.SET_OUTPUT (TEXT_IO.STANDARD_OUTPUT);
exception
   when others =>
      raise CLOSE_ERROR;

end CLOSE;

------------------------------------PROCEDURE ADD_LINE--------------------------------
-- This procedure adds a line to the CMRL report file.
--------------------------------------------------------------------------------------
procedure ADD_LINE (LINE : in STRING) is

begin
   PUT_LINE (REPORT_FILE, LINE);

end ADD_LINE;

-----------------------PROCEDURE FORMAT_EXCEPTION_REPORT-----------------------
-- This procedure formats the exception report.
--------------------------------------------------------------------------------------
procedure FORMAT_EXCEPTION_REPORT is

    TITLE       : STRING (1..16):= "EXCEPTION REPORT";
    HEADER1     : STRING (1..13):= "Serial Number";
    HEADER2     : STRING (1..11):= "Description";
    HEADER3     : STRING (1..8):= "Location";

begin
   NEW_LINE(3);
   SET_COL(32);
   PUT_LINE(TITLE);
   NEW_LINE(2);
   SET_COL(10);
   PUT(HEADER1);
   SET_COL(28);
   PUT(HEADER2);
   SET_COL(62);
   PUT_LINE(HEADER3);
   NEW_LINE;
end FORMAT_EXCEPTION_REPORT;
--------------------------------------------------------------------------------------
 end REPORT;
```

```
---------------------------------------------------------------------------------------------
-- TITLE           : Serial Number Array definition package specifications
-- NAME            : Richard Hancock
-- DATE            : 17 July 1993
-- DESCRIPTION     : This package contains all operations associated with the object Serial
--                 : Number Array. This object was not developed during the design phase,
--                 : but  was created to help with the implementation of the system.  This
--                 : object: is used to sort the serial number files and flag a serial number as
--                 : found. The package contains operations that will CREATE an array,
--                 : INSERT a serial number into the array, SORT an array by value, SORT
--                 : an array by initial file position, MARK a serial number as found, and
--                 : removes as serial number from the Array.  A detailed description of each
--                 : of the operations is provided in the package body.
---------------------------------------------------------------------------------------------

 with SERIAL_NUMBERS; use SERIAL_NUMBERS;

package SN_ARRAY is

   type ARRAY_RECORD is
      record
         DATA : SERIAL_NUMBER;
        KEY  : INTEGER;
     end record;

   type SER_ARRAY is array (INTEGER RANGE <>) of ARRAY_RECORD;
   type ARRAY_POINTER is access SER_ARRAY;


  procedure CREATE_ARRAY (SN_COUNT : in INTEGER; ARR_PTR : out
                                    ARRAY_POINTER);
  procedure INSERT_SN (INDEX : in INTEGER; VALUE : in SERIAL_NUMBER;
                                 ARR_PTR : in ARRAY_POINTER);
  procedure SORT_BY_VALUE(ARRAY_SIZE : in INTEGER; ARR_PTR : in
                                    ARRAY_POINTER);
  procedure SORT_BY_KEY (ARRAY_SIZE : in INTEGER; ARR_PTR : in
                                 ARRAY_POINTER);
  procedure MARK_SN_FOUND (SN_STR : in STRING; ARR_PTR : in
                                      ARRAY_POINTER);
  function REMOVE_SN (INDEX : in INTEGER; ARR_PTR : in ARRAY_POINTER)
                          return SERIAL_NUMBER;


 end SN_ARRAY;
```

```
-------------------------------------------------------------------------------
-- TITLE          : Serial Number Array definition package body
-- NAME           : Richard Hancock
-- DATE           : 17 July 1993
-- DESCRIPTION    : This package contains all operations associated with the object Serial
--                : Number Array. This object was not developed during the design phase,
--                : but was created to help with the implementation of the system. This
--                : object is used to sort the serial number files and flag a serial number as
--                : found. The package contains operations that will CREATE an array,
--                : INSERT a serial number into the array, SORT an array by value, SORT
--                : an array by initial file position, MARK a serial number as found, and
--                : removes a serial number from the Array. A detailed description of each
--                : of the operations is provided in the package body.
-------------------------------------------------------------------------------

with TEXT_IO; use TEXT_IO;

package body SN_ARRAY is

-----------------------------PROCEDURE CREATE_ARRAY-----------------------------
-- This procedure creates an array with a size equal to the number of serial numbers found in the
-- CMRL. The procedure returns an array pointer to the calling procedure, which points to the
-- memory location of the array. This pointer allows us to access and preserve the contents of
-- the array after a procedure is closed. This is a benefit because we do not have to pass an entire
-- array from procedure to procedure, which could result in the data being erroneously
-- modified.
-------------------------------------------------------------------------------


 procedure CREATE_ARRAY (SN_COUNT : in INTEGER; ARR_PTR : out
                          ARRAY_POINTER) is

begin
   ARR_PTR := new SER_ARRAY (1..SN_COUNT);
end CREATE_ARRAY;


-----------------------------PROCEDURE INSERT_SN--------------------------------
 --  This procedure inserts the serial number value and the original file position of the serial
 --  number into an array record element. The file position is saved for when we put the serial
 --  numbers back in the CMRL with a found flag.
-------------------------------------------------------------------------------
```

```
procedure INSERT_SN (INDEX : in INTEGER; VALUE : in SERIAL_NUMBER;
                                        ARR_PTR : in ARRAY_POINTER) is

begin
    ARR_PTR(INDEX).DATA := VALUE;
    ARR_PTR(INDEX).KEY  := INDEX;
end INSERT_SN;


------------------------------PROCEDURE SORT_BY_VALUE------------------------------
-- This procedure conducts a shell sort of the array by serial number value, placing the serial
-- numbers in ascending alpha-numeric order.  This procedure was obtained from a book called
-- ALGORITHMS written by ROBERT SEDGEWICK (1984).  This procedure is found on
-- page 98.
-------------------------------------------------------------------------------------


procedure SORT_BY_VALUE (ARRAY_SIZE : in INTEGER; ARR_PTR : in
                                             ARRAY_POINTER) is

   TEMP_REC : ARRAY_RECORD;
   TEMP_SN  : SERIAL_NUMBER;
   H, J     : INTEGER;

begin
  H := 1;
  loop
    H := 3*H+1;
    exit when H > ARRAY_SIZE;
  end loop;
  loop
    H := H / 3;
    for I in H+1..ARRAY_SIZE loop
      TEMP_REC := ARR_PTR (I);
      TEMP_SN := ARR_PTR(I).DATA;
      J := I;
      while ARR_PTR(J-H).DATA > TEMP_SN loop
         ARR_PTR (J) := ARR_PTR (J-H);
         J := J-H;
         exit when J <= H;
      end loop;
      ARR_PTR (J) := TEMP_REC;
    end loop;
    exit when H = 1;
  end loop;
```

```
exception
  when CONSTRAINT_ERROR =>
    text_io.put_line (standard_output, "--CONSTRAINT_ERROR in
                         ARRAY_SORT_BY_BALUE procedure.--");
end SORT_BY_VALUE;


-------------------------------PROCEDURE SORT_BY_KEY------------------------------
--  This procedure conducts a shell sort of an array by key, which places the serial numbers back
--  in the order that they were read in from the CMRL file. This procedure was obtained from a
--  book called ALGORITHMS written by ROBERT SEDGEWICK (1984). This procedure is
--  found on page 98.
-----------------------------------------------------------------------------------


procedure SORT_BY_KEY (ARRAY_SIZE : in INTEGER; ARR_PTR : in
                              ARRAY_POINTER) is

   TEMP_REC    : ARRAY_RECORD;
   TEMP_KEY    : INTEGER;
   H, J        : INTEGER;

begin
  H := 1;
  loop
    H := 3*H+1;
    exit when H > ARRAY_SIZE;
  end loop;
  loop
    H := H / 3;
    for I in H+1..ARRAY_SIZE loop
      TEMP_REC := ARR_PTR (I);
      TEMP_KEY := ARR_PTR(I).KEY;
      J := I;
      while ARR_PTR(J-H).KEY > TEMP_KEY loop
        ARR_PTR (J) := ARR_PTR (J-H);
        J := J-H;
        exit when J <= H;
      end loop;
      ARR_PTR (J) := TEMP_REC;
    end loop;
   exit when H = 1;
  end loop;
```

```
exception

   when CONSTRAINT_ERROR =>
       text_io.put_line (standard_output, "-- CONSTRAIN_ERROR in ARRAY
SORT_BY_KEY                               procedure.--");

end SORT_BY_KEY;


------------------------------PROCEDURE MARK_FOUND--------------------------------
--  This procedure is passed a serial number (string of 22 chars). The procedure opens the serial
--  number array and searches the array for a serial number that matches.  When a match is
--  found the MARK procedure in the package SERIAL NUMBERS is called and appends a
--  flag to the end of the serial number.  This serial number is then placed back into the array
.---------------------------------------------------------------------------------------


 procedure MARK_SN_FOUND (SN_STR:in STRING; ARR_PTR:in ARRAY_POINTER) is


   I           : INTEGER := 1;
   VALUE   : STRING (1..22);


begin
   VALUE := SERIAL_NUMBERS.VALUE (ARR_PTR (I).DATA);
   while ( VALUE (22..22) = "F" ) or ( SN_STR /= SERIAL_NUMBERS.VALUE (ARR_PTR
              (I).DATA) ) loop
     I := I + 1;
     VALUE := SERIAL_NUMBERS.VALUE (ARR_PTR (I).DATA);
   end loop;
   SERIAL_NUMBERS.MARK (ARR_PTR(I).DATA);

exception

   when CONSTRAINT_ERROR =>
     text_io.put_line (standard_output, "--CONSTRAINT_ERROR in ARRAY
                          MARK_SN_FOUND procedure.--");

end MARK_SN_FOUND;
```

```
---------------------------------FUNCTION  REMOVE_SN--------------------------------------
-- This function given an index value and the location of an array goes into the array and returns
-- the serial number value found in the position specified by the index value.
------------------------------------------------------------------------------------------


function REMOVE_SN (INDEX : in INTEGER; ARR_PTR : in  ARRAY_POINTER)
                             return SERIAL_NUMBER is

begin
   return ARR_PTR(INDEX).DATA;

end REMOVE_SN;



-----------------------------------------------------------------
 end SN_ARRAY;
```

```
------------------------------------------------------------------------------------------------
-- TITLE          : SERIAL NUMBERS package definition specifications
-- NAME           : Richard Hancock
-- DATE           : 28 July 1993
-- DESCRIPTION    : This package contains all operations associated with the object serial
--                : numbers.  Serial numbers are alphanumeric and can be up to 23
--                : characters in length. This package defines serial numbers as a private
--                : type to enhance the object oriented principle of information hiding.  By
--                : declaring serial numbers a private type we limit the operations that can be
--                : performed on serial numbers outside this package.  The operations
--                : contained in this package are:  CREATE (converts type string to type
--                : serial number), MARK (places a flag at the end of found serial numbers),
--                : VALUE (converts type serial number to type string), and ">" (compares
--                : two serial numbers and returns a boolean). A detailed description of each
--                : of these operations are provided in the Package Body.
------------------------------------------------------------------------------------------------

package SERIAL_NUMBERS is

   type SERIAL_NUMBER is private;


  procedure CREATE (VALUE : in STRING; SN : out SERIAL_NUMBER);

  procedure MARK (SN : in out SERIAL_NUMBER);

  function VALUE (SN : SERIAL_NUMBER) return STRING;

  function ">" (SN1,SN2 : in SERIAL_NUMBER) return BOOLEAN;


  private   type SERIAL_NUMBER is
     record
        VALUE : STRING (1..22);
     end record;


end SERIAL_NUMBERS;
```

```
------------------------------------------------------------------------------
-- TITLE          : SERIAL NUMBERS package definition body
-- NAME           : Richard Hancock
-- DATE           : 28 July 1993
-- DESCRIPTION    : This package contains all operations associated with the object serial
--                : number.  Serial numbers are alphanumeric and can be up to 23 characters
--                : in length. This package defines serial numbers as a private type to
--                : enhance the object oriented principle of information hiding.  By declaring
--                : serial numbers a private type we limit the operations that can be
--                : performed on serial numbers outside this package.  The operations
--                : contained in this package are CREATE (converts type string to type
--                : serial number), MARK (places a flag at the end of found serial numbers)
--                : and VALUE (converts type serial number to type string). A detailed
--                : description of each of these operations are provided in the Package Body
------------------------------------------------------------------------------

 with TEXT_IO; use TEXT_IO;

package body SERIAL_NUMBERS is

--------------------------------PROCEDURE CREATE--------------------------------
-- This procedure accepts a serial number value (of type string) and creates a new serial number
-- (of private type serial number) with the same value.  This preserves the integrity of the serial
-- numbers because using this type limits the operations performed outside this package.
------------------------------------------------------------------------------

procedure CREATE (VALUE : in STRING; SN : out SERIAL_NUMBER) is

begin
   SN.VALUE := VALUE;

end CREATE;

--------------------------------PROCEDURE MARK--------------------------------
-- This procedure removes two blanks from the front of the serial number and appends a blank
-- and an F to flag the serial number as found.  This procedure assumes the length of a serial
-- number will not exceed 22 characters.
------------------------------------------------------------------------------

procedure MARK (SN : in out SERIAL_NUMBER) is

begin
   SN.VALUE := SN.VALUE (3..22) & " F";

end MARK;
```

```
----------------------------------FUNCTION  VALUE----------------------------------------------
-- This function accepts a serial number (of private type serial  number) and converts it to type
-- string.  This is necessary so we  can perform necessary functions to the serial number outsidc
-- this package.
-----------------------------------------------------------------------------------------------

 function VALUE (SN : SERIAL_NUMBER) return STRING is

begin
    return SN.VALUE;

end VALUE;

------------------------------------------FUNCTION ">"-----------------------------------------------
-- This function is used to compare the values of two serial numbers and return a boolean value
-- if the first serial number is greater that the second.  This function is used in the Package
--  SN_ARRAY during the procedure SORT_BY_VALUE.
-----------------------------------------------------------------------------------------------

function ">" (SN1,SN2 : in SERIAL_NUMBER) return BOOLEAN is

begin
   return SN1.VALUE > SN2.VALUE;

end ">";


-----------------------------------------------------------------------------------------------
end SERIAL_NUMBERS:
```

APPENDIX B

BCIS USER'S MANUAL

A. INTRODUCTION

The Bar Code Inventory System (BCIS) is an automated inventory system designed to provide Marine Corps Systems Command (MARCORSYSCOM) with an efficient and effective tool with which to manage the unit's serialized assets.

The following special topics are presented to facilitate introduction to this system:

1. INSTALLATION AND SETUP
2. STARTING THE SYSTEM
3. DOWNLOAD FILE TO BAR CODE READER
4. CONDUCTING AN INVENTORY
5. UPLOADING INVENTORY RESULTS
6. PRINTING REPORTS
7. PRINTING BAR CODE LABELS

B. INSTALLATION AND SETUP

1. Compatibility and Requirements

The BCIS is compatible with any IBM personal computer or IBM compatible personal computer. In addition, the BCIS requires the following configuration:

1. MS-DOS or PC-DOS (Versions 5.0 or 6.0).
2. 640K RAM (1M recommended).
3. Two serial ports and one parallel port
4. Intermec Bar Code Reader
5. Intermec Bar Code Printer
6. 10-9 Null Modem Cable
7. 25-Pin Printer to PC Cable
8. Visible Laser Scanner

2. Installation

To install the Bar Code Inventory System on a personal computer, place the BCIS diskette in **B drive** (or **Drive A**) of the computer and type **B:\INSTALL** (or **A:\INSTALL**). The install program will create four directories on the computer's hard drive and copy the system programs to those directories. When the installation process is complete check the computer's hard drive for the following **directories** and programs:

```
BCIS
   BCIS    .COM
   BATCHKEY.COM
   CLRSCR  .COM
   TEXTOUT .COM
   SETPOS  .COM
   DRAWBOX .COM

BCIS/ADA
   CMR_RPT  .EXE
   DOWNLOAD .EXE
   PRN_BATCH.EXE
   X_REPORT .EXE
   PRN_INDIV.EXE

INTRSCAN
   README
   COMMDRV .EXE
   INTRSCAN.EXE
   PCHOST  .EXE
   TERMDRV .EXE
   UTILDRV .EXE
   INV     .IRL
   INTRSCAN.INI

PCIRL
   PCIRL  .EXE
   PIPE   .EXE
   README
   PCIRL  .CFN
   BCIS   .IRL
   INV    .IRL
```

In addition to the system files, a file called PCDRIVER.SYS will be added to the computer's Root directory. PCDRIVER.SYS is a PCIRL device driver that will manage the communication protocol for uploading and downloading programs and files.

3. Setup

Before PCIRL can be utilized, ANSI.SYS must be added to the Root directory and two lines must be added to the CONFIG.SYS File. ANSI.SYS is a DOS device driver that will manage the PCIRL screen output.

To add the required lines to the CONFIG.SYS file, perform the following steps:

1. Move to the Root directory by typing **CD\**.
2. At the DOS prompt, type **EDIT CONFIG.SYS**.
3. Press **[Enter]**.
4. At the end of the CONFIG.SYS lines, type the following: **DEVICE=ANSI.SYS**.
5. Press **[Enter]**.
6. On the next line, type **DEVICE=PCDRIVER.SYS**.
7. Press **[Enter]**.
8. Press **[Alt]** to activate the menu. Use the arrow keys to select FILE and press **[Enter]**.
9. Select SAVE and press **[Enter]**.
10. Press **[Alt]** to activate the menu. Use the arrow keys to select FILE and press **[Enter]**.
11. Select EXIT and press **[Enter]**.

12. At the DOS prompt, type the following: **TYPE CONFIG.SYS**.

13. Press **[Enter]**.

14. Verify that CONFIG.SYS now includes the two required lines:

```
DEVICE=ANSI.SYS
DEVICE=PCDRIVER.SYS
```

15. Reboot the computer by pressing **[Ctrl]-[Alt]-[Del]**.

C. STARTING THE SYSTEM

To initiate the Bar Code Inventory System, move to the BCIS directory by typing **CD\BCIS**. The system starts when **BCIS** is entered at the *>C:\BCIS* prompt.

Once in the Bar Code Inventory System, the user is directed to the Main Menu. Figure B.1 illustrates the options available to the user. To navigate through the system, type the number in front of the desired option. For example, to exit the system press [5] and hit **[Enter]**.

```
              BAR CODE INVENTORY SYSTEM

        1.  Download File to Reader

        2.  Receive Inventory Results

        3.  Print Reports

        4.  Print Bar Code Labels

        5.  Exit to DOS

                          ENTER (1..5)
```

FIGURE B.1 **Main Menu**

D. DOWNLOAD FILE TO BAR CODE READER

This portion of the system extracts the serial numbers from the CMR Listing and creates a serial number file. The file is formatted for the reader and is passed to the PCIRL directory for download. After the file is downloaded to the reader, the user is returned to the Main Menu, (see Figure B.1). The serial number file must be downloaded to the reader before conducting an inventory.

1. Establish a Connection

Before commencing the download portion of the system, connect the male end of the of the 10-9 Null Modem Cable into the reader's communication dock and the female end into the COM1 port of the personal computer.

2. Set Protocol On Reader

The communication protocol, on the reader, must be set to "POLLING MODE D" before downloading a file. The following example illustrates the steps required to set the reader's protocol to "POLLING MODE D":

1. Press the reader's **ON-OFF** key to turn the reader on.
2. At the "READY" prompt, press **[Ctrl]-[E]** to enter configuration mode. The prompt in Figure B.2 appears on the reader's screen:

```
CONFIGURATION MENU:
Press <?> for help,
<ENTER> to continue
<CNTRL-Z> to exit
```

<u>FIGURE B.2</u> Reader Configuration Prompt

3. Press **[ENTER]** to continue. The prompt in Figure B.3 appears:

```
CONFIGURATION MENU:
Select or modify
bar codes?
NO _
```

FIGURE B.3 Bar Code Option

4. Press **[ENTER]** for no. The screen in Figure B.4 appears:

```
CONFIGURATION MENU:
Select or modify
operating parms?
NO _
```

FIGURE B.4 Operating Parameter Option

5. Press **[ENTER]** for no. The screen in Figure B.5 appears:

```
CONFIGURATION MENU:
Select or modify
comm protocol?
NO (now=PT. TO PT.)
```

FIGURE B.5 Comm Protocol Option

6. Press **[SPACE]** twice to change to "POLLING MODE D" Protocol.

7. Press **[Alt]-[E]** to save changes and exit Configuration Menu.

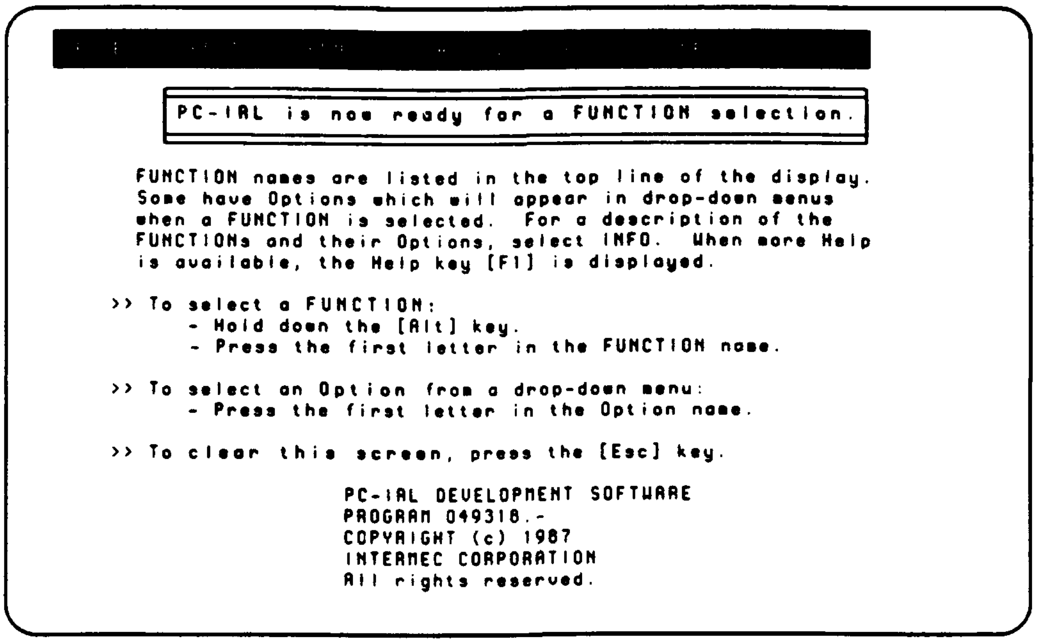
3. Place CMR Listing in Proper Directory

Before selecting the download option from the Main Menu, the CMR Listing must be copied to the **BCIS\ADA** directory under the File Name **CMRL**. If this step is not accomplished prior to initiating download, the user will receive a quick error message and the program will return to the main menu.

4. Initiating Download Option

Once the **CMRL** File is copied to the **BCIS\ADA** directory, the download can be started by pressing **[1]** then **[Enter]**. The system will then display a message that Serial Numbers are being extracted from the CMR.

When all the serial numbers have been extracted from the CMR the screen in Figure B.6 will be displayed. This screen is the PCIRL Set-Up Screen. To proceed with downloading the file to



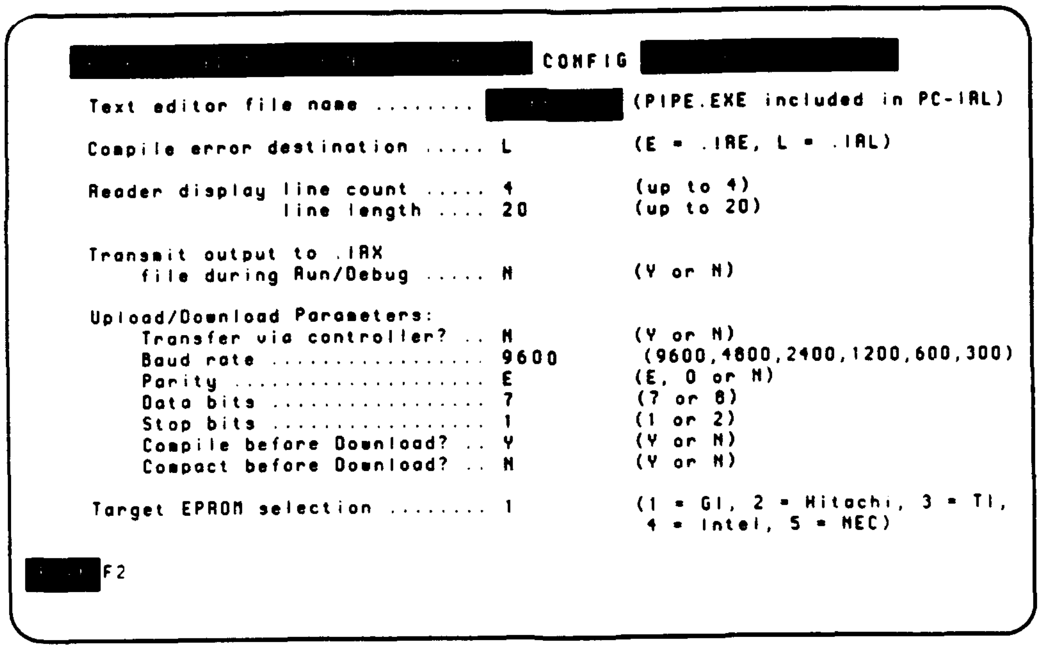
```
PC-IRL is now ready for a FUNCTION selection.

FUNCTION names are listed in the top line of the display.
Some have Options which will appear in drop-down menus
when a FUNCTION is selected.  For a description of the
FUNCTIONs and their Options, select INFO.  When more Help
is available, the Help key [F1] is displayed.

>> To select a FUNCTION:
     - Hold down the [Alt] key.
     - Press the first letter in the FUNCTION name.

>> To select an Option from a drop-down menu:
     - Press the first letter in the Option name.

>> To clear this screen, press the [Esc] key.

                    PC-IRL DEVELOPMENT SOFTWARE
                    PROGRAM 049318.-
                    COPYRIGHT (c) 1987
                    INTERMEC CORPORATION
                    All rights reserved.
```

FIGURE B.6 **PCIRL Set-Up Screen**

the reader, this screen must be cleared before a function can be selected. To clear the Set-Up Screen press **[Esc]**.

5. Configure PCIRL for Communication

The first time the Bar Code Inventory System is used, PCIRL must be configured prior to downloading the file. To select the CONFIG function enter **[Alt]-[C]**. Verify that the CONFIG Screen looks like Figure B.7. To make configuration changes, use the arrow keys to move to the appropriate field and manually enter the correct setting.

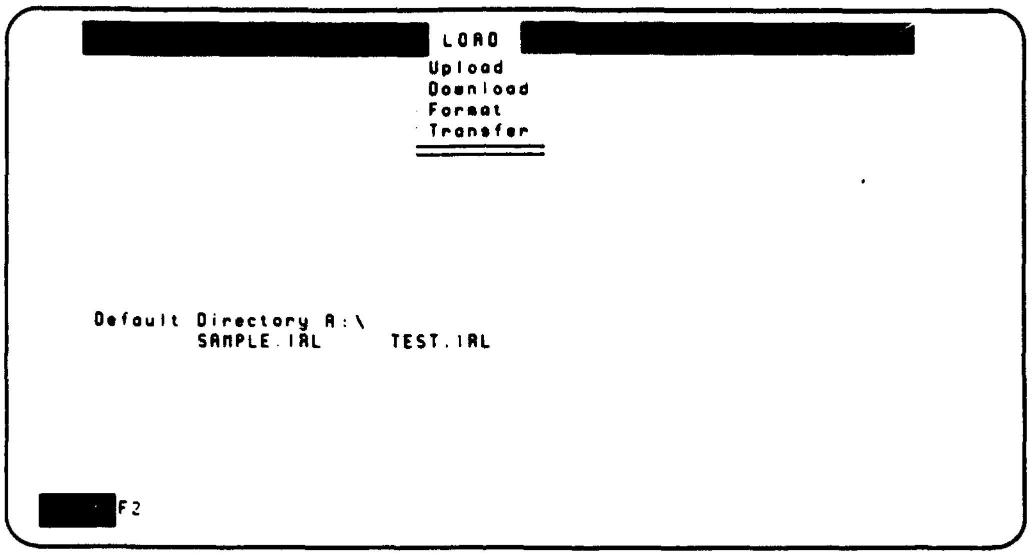


```
                                        CONFIG
  Text editor file name ........              (PIPE.EXE included in PC-IRL)

  Compile error destination ..... L           (E = .IRE, L = .IRL)

  Reader display line count ..... 4           (up to 4)
                 line length .... 20          (up to 20)

  Transmit output to .IRX
      file during Run/Debug ..... N           (Y or N)

  Upload/Download Parameters:
      Transfer via controller? .. N           (Y or N)
      Baud rate ................. 9600        (9600,4800,2400,1200,600,300)
      Parity .................... E           (E, O or N)
      Data bits ................. 7           (7 or 8)
      Stop bits ................. 1           (1 or 2)
      Compile before Download? .. Y           (Y or N)
      Compact before Download? .. N           (Y or N)

  Target EPROM selection ........ 1           (1 = GI, 2 = Hitachi, 3 = TI,
                                               4 = Intel, 5 = NEC)

F2
```

FIGURE B.7 **PCIRL Configuration Screen**

6. Downloading File to Reader

After configuring PCIRL, move to the LOAD function by pressing **[Alt]-[L]**. This creates a drop-down menu as depicted in Figure B.8. Select **[D]** for download and type **READIN** for the



```
Default Directory A:\
        SAMPLE.IRL      TEST.IRL
```

FIGURE B.8 **PCIRL Load Screen**

download file name. Using the down arrow, move to the file destination field and enter **[A]**. Turn the bar code reader on by Pressing the **[ON-OFF]** key and wait for the ready prompt. Hit **[ENTER]** on the computer keyboard and PCIRL will download the file to the reader. The download process is finished when PCIRL displays the message "DOWNLOAD COMPLETE..." in the lower right corner of the screen. Enter **[F2]** to exit the download menu and [Alt]-[Q] to return to the BCIS Main Menu. This completes the download portion of the system.

E. CONDUCTING THE INVENTORY

This portion of the system is performed with the bar code reader. The reader scans bar code labels affixed to the serialized assets and checks to see if the asset is on the CMR. If the asset inventoried is on the CMR the serial number is recorded in the Found File. If the asset is not on the CMR the user is prompted to enter a description and location of the asset. This information along with the serial number is then recorded in the Not Found File.

To begin the inventory, connect the laser scanner to the bar code reader. Press the **[ON-OFF]** key to turn the reader on and wait for the ready prompt. At the ready prompt enter **[Ctrl]-[ENTER] [B]**. Press the **[Ctrl]** and **[ENTER]** keys simultaneously and then press **[B]**. This key sequence initiates the IRL program stored in the reader's memory. When the program is initiated the prompt in Figure B.9 appears on the screen:

```
   MARCORSYSCOM
     INVENTORY
```

FIGURE B.9 **Start of Program Screen**

After three seconds the screen clears and the user is presented with the prompt in Figure B.10:

```
ENTER SERIAL # OR
F3 TO TRANSMIT FILES
```

FIGURE B.10 **Enter Serial Number or Transmit Prompt**

At this prompt, the user scans or manually enters the serial number of the asset being inventoried. If the asset inventoried is on the CMR the user is returned to the prompt in Figure B.10.

If the asset inventoried is not on the CMR the reader emits a sequence of beeps and the screen depicted in Figure B.11 appears:

```
ITEM NOT ON CMR
```

FIGURE B.11 **Item Not On Inventory Screen**

After three seconds, the screen is cleared and the user is presented with the prompts illustrated in Figures B.12 and B.13.

```
ENTER DESCRIPTION
OF THE ASSET
```

FIGURE B.12 **Enter Description Prompt**

At these prompts, manually enter the description and location of the asset just inventoried. After this information is recorded, the user is returned to the Enter Serial Number Prompt. This process is continued until the inventory is completed.

```
ENTER LOCATION
OF THE ASSET
```

FIGURE B.13 **Enter Location Prompt**

If the reader is turned off during the inventory, the data is not lost. To resume the inventory, press the **[ON-OFF]** key and at the ready prompt enter [Ctrl]-[**ENTER**] [B]. The IRL program is resumed and the user can continue the inventory.

F. RECEIVE INVENTORY RESULTS

This phase of the system uploads the results of an inventory from the reader to the personal computer. The results are then processed and two reports are created, the CMR Report and the Exception Report.

To Receive the inventory results, select option [**2**] from the Main Menu, (see Figure B.1). Press [**ENTER**] and the Receive File Option Menu appears, (Figure B.14).

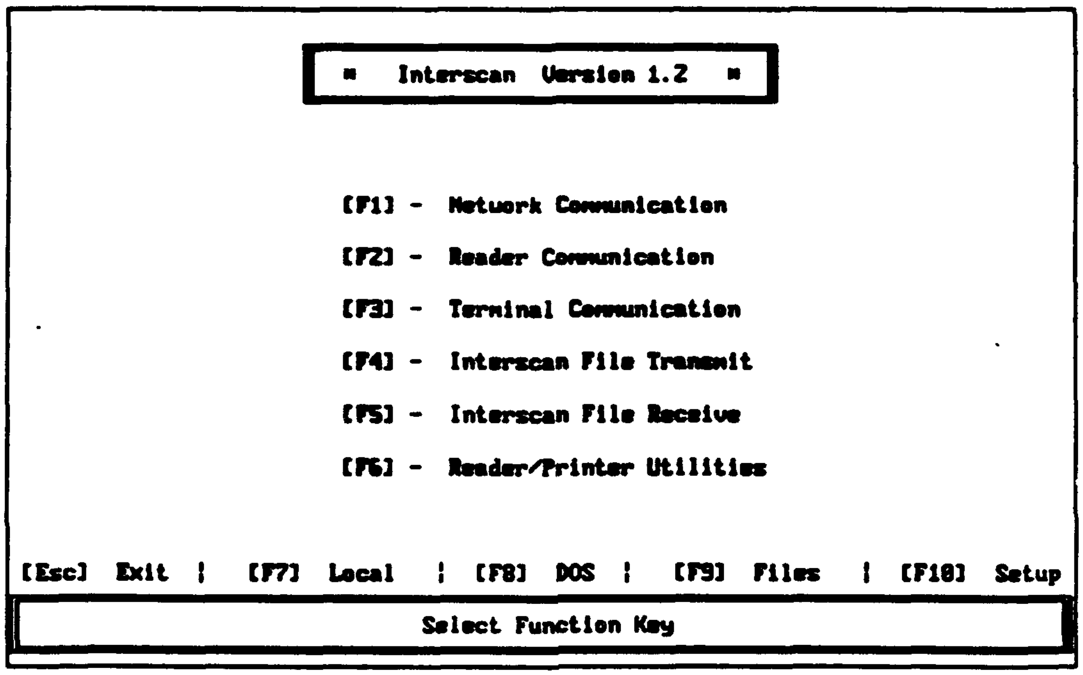
This Menu allows the user to select which file to upload--the Found or Not Found File--or to upload both files. The proper selection depends on the needs of the user.

```
                RECEIVE FILE OPTIONS

        1.  Receive Found File

        2.  Receive Not Found File

        3.  Receive Both Files

        4.  Exit to Main Menu

                          ENTER (1..4)
```

FIGURE B.14 **Receive File Options Menu**

1. Receive Found File

If all the assets inventoried were on the CMR (i.e., the Not Found File is empty), select option [1] and press [ENTER]. The Interscan Menu illustrated in Figure B.15 will appear on the screen.



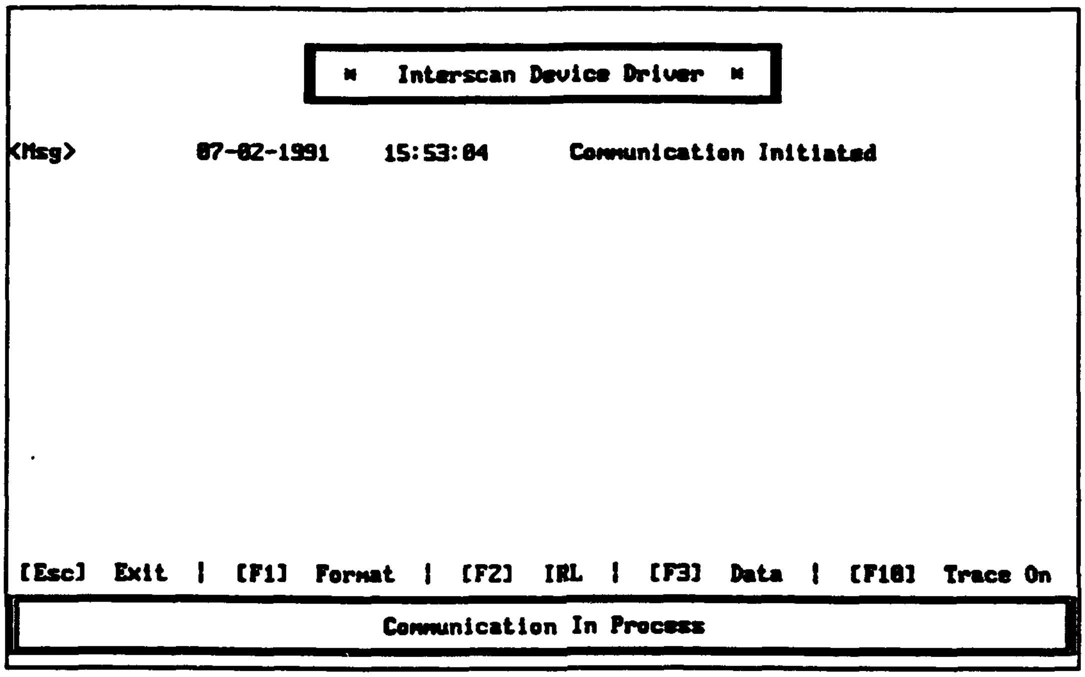
```
* Interscan Version 1.2 *

[F1] - Network Communication
[F2] - Reader Communication
[F3] - Terminal Communication
[F4] - Interscan File Transmit
[F5] - Interscan File Receive
[F6] - Reader/Printer Utilities

[Esc] Exit | [F7] Local | [F8] DOS | [F9] Files | [F10] Setup
Select Function Key
```

FIGURE B.15 Interscan Interactive Mode Menu

The first time the Bar Code Inventory System is used, Intrscan must be configured prior to uploading a file. To configure the Intrscan software for communication with a reader refer to the Configure Intrscan Section, at the end of Section F.

Select [F2] for reader communication and the Intrscan Communication Screen will appear, (Figure B.16). When in this screen the computer is ready to receive the Found File.

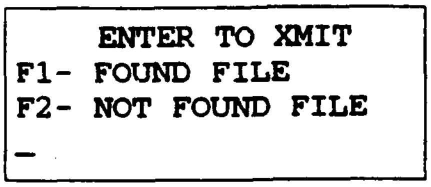
Before uploading the file to the computer, connect the male end of the of the 10-9 Null Modem Cable into the reader's communication dock and the female end into the COM1 port of the personal computer.



```
                    * Interscan Device Driver *

<Msg>        07-02-1991    15:53:04      Communication Initiated






[Esc] Exit | [F1] Format | [F2] IRL | [F3] Data | [F10] Trace On
                    Communication In Process
```

FIGURE B.16 Interscan Communication Screen

Turn the bar code reader on and at the "READY PROMPT" type [Ctrl]-[ENTER] [B] to start the program. The start of program screen, (see Figure B.9), will appear for three seconds followed by the "enter serial number or F3 to transmit" prompt,(see Figure B.10). At this prompt enter [F3] to transmit and the screen in Figure B.17 will appear.



```
    ENTER TO XMIT
F1- FOUND FILE
F2- NOT FOUND FILE

_
```

FIGURE B.17 Transmit Screen

Enter **[F1]** at this prompt and the reader will upload the file to the computer. The serial numbers will scroll down the Intermec Communication Screen (Figure B.16). The serial numbers will be stored in a file called Found on the personal computer. The upload is complete when serial numbers stop scrolling across the computer screen.

Press **[Esc]** twice and the program will process the serial numbers in the Found file and create the CMR Report. When the CMR Report is finished the user is returned to the Receive File Options Menu, (Figure B.14).

2. Receive Not Found File

This option uploads the Not Found File from the bar code reader and creates an Exception Report. The Exception Report is a listing of all of the serial numbers found during the inventory that are not on the CMR. A short description and the location of the item is also provided.

To initiate this option press **[2]** then **[ENTER]** from the Receive File Options Menu, (see Figure B.14). The Intrscan Menu illustrated in Figure B.15 will appear on the screen.

The first time the Bar Code Inventory System is used, Intrscan must be configured prior to uploading a file. To configure the Intrscan software for communication with a reader refer to the Configure Intrscan Section, at the end of Section F.

Select **[F2]** for reader communication and the Intrscan Communication Screen will appear, (see Figure B.16). When in this screen the computer is ready to receive a File.

Before uploading the file to the computer, connect the male end of the of the 10-9 Null Modem Cable into the reader's communication dock and the female end into the COM1 port of the personal computer.

Turn the bar code reader on and at the "ready prompt" type **[Ctrl]-[ENTER] [B]** to start the program. The Start of Program screen, (see Figure B.9), will appear for three seconds followed by the "enter serial number or F3 to transmit" prompt,(see Figure B.10). At this prompt enter **[F3]** to transmit and the screen in Figure B.17 will appear.

Enter **[F2]** to transmit the Not Found File and the reader will upload the file to the computer. The data uploaded will scroll down the Intermec Communication Screen (see Figure B.16). This information will be stored in a file called NoFind on the personal computer. The upload is complete when the data stops scrolling down the computer screen.

Press **[Esc]** twice and the program will process the information in the NoFind file and create an Exception Report. When the Exception Report is finished the user is returned to the Receive File Options Menu, (see Figure B.14).

3. Receive Both Files

This option uploads the Found and Not Found File from the bar code reader and creates both the CMR and Exception Reports. To initiate this option press **[3]** then **[ENTER]** from the Receive File Options Menu, (see Figure B.14). The Intrscan Menu illustrated in Figure B.15 will appear on the screen.

The first time the Bar Code Inventory System is used, Intrscan must be configured prior to uploading a file. To configure the Intrscan software for communication with a reader refer to the Configure Intrscan Section, at the end of Section F.

Select **[F2]** for reader communication and the Intrscan Communication Screen will appear, (see Figure B.16). When in this screen the computer is ready to receive a File.

Before uploading the file to the computer, connect the male end of the of the 10-9 Null Modem Cable into the reader's communication dock and the female end into the COM1 port of the personal computer.

Turn the bar code reader on and at the "READY PROMPT" type **[Ctrl]-[ENTER] [B]** to start the program. The Start of Program screen, (see Figure B.9), will appear for three seconds followed by the "enter serial number or F3 to transmit" prompt,(see Figure B.10). At this prompt enter **[F3]** to transmit and the screen in Figure B.17 will appear.

Enter **[F1]** to transmit the Found File and the reader will upload the file to the computer. When the serial numbers stop scrolling down the Intermec Communication Screen (see Figure B.16) the file is uploaded. At the "ENTER SERIAL NUMBER OR F3" prompt enter **[F3]**. Press **[F2]** and the Not Found File will be transmitted to the computer. The upload is complete when the data stops scrolling down the computer screen.

Press **[Esc]** twice and the program will process the received information into the CMR and Exception Reports. When the Reports are created the user is returned to the Receive File Options Menu, (see Figure B.14).

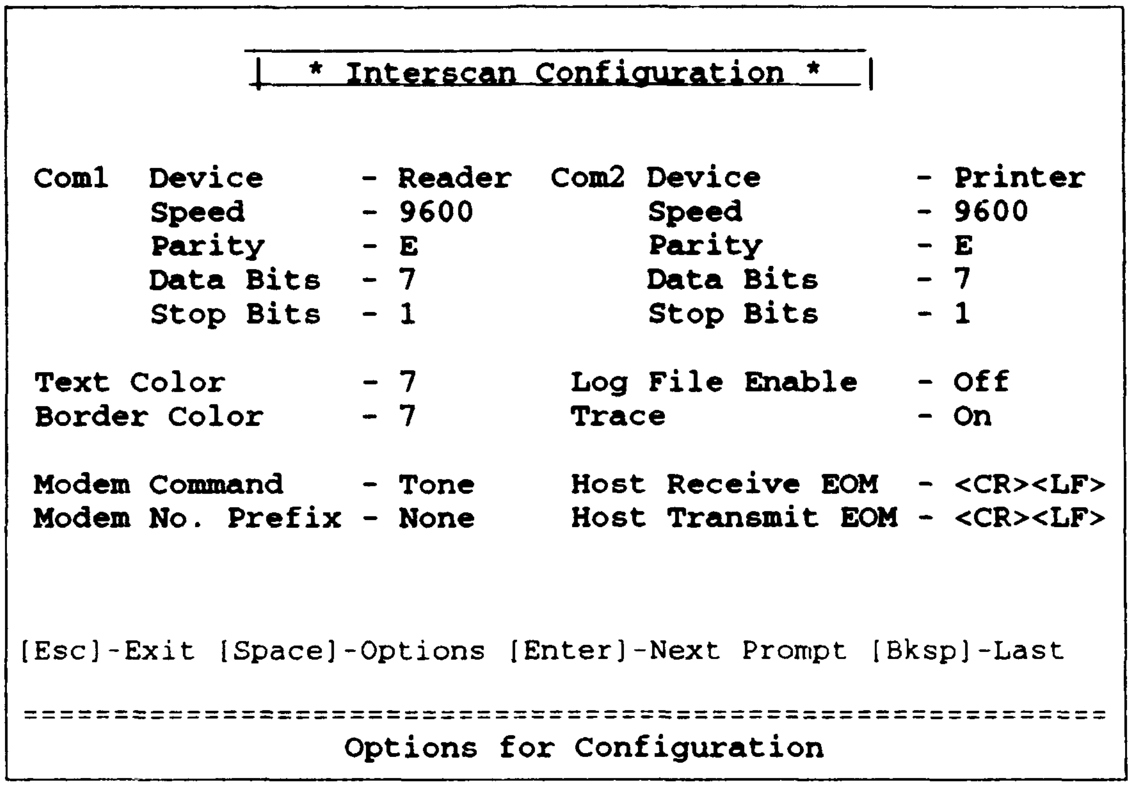
4. Configure Interscan for Communication

To enter the Configuration Menu, select **[F10]** from the Interactive Mode Menu, (see Figure B.15). The Configuration Menu is illustrated in Figure B.18. Use the **[SPACE]** key to change the options, the **[ENTER]** key to move to the next field, and the **[Bksp]** key to move to the prior field. Change the options on the screen to match Figure B.18 and enter **[Esc]** to return to the Interactive Mode Menu.

G. PRINT REPORTS

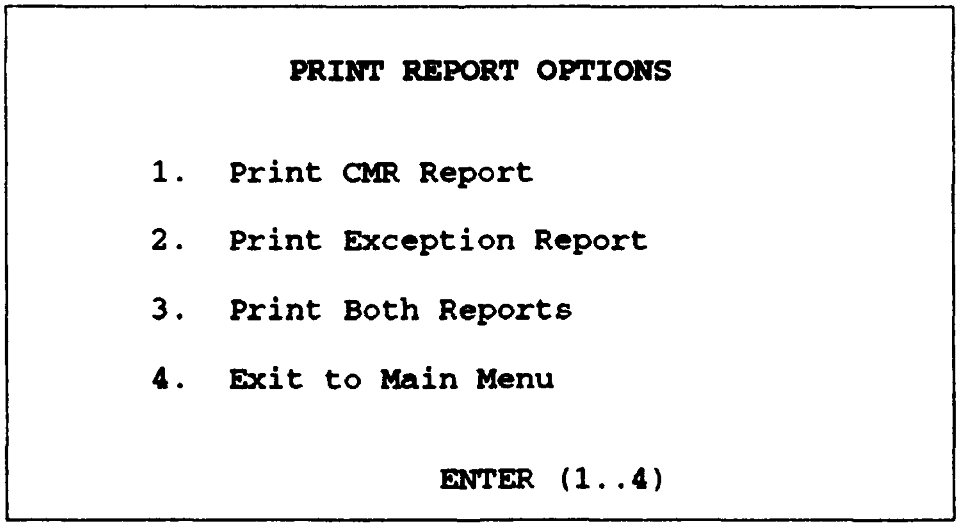
This option allow the user to print the CMR and Exception Reports. At the Bar Code Inventory System Main Menu type **[3]** then **[ENTER]**. The Print Report Options Menu, illustrated in figure B.19 will appear.

This Menu allows the user to select which Report to print--the CMR Report or Exception Report--or to print both



```
            | * Interscan Configuration * |

Com1 Device       - Reader  Com2 Device        - Printer
     Speed        - 9600         Speed         - 9600
     Parity       - E            Parity        - E
     Data Bits    - 7            Data Bits     - 7
     Stop Bits    - 1            Stop Bits     - 1

Text Color        - 7         Log File Enable   - Off
Border Color      - 7         Trace             - On

Modem Command     - Tone      Host Receive EOM  - <CR><LF>
Modem No. Prefix  - None      Host Transmit EOM - <CR><LF>


[Esc]-Exit [Space]-Options [Enter]-Next Prompt [Bksp]-Last

========================================================
               Options for Configuration
```

FIGURE B.18 Interscan Configuration Menu



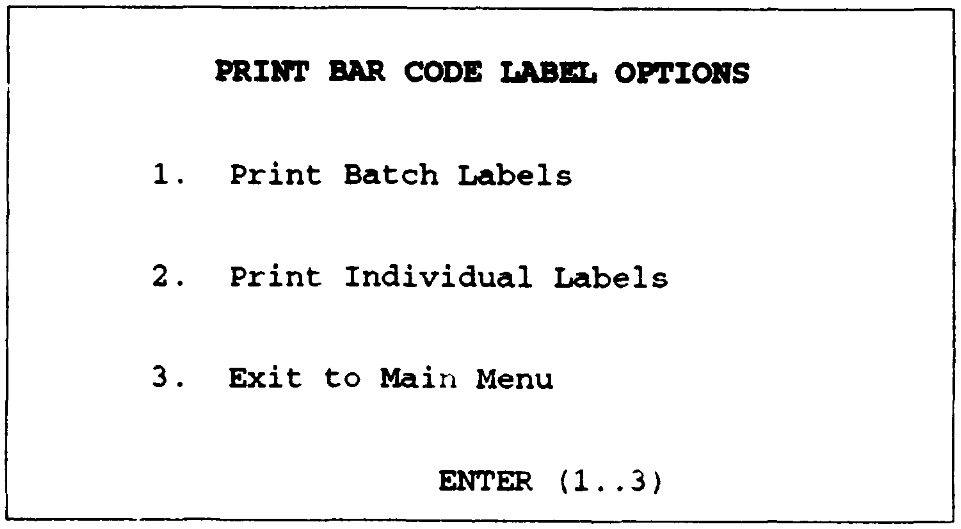
```
              PRINT REPORT OPTIONS

        1.  Print CMR Report

        2.  Print Exception Report

        3.  Print Both Reports

        4.  Exit to Main Menu

                        ENTER (1..4)
```

FIGURE B.19 Print Report Options Menu

Reports. To select which report to print type the number in front of the desired option and press **[ENTER]**. The proper selection depends on the desires of the user.

H. PRINT BAR CODE LABELS

This option prints the bar code labels that will be attached to the serialized assets. At the Main Menu type **[4]** then **[ENTER]** to access the Print Bar Code Label Menu, (Figure B.20).



PRINT BAR CODE LABEL OPTIONS

1. Print Batch Labels

2. Print Individual Labels

3. Exit to Main Menu

ENTER (1..3)

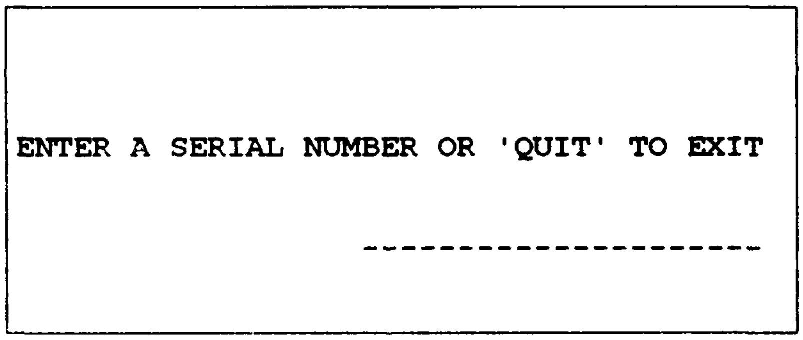
__FIGURE B.20__ **Print Bar Code Label Options Menu**

Before printing any labels, connect the female end of the of the 25-Pin printer cable into the printer's interface connector port and the male end into the COM2 port of the personal computer.

The Print Batch Labels option allows the user to print bar code labels for all the assets on the CMR. Before selecting this option, the CMRL file must be placed in the BCIS\ADA directory. To execute this option type [1] then [**ENTER**].

The Print Individual Labels option allow the user to print individual labels. When a new asset arrives, the user can select this option to create a new label and affix this label to the item. To execute this option type [2] then [ENTER]. The prompt is figure B.21 will appear on the screen.

At this prompt, type the serial number of the asset and hit [**ENTER**]. A bar code label will be printed and figure B.21 will return to the screen. To return to the Print Bar Code Label Options type **QUIT** instead of a serial number.



```
ENTER A SERIAL NUMBER OR 'QUIT' TO EXIT

                    --------------------
```

FIGURE B.21 Individual Label Prompt

I. EXIT THE SYSTEM

To exit the Bar Code Inventory System, select the option to return to the Main Menu. At this screen type [5] then [**ENTER**] and the user will be returned to the DOS prompt.

LIST OF REFERENCES

Page-Jones, M., *Practical Guide to Structured Systems Design*, 2nd ed., Prentice-Hall, Inc., 1988.

Pressman, R.S., *Software Engineering: A Practitioner's Approach*, 3rd ed., McGraw-Hill, Inc., 1992.

Sedgewick, R., *Algorithms*, Addison-Wesley Publishing Company, Inc., 1984.

Whitten, J.L., Bentley, L.D., and Barlow, V.M., *Systems Analysis and Design Methods*, 2nd ed., Irwin, Inc., 1989.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Director, Training and Education MCCDC, Code C46 1019 Elliot Road Quantico, VA 22134-5027	1
4.	Director ISMD MARCORSYSCOM 2033 Barnett Avenue, Suite 315 Quantico, VA 22134-5010	1
5.	Professor William Haga, Code AS/HA Naval Postgraduate School Monterey, CA 93943-5002	1
6.	Professor Shu S. Liao, Code AS/LC Naval Postgraduate School Monterey, CA 93943-5002	1
7.	Computer Technology Programs, Code 370 Naval Postgraduate School Monterey, CA 93943-5002	1
8.	Capt. Richard M. Hancock 206 Greensview Drive Cary, NC 27511	1