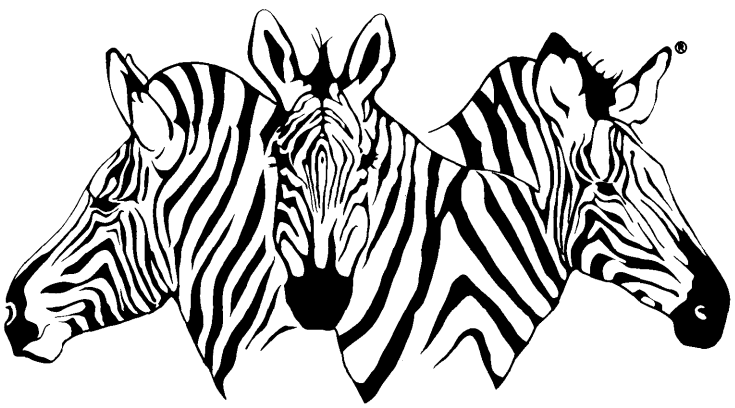
PICK
operator guide

88A00757A06



ZEBRA® BY GA

RECORD OF REVISIONS

Title: PICK Operator Guide

Document No. 88A00757A06

Date	Revision Record
Apr 83	Original Issue
Sep 83	Revision B - (Covering ZEBRA/PICK 2.0)
Apr 84	Revision A03
Oct 84	Revision A04
Feb 85	Revision A05 - Change Package (85A00514A01)
Apr 85	Revision A06 - Change Package (85A00520A01)

PICK
operator guide

88A00757A06

1045 South East Street P.O. Box 4883
Anaheim, California 92803
(714)778-4800 (800)854-6234
TWX 910-591-1695 TELEX 685-513

RECORD OF REVISIONS

Title: PICK Operator Guide

Document No. 88A00757A06

Date	Revision Record
Apr 83	Original Issue
Sep 83	Revision B - (Covering ZEBRA/PICK 2.0)
Apr 84	Revision A03
Oct 84	Revision A04
Feb 85	Revision A05 - Change Package (85A00514A01)
Apr 85	Revision A06 - Change Package (85A00520A01)

FOREWORD

This manual provides operator guidance in the use of the General Automation ZEBRA/PICK operating system. It is intended for user persons responsible for standard day-to-day operations; and for user persons who must carry out the more specialized system operator functions. In today's computer environment, both of these may be in the hands of one person.

The major subjects arranged in that way are as follows:

Section 1 - Introduction

Section 2 - System Startup

Section 3 - Terminal Control Language

Section 4 - System File Management

Section 5 - System Memory Management

Section 6 - Dictionaries and Files

Section 7 - Support Processors

Appendix A - ERRMSG (Error Messages)

Appendix B - ASCII Codes

Appendix C - ZEBRA Series Firmware Executives

Appendix D - Tape Operation

Appendix E - System-Cursor Definition Utility

Related ZEBRA/PICK documents that are available to the user:

Document No.	Title
88A00751A	Overview of the PICK Operating System
88A00758A	ACCU-PLOT Operator Guide
88A00759A	COMPU-SHEET Operator Guide
88A00760A	Quick Guide for the PICK Operating System
88A00774A	PICK Utilities Guide
88A00775A	ZEBRA Hardware Reference Manual
88A00776A	PICK ACCESS Reference Manual
88A00777A	PICK SPOOLER Reference Manual
88A00778A	PICK BASIC Reference Manual
88A00779A	PICK EDITOR Reference Manual
88A00780A	PICK PROC Reference Manual
88A00781A	PICK RUNOFF Reference Manual
88A00782A	Introduction to PICK TCL and FILE STRUCTURE
88A00783A	PICK JET Word Processor Guide

TABLE OF CONTENTS

LIST OF APPENDIXES

1 introduction

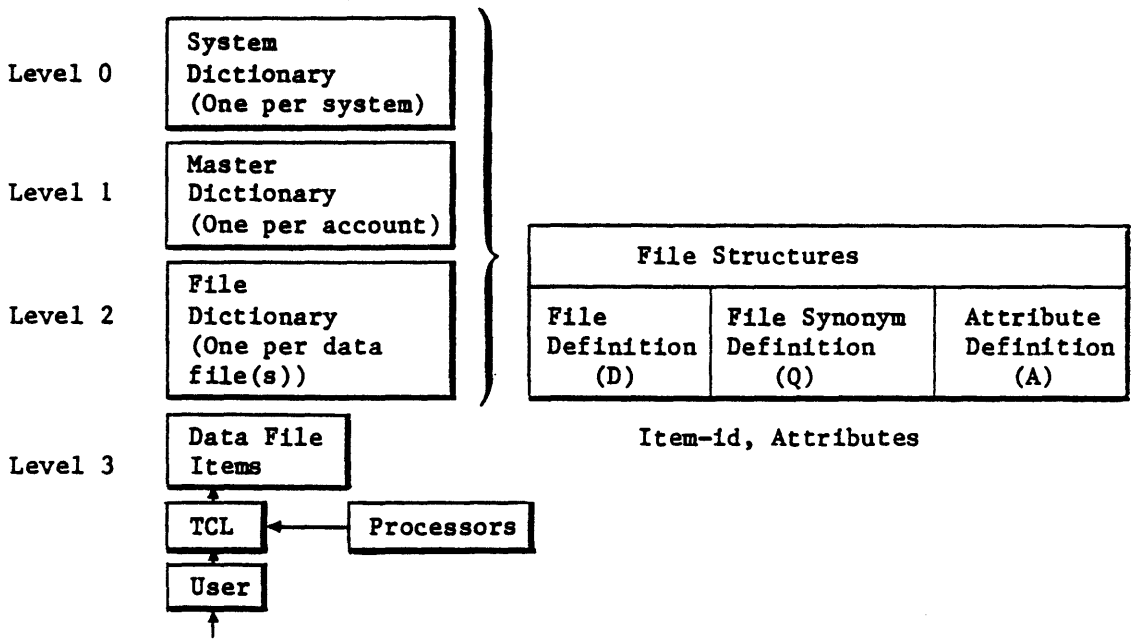
This document provides guidance for the user in turning on the ZEBRA™ system, and in bringing up and using the PICK operating system.

Since you, the system operator, will be establishing and controlling dictionaries and files, creating new accounts and supervising overall system operation, it is appropriate that we define the basic system structure and key words used throughout the text. Detailed description of all aspects of system operations will be provided in later sections.

1.1 SYSTEM STRUCTURE

The primary elements of the PICK operating system are a set of files called "dictionaries". These dictionaries exist at four levels, and are used to describe the structure of files and "point" to their location. This pointer data is a key link between a file and its dictionary. Access to a file cannot take place without a dictionary to guide that access; and, where a single dictionary can serve several files, no file can exist without a dictionary to define its location and structure.

Pictorially, the dictionaries, their principle component levels and their interface to the user are:



1.2 DICTIONARIES/FILES

This section describes the hierarchical nature of the dictionaries and files in the PICK system. Throughout these sections, the following terms will be used:

Name	Conventional Data Processing Name
Item	Record
Item-id	Record Key
Attribute	Field
Value	Subfield

Files are organized in a hierarchical structure, with files at each level pointing to multiple files at the next lower level. Four distinct file levels exist: System Dictionary, Master Dictionary, File Level Dictionary, and Data File.

The term "file" as used in the context of this system refers to a mechanism for maintaining a set of like items logically together. The data in a file must be accessed via the dictionary associated with it. A "dictionary" is like the "index" to a file. Since the dictionary itself is also a file, it contains items just as a data file does. The items in a dictionary serve to define lower level dictionaries or data files.

The system can contain any number of files. Files can contain any number of items, and can automatically expand to any size. Items are variable length, and can contain any number of fields and characters so long as the data in an item does not exceed a maximum of 32,267 bytes.

1.2.1 SYSTEM DICTIONARY (SYSTEM)

The highest level dictionary is called the System Dictionary (SYSTEM). The System Dictionary contains all legitimate user Logon names, along with associated passwords, security codes, and system privileges. The Logon names and related information are stored as items in the System Dictionary. These items function as pointers to the user's Master Dictionary.

1.2.2 USER MASTER DICTIONARIES (MD)

The Master Dictionaries (MD) comprise the next dictionary level. Each user's account will normally have a unique MD associated with it. The MD contains items which store the definitions of all user vocabulary, (verbs, PROCs, etc.) and items which function as pointers to accessible files.

When an account is created, a standard set of MD vocabulary items are stored in the account's MD. A user may, however, create synonyms to, and abbreviated forms of any or all of these standard vocabulary words. Since they are merely items within his MD file, he may create copies of their elements and rename the words. The user can also create his own prestored vocabulary statements, called PROCs.

The file pointers can reference any file or dictionary in the system, that is, they are not restricted to files defined within the user's account alone.

1.2.3 FILE LEVEL DICTIONARIES

The File Level Dictionaries describe the structure of the data within the associated data files. They also contain pointers to the associated data-level files. A file-level dictionary may be shared by more than one data-level file.

Some dictionaries do not have an associated data-file; these are called "single-level" files. Data in a single-level file is stored within the dictionary itself.

1.2.4 DATA FILES

The Data Files contain the actual data stored in variable record/field length format. In addition to the normal record/field data structure, an attribute (field) can contain multiple values, and a value, in turn, can consist of multiple subvalues. Thus, data may be stored in a three-dimensional variable length format.

1.3 FILE STRUCTURES

1.3.1 FRAMES

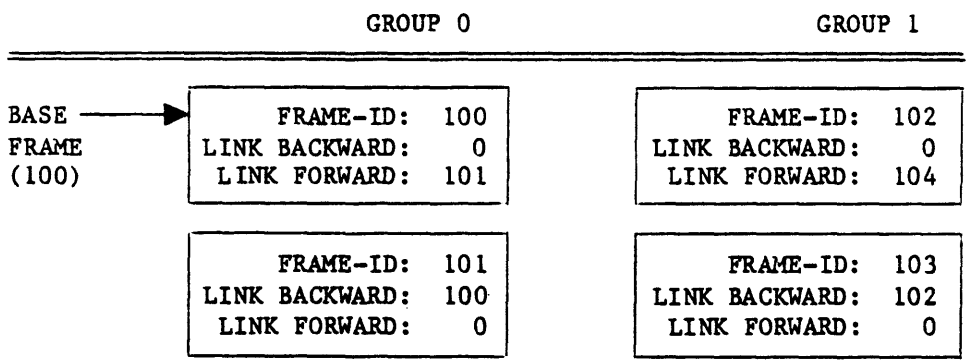
The PICK operating system "addresses" disk storage in 512-byte units called "frames". These frames are moved in and out of memory dynamically on an as-needed basis. This process is totally transparent to the individual user, who, for all practical purposes, can manipulate data within the capacity range of the given disk configuration.

Each frame has a logical address known as the "frame-id" and each frame contains 12 bytes of link information, including the number and location of forward and backward "links" in the chain of frames that make up an individual file. The remaining five hundred bytes per frame are for the storage of user data.

When a new file is created, space for it on disk is reserved in one contiguous group of frames. These frames are chosen by consulting the available space pool using two figures supplied by the user, the "modulo" and the "separation". This space is called the "primary" space, and in no way represents a limit on how large the file can grow.

The following illustrates the relation and provides further definition of frame, modulo, separation and group.

MODULO 2
SEPARATION 2
BASE FRAME 100



1.3.2 MODULO AND SEPARATION

The modulo and separation are a method of dividing up a whole file into smaller groups. The purpose of dividing a file into groups is to focus a search for a given item of data in a smaller amount of storage, and thus, minimize the search time. The modulo represents the number of groups the file is to be divided into. The separation represents the number of frames that will initially be allocated to each group. If a file is "modulo three," that means the file is divided into three groups. If the separation is three, then each group is three frames long, and the file would reserve a total of nine frames as the "primary" file space.

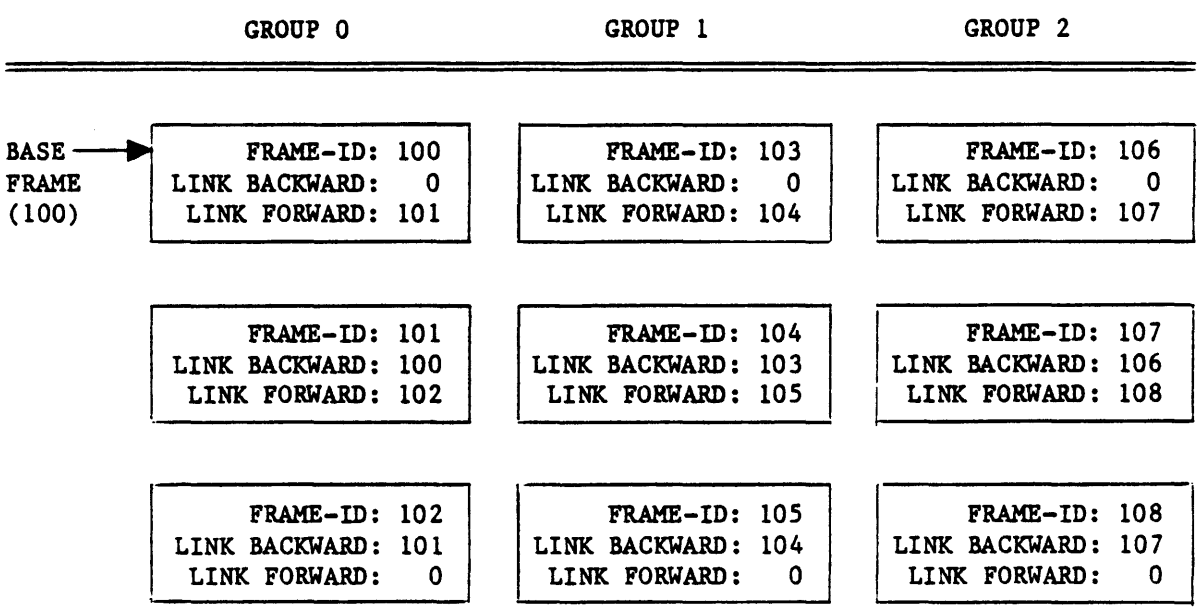
In short, the modulo*separation product represents the total number of frames allocated for the file and these frames are always contiguous.

It is important to emphasize that this pre-allocation places no limit on the growth potential in a file. Because each frame has forward and backward linking pointers, new frames are automatically added to an expanding group, as required.

While using the modulo*separation product to determine the amount of contiguous space to allocate to a file, the system also uses these figures to build the file dictionary. The file dictionary will contain the modulo, the separation, and the disk address of the first frame in the file, called the "base frame" or "file base".

The conceptual effect of dividing a file into groups is to create a number of separate files. The following sketch diagrams the primary space of a modulo-3 separation-3 file. This file is divided into three frames each. The "base-frame" is the first frame of the reserved, contiguous primary file space.

MODULO	3
SEPARATION	3
BASE FRAME	100



Notice that the file effectively contains three "starting points" in frames 100, 103, and 106. These frames link backward to frame 0, which is to say they do not "link back" at all. Because only the first frame of Group 0 is stored in the file dictionary as the base frame, the starting frames of the following groups are computed. Since the primary space frames are always contiguous, the base frame of any given group can easily be determined using the following formula:

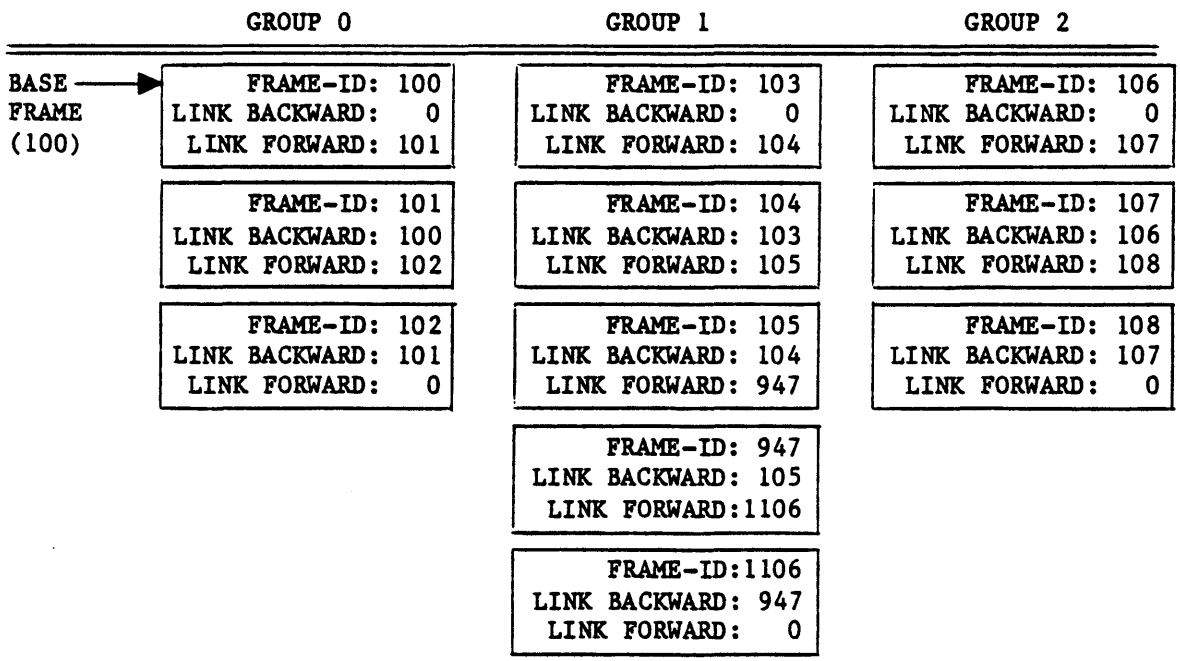
Base of any Group = (Group * Separation) + Filebase

Thus, the starting frame of Group 1 is 103:

103 = (1 * 3) + 100

Now suppose an item is to be added to the file. First the item-id is hashed to determine which group it will be stored in. The hashing algorithm will return a value within the range of the modulo (the number of groups in the file). In this case, we will assume the result is Group 1. Also, we will assume that the primary space allocated to Group 1 is almost full and the addition of this item will cause it to overflow two frames. The following sketch diagrams the result.

MODULO	3
SEPARATION	3
BASE FRAME	100



As shown, when Group 1 overflowed, the additional data was written into frames 947 and 1106 and the linking information was used to chain these frames to the last frame of the "primary" space in Group 1. Should the total size of Group 1 be reduced by future deletions, frames 947 and 1106 would be "un-linked" and returned to the free space pool. The "primary space" will always be associated with the file, even if one group, or the entire file, has all items deleted.

This process of defining the modulo and separation for a file can be used to optimize the file access process. Utility programs are available to choose the optimum modulo and separation based on average item size and the total number of items stored. Since the search for any one item is always limited to only one group, it is group size, not file size, that determines the speed of retrieval. The separation, involving reservation of contiguous space, is an attempt to ensure that the "next access" will be physically the next sector on the disk. Since most PICK Operating Systems are used in a timesharing environment, contention for the disk will almost always cause head movement before the next access for any one user. Therefore, in most instances, the separation is left as one, and the linking process begins as soon as the first frame of a group is filled; each group being free to grow and shrink independently.

1.4 ITEM-ID AND ATTRIBUTES

The file structures just described are "transparent" to the user once a file has been divided into frames by using the modulo and separation. The item is where the real mechanics of the PICK Operating System file structure become obvious to the user.

1.4.1 ITEM-ID

Item-id is a keyword that identifies a related group of fields called an item. A file is a collection of related items. A file can be any length, consisting of any number of items, with any one item limited to 32K bytes. An item is divided into attributes. These attributes can contain any data except for attribute 0, the 0th attribute, which contains the keyword that identifies the item. This attribute 0, called item-id, can be up to 50 bytes long. No length restrictions apply to other individual attributes, except for their collective limit of 32K bytes.

The data that follows item-id can be further divided into more attributes; attributes can contain multiple values; and values can contain multiple subvalues. Attributes are delimited by physically writing the ^ character between attributes. Multiple values within attributes are delimited by the] character and multiple subvalues within values are delimited by the \ character. Attributes, values, and subvalues are individually variable in length, can grow or shrink as required by the application, and occupy only as much disk storage as they require plus the one-byte delimiter marks that separate them. All of the information pertaining to what attributes are in a file is contained in the file dictionary. Whereas attributes, values, and subvalues can potentially contain the same data, attribute 0, the item-id, must be unique within one file.

Attribute 0, or item-id, plays a key role in the location of an item. The retrieval of a given item follows this basic procedure:

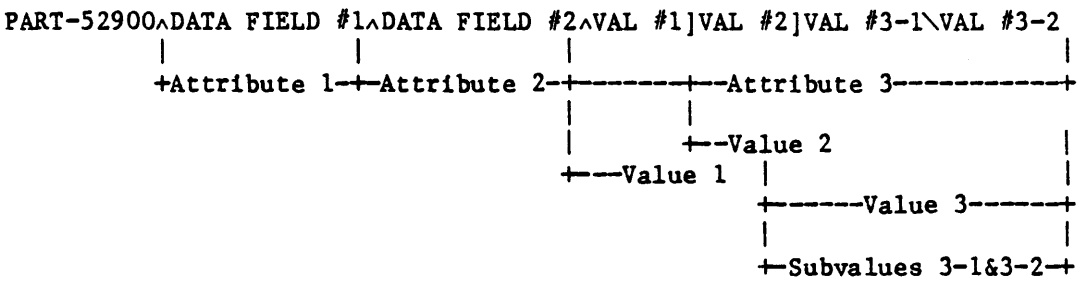
1. Given the file name and item-id of the data to be retrieved, the operating system consults the file dictionary and determines the base frame, modulo, and separation of the file.

2. Using the item-id supplied by the user and the hashing algorithm, the base frame of the group in which the item is stored is determined.

3. The base frame and associated linking pointers to the following frames are read and the group is searched until the item-id is found.

Shown below is an example of an item in an inventory file. This example illustrates the role of the attribute mark ^, the value mark], and the subvalue mark \. Both attributes 1 and 2 contain one item of data.

Attribute 3 illustrates the full range of complexity that can be managed by the operating system; it contains three values delimited by the value mark], and the third of these, Value 3, consists of two subvalues.



1.4.2 ATTRIBUTES

Within the PICK operating system, file dictionaries and file data items are composed of a string of attributes, delimited by the character ^. In all cases, attribute 0 serves as item-id. A brief description of attributes follows; detailed description is provided in Section 6.

1.4.2.1 Dictionary Attributes

An important concept to remember is that dictionaries are files. Their structure follows the same pattern of item-id followed by attributes, like any other file. Dictionaries achieve special significance by following a relatively rigid structure, unlike data files which follow whatever structure the data is suited to. Dictionary files reserve certain characters for attribute 1. If one of these reserved characters appear as attribute 1 in the item, then the item-id and the following attributes in that item take on special significance in the definition of the item and its purpose.

There are three classes of items that can appear in a dictionary file. The following is a discussion of these types.

1. File Definition (D) Items

   When an item in a dictionary file is used to define another "lower level" file, the item-id, or attribute 0, of the item becomes the name of the file being pointed to. This lower level file can be either a true data file, or another dictionary.

   When a file definition item appears in the system dictionary, the "file" being defined is the master dictionary of a user account. Some of the attributes take on special roles in the system dictionary, such as the establishment of logon passwords.

   When a file definition item appears in a master dictionary, the "file" being defined is the dictionary establishing a data file. One dictionary file can contain one or more file definition items pointing to data files that have identical structure.

2. File Synonym Definition (Q) Items

When an item in a dictionary file is used to define another "lower level" file, the item-id, or attribute 0, of the item becomes the name of the file being defined. It is sometimes convenient to give a file more than one name, as in giving the file "INVENTORY", the name "INV" for short. File synonyms can also be useful in a master dictionary (MD), to alter the command language terminology and/or create abbreviations. PICK makes this possible with the "file-synonym definition" item. Aside from giving a file an alternate name within the same user account, the file-synonym definition item can also point "outside" its account and reference files in other accounts, providing security restrictions are met. The item-id of the synonym definition item is the new version of the name. The "real" name is placed in attribute 3.

3. Attribute Definition (A) Items

The purpose of the attribute definition item in the file dictionary is to define the nature of the data contained within a specific attribute. Thus, the data, and the inter-data relationships, can be defined on an attribute by attribute basis. The item-id, or attribute 0, of the attribute definition item is a mnemonic "name" for that attribute. This feature is used extensively by the English-like ACCESS processor as a user-oriented means to identify the data. For instance, a user could refer to the "LIST-PRICE" rather than "ATTRIBUTE 14".

1.4.2.2 Data Attributes

A data file, like all files, consists of item identified by the item-id followed by a number of other attributes, 1-n (or no attributes). These attributes make up the data in the file. The handling of this data is dependent upon the attribute definition and the processor selected for the handling of the data.

system startup 2

Prior to turning on ZEBRA, make sure that the system is properly set up (refer to your Installation Guide). Next, make sure that the disk heads are unlocked. If in doubt verify and, if necessary, refer to the manufacturer's instructions provided with the drive.

2.1 TURNING ZEBRA ON

2.1.1 ZEBRA 1500, 2500, 3500, 5500

Prior to turning on ZEBRA, make sure that your Installation Checklist is complete through item #7. Then turn on the POWER ON switch, then press RESET. These controls are on the front panel of all ZEBRA models. The console CRT will then display:

```
GENERAL AUTOMATION EXECUTIVE - VER n.n., P/N 1561-X
    (individual configuration statistics)

Enter BOOT, BACKUP or RESTORE
Ok,
```

If nothing is entered within 10 to 15 seconds, automatic bootstrap will begin.

2.1.1.1 PICK Operating System Load

The ZEBRA operating system loader and its execution will then begin, indicated by the display of:

```
PICK OS LOADED


nnnnK MEMORY
   cc COMM LINES
```

where:

nnnn = Amount of RAM
cc = Number of comm lines physically present

Following the successful load of PICK, the operator will be prompted:

```
OPTIONS (X,M,A,F,B) >
```

During startup and normal operation, the operator will enter X. This will initialize the system and start the COLDSTART procedure. A summary of these options is given below.

Option

M Loads only the PICK monitor onto your system. Not valid for the ZEBRA 750.

A Loads the operating system object code software consisting of PICK processors (ACCESS, BASIC, PROC, etc.) and various utility programs. A binary image cartridge must be mounted to use. Before selecting this option, a backup of the master cartridge containing the operating system should be prepared.

F Loads entire system from a FILE-SAVE tape or cartridge disk containing the data and dictionary files. Prompt: "(m)ag tape or (c)artridge" will be given. User should enter "m" or "c" as appropriate and mount tape or cartridge with blocksize of 4000 bytes.

B Loads Utilities and Diagnostics program for the selection of tests.

The entry of "X" will result in IPL from disk, automatic logon and display:

```
SPOOLER STARTED
LINKING WORKSPACE FOR LINE 0

<<< R80 GENERAL AUTOMATION  REV.  n.m >>>
<<< HH:MM:SS     ZEBRA       DD:MM:YY >>>
```

The HH:MM:SS shown will be the actual time since startup was initiated. The display message will continue with display of:

```
THIS IS A COLDSTART PROCEDURE
HH:MM:SS      DD MMM YYYY
```

and an operator prompt to enter the current time and the current date.

```
TIME = HH:MM:SS    [CR]
DATE = DD MMM YYYY [CR]
```

The display message will continue with:

```
NOW CLEARING ACC FILE
NOW VERIFYING THE SYSTEM
[341]  ZEBRA PICK R80 REV. m.n SYSTEM VERIFIED
```

Automatic logoff will then take place. This will occur automatically at one time only during COLDSTART. Logon and Logoff will otherwise be carried out as described in the following section.

2.1.2 ZEBRA 750

Turn on the POWER ON switch, then press RESET. These controls are on the back panel of the ZEBRA 750 model. The console CRT will then display:

```
GENERAL AUTOMATION EXECUTIVE - VER n.n., P/N 1563-X
    (individual configuration statistics)

Enter BOOT, BACKUP or RESTORE

Ok, _
```

If a command is not entered within approximately 20 seconds, automatic bootstrap of PICK will begin, indicated by the display of

```
LOADING AND VERIFYING PICK MONITOR
```

2.1.2.1 PICK Operating System Load

The following describes the load for the 750 model. For creation of a formatted operating system cartridge tape or disk, see Appendix D, "CREATING A SYSGEN CARTRIDGE TAPE OR DISK FOR ZEBRA 750." The execution of the ZEBRA 750 operating system loader will begin with the display of:

```
PICK MONITOR LOADED AND VERIFIED

    mmmK MEMORY
      cc COMM LINES
```

where:

```
mmmK = Amount of RAM
  cc = Number of comm lines physically present
```

Following the successful load of PICK, the operator will be prompted to select an option:

```
OPTIONS (X, F, B)=
```

For startup and normal operation, the operator should enter X or only a carriage return. This will initialize the system and start the COLDSTART procedure. The following will be displayed:

```
SPOOLER STARTED
LINKING WORKSPACE FOR LINE 0

<<< R80 GENERAL AUTOMATION  REV. 2.1 >>>
<<<  time        ZEBRA        date   >>>

THIS IS THE COLD-START PROCEDURE
00:12:03         31 DEC 1967
```

The time will be the time since STARTUP was initiated, and the date will be the start date of the PICK Operating System. The operator will then be prompted to enter the current time in the form hh:mm:ss, using 24-hour format. The new time and old date will then be displayed.

```
TIME=
newtime          31 DEC 1967
```

The operator will then be prompted to enter the current date in the form mm:dd:yy. The new time and new date will then be displayed.

```
DATE=
newtime          newdate
```

Display messages will continue until the LOGON message, at which time your system is loaded, verified, and ready for LOGON.

```
NOW CLEANING UP ACC FILE
NOW VERIFYING THE SYSTEM

[341]  ZEBRA PICK R80 rev. no SYSTEM VERIFIED!!!

<CONNECT TIME=min; CPU=    ;UNITS=    ;LPTR PAGES =   >
<LOGGED OFF AT time      ON  date>

LOGON TO THE GA 2.1 ZEBRA AT time
PLEASE ENTER ACCOUNT NAME >
```

F option provides FILE RESTORE of user files.

B option returns you to the Firmware Executive. At the 'Ok,' prompt, type in either '?', 'COMMANDS' or 'HELP' to receive the following list of Executive commands.

```
? BOOT  BACKUP  COMMANDS  CONNECT  CONTEXT  DIAGNOSTIC  DUMP  ERASE
FORMAT  GOTO  HELP  LOAD  MEMORY  PAGE  REGISTER  RESTORE  RESE  RESET
RETENSION  REWIND  SAVE  SEGMENT  SRECORD  SYSTEM
```

2.1.2.2 PICK OS RESTORE

To carry out ABS RESTORE, enter BOOT CD (for Cartridge Disk) or BOOT CT (for Cartridge Tape) at the 'Ok,' prompt during startup. This will result with the prompt:

```
Mount Cartridge 1 (y/n):
```

There are no default entries. If you enter 'N', you will return to 'Ok,' prompt. If you enter 'Y', you will receive the following prompts:

```
ZEBRA 750 SYSGEN LOADER

RESTORE SYSTEM-R
RESTORE ABS-A
```

Enter one of the options. Option A loads ABS (operating system). After entering 'A', the screen will display:

```
LOADING AND VERIFYING PICK MONITOR
PICK MONITOR LOADED AND VERIFIED
LOADING AND VERIFYING ABS
LOADING ABS FRAME > xxxx
```

The display of xxxx will flicker with numbers as frames are loaded. When loaded, the system will jump to BOOT and display the message:

```
ABS LOADED AND VERIFIED

mmm MEMORY
 cc COMM LINES

OPTIONS(X, F, B)
```

Option R is a combination of options A and F. It loads ABS (operating system) and FILES (user files). Option R steps are the same as ABS through "mmm MEMORY, cc COMM LINES." Following this, the prompt will be displayed:

```
SPOOLER STARTED
MOUNT CARTRIDGE AND PRESS RETURN
```

If an ABS (operating system) is recorded at the front of the FILE-SAVE tape, the system will take a few minutes to get past the ABS. The blinking red light on the front of the panel indicates that the cartridge is being accessed. Once ABS is passed, the names and sizes of the files will be brought to the screen. When loaded, the system responds as it does for COLD-START procedure.

2.1.2.3 Binary Backup and RESTORE

In order to perform a binary backup, enter (after the 'Ok,' prompt):

```
Ok, BACKUP CD (or BACKUP CT for Cartridge Tape)
```

The system then responds with:

```
Mount Cartridge 1 (y/n):
```

A 'Y' response initiates the saving of data. The response may be either upper- or lowercase. Since each IOMEGA cartridge is 5M bytes, it will take four cartridges to backup and entire 20M byte system. Once the data is saved, it can be restored with the command (given after the 'Ok,' prompt):

```
Ok, RESTORE CD (or RESTORE CT for Cartridge Tape)
```

Again, the system responds with:

```
Mount Cartridge 1 (y/n):
```

A 'Y' response will initiate the restoring of data.

2.1.2.4 BOOT

At the 'Ok,' prompt, enter 'BOOT'

```
Ok, BOOT
```

The system responds with:

```
LOADING AND VERIFYING PICK MONITOR
PICK MONITOR LOADED AND VERIFIED

256K MEMORY
  cc COMM LINES

OPTIONS (X, F, B)=
```

At this point, a coldstart may be performed by entering 'X' for the X option, or a fileload by entering 'F' for the F option. Currently, the B option returns to the firmware prompt 'Ok,'.

NOTE

The sequence up to the Options message will occur automatically if nothing is keyed for 20 seconds at the initial 'Ok,' prompt.

2.1.2.5 Cartridge and Hard Disk Format Procedure

Before using the removable cartridge disk to perform file-saves or T-DUMPs, the cartridge must be formatted. To format an IOMEGA cartridge disk, you must be at the firmware prompt 'Ok,'. You must ensure that the cartridge is not write protected by ensuring that the write-protect switch on the cartridge is not adjacent to the circle mark. At the 'Ok,' prompt, enter:

```
Ok, FORMAT CD MODEL 0
```

The system responds with:

```
Disk Configured, Proceed With Format (y/<n>):
```

At the '(y/<n>)' prompt, enter 'Y'. The system will respond with the following messages. Note that messages within braces ({}) will appear only if any tracks are to be relocated. (In most of these cases, the system will continue.)

```
Initializing..
Checking..
{Defective Track At Head: 'X' (n),   Cylinder: 'XX' (nnn),   Status: 'XX'}
Format Complete, {Defective Tracks:}  'XX'
Add Defect (Head, Cylinder):
{Mapping Alternate Tracks}
```

At the 'Add Defect (Head, Cylinder):' prompt, hit RETURN and if any tracks are to be relocated, it will be done automatically. (Or you may specify tracks that the system has not found by entering Head and Cylinder numbers in hexadecimal, separated by commas.)

Should there be a need to format the 20MB hard disk drive, then, at the firmware prompt 'Ok,', enter:

```
Ok, FORMAT DISK 0 MODEL 3
```

During formatting, the defective head and track numbers are displayed. When formatting is completed, the number of defective tracks is displayed and the user is prompted to enter additional tracks from the manufacturer's defect list. The format of the response is defined as follows:

```
Add Defect (Head, Cylinder):  [!] head cylinder
```

The user responds with the head and cylinder number of the additional defective track. If the numbers are preceded with an exclamation point (!), they are taken to be decimal. A null input will terminate the defect list and all defective tracks are then remapped. After mapping, a 2048-byte configuration table/defect map are written into the first secton on track 0, head 0. See 700/750 Hardware Reference Manual (88A00785A) for further information.

After the hard disk has been formatted, you will need to reload the entire system, either with a SYSGEN cartridge or a previously binary-saved cartridge.

2.2 LOGGING OPERATION

The Logon processor initiates user sessions by identifying valid users and their associated passwords. The Logoff processor is used to terminate the session and should always be invoked via the verb OFF when the user wishes to terminate. These processors can accumulate accounting statistics for billing purposes and associate the user with his privileges and security codes.

2.2.1 LOGON

The user may logon to the PICK system when the following message is displayed:

```
LOGON TO THE GA ZEBRA AT 00:00:00
PLEASE ENTER ACCOUNT NAME >
```

The actual form of this message will vary since the message format is obtained from an entry called "LOGON" in the SYSTEM dictionary.

The user enters the name (identification) established for him in the system, followed by a carriage-return. If a password has also been established, there will be a prompt:

```
PASSWORD:
```

The user then enters the password, followed by a carriage-return. If a valid password is not entered, the system will display the message:

```
PASSWORD?
```

Note that it is possible to enter identification followed by a comma and the password on one line, but then the password will be visible.

The system validates the user's identification against the entries in the SYSTEM dictionary; if it is illegal, the following message is returned:

```
USER-ID?
```

The user must reenter his identification and password. If the identification is valid but the password is not, the user must then reenter both his identification and password. If the user has successfully logged onto the system (i.e., both the identification and the password have been accepted), the following message is displayed:

```
<<<R80 GENERAL AUTOMATION REV: m.n>>>
<<<time   ZEBRA              date>>>
  >
```

where "time" is the current time, "date" is the current date, and "R80" is the current release level. The ">" is the TCL prompt character which indicates that the user may enter any valid TCL command.

2.2.2 LOGOFF

Logoff is achieved by entering the word OFF either at the TCL level or at the DEBUG level. A message indicating the connect time (i.e., number of minutes the user was logged on) and the appropriate charge units will be displayed. The system then displays the LOGON PLEASE message and waits for the next user session to be initiated. The general form of the logoff message is:

```
<CONNECT TIME = n MINS.; CPU = m UNITS, LPTR PAGES = x>
<LOGGED OFF AT   time         ON   date              >
```

where 'n' is the number of minutes of connect time, "m" is the number of charge units, "time" is the current time, and "date" is the current date, and "x" is the number of line-printer pages generated. The charge-units represent usage of the CPU; it is in tenths of a CPU second. An example of Logon, Logoff:

```
PLEASE ENTER ACCOUNT NAME> TEST [CR] <---------- Valid identification.
PASSWORD:   <-------------------- Valid password.  [CR]

<<<R80 GENERAL AUTOMATION REV: m.n>>>
<<<14:33:08   ZEBRA     3 JAN 1982>>>

>OFF  [CR]

<CONNECT TIME = 5 MINS.;  CPU = 6 UNITS;  LPTR PAGES = 15 >
<LOGGED OFF AT 17:55:44  ON 3 JAN 1983>

LOGON TO THE GA ZEBRA AT 17:56:01
PLEASE ENTER ACCOUNT NAME >
```

2.2.3 ADDITIONAL LOGON FUNCTIONS

The LOGTO verb allows the user to log to another account faster than by going through the OFF and LOGON process. The CHARGE-TO verb allows the user to charge a particular logon session to a specific charge number or name; the CHARGES verb displays the charge statistics for the current logon session.

2.2.3.1 LOGTO

The general form of the LOGTO verb is as follows:

```
LOGTO acctname  [CR]
```

where "acctname" is that of the new account that the user wishes to logon to. If the account has a password defined, the message:

```
PASSWORD:
```

will be displayed and the password may then be entered.

If the account name is illegal, the message "USER ID?" will be printed and the user will be back at TCL. If the password is incorrect, the message "PASSWORD?" will be displayed.

If the account name and password are both correct, the current logon sessions will be terminated by updating the accounting file with the appropriate statistics and a new session started. The message:

```
<CONNECT TIME = n MINS.;  CPU = m UNITS;  LPTR PAGES = x >
```

will be displayed.

Note that it is possible to enter the acctname and password separated by a comma on the same line, but that then the password will be visible.

Also, the tape unit and line-printer will be detached if the user had them attached to this line prior to the LOGTO.

2.2.3.2 CHARGE-TO

The CHARGE-TO verb is used to keep track of computer usage for several projects associated with the same logon name. This verb performs the following:

1. Terminates the current charge session by updating the ACC file with the user's accumulated charge-units, line printer pages and connect-time statistics.

2. Changes the logon name to the original name concatenated with an asterisk and then the name following "CHARGE-TO".

The CHARGE-TO verb has the following general form:

```
CHARGE-TO {acctname}
```

where "acctname" is any sequence of non-blank characters. This statement will cause the current logon session to be terminated and the account file to be updated with the appropriate statistics; a new session is started with the new user identification of the form:

```
logon account name*acctname
```

where "acctname" is the account name specified in the CHARGE-TO statement. This allows the user to charge his logon sessions to specific names or numbers. If "acctname" is null in the CHARGE-TO statement, the user identification will revert to the logon account name alone.

The CHARGE-TO statement will also cause the following message to be displayed:

```
<CONNECT TIME = n MINS.;  CPU = m UNITS;  LPTR PAGES = x >
```

2.2.3.3 CHARGES

The CHARGES verb prints the current computer usage since logon as connect time in minutes and CPU usage in charge-units. The general form of this verb is:

```
CHARGES
```

This will display the logon statistics with the following message:

```
<CONNECT TIME = n MINS.;  CPU = m UNITS;  LPTR PAGES = x >
```

Sample usage of LOGTO, CHARGE-TO and CHARGES verb:

```
*   PLEASE ENTER ACCOUNT NAME >  SMITH,XYZ  [CR]

    <<<R80 GENERAL AUTOMATION REV: m.n>>>
    <<<09:15:33       ZEBRA          11 DEC 1982>>>

*   >WHO  [CR]
    7 SMITH

*   >TIME  [CR]
    09:17:00  11 DEC 1982

*   >LOGTO JONEA  [CR]

    USER-ID?

*   >LOGTO JONES  [CR]
*   PASSWORD:  ABC  [CR]

 <CONNECT TIME = 3 MINS.;  CPU = 11 UNITS;  LPTR PAGES = 0 >

*   >WHO  [CR]
    7 JONES

*   >CHARGE-TO A001  [CR]

 <CONNECT TIME = 0 MINS.;  CPU = 7 UNITS;  LPTR PAGES = 0 >

*   >WHO  [CR]
    7 JONES*A001

*   >CHARGES  [CR]

 <CONNECT TIME = 0 MINS.;  CPU = 8 UNITS;  LPTR PAGES = 0 >

*   >CHARGE-TO  [CR]

 <CONNECT TIME = 0 MINS.;  CPU = 9 UNITS;  LPTR PAGES = 0 >

*   >WHO  [CR]
    7 JONES
```

2.2.4 THE LOGON PROC AND GENERAL SYSTEM MESSAGES

Upon logon, PICK allows for the execution of a PROC with an item-id identical to the user's identification. PICK also allows a general message to be sent to each user as he logs onto the system.

When the user has logged on to this account, PICK permits the automatic execution of PROC whose item-id is the same as the user's identification. That is, the Master Dictionary (MD) of the account will be searched for a PROC matching the identification which was used to log on to the account; if it is found, it will be executed. (For further information regarding PROC's, refer to PICK PROC Reference Manual, 88A00730A.)

Typically, the Logon PROC is used to perform standard functions that are always associated with the particular user's needs. For example, setting of terminal characteristics could be performed by the Logon PROC. When the user logs on to the system, his terminal characteristics are set to the following initial conditions:

	Terminal	Printer
Page Width:	79	132 (characters)
Page Depth:	24	60 (lines)
Line Skip:	0	
LF Delay:	1	
FF Delay:	5	
Backspace:	8	
Term Type:	M	

These conditions can subsequently be displayed and altered by the TCL verb TERM. As an example, assume that the following PROC:

```
item 'SMITH' in MD of user SMITH

001   PQ
002   HTERM 118,44,7,6
003   P
004   X***  TERMINAL CHARACTERISTICS SET ***
```

is stored as item SMITH in the user's Master Dictionary (MD).

If the user's identification is the word SMITH, then the SMITH PROC will be executed automatically every time the user logs on (i.e., the user's particular terminal characteristics will automatically be set). This is illustrated as follows:

```
PLEASE ENTER ACCOUNT NAME > SMITH,XYZ [CR] <---------- Logon sequence.

**** NOTE: SYSTEM PRINTER WILL BE DOWN AT 12:00    <- General system
**** TODAY FOR ROUTINE MAINTENANCE UNTIL 2:00PM        message.

<<<R80 GENERAL AUTOMATION REV: m.n>>>
<<<17:09:50    ZEBRA          05 DEC 1982>>>

***   TERMINAL CHARACTERISTICS SET  *** <---------- Message from SMITH
                                                    PROC.
>  <------------------------------------------------ TCL prompt
                                                    character.
```

A general system message may be stored in the special item "LOGON" of the system error-message file ERRMSG; this file and the formats of the data in it are described in Section 5.6. This system message will be printed immediately before the standard logon message.

Typically, this facility is used to broadcast messages relating to the system to everybody who logs on to it. Note that the item "LOGON" must exist in the ERRMSG file even if there is to be no system-wide message; in this case, it will be a null item.

2.2.5 POWERFAIL AND GENERAL SYSTEM MESSAGE

In the event of a power failure, battery backup will be provided for a limited period of time in order to permit all users to log off for an orderly shutdown. (Instructions to implement this procedure are in the Enhancement section of the 2.1 Release Notes.)

After a power failure, all logged-on users will receive the following general system message:

```
time:          date:          from:  SYSTEM
POWERFAIL - SHUTDOWN IMMINENT, LOGOFF NOW!
```

Users should log off immediately after receiving this message in order to prevent loss of data.

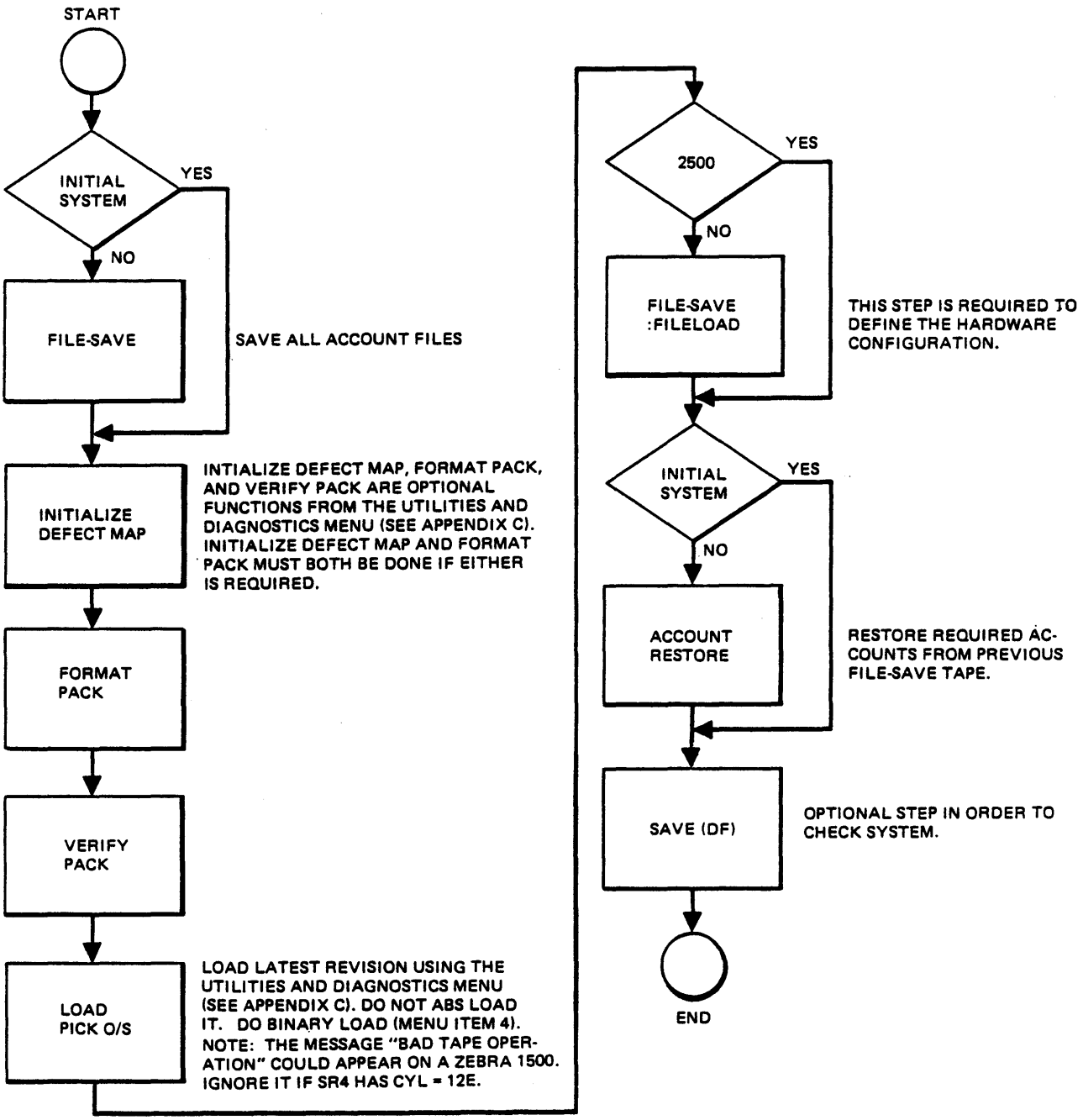


Figure 2-1. Initial Load/Reload Operating System

terminal control language (TCL) 3

TCL is a language processor that provides the communication media between a user-operator and the PICK operating system. All other processors, such as EDITOR, BASIC, and ACCESS, are invoked by TCL. In addition, any PICK verb in a command sentence is invoked from a keyboard, or from direct execution by a PROC, which is a series of stored instructions.

TCL is automatically entered during LOGON and returned to whenever a particular verb process (such as LIST, DUMP, COUNT), or a series of processes are complete. The display prompt for TCL is ">". This prompt is displayed at the upper left of the display and indicates that the system awaits input from the keyboard.

A TCL input statement will implement one of a set of verbs residing in the user's Master Dictionary (MD) (see Section 6). This verb will perform a set of steps specified at the keyboard, or will involve another processor to carry out specified steps. The latter is often the case where the "specified steps" are commonly used and required frequently. For example, the "RUN PROGRAMS ABC" causes the BASIC program ABC in the PROGRAMS file to be executed.

3.1 TCL VERB TYPES

The TCL verbs are of three types: TYPE-I verbs perform specific functions, but do not access file data. For example, TIME is a TYPE-I verb. TYPE-II verbs are those which access file data. For example, RUN, COPY, COMPILE are TYPE-II verbs. The TYPE-III verbs are for ACCESS, providing the user with immediate access, via keyboard, to his file data. Using the TYPE-III verbs, he has an unlimited flexibility for selecting the desired data, formatting and presenting it. These verbs are totally covered in the PICK Reference Manual, ACCESS section.

3.2 TCL VERB ATTRIBUTES

Table 3-1 identifies PICK verbs in alphabetical sequence and by type. The key attribute identifiers shown in this table (such as attribute 1 for DIVD is "PD") are defined as follows:

Attribute Number	Description
0	This is the item-id, which is the name of the verb.
1	Must contain a P optionally followed by another letter. P identifies the MD item as a verb definition item. The letter following P is passed to the defined processor. Some of these are: A defines an ACCESS verb. Q defines a PROC verb. Y defines a TCL-II verb. Z defines a TCL-I verb.
2	This attribute defines the processor entry point to which TCL passes control (i.e., the mode-id in hex). An ACCESS verb will have an entry of: 35 A Type-II verb will have an entry of: 2 A Type-I verb will have an entry of: xxxx where xxxx is the entry point.
3	Secondary transfer point. Use depends on attributes 1 and 2.
4	Tertiary transfer point. Use depends on attributes 1 and 2.

5 TCL-II parameter string. These parameters govern treatment of the items retrieval by TCL-II verbs to be passed to the processor whose entry point is defined in attribute three. Parameter may be any of the following:

- C Copy item to a work area.
- F Pick up file parameters only (ignore item-list).
- N Okay if item is not on file.
- P Print item-id if item-list is "*" (all items) or if SELECT-ed item-list.
- S Ignore the select-list; item-list is mandatory.
- U Items will be updated by processor.
- Z Final entry required on EOI.

Attribute zero, for all verbs, is item-id, the name assigned to an item as shown in left-most column of the table.

3.3 TCL VERB LIST

Table 3-1. PICK Verbs (Sheet 1 of 5)

Item-Id	Privilege Level	Attribute A1	A2	A3	A4	A5
%%SELP%%	0	P	3146			
:FILELOAD	2	P	20D7			
:STARTSPOOLER	2	P	500A	60B6	10AF	
:TASKINIT	S	P	13C			
ACCOUNT-RESTORE	2	P	80D7			
ADDD	0	PA	40A0			
ADDX	0	PA	A0			
B/ADD	1	PZ	2	53	57	CS
B/DEL	1	PD	2	53	57	CS
B/UNLOCK	2	P	4058			
BASIC	0	P	2	BE		UP
BLOCK-PRINT	0	P	500A	41		
BREAK-KEY-OFF	S	P	418C			
BREAK-KEY-ON	S	P	518C			
CATALOG	0	P	2	E0		
CHARGE-TO	0	P	6032			
CHARGES	0	P	5032			
CHECK-SUM	0	PX	35	4F		
CLEAR-FILE	1	PO	8F			
COMPARE	2	P	2	17D		
COMPILE	0	P	2	BE		UP
COPY	0	PZ	2	8C		UZ
COPY-LIST	0	PL	3013			
COUNT	0	PB	35	1069		
CREATE-FILE	1	PO	8D			
CREATE-PFILE	1	PO	408D			
CROSS-INDEX	2	PZ	2	92		C
DATE	0	P	618C			
DATE-FLAG	S	P	305B			
DECATALOG	1	PY	2	E0		N
DELETE	1	PD	2	509E		CUZ
DELETE-ACCOUNT	S	P	0DE			
DELETE-FILE	1	PO	108F			
DELETE-LIST	1	PL	006D			
DIVD	0	PD	40A0			
DIVX	0	PD	A0			

Table 3-1. PICK Verbs (Sheet 2 of 5)

Item-Id	Privilege Level	Attribute A1	A2	A3	A4	A5
DTR	0	P	10A0			
DTX	0	P	10A0			
DUMP	2	PZ	42			
ECHO	0	P	5080			
ED{IT}	0	PE	2	D		CUPN
EDIT-LIST	0	PL	13			
EXCHANGE	0	PQ				
FILE-SAVE	S	PQ				
FORMAT	0	PQ				
GET	0	PZ	2	142		FUN
GET-LIST	0	PL	1064			
GROUP	0	P	2	50A0		F
HASH-TEST	0	PA	35	106A		
INIT-CURSOR	S	P	2	118F		
ISTAT	0	PA	35	6A		
ITEM	0	P	2	30A0		N
JET-EDIT	0	P	2	2154		CUN
JET-IN	0	P	2	154		CUN
JET-OUT	0	P	2	1154		CS
LAN	0	P	143			
LANOFF	0	P	2143			
LINK-WS	2	P	40AC			
LIST	0	PA	35	4D		
LIST-ITEM	0	PA	35	4D	508E	
LIST-LABEL	0	PA	35	4D	7D	
LIST-LOCKS	0	P	500A	30CD		
LISTABS	0	P	20AE			
LISTPEQS	0	P	AB			
LISTPTR	0	P	AE			
LOAD	0	PZ	2	142		FUN
LOCK-FRAME	2	P	3041			
LOGOFF	2	P	1116			
LOGON	2	P	116			
LOGTO	0	PG	7032			
MAS	S	P	2	17	32	CUP
MESSAGE	2	PG	1034			
MLIST	2	PY	2	20		C
MLOAD	2	P	2	1F		

Table 3-1. PICK Verbs (Sheet 3 of 5)

Item-Id	Privilege Level	Attribute A1	A2	A3	A4	A5
MSG	2	PG	1034			
MULD	0	PM	40A0			
MULX	0	PM	A0			
MVERIFY	2	P	2	101F		
NODE	0	P	147			
OFF	0	PZ	32			
P	0	P	2080			
PAS	S	PA	2	17	1B	CUP
PASSWORD	S	P	186			
PIE	0	PA35	4D	11CC		
PLOT	0	PA35	4D	01CC		
POVF	0	P	40BB			
PRINT-ERR	0	P	2	43		
PRINTRONIX	1	P	40B4			
PVERIFY	0	PZ	2	E0		N
QSELECT	0	P	2	303A	0	Z
RECOVER-FD	0	P	1012			
REFORMAT	0	PA	35	4D	12C	
RESET-PORT	S	P	B18C			
RTD	0	P	20A0			
RUN	0	PR	2	E6		F
RUNOFF	0	P	2	126		S
S-DUMP	1	PF	35	4E	25	
SAVE	2	P	C8			
SAVE-LIST	0	PL	2064			
SEL-RESTORE	1	P	90D7			
SELECT	0	PB	35	3076		
SELP	0	P	2146			
SEND	0	PZ	2	140		UZ
SET-1/2	1	P	20A8			
SET-1/4	1	P	D0A8			
SET-BAUD	2	P	10E			
SET-CD	1	P	F0A8			
SET-CT	1	P	D0A8			
SET-DATE	S	PZ	2033			
SET-MT	1	P	20A8			
SET-SYM	0	P	2	89		F
SET-TERM	S	PL	607A			
SET-TIME	S	PZ	1033			

Table 3-1. PICK Verbs (Sheet 4 of 5)

Item-Id	Privilege Level	Attribute A1	A2	A3	A4	A5
SLEEP	0	P	107A			
SORT	0	PA	35	4E		
SORT-ITEM	0	PA	35	4E	508E	
SORT-LABEL	0	PA	35	4E	7D	
SP-ASSIGN	0	P	10A7			
SP-CLOSE	0	P	A7			
SP-EDIT	0	P	A9			
SP-SKILL	0	P	10AD			
SP-OPEN	0	P	20A7			
SP-STATUS	0	P	50A6			
SP-TAPEOUT	1	P	20B4			
SPIE	0	PA	35	4E	11CC	
SPLOT	0	PA	35	4E	01CC	
SREFORMAT	0	PA	35	4E	12C	
SSELECT	0	PB	35	4E	3076	
STACK-OFF	0	P	2188			
STACK-ON	0	P	1188			
STARTPTR	2	P	B2			
STAT	1	PF	35	4F		
STOPPTR	2	P	AD			
STRIP-SOURCE	S	P	2	1091		CUP
SUBD	0	PS	40A0			
SUBX	0	PS	A0			
SUM	0	PW	35	4F		
T-ASSIGN	2	P	9A			
T-ATT	1	P	80A8			
T-BCK	1	PI	6033	B023		
T-CHK	1	P	B09A			
T-DET	1	P	A0A8			
T-DUMP	1	PA	35	25		
T-EOD	1	P	709A			
T-EOF	1	P	C09B			
T-EOFD	1	P	D09B			
T-FWD	1	PI	6033	9023		
T-LOAD	1	PT	35	73		
T-RDLBL	1	PO	6033	2024	A09A	
T-READ	1	P	99			

Table 3-1. PICK Verbs (Sheet 5 of 5)

Item-Id	Privilege Level	Attribute A1	A2	A3	A4	A5
T-RET	1	P	B09B			
T-REW	1	PO	6033	8023		
T-STATUS	1	P	E0A8			
T-UNLOAD	1	PO	6033	909B		
T-WEOF	1	PN	6033	1024		
T-WTLBL	1	PO	6033	2099	A09A	
TABS	0	P	80			
TA-ON	0	P	110E			
TA-OFF	0	P	210E			
TERM	0	PZ	607A			
TIME	1	PZ	3033			
UNLOCK-FRAME	2	P	4041			
VERIFY-SYSTEM	0	P	11F			
WHAT	2	P	30BB			
WHERE	2	P	1079			
WHICH	0	P	409E			
WHO	0	P	10BB			
XOFF-DISABLE	1	P	918C			
XOFF-ENABLE	1	P	A18C			
XTD	0	P	20A0			

NOTES:

Privilege Level indicates the minimum system privileges needed to use a verb.

Levels are: 0, 1, 2, and S.

The lowest is 0.

The highest, S, indicates that the verb may only be used from SYSPROG.

3.4 VERB DEFINITION FOR MD

Examples of verb definition items using the attributes shown in Table 3-1:

1. Define the ACCESS verb LIST:

```
item 'LIST' in MD

    001  PA
    002  35
    003  4D
```

2. Define the TCL-II verb COPY:

```
item 'COPY' in MD

    001  PZ
    002  2
    003  8C
    004
    005  UZ
```

3. Define the TCL-I verb TIME:

```
item 'TIME' in MD

    001  PZ
    002  3033
```

Any number of synonyms for such verb definition items may be created (or removed or changed) by the user. This can be accomplished by copying the verb definition item to another master dictionary item with the new item-id for the desired synonym name.

3.5 TCL VERB STATEMENT

3.5.1 TYPE-I STATEMENTS

Type-I statements do not access a file. The general form is simply a prompt (>) followed by the verb and a carriage return.

3.5.2 TYPE-II STATEMENTS

TCL Type-II verbs allow access to a specified file. The format for forming a TCL-II input statement is more restrictive than for an ACCESS statement. The advantage gained by this restricted format is an enhancement in processing speed since statement parsing is quicker. The general form of TCL-II statement:

```
>VERB {DICT} file-name {item-list} {(options)}
```

A file-name (or DICT file-name) must immediately follow the TCL-II verb. Item selection is more restricted than in ACCESS statements, since each item-id must be explicitly named in the statement (or, alternatively, all items may be specified via use of the asterisk (*) character).

File-name specifies the desired file. The DICT option specifies the dictionary portion of the file. The item-list is made up of one or more item-id's, separated by one or more blanks. If an item-id contains embedded blanks or parentheses, it must be surrounded by single quotes. All items in a file may be specified by using an asterisk (*) character as the item-list. Options, if specified, must be enclosed in parentheses at the end of the input line. Multiple options may be separated by commas. The specified options are passed to the appropriate TCL-II processor.

The item-list may be omitted entirely, if the TCL-II statement is preceded by a statement that generates a "select-list"; the item-ids are then obtained from this preselected list. Statements that generate select-lists are SELECT, SSELECT, QSELECT and GET-LIST, and are described in ACCESS section of the PICK Reference Manual.

3.6 TERMINAL/PRINTER CONTROLS

3.6.1 KEYBOARD CONTROLS

The description of TCL in previous sections has implied the use of a standard typewriter keyboard. Certain keys and combinations of keys, however, are unique for PICK operation. NOTE: Some terminals may use different letters than the ones shown below for these functions. If a control key does not work as described, the ASCII equivalent will have to be looked up in the manual for the terminal. ASCII equivalents are given in square brackets.

CONTROL

H — Backspaces your cursor one character. [BS]

R — (Retype) Causes the line you are on to be redisplayed on the next line. This is useful if you have backspaced and retyped since you can confirm what is actually there and that the characters you wanted erased were erased. [DC2]

W — Backspaces your cursor one word. [ETB]

X — Causes the current line to be ignored and moves your cursor back to the prompt. This is also used if you are displaying a long ACCESS listing and wish to end it at the end of a page. You can also end a display by using the Control-X.

0 Sp — (With both the Control and space bar pressed) allows you to continue entering but on the next line. When you use this, the prompt on the next line will be a colon (:). [US]

BREAK — Usually this stops whatever process the computer is doing as a result of input from your terminal (line). Sometimes the BREAK key and the CONTROL key must be pressed together to get the break result. In the break condition, the prompt is an exclamation mark rather than the greater than sign. You have two options for getting out of the break condition: you can enter two "line feeds" (not "New Line") to pick up from where you were when you "broke", or you can type in "END" which will return you to the regular (>) prompt.

NEW LINE or ENTER — This key, like carriage return, sends whatever you have keyed in on your keyboard to the computer.

ESC or ESCAPE — This key, like the Control key, is used with other keys to cause certain results. It is used in some word processing functions but not in normal operator procedures. It is used to chain editor commands together as described in the section on the Editor in the PICK Reference Manual.

3.6.2 SETTING TERMINAL/PRINTER CONTROLS

The terminal and/or line printer characteristics may be displayed or set by a process via the TERM command. The general form of the TERM command:

TERM {a,b,c,d,e,f,g,h,t}

where:

a — is the terminal line length (i.e., number of characters per line). The a parameter must be in the following range: 16<a<140.

b — is the number of print lines per page on the terminal.

c — is the number of blank lines per page on the terminal (sum of b and c equals page length).

d — is the number of delay or idle characters following each carriage return or line feed. This is used for terminals that require a pause after a carriage return or line feed (i.e., since the CPU generates characters faster than the terminal can accept them).

e — is the number of delay characters following each top-of-form. If e is zero, no form-feed character will be sent to either the terminal or the printer.

If e is non-zero, a form-feed character is also output before each page; if e is 1, this character is sent to the line-printer, but not to the terminal.

If e is greater than 1, the form-feed character is also sent to the terminal at the beginning of each page, and that many delay or idle characters is also sent to allow the terminal time to settle after the form-feed.

The form-feed character sent to the printer is always a hexadecimal 'OC' (ASCII FF character).

f — is the backspace character. An ASCII backspace (Control-H) is always input to backspace over (or erase the last character that was input; however, the user may set the actual character echoed to his terminal). This accommodates terminals that cannot physically backspace, or that have a backspace character other than the ASCII backspace. The f parameter should be 21 for the ADDS REGENT terminal, and d for the TEC 2404 terminal.

g is the line printer line length.

h is the line printer page length.

t is the terminal type code; this changes the form-feed character sent by the system to match the terminal requirements, and, more importantly, sets the appropriate cursor addressing for the BASIC cursor functions. The letters used for supported terminals are:

L = LEAR-SIEGLER ADM-11, ADM-12

M = AMPEX, DIALOGUE 80

T = TELEVIDEO 925, 950

V = ADDS VIEWPOINT

X = No cursor addressing functions needed.

Individual parameters may be null (i.e., as specified by two adjacent commas in the TERM command). If so, the previously defined parameter remains in force. A TERM command without a parameter list causes display of the current characteristics. To function properly, the t parameter must be the last element in any TERM string. It may be the only statement if no other elements are to be changed. The other parameters are positional, however.

An example of the use of TERM:

```
>TERM  [CR]

              TERMINAL PRINTER
PAGE WIDTH:       79      132
PAGE DEPTH:       24       64
LINE SKIP:         0
LF DELAY:          1
FF DELAY:          2
BACKSPACE:         8
TERM TYPE:         M

Standard characteristics set for DIALOGUE terminal.

>TERM,,,,,,120,48 [CR]

Resets the line-printer page size to 120x48.

>TERM  [CR]
              TERMINAL PRINTER
PAGE WIDTH:       79      120
PAGE DEPTH:       24       48
LINE SKIP:         0
LF DELAY:          1
FF DELAY:          2
BACKSPACE:         8
TERM TYPE:         M
```

Note that an FF delay of 0 will cause the printer not to do a top-of-form and will not clear the screen on the terminal. An FF delay of 1 will not clear the screen on the terminal.

3.6.2.1 SET-TERM

The general form of the SET-TERM command:

```
SET-TERM {a,b,c,d,e,f,g,h,t}
```

The SET-TERM verb sets the default printer and terminal characteristics for subsequent logons on all terminals. This verb is present only on the SYSPROG account; the parameters are the same as for those for the TERM command.

3.6.2.2 Changing Baud Rate (SET-BAUD)

On IPL, this baud rate of the communication ports for terminals and printers are default set to 9600. If you wish to change the rate for a particular terminal on a printer, the following verb should be used:

```
SET-BAUD  {p,}n
```

where: p is the port or channel number to which the terminal or printer you wish to change is physically connected. If p is not specified, the baud rate of the line currently being used will be changed.

n is the baud rate desired.

3.6.3 SETTING TAB STOPS

Tab stops may be set with the TABS statement. The general form of the TABS command is as follows:

```
TABS I or O  n1,n2,.....n15
```
or
```
TABS I or O {S}
```

where the tabs may be set for input or output depending on the parameter "I" or "O" following the TABS verb. n1, n2, etc., are up to fifteen tab-stop positions, which must be entered in ascending numerical sequence.

Tabs set for input are then available at any time that the system requests input from the terminal. By entering a Control-I ([cI]), the system will space over to the next tab-stop position, if any. If there are no more tab-stop positions, the [cI] is ignored (Control-I is also generated by the TAB key on some terminals). The TAB stops set by the TABS I statement are identical to those set by the TB statement in the EDITOR.

Tabs set for output are only useful for those printing terminals that have a physical tabbing capability. Do not set output tabs for a CRT. If output tab stops are set, the system will replace blank sequences in any output generated by the system by an appropriate tab character ([cI]), thus, reducing the data output. The user must also set up the physical tab stops on the terminal to correspond to those set in the TABS O statement. On many terminals, this entails positioning the carriage and entering a set-tabs sequence from the keyboard.

Input or output tab stops may be disabled by entering "TABS I" or "TABS O", respectively. Previously set tab stops may then be recalled by entering "TABS I S" or "TABS O S" for input and output tab stops, respectively. Current tab stops can be displayed by entering "TABS" alone.

Examples of the use of TABS:

```
>TABS I  4,8,12,16,20,24,28  [CR]          (sets input tab stops)

>TABS O  10,20,30,40,50,60   [CR]          (sets output tab stops)

>TABS  [CR]                                (displays current tab stops)

         1         2         3         4         5         6         7
12345678901234567890123456789012345678901234567890123456789012345678901234567890
   I   I   I   I   I   I   I
         0         0         0         0         0         0

>TABS O  [CR]                              (turns off output tab stops)

>TABS  [CR]

         1         2         3         4         5         6         7
12345678901234567890123456789012345678901234567890123456789012345678901234567890
   I   I   I   I   I   I   I

>TABS I  5,15,20,40,50  [CR]

>TABS O S  [CR]                        (recalls previous output tabs stops)

>TABS  [CR]

         1         2         3         4         5         6         7
12345678901234567890123456789012345678901234567890123456789012345678901234567890
    I         I    I                   I         I
         0         0         0         0         0         0
```

3.6.4 ENABLE/DISENABLE CHARACTER ECHO

3.6.4.1 The ECHO Verb

The function of this verb is to toggle the switch in each user's PIB indicating whether or not characters typed in are to be echoed to the terminal. Thus, typing ECHO in normal mode will cause all further typing to be echo-suppressed. Similarly, typing ECHO in suppressed mode will cause echoing to resume. The user may also force a particular echo status. Typing ECHO (I) will force echo suppression, just as ECHO (L) will force echoing.

3.6.5 BLOCK PRINTING

3.6.5.1 Block-Print

The BLOCK-PRINT command will print characters in block-form on the line printer or the user's terminal. Any ASCII characters may be printed. The general form of the BLOCK-PRINT command:

BLOCK-PRINT character-string {(P)}

This command causes the specified character-string to be block-printed on the terminal. Any character-string containing single quotes (') must be enclosed in double quotes ("), and vice versa. The surrounding quotes will not be printed. A character-string not containing quotes as part of the string need not be surrounded by quotes.

The option (P) will route the output to the line printer.

Character-strings to be blocked cannot have more than nine characters. For the BLOCK-PRINT command, the total number of characters must not exceed the current line length set by the most recent TERM command.

If a BLOCK-PRINT command is illegally formed, any of the error messages 521 through 525 may be displayed.

The BLOCK-PRINT commands use a file named BLOCK-CONVERT to create the blocked characters. A BLOCK-CONVERT file already exists which contains the conversion specifications for all printable ASCII characters (no lower case alphas). With this file, characters will be printed as 9-by-12 to 9-by-20 blocks.

If it is desired to change the way any character is printed, it is necessary to change the corresponding item in the BLOCK-CONVERT file. (See Section 5.3.2, BLOCK-CONVERT FILE.)

3.6.6 GENERAL SERVICE VERBS

This section discusses TCL-I verbs such as TIME, SLEEP, WHO, and MSG. These verbs are used for general service to the terminal operators.

3.6.6.1 TIME

The TIME statement displays the current system time and date. Its general form is:

```
TIME
```

For example:

```
>TIME  [CR]
09:11:23  11 DEC 1982
```

3.6.6.2 WHO

The WHO statement is used to display the account-name that a terminal is logged on to. Its general form is:

```
WHO {n}
```

If WHO is entered without the "n", the line-number (channel number) of the user's terminal is displayed, along with the account-name that he is logged on to. If the "n" is specified, the same data is displayed for line-number "n", where n ranges from 0 to the maximum number of lines on the current system. If the line is non-existent, or if no user is logged on to that line, the account-name is replaced with "UNKNOWN".

Lines which have the name UNKNOWN and which are actively processing suggest that something wrong is happening on the system, unless these lines are spoolers or you have an account on the system by the name of UNKNOWN.

You may specify a line or a range of lines. Any non-numeric character will cause WHO to display all lines and their logon name. For example:

```
>WHO  [CR]
07 SMITH                (this is line-number 7, logged on to "SMITH")

>WHO 0  [CR]
00 SYSPROG              (line number 0 is logged on to SYSPROG)

>WHO 11  [CR]
11 UNKNOWN

>WHO *                  (displays accounts using all lines; lines which are
                         not logged on display UNKNOWN.)

>WHO 1-3

01 JOHN
02 SYSPROG
03 UNKNOWN

>WHO 'SYSPROG'          (displays all lines logged onto the SYSPROG
                         account.
```

3.6.6.3 SLEEP

The SLEEP verb is used to put a terminal to "sleep", that is, to enter a quiescent state for a specified period of time, or until a specified time. Its general form is:

```
SLEEP x
```

where "x" is either a decimal number specifying the number of seconds to sleep, or is of the form "hh:mm:ss" or "hh:mm", specifying a time in 24-hour format until which to sleep. SLEEP is useful to cause a terminal to wait until some time to run a task, for instance, the FILE-SAVE may be run at 23:00 (11:00PM) every night. For example:

```
>SLEEP 100  [CR]          (terminal will sleep for 100 seconds)
>SLEEP 23:00  [CR]        (terminal will wake up at 11:00 pm)
```

The form of SLEEP with a wake-up time is usable for a maximum of 24 hours.

3.6.6.4 MESSAGES

The MSG (or MESSAGE) statement allows one user to send a message to another. The general form of the MSG statement is:

```
    MSG[destination-account][text]
or
    MSG[!line#]
```

where "destination-account" is the name of the account that the other user is logged on to, and "text" is the message that follows. The message text is not edited in any way; there is no "options" parameter in the MSG statement.

Note that ALL users who are logged on to the specified destination-account will receive the message.

Users with system level 2 privileges (see Section 5.2.1) can broadcast a message to all users by substituting an asterisk (*) for the "desination-account" in the MSG statement. This message will be received by the sending user's terminal also.

A user who was entering data when a message is received will lose up to 16 characters due to the interference of the message; he should use the Control-R to see exactly what data is left. Some examples:

```
    >MSG MARY*A0001 WHAT'S THE STATUS OF THE INVENTORY REPORT???  [CR]

    >MESSAGE JONES HELLO THERE!"%%%%'''%  [CR]

    USER NOT LOGGED ON                          (JONES is not logged on).

    >MSG * SYSTEM FILE-SAVE WILL START IN 5 MINUTES!!!  [CR]
```

MESSAGE and MSG verbs may direct a message to a particular line as well as to a particular user by preceding the line number with an exclamation mark (!). This form of the verb will send messages to terminals which are not logged-on. Further, the user may send a message to all lines, signed on or not, through a special form of this verb. For example:

```
    MESSAGE !12 HELLO       Send the message 'HELLO' to the user on line 12.

    MSG !* SIGN OFF NOW     Sends a message to all terminals connected to the
                            computer.
```

3.7 TAPE OPERATION AND CARTRIDGE DISK: SET-1/2 (OR SET-MT), SET-1/4 (OR SET-CT), AND SET-CD

3.7.1 SETTING TAPE SIZE

PICK allows you to use either 1/2-inch or 1/4-inch tape. It is necessary, however, to tell the system which tape size is currently being used. The following verbs will specify the type of tape and attach the drive at the same time:

SET-1/2 {n} {U} which is the same as SET-MT {n} {U}

or

SET-1/4 {n} {U} which is the same as SET-CT {n} {U}

Block size may be specified by n. If it is not, the last block size used, or if none, the system default of 4000 bytes will be used. Note that 4000 bytes is the most efficient blocksize to use for 1/4-inch tape. If it is necessary to use a larger blocksize, 8000 bytes is the maximum recommended.

The U option unconditionally attaches the tape drive by detaching any other line that is attached to the drive. SYS2 privileges are necessary to use the 'U' option.

The verb will remain in effect until you change it or IPL is performed.

3.7.1.1 Attaching Cartridge Disk

The following verb tells the system you are using cartridge disk and attaches the drive at block size 1024:

SET-CD

3.7.2 FULLY UTILIZING 1/4" TAPE BY LOCATING EOF'S: T-EOF, T-EOFD

A physical EOF is automatically generated when a T-REW command is issued immediately following a WRITE operation. This is different from a software EOF because the PICK operating system treats each physical EOF as the actual end of the tape. A T-FWD command will advance the tape over software EOFs, but will not move past the first physical EOF.

To utilize tape beyond the first physical EOF, the following verbs should be used:

T-EOF and T-EOFD

T-EOF will locate each physical EOF on a tape or perform a T-FWD on a cartridge disk. T-EOFD will locate the last physical EOF on a tape. When you wish to use a partially filled tape for another operation such as ACCOUNT-SAVE or T-DUMP, the T-EOFD verb should be used to position the tape correctly at the end of the last filled portion of tape. To search a tape which contains multiple tape operations for a particular section, the T-EOF verb should be used.

Note that use of the T-EOF and T-EOFD verbs will shorten the physical tape. Since the T-REW verb will reset the logical tape pointer to zero, it is possible to run over the logical tape EOT. Be sure to keep track of how much you put on a tape so this does not happen. Also note that filling a tape in this fashion does not also allow you to use the multi-reel capability.

NOTE: T-EOF may now be used with 1/2" tape (but T-EOFD may not be used).

Note that T-EOFD is not valid for cartridge disk.

3.8 PROGRAM INTERRUPTION: DEBUG FACILITY

Debug is a system facility for debugging assembler programs. Since an assembler is not an official part of ZEBRA/PICK, the use of DEBUG is not supported.

Debug, however, will be entered when the BREAK key on the terminal (INT key on some terminals) is pressed. This causes an interrupt in the current processing, entry into the DEBUG state and display of the following message:

```
I x.d
!
```

The instruction location of the interruption is x.d. The DEBUG prompt character (!) is displayed to prompt the user for a DEBUG command. For users with system privilege levels 0 or 1, the only DEBUG commands allowed are:

Command	Description
P	Print on/off. Each entry of a P command switches (toggles) from print suppression to print non-suppression. The message OFF is displayed if output is currently suppressed. The message ON is displayed if output is resumed. This feature is useful in limiting the output at the terminal.
G [LF]	Go or line feed. This command causes resumption of process execution from the point of interruption. G cannot be used if a process ABORT condition caused the entry to DEBUG.
END	Terminates current process and causes immediate return to TCL.
OFF	Terminates current process and causes the user to be logged off the system.

Pressing the BREAK key while in the terminal input or output mode will cause a loss of up to 16 characters. If in the input mode, the retype-line character (Control-R) should be used to check the loss of data after returning from DEBUG via the G command.

If the system is executing in virtual mode and an unrecoverable error occurs, the system debugger will be entered and the message "PRIVILEGED OPCODE ABORT @ xxx.xxx" will be displayed. The FID number xxx.xxx gives the location where the abort occurred.

When a hardware abnormal condition exists, the system will also trap to the DEBUG state with a message indicating the nature and location of the abort.

For both error conditions, if the user has system privileges level 0 or 1, he should enter END or OFF to exit from the DEBUG state.

system file management 4

The PICK File Management processors provide for generating, managing, and manipulating files and items within the system. The processors include the CREATE-FILE processor, the CLEAR-FILE processor and the DELETE-FILE processor.

1. The CREATE-FILE processor is used to generate new dictionaries and/or data files. The processor creates the file dictionaries which exist as the "D" entries (pointers) in the user's Master Dictionary. The processor reserves and links primary file space. The user need only specify values for the desired modulo (number of groups in the file) and separation (number of frames per group).

2. The CLEAR-FILE processor clears the data from a file (i.e., it sets the file to the "empty" state by placing an attribute mark in the first data position of each group of the file). "Overflow" frames that may be linked to the primary frame space of the file will be released to the system's overflow space pool. Either the data section or the dictionary section of a file may be cleared.

3. The DELETE-FILE processor allows for the deletion of a file. Either the data section or the dictionary section (or both) of the file may be deleted.

If the file level dictionary is shared by several data files, each data file can be created, cleared or deleted independently of the other data files associated with the dictionary.

As a general introduction to this section, the following examples illustrate use of the File Management processors:

```
*  >CREATE-FILE WORK-IN-PROGRESS  3,1 1743,1  [CR]

*  >CREATE-FILE DICT TRANSLATION  23,1  [CR]

*  >CREATE-FILE DICT DEPT  3,1  [CR]

*  >CREATE-FILE DATA DEPT,PROGRAMMING  123,1  [CR]

*  >CLEAR-FILE DICT TRANSLATION  [CR]

*  >CLEAR-FILE DATA DEPT,ACCOUNTING  [CR]

*  >DELETE-FILE WORK-IN-PROGRESS  [CR]

*  >DELETE-FILE DATA DEPT,MARKETING  [CR]

*  >DELETE-FILE DEPT  [CR]
```

Additional file management procedures (such as the creation of new user accounts, the saving and restoring of files, etc.), have been described in Section 5.

4.1 CREATING NEW FILES: CREATE-FILE AND CREATE-PFILE

The CREATE-FILE processor provides the capability for generating new files and dictionaries in the system. CREATE-FILE and CREATE-PFILE are used to create file dictionaries by reserving disk space and inserting a "D" or "DC" entry, respectively, in the user's Master Dictionary which points to the file-level dictionary, and to create data files by reserving disk space and placing a pointer to the space in the file-level dictionary. CREATE-FILE will automatically locate and reserve a contiguous block of disk frames from the available space pool. The user need only specify values for the modulo and the separation of both the file dictionary and the data area.

For a discussion of the values to use for modulo and separation, refer to Sections 1.3 and 6.3.

There may not be a data file without a file level dictionary pointing to it. Therefore, the file-level dictionary must be created prior to or concurrently with the data file. The latter is the preferred method for creating files and this form of the CREATE-FILE command is shown below. This enables the creation of both the dictionary and a data area with one command. The general form is:

```
CREATE-FILE  file-name m1{,s1} m2{,s2}

CREATE-FILE  dict-name,data-name m1{,s1} m2{,s2}
```

where "file-name" is the name of the file, m1 and s1 are the modulo and separation values of the dictionary (DICT) portion, and m2 and s2 are the modulo and separation of the data portion. If s1 and/or s2 are not given, then separation will be 1. A dict-name and data-name are optional dictionary and data file names to be used if multiple data files will be pointed to by the file dictionary. In either case, a pointer to the data file is placed in the file-level dictionary.

A file dictionary may be created without a data file by the command:

```
CREATE-FILE  DICT file-name m1{,s1}
```

The term 'DICT' specifies creation of the dictionary only with modulo m1 and separation s1, and a pointer to file-name is placed in the user's Master Dictionary. The user should note that a data area need not be reserved for a single-level file, in which case, the data is to be stored in the dictionary, as in the case of the POINTER-FILE.

Once the DICT (Dictionary file) has been created, the primary file space for the data section of the file can be reserved. The general form of the command:

CREATE-FILE DATA dict-name{,data-name} m2{,s2}

The term 'DATA' specifies creation of the data file data-name, if the data file is unique to the file-level dictionary, or creation of the data file data-name under dictionary dict-name, if the multiple data file option is desired. The data file has modulo m2 and separation s2, and the pointer to the reserved space is placed in the file-level dictionary. This form is also used to create new data files pointed to by a shared dictionary using the option {data-name}.

If you wish to create a pointer-file or a BASIC program file, use the CREATE-PFILE verb, which has the same general form as the CREATE-FILE verb.

* >CREATE-FILE INVENTORY 3,1 373 [CR]

Creates a new file called "INVENTORY", with a DICTIONARY section with modulo of 3 and separation of 1, and a DATA section with a modulo of 373 and a separation of 1. An item called "INVENTORY" will be placed in the Master Dictionary, and a D-item called "INVENTORY" will be placed in the INVENTORY dictionary.

* >CREATE-FILE DICT TEST/FILE 7,1 [CR]

Creates a single-level file called "TEST/FILE"; a D-item "TEST/FILE" will be placed in the Master Dictionary, and a Q-item "TEST/FILE" will also be placed in the dictionary created, pointing back to itself.

* >CREATE-FILE DATA DEPT,ACCOUNTING 3 73,2 [CR]

Creates a dictionary called "DEPT" and a DATA section called "ACCOUNTING" for the dictionary DEPT; a D-item called "ACCOUNTING" will be placed in the DEPT dictionary. The data file created will have to be referenced as "DEPT,ACCOUNTING" since it has the shared dictionary structure.

* >CREATE-FILE DATA DEPT,MAINTENANCE 57 [CR]

Creates a new DATA section called "MAINTENANCE" for the dictionary DEPT. This data file will have to be referenced as "DEPT,MAINTENANCE".

4.2 CLEARING FILES

4.2.1 CLEAR-FILE

The CLEAR-FILE processor clears the data from a file (i.e., it sets the file to the "empty" state by placing an attribute mark in the first data position of each group of the file). Overflow frames that may be linked to the primary file space will be released to the system's additional space pool. Either the data section or the dictionary (DICT) section of a file may be cleared using the CLEAR-FILE command. If the dictionary section is cleared and a corresponding data section exists (as implied by the presence of a file definition item in the dictionary), then it will be maintained in the dictionary. The BREAK key is inhibited during the DELETE process, but not during the CLEAR process.

To clear the data section of a file, the following command is used:

```
CLEAR-FILE DATA file-name{,data-name}.
```

In the case that the data file is unique to dictionary file-name the data file "file-name" is cleared; in the case that data file "data-name" is one of multiple data files under dictionary file-name, then "data-name" will be cleared.

To clear the dictionary section of a file, the following command is used:

```
CLEAR-FILE DICT file-name.
```

CLEAR-FILE examples:

```
*   >CLEAR-FILE DATA INVENTORY  [CR]
```

Clears the data section of the INVENTORY file.

```
*   >CLEAR-FILE DICT TEST/FILE  [CR]
```

Clears the dictionary of the TEST/FILE of all non-D-items; all D-items are maintained in the dictionary.

```
*   >CLEAR-FILE DATA DEPT,ACCOUNTING
```

Clears the ACCOUNTING data section of the DEPT file.

4.2.2 DELETE-FILE

The DELETE-FILE processor allows the deletion of the whole file, dictionary and data files, the dictionary only (if the dictionary has no attached data file), the data file in the case of a unique data file, or any data file in the multiple data file case. A file-level dictionary which points to a data file cannot be deleted. All frames owned by the deleted file are returned to the available space pool. To delete a file-level dictionary and ALL its attached data file(s), use the command:

```
DELETE-FILE file-name.
```

To delete a file-level dictionary without an attached data file, use the command:

```
DELETE-FILE DICT file-name.
```

In both cases, the file-definition item (D-pointer) in the user's Master Dictionary is deleted, and the space owned by the deleted file is returned to the available space pool.

To delete the data file, the following command is used:

```
DELETE-FILE DATA file-name{,data-name}.
```

This will delete the pointer to the data file from the file-level dictionary and return the space owned by the data file to the available space pool. The parameter "data-name" is necessary to delete a file from a dictionary with multiple data files.

Files that are defined by file-synonym definitions (Q-pointers) in the user's MD cannot be specified in a DELETE-FILE command.

DELETE-FILE examples:

```
*   >DELETE-FILE INVENTORY  [CR]
```

Deletes the INVENTORY dictionary and all associated data files.

```
*   >DELETE-FILE (DICT TEST/FILE)  [CR]
```

Deletes dictionary TEST/FILE. If there are any data sections associated with this dictionary (if there are any D-items in the dictionary) this command will not be executed. (Parentheses are optional.)

```
*   >DELETE-FILE DATA DEPT,ACCOUNTING  [CR]
```

Deletes the DATA section ACCOUNTING from the shared dictionary structure whose shared dictionary name is DEPT.

4.3 COPYING FILE DATA

4.3.1 COPY PROCESSOR

The COPY processor allows the user to copy items from a file to the terminal, to the line-printer, to the same file, or to another file (either in his account or in the some other user-account).

The COPY processor is invoked via the COPY verb, which is a TYPE-II verb and, therefore, takes on the TYPE-II format. The general form of the COPY command:

```
COPY {DICT} file-name item-list {(options)}
```

The file-name parameter specifies the source file. The item-list specifies the items to be copied and consists of one or more item-ids separated by blanks, or an asterisk (*) specifying all items. The options parameter, if used, must be enclosed in parentheses. Options are described in the next section.

Once a COPY command has been issued, the COPY processor will respond differently depending on whether the copy is to the terminal or line-printer, or to a file. Copy to the terminal is specified by the "T" option and copy to the printer by the "P" option. If neither of these options is specified, the copy is to a file.

If the copy is a file-to-file copy, the processor will respond with:

```
TO:
```

The response to this request is in the form:

```
{(}{DICT} file-name {item-list}
```

where:

1. If the data is to be copied to a different file, the destination file-name is preceded by a left parenthesis; the word DICT may optionally precede the file-name if the data is being copied to a destination dictionary file instead of a data file.

2. If the data is being copied to the SAME file, the left parenthesis is omitted.

3. If the item-ids of the items being copied are to be changed, the list of new item-ids must follow.

4. If a null is entered to the "TO" request, a copy to the terminal is performed (just as if the original COPY statement had the "T" option).

4.3.2 COPY OPTIONS

The COPY "options" parameter, if used, must be enclosed in parentheses. Options are single alphabetic characters. Multiple options may be strung together, or separated by commas for clarity. Table 4-1 describes the options used by the COPY processor. Note that some options operate differently depending on whether the copy is to the terminal or line-printer, or is a file copy.

On a terminal or line-printer copy, the data is displayed in the following format:

```
    item-id
001 attribute one
002 attribute two
003 attribute three
.......
nnn last attribute
```

For example, the item "ITEMX" in the SAMPLE-FILE may be copied to the terminal as follows:

```
>COPY SAMPLE-FILE ITEMX (T)  [CR]

    ITEMX
001 3745
002 SMITH, JOHN
003 1234 MAIN STREET
```

Table 4-1. COPY Processor Options

Option	Note	Description
A		Activates Assembly MLIST format.
D	1	Delete item. On a file copy, the original (source item) is deleted from the file after it is copied.
F	2	Form-feed. On a copy to the terminal or the line-printer, item will cause a new page to begin.
I	1	Item-id list suppress. On a file copy, will inhibit the listing of item-ids.
M		Activates Macro (Assembly) format.
N	1	New item inhibit. On a file copy, will not copy the items to the destination file unless the item already exists there. That is, new items will not be created if this option is set.
N	2	No Page. On a terminal copy, will inhibit the automatic end-of-page wait.
O	1	Overwrite items. On a file copy, will copy the item to the destination file even if it already exists on file. Note that you may not use the "O" option to copy an item which has the same name as the destination file to that file if that file's dictionary contains a "D" pointer.
P		Printer copy. Copies the data to the line-printer.
S	1	Suppress error messages. On a file copy, messages indicating that items were not copied (messages 409, 415, and 418) will not be printed.
S	2	Suppress line-numbers. On a copy to the terminal or line printer, the line-numbers will not be displayed.
T		Terminal copy. Copies the data to the terminal.
X	1	Hexadecimal format. On a terminal or line-printer copy, the data is displayed in hexadecimal form.
n		An integer number indicating the number of items to be copied. Typically used for copying data for test files.

NOTES: 1. Valid only on a FILE copy.
2. Valid only on a NON-FILE (terminal or line-printer) copy.

4.3.3 FILE-TO-FILE COPY

It is frequently required to transfer data from one file to another or to different locations within the same file.

Multiple items may be specified as the source and as the destination in the COPY statement. Multiple item-ids are separated by blanks, unless the item-id itself has embedded blanks, in which case the entire item-id may be enclosed in double-quotes (").

For example, the item-list may be:

```
1024-24 1024-25 "TEST ITEM" ABC
```

which specifies four item-ids, "1024-24", "1024-25", "TEST ITEM" and "ABC".

Item-ids may be repeated within the item-list. There may be different numbers of items within the source and destination lists. If the source item-list is exhausted first, the COPY terminates. If the destination item-list is exhausted first, the remainder of the items are copied with no change in item-id.

If the items are to be copied without any change in the item-ids, the destination file item-list may be null.

If it is desired to copy all existing items, an asterisk (*) may be used as the source file item-list.

If a preselected list of items is to be copied, the source item-list should be null. In this case, the COPY statement must be preceded by a SELECT, SSELECT, QSELECT or GET-LIST statement. See the ACCESS Manual for a discussion of these verbs.

When copying from one dictionary to another, the COPY processor does not copy dictionary items with a D/CODE of "D" (that is, the D-pointers). D-pointers must only be created by the CREATE-FILE processor. To recreate both the dictionary and the data sections of a file in a new file, a command sequence, such as shown in the examples must be used.

Examples of COPY under different conditions are listed on the next page.

4.3.3.1 Copy to Another Account

When copying to a file in another account, it is neceesary to set up a Q-pointer to that account. Q-pointers are discussed in Section 6.6.2.

Copying items to the same file:

```
>COPY DICT SAMPLE COST  (I)  [CR]  <------ Single dictionary item
TO:  PRICE  [CR]                           copied.

1 ITEMS COPIED

>COPY SAMPLE 1242-01  [CR]       <--------- Single data item copied.
TO:  1242-99  [CR]

     1 1242-01        <-------------------- Item-id is listed.
1 ITEMS COPIED

>COPY FLAVORS RED WHITE BLUE  [CR]  <----- Multiple data items copied.
TO:  ALPHA BETA GAMMA  [CR]

     1 RED
     2 WHITE
     3 BLUE

3 ITEMS COPIED
```

Copying items to a different file:

```
>COPY DICT SAMPLE * (I)  [CR]     <-------- All dictionary items copied.
TO:  (DICT FLAVORS   [CR]
[418] FILE DEFINITION ITEM 'SAMPLE' WAS NOT COPIED.

2 ITEMS COPIED
```

Recreation of entire dictionary and data sections:

```
>CREATE-FILE (NEW-SAMPLE 1,1 3,1)  [CR]   <--- New file created.

[417] FILE 'NEW-SAMPLE' CREATED; BASE = 15417, MODULO = 1, SEPAR = 1.
[417] FILE 'NEW-SAMPLE' CREATED; BASE = 15418, MODULO = 3, SEPAR = 1.

>COPY DICT SAMPLE * (I)  [CR]   <------ All dictionary items (except
TO:  (DICT NEW-SAMPLE  [CR]             D-pointer) copied.
[418] FILE DEFINITION ITEM 'SAMPLE' WAS NOT COPIED

3 ITEMS COPIED
>COPY SAMPLE * (I)  [CR]     <---------- All data items copied.
TO:  (NEW-SAMPLE  [CR]

22 ITEMS COPIED
```

4.4 RESTORING FILE DATA

4.4.1 SELECTIVE RESTORES: SEL-RESTORE

The Selective-Restore capability allows individual files or items to be loaded onto a PICK system from a file-save tape or cartridge disk. This verb is used to restore items from either a FILE-SAVE or ACCOUNT-SAVE media. Selective restores are performed as follows:

1. Log on to the account with the file to be restored.
2. Mount the tape or cartridge disk.

NOTE: Selective-restores may be started from any tape or cartridge disk of a multi- tape or cartridge disk file save. To save time in searching a tape or cartridge disk, the STAT-FILE listing may be consulted to determine which reel or cartridge the file's data starts on, and that reel or cartridge may be mounted. A SEL-RESTORE may be started at any place on any reel or cartridge of the file-save media. Any coldstart or ABS sections will be skipped automatically.

3. Attach the tape or cartridge disk unit (T-ATT).
4. To start the restore, enter:

```
>SEL-RESTORE file-name {item-list} {(options)}  [CR]
```

where "file-name" is the file in which items will be placed. This file must be defined on the account from which the restore is run. The optional item-list enumerates those items eligible for restore. A '*' symbol may be used as the item-list to indicate all items on the tape or cartridge disk. The options are:

A The tape or cartridge disk is already positioned in the desired account. In this case, the "ACCOUNT NAME ON TAPE" prompt will not appear.

C This option has effect when the 'N' option is used. It causes every item before the next End Of File to be a candidate for restore. This ensures that data can be restored even if a D-pointer is damaged on the tape or cartridge disk.

I The item-ids of the restored items will not be printed.

N The file is to be identified on tape or cartridge disk by its file number, in which case, the prompt will be FILE#?. The required file # is the one which accompanies the file on the statistics file printout for the appropriate file-save.

O Overwrites duplicate items.

S Skips label search of the tape or cartridge disk. This is used when beginning at the second or later reels or cartridges of a file-save.

If the N option is not used, the operator will be prompted:

```
ACCOUNT NAME ON TAPE?account-name

FILE NAME?file-name
```

where 'account-name' is the name of the account under which the file was saved on tape or cartridge disk, and 'file-name' is the name of the file as it appears on the tape or cartridge disk. Entering [CR] to 'FILE NAME?' causes the account Master Dictionary to be restored. The file-name may be of the form file-name, DICT file-name, or file-name,data-name.

If the N option is used, the prompt will be:

```
FILE #?
```

and the file-number must then be entered.

As the media is searched, the file-names on it are printed along with the file-numbers; names are indented one space for account-names, two spaces for dictionaries, and three spaces for data-file-names.

If a STAT-FILE listing for the tape or cartridge disk is available, ensure that the account-name and file-name are on the media in the form you want. In the case of multiple D-pointers in the SYSTEM dictionary to an account, or multiple D-pointers in the MD to the file, the account-name or file-name on the tape or cartridge disk will be the first one the save processor encounters, and may be different from the one you commonly use. All other names will appear in the STAT-FILE listing with no data (null SIZE field), and cannot be specified in the SEL-RESTORE.

If in doubt about the contents of the tape or cartridge disk, the files can be listed by using a SEL-RESTORE of the form:

```
>SEL-RESTORE TEMP *

ACCOUNT-NAME ON TAPE?  XXXXX

FILE-NAME?  YYYYY
```

where XXXXX and YYYYY are fake names that will cause the SEL-RESTORE to search the media for non-existent data. Files will be printed out as encountered along with the file-numbers. Files with an (S) are synonyms and should be ignored.

In restoring both the dictionary and data section of a file, restore the dictionary first (DICT filename). Remember that the dictionary items follow the data items, so for large files, there may be a considerable pause after the time that the system has found the file (it stops the printout) and the actual restore of the items.

At any point, the tape or cartridge disk may be backed up (T-BCK (n)), or forward-spaced (T-FWD(n)) to position it, and a SEL-RESTORE with the A or N options may be started; this may be faster than restarting the tape or cartridge disk from the beginning when restoring both the dictionary and the data sections of a file, or when restoring multiple files.

Remember also that account dictionaries (MD items) follow all other files for the account on the tape or cartridge disk.

To restore the Q-pointers in the SYSTEM dictionary, use the N option with FILE# = 1. Remember that this will be the last file on the tape or cartridge disk. On a multi- reel or cartridge file-save, mount reel or cartridge #1 first, and start the SEL-RESTORE as usual; when the file-name "SYSTEM" has printed out, use the BREAK key to interrupt the restore, then mount the last reel or cartridge of the set and type "G[CR]" to continue the process. This saves searching the entire first and any intermediate tape reels, tape cartridges, or cartridge disks.

4.4.2 GROUP FORMAT ERROR

As described in Section 1.2.4, a file consists of a number of groups specified by the modulo of the file. Using the modulo and the separation/baseframe parameters, the hashing algorithm determines the group in which an item-id is to be stored. The file retrieval (or storage) routine then searches that group for a specific item. Within each group, the items are physically stored end to end. Each item consists of a count field, a key (item-id) and the data.

The count field is a 16-bit binary number. The high-order bit is 0, represented in the file in ASCII hexadecimal, and requires four bytes of storage. It immediately precedes the item-id in the file. If the item in question is the first item in the group, the count field starts in the first data byte in the frame. If the item is not the first item in the group, then the count field starts at the first byte after the segment mark of the previous item.

The count field is used as a pointer to the end of the item. The end of the item must be an attribute mark followed by a segment mark (^_). If the count field does not point to this pattern, there is a group format error, and the group format error handler will be entered.

4.4.2.1 Transient Format Error

The group format error may be transient or real. Transient group format errors will be encountered if another process is writing an item into the group at the same time that you are trying to read an item in the group. The read without update routines, notably ACCESS, RUNOFF, and PROC, will not check the group locks which are set by the update processor. In the case that the update processor is in the middle of an update, the various frames in the chain which make up the group may not all be updated synchronously. There is, in other words, a random set of conditions under which a phantom group format error will occur, in which case the error handler will be entered. It will normally not find a group format error and will exit back to the process it was executing when it sensed the group format error.

4.4.2.2 Real Format Error

A real group format error is sensed if the count field does not point at an attribute mark, segment mark sequence (^__). This can occur if count or end of item data is wrong.

The count is in error if one or more of the ASCII characters which make up count are incorrect. The correct ASCII count characters are 0-9, A-F.

The effect of the group format error handler is to terminate the group at the end of the last consistent item and cut the forward link out of the last acceptable frame in the group. The rest of the overflow is intentionally lost, because of the effect of having two copies of the same frame referenced in the overflow chain.

The one case in which the group will not be terminated is when a print file has meandered across the base of the file. In this case, it is probably best to recreate the file and selectively restore it. The old file pointer should be thrown away. Do not use the DELETE-FILE verb on the old file, because this will further muddy the condition of the overflow handler.

4.4.2.3 Recovery from Group-Error

Since the overflow handler will chop off groups at the end of good data, the recovery strategy is to identify the file affected and do a SEL-RESTORE on the file. It is best to do this as soon after the group format error is noticed as possible.

In this conflict, note that the organization of file-save tapes written by R80 puts an end-of-file mark at the end of each account, and a tape label at the beginning of each account. This means that the reel upon which the needed file starts may be mounted, rather than starting at the beginning of the tape. If the beginning of the required file occurs in the middle of the desired account, then an A option is to be used.

4.4.2.4 Preventing Group Format Error

When Powering Off or IPL takes place and a user's files are still being written back to disk, the system will not continue writing. Data will be lost and Group Format Errors may possibly occur. To prevent this, wait at least 5 minutes after the last person logs off (longer if there was a lot of activity on the system) before either Powering Down or IPL.

4.5 SYSTEM FILE BACKUP

PICK can save the entire disk data base on tape or cartridge disk and restore this copy, entirely or selectively, to the fixed disk. It is this procedure that provides backup in the event of a catastrophic failure or error.

4.5.1 FILE-SAVE PROC

The FILE-SAVE procedure protects your valuable data base by creating an off-line copy of it on tape or cartridge disk. For adequate backup, you should have separate daily backup sets for one week's time and a monthly backup for each month in the year. Some situations may also need a weekly backup cycle for the past month. That is, use a separate tape or cartridge disk set for each day of the week, one for each week of the month, and one for each month of the year. The longer cycle sets should be stored off premises to provide protection in the event of physical damage such as fire.

The FILE-SAVE procedure requires that you mount the media that is to save your data, and then LOGON to the SYSPROG account. Note that cartridge disks must be formatted before using for the first time. (See Section 2.1.2.5, Cartridge and Hard Disk Format Procedure.) The FILE-SAVE verb calls a PROC which sets up a sentence using the SAVE verb with options D, F, L, S, and T. It also performs a T-REW. The general form of the SAVE verb is:

SAVE {(options)}

Options	Meaning
D	Data area is saved. This option must be present if any files are to be saved.
F	File names are printed. If (F) is not specified, just the SYSTEM file and account-names are listed.
I	Individual account saved. The prompt "ACCOUNT-NAME:" is given.
L	Suppresses prompting for Group Format Errors. Logs them in the GFE and STAT files.
N	No overflow space is required to run the SAVE. This makes it possible to perform a FILE-SAVE on a system that has no overflow space available. Note: If there are more than 1500 files on the system, one (1) frame of overflow space will be needed for every 125 files above 1500.
P	Output (list of file names) goes to the line printer. If (P) is not specified, all output goes to the user's terminal.

Options (Cont)

S STAT-FILE items are stored, one for each file saved. Must be present if a STAT-FILE listing is made after the FILE-SAVE.

T Output to Tape or Cartridge Disk. If the (T) option is not specified, nothing will be written on tape or cartridge disk. However, the STAT-FILE will be generated if the (S) option is used. Note that a cartridge disk must be formatted before using the first time.

Files whose file definition items have a "DX" in attribute 1 will not be saved. Thus, any data file, dictionary or even an entire account, may be prevented from taking up space on the FILE-SAVE tape.

Files whose file definition items have a "DY" in attribute 1 will be saved, but none of the items in the file or sub-files will be saved. The data section of the STAT-FILE, for instance, has a "DY" code because the data is not valid after a file-restore and need not be saved.

To prevent erroneous Group Format Error (GFE) messages from occurring on other lines while the FILE-SAVE is running, the SAVE processor locks groups as it saves them. Up to 4 groups may be locked at one time by a file-save process. These groups would be the ones containing:

1. The SYSTEM dictionary pointer for the account being saved.

2. The file dictionary pointer for the dictionary of the file being saved. This would be a group in the account's MD.

3. The group in the data file of the ACC file.

4. A group in the dictionary of the ACC file.

If a user on another line tries to access data in a locked group, his terminal will hang until the file-save process finishes saving all the items in that group and unlocks it.

If the (T) option is specified, the SAVE processor will prompt the user's terminal:

```
FILE-SAVE TAPE LABEL =
```

The response will be written on the tape or cartridge disk as part of the tape or cartridge disk label.

4.5.1.1 Customizing the FILE-SAVE PROC: CREATE-FILE-SAVE

This program allows you to customize the FILE-SAVE PROC for a more versatile system. To run the program, execute the following TCL command from SYSPROG:

```
>RUN SYSPROG-PL CREATE-FILE-SAVE
```

The following will be displayed:

```
                CUSTOMIZE FILE-SAVE PROC CREATOR
```

Enter Y or N to the following prompts if you want them in the FILE-SAVE PROC:

```
DO YOU WANT A LISTING TO THE PRINTER (Y,N)?

Enter Y or N=(CR):
```

Answer this the same way as for the previous prompts, and do the same for the following:

```
ENTER TIME TO START FILE-SAVE OR (RETURN) FOR IMMEDIATELY:

Enter Y or N=(CR):
```

You will then be asked whether or not you would like to have a spooler assignment for any type output placed in the FILE-SAVE PROC.

```
Enter Spooler Assignment (i.e., SP-ASSIGN HS) or (RETURN):
```

This will give you the option of preassigning the spooler assignment, or if you enter (RETURN), of having no spooler assignment in the PROC and defaulting to what was previously set up. This option, however, requires you to type in the complete SP-ASSIGN verb, like in the preceding example.

The next inquiry allows you to select whether or not you wish the FILE-SAVE to stop on a Group Format Error, or not to stop on any Group Format Errors, but to log them into the STAT-FILE and the GFE files. (It does this by adding the option "L" to the SAVE verb in the PROC.)

Once all the questions have been answered, the program will move the previous FILE-SAVE PROC to an item called OLD-FILE-SAVE in the SYSPROG-PL file, and then create a new FILE-SAVE PROC. Now, when the new FILE-SAVE PROC is run, whatever new prompts have been added will be displayed.

4.5.1.2 Customizing The FILE-SAVE PROC: NEW-FILE-SAVE

A second, newer FILE-SAVE PROC called NEW-FILE-SAVE has also been created. It includes additional options which allow specification of FILE-SAVE media, time to start the FILE-SAVE, and a request for a File-Stat Report. These options will be prompted for by the PROC.

If you wish to use this new PROC in place of the old one, COPY NEW-FILE-SAVE to FILE-SAVE and include the overwrite option in your statement. If you wish to return to using the previous PROC, it can be obtained from SYSPROG-PL where it is named ORIG-FILE-SAVE.

4.5.2 ACCOUNT-SAVE AND ACCOUNT-RESTORE

The PICK system provides the facility to save and restore single accounts. The ACCOUNT-SAVE PROC allows you to generate a backup with only one account on it. The ACCOUNT-RESTORE verb is used to add a single account to an already existing PICK system.

4.5.2.1 ACCOUNT-SAVE PROC

The 'ACCOUNT-SAVE' PROC functions similarly to the 'FILE-SAVE' PROC. The files section contains no System Dictionary pointer or items, and only one account is saved. No synonym D or Q pointers will be saved. If STAT-FILE items are generated, they will pertain only to the saved account.

Account saves are performed as follows:

1. Log onto SYSPROG

2. Mount a tape or cartridge disk which is not write-protected.

3. Type: ACCOUNT-SAVE [CR]

   System responds: TAPE LABEL IF DESIRED

   Type: tape or cartridge disk label [CR] or [CR] (if no label is desired).

   System responds: ACCOUNT NAME?

   Type: account-name [CR] (of account to be saved).

The account-name used must be in the System Dictionary. Note that a cartridge disk must be formatted before it is used for the first time.

4.5.2.2 ACCOUNT-RESTORE

An 'ACCOUNT-RESTORE' can be performed from a file-save of a whole system or from an 'ACCOUNT-SAVE' backup. In either case, the account will be restored and a pointer to the account will be created in the System Dictionary.

Note that the account must not already exist on the system. Account-restores may be started from any reel or cartridge of a multi- tape or cartridge disk file-save. To save time in searching for data, the STAT-FILE listing may be consulted to determine which reel or cartridge the account's data starts on, and that unit may be mounted.

Account restores are performed as follows:

1. Log on to SYSPROG.

2. Mount the tape or cartridge disk with the account on it.

3. Type: ACCOUNT-RESTORE new-account-name [CR]

   System responds: ACCOUNT NAME ON TAPE?

   or

   ACCOUNT NAME ON CARTRIDGE?

   Type: old-account-name [CR]

The operator must respond with the name of the account, and must use the same name under which the account was saved. The media will be searched for the account, and the restore will proceed automatically.

A 'Synonym' segment may be encountered with a base which has not been found on the tape or cartridge disk. This happens when a D-pointer on the saved account points to a file on another account, or if a 'D' segment on the tape or cartridge disk is unrecognizable because of a parity error. In this case, the message 'SYNONYM NOT FOUND' will appear. The synonym D-pointer will not be created and the restore will continue.

4.5.2.3 Multi-Unit Tape or Cartridge Disk Operation

At the end of FILE-SAVE, :FILELOAD, ACCOUNT-SAVE and ACCOUNT-RESTORE tape or cartridge disk operation, you will receive a message:

```
MOUNT REEL #n      or      MOUNT CARTRIDGE #n
LABEL date DATA file-name file-label reel-no. or cartridge-no.
```

The message gives the number of the unit to mount next and displays the file label. After mounting the appropriate unit, you should type "C" to continue.

system memory management 5

5.1 MEMORY STRUCTURE

PICK is a multi-programmable virtual memory operating system with all of virtual memory (disk) being directly addressable as if it were real memory. The virtual memory of PICK resides on a disk drive, divided into 512 byte frames. The frames are addressed by frame-ids (FID), numbered 1, 2, 3,... up to a maximum, depending upon the disk being used.

1. ABS Frames - The lower-numbered frames on the disk are "ABS" frames, which contain system software and workspaces. All frames above the ABS area are available for use in files. Those frames not used for files make up the Available Space, sometimes called "Overflow".

2. Work Area - The PICK operating system allows multi-programming, which means more than one different program may be executed on a time-sharing basis by the CPU. Each running program, or process, has a work area of 32 contiguous frames, the first of which is called the "Process Control Block" (PCB) as shown in Figure 5-1.

   The PCB of channel zero is normally frame 1024 (400 hexadecimal). PCB's for succeeding processes are separated by 32, and therefore, the PCB for line one is 1056 (420 hexadecimal), line two is 1088 hexadecimal, etc.

   Additionally, larger "Secondary" workspace blocks are reserved in the ABS area following the last primary workspace, which is that of the SPOOLER. WSSTART is the starting FID of the secondary workspaces, which continue to the end of the work area. Each process has three secondary workspaces, usually consisting of 127 frames each.

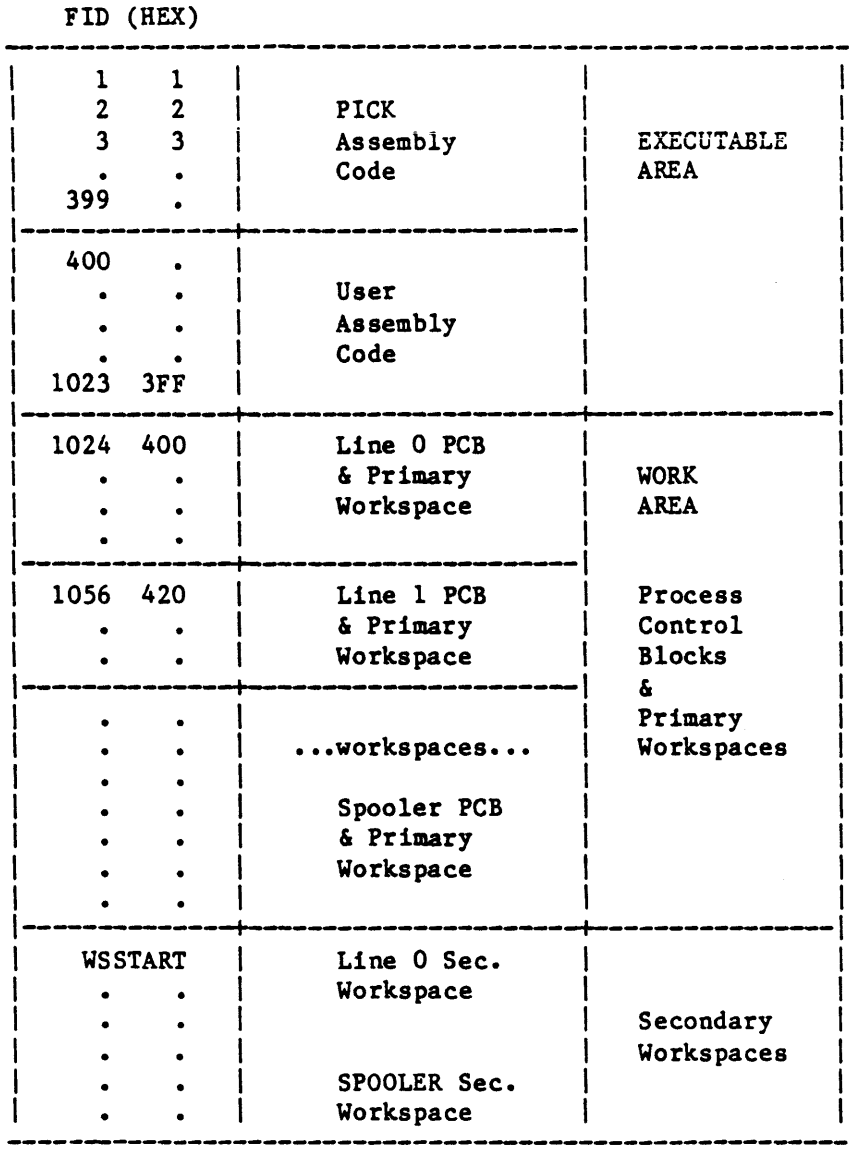
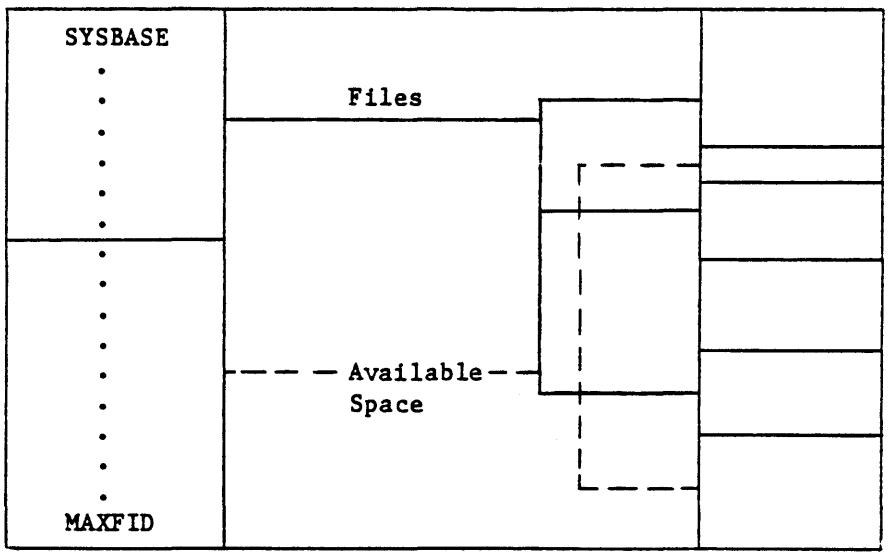


Figure 5-1. ABS Area, Including Executable Area and Work Area

3. Files and Overflow - Following the work area are PICK files, beginning with the SYSTEM file. The base of the SYSTEM file, SYSBASE, is the beginning of the file space. On a newly generated or restored system, all other files on the system immediately follow the SYSTEM file. At the end of the files is the start of Available Space (overflow), which then continues until the end of the disk (MAXFID) as shown in the illustration below.

   On a running system, the overflow area will become "fragmented" as frames are taken from and returned to the overflow pool. The following illustration shows the effect on files and available space following a file-restore (left) and after undergoing normal fragmentation (right).



5.1.1 ADDED WORK SPACE

The "additional workspace" is a set of contiguous, linked frames that is initialized by the system at coldstart or system-generation time.

There are several processors in the system that require large amounts of workspace or buffer area. The workspace is preassigned and need not be linked up at LOGON time. The workspace is linked after a file-restore, or it may be linked from TCL by use of the LINK-WS verb. The SPOOLER process links the workspace for all the other lines and no other user can log on the system while this linkage is taking place.

The following message will appear until the spooler has finished the linkage:

```
LINKING WORK-SPACE;  WAIT
```

The starting FID of the secondary workspace for line 0 may be computed:

```
WSSTART = (number of lines) * (32) + 1024
```

(Each line has 3 secondary workspaces of 127 contiguous frames.)

The workspace may be linked on a live system using the LINK-WS verb on the SYSPROG account. This may be done if it is suspected that the links of the additional workspace have been destroyed. One manifestation of this situation is that BASIC programs terminate with the "NOT ENOUGH WORK SPACE" message. Work-space links should be particularly suspect if a program or process aborts on one channel but works correctly on others.

The general form of the verb to relink the workspace is:

```
LINK-WS {(n{-m})}
```

If the "(n)" or "(n-m)" is omitted, the workspace of ALL lines will be relinked, except those of lines logged on and that of the spooler process. The parenthetical specification may be used to limit the relinking process to lines "n", or "n" through "m" only.

As the linkage proceeds, the line-number of the process whose workspace is currently being linked is displayed on the terminal; if the line is logged on, the message "ON!" will be displayed, and also THE WORK-SPACE IS NOT RELINKED. The spooler's workspace can only be relinked via a coldstart.

5.1.2 THE FILE AREA

Immediately following Work Area, the remainder of the virtual memory, the File Area is available for the storage of data in files. The portions of the File Area that are not allocated to the files are maintained as a pool of Available Space. The beginning of the File Area is a system generation parameter. It may be computed as follows:

```
Start of File Area (SYSBASE) = (FID of first PCB) +
((number of processes)*32 +
((number of processes)*(pre-assigned work-space)*3
```

Pre-assigned work-space is set to 127 frames per process per work-space. Each process (including the spooler) has 3 secondary workspaces of 127 frames each.

As an example, a system with 18 communication lines (therefore 19 processes including the spooler) will have the start of the file area at frame:

```
1024 + (19 * 32) + (19 * 381) = 8871
```

The end of the File Area is the highest available disk frame, MAXFID.

Disk Configuration	Highest Disk Frame (MAXFID)
One 16MB disk	25,427
One 32MB disk	56,423
One 64MB disk	118,823
One 142MB disk	272,075

File Area frames which are not allocated to the files are maintained as a pool of Available Space, often called "Overflow". Available Space is used by the PICK system file management routines as additional data space, and also by other processors as scratch work space. The PICK system maintains a table of pointers that define the Available Space, which may be either in a "linked" form, or in a "contiguous" form. Contiguous Available Space consists of blocks of contiguous frames (defined by starting and ending numbers) that can be taken out of the pool either singly or as a block. Linked Available Space can only be taken a frame-at-a-time. Conversely, space may be released by processors to the linked available pool a frame-at-a-time, or to the contiguous pool as a block.

At the conclusion of a File-Restore process, an initial condition exists; there will be one principle block of contiguous available space, extending from the end of the current data space through the last available data frame. This is illustrated as follows:

```
>POVF  [CR]

 23987 - 56423: 32437

.TOTAL NUMBER OF CONTIGUOUS FRAMES AVAILABLE= 32437
```

The results of the POVF (print overflow) verb indicate that there is no linked overflow space (blank line at the top of the output), and only one contiguous block of space. See Section 5.1.5.1 for further description of POVF.

As the system obtains and releases Available Space (and as files are created and deleted), the Available Space becomes fragmented; at any particular time there may be several blocks of contiguous Available Space, and a chain of linked Available Space. Available frames will be placed in the linked Available Chain only when there are 31 sets of contiguous Available space (representing the maximum that the system space management routines can maintain). For example:

```
>POVF  [CR]
   23459 (400)
    8112-  8117  :    6     9000-  9000  :    1
   23789- 23801  :   13    25000- 25678  :  679
   25681- 25692  :   12    27123- 27323  :  201
   34502- 35123  :  522    35800- 35801  :    2
   37091- 37091  :    1    37093- 37093  :    1
   37099- 37100  :    2    38100- 38100  :    1
   43100- 44234  : 1135    45680- 45681  :    2
   46343- 46443  :  101    46445- 46445  :    1
   46448- 46448  :    1    46451- 46451  :    1
   46454- 46454  :    1    46458- 46474  :   17
   47011- 47444  :  434    47460- 47492  :   33
   47661- 47750  :   90    48012- 48017  :    6
   48018- 48018  :    1    48020- 48101  :   82
   48233- 48268  :   36    48299- 48299  :    1
   51111- 53234  : 2124    53400- 53601  :  202
   60000- 97799  :37800
  TOTAL NUMBER OF CONTIGUOUS AVAILABLE FRAMES= 43509
```

In this example, the linked Available chain starts at FID 23459 and contains 400 frames. There are also several sets of contiguous Available space as shown by the pairs of FIDs displayed.

Logically, there is no difference between Available space in linked chain and that in the contiguous sets; however, certain processors obtain frames from the contiguous space only, for example, the CREATE-FILE processor. Therefore, if the system Available space is severely fragmented, while there may actually be enough disk space to create a large file, there may not be enough available as a contiguous block. For further explanation, see the section on the POVF verb.

5.1.3 FRAME FORMATS

A Frame is a block of 512 bytes that is referenced by a unique number called the Frame Identifier, or FID. There are two types of PICK frames; ABS frames and FILE frames. The FILE frames contain 500 bytes of data, the remaining 12 bytes being the "link fields". ABS frames may be object-code (assembly or BASIC-compiled object code), buffers or other workspaces required by the system.

Linked frames are used to define data areas which may be greater than 1 frame in length. Unlinked frames have no specified format; all 512 bytes of the frame may be used by the system.

The groups in data files may expand as more data is placed in the group, so when the end of a frame is reached, another frame is obtained from the system Available Space pool and linked to the end of the group. The format of the linked frame is as follows:

byte: 0	1	2	3	4	5	6	7	8	9	A	B	C
*	nncf	..forward		link...		..backward		link...		npcf	*	start of data

where:

*	Unused byte.
nncf	Number of next contiguous frames (count of frames that are linked forward of this one, whose FID's are sequential to this FID).
npcf	Number of previous contiguous frames (count of frames that are linked backward to this one, whose FID's are sequential to this FID).
forward link	FID of the frame that is next in logical sequence to this one.
backward link	FID of frame that is logically previous to this one.

The first frame of a linked set of frames will have zero "npcf" and "backward link" fields. The last frame of such a set will have zero "nncf" and "forward link" fields. The "nncf" and "npcf" fields are also normally zero, except in the "linked workspace" allocated to each process, and in files that have a separation greater than one.

5.1.3.1 Frame Format Display: DUMP

The DUMP verb may be used to display data in a frame or to display absolute core locations. The data display may be specified in either character or hexadecimal format. The general form of the DUMP verb is as follows:

```
DUMP n1{-n2},{options}
```

"n1" and "n2" are numbers that may be specified in decimal or in hexadecimal by preceding the hex number with a period (.). The n1 and n2 parameters contain the beginning (and ending) FID(s) of the frame(s) being dumped. After the first entry, if n1 is not entered, the next frame will be displayed. If n1 is the ABS area, the ABS frame will be displayed in 2048 byte format without links. Otherwise, the data frame will be displayed in 500 byte format with links. The left-hand column gives the absolute decimal displacement. Options are specified like normal statement options, as single characters, optionally separated by commas. Valid options are:

Option	Description
C	Core dump; specifies that absolute memory locations are to be dumped. In this case, a 512-byte block starting at (n1 modulo 512) is dumped. n2 is ignored. This option automatically sets the X (hex) option also.
G	Group; specifies that the data starting at frame n1 is to be dumped and that the dump continue following either the forward or backward links (depending on whether the U option is not or is specified). The dump will terminate when the last frame in the logical chain has been found.
L	Links; specifies that the dump be confined to the "links" of the frame(s) concerned; no data is displayed.
N	Nostop; if the data is printed on the terminal, specifies that the end-of-page stop be inhibited.
P	Printer; the display is routed to the line-printer.
U	The data or links are traced logically upwards; that is, the backward links are used to continue the display.
X	Outputs in hexadecimal and character format.

An example of the use of DUMP:

```
>DUMP 6950,L  [CR]
 FID:    6950 :    0   6967     0    0    (  1B26 :   0    1B37     0    0 )
+FID:    6967 :    0      0  6950    0    (  1B37 :   0       0  1B26    0 )
```

In the above example, the display indicates that 6950 is the FID whose links are being dumped; the "nncf"* field is 0; the "forward link" field is 6967; the "backward link" field is 0; the "npcf"** field is 0. Data in parentheses displays the same information in hexadecimal.

The next line displays the link fields of FID 6967; the "+" indicates that this FID is logically "forward" of the preceding one.

5.1.3.2 Frame Lock in Memory: LOCK-FRAME, UNLOCK-FRAME

The LOCK-FRAME verb may be used to lock a frame in memory. The general form of this verb is:

```
LOCK-FRAME number
```

where "number" is a decimal frame number. The LOCK-FRAME verb responds with the absolute hexadecimal work address of the memory buffer in which the frame is locked.

The frame remains locked until it is released by the UNLOCK-FRAME verb (of the same general form), or RESET from the system front panel, which releases all memory frames locked by the LOCK-FRAME verb.

*nncf = Number of next contiguous frames. (Count of frames that are linked forward to this frame, whose FID's are sequential to this FID.)

**npcf = Number of previous contiguous frames. (Count of frames that are linked backward to this one, whose FID's are sequential to this FID.)

5.1.4 DISPLAY OF SYSTEM STATUS: WHAT, WHERE

The WHAT verb is used to display the system configuration, the current status of its locks and tables, and the location of the processes that are logged on. The WHERE verb is a subset of the WHAT verb; both verbs require SYS2 privileges to use. The WHAT and WHERE verbs have the following general form:

```
WHAT {options}   WHERE {n} {options}
```

WHAT and WHERE options:

WHAT Options	Explanation
L	Suppresses lock status display.
P	Statistics printed on printer.
S	Suppresses SP-STATUS statistics.
W	Suppresses current processes information (WHERE component).

WHERE Options	Explanation
"account-name	Displays informaion only for lines logged onto specified account (quotes must be used).
n{-m}	Diplays information only for lines logged on in range specified.
Z	Displays information for all lines whether logged on or not.

WHAT and WHERE examples:

WHERE 3-5	Displays the return stack for users three through five.
WHERE 'DP'	Displays the return stack for all lines logged onto DP.
WHAT L	Will suppress the locks section of the WHAT verb.
WHAT W	Will suppress the WHERE section of the WHAT verb.
WHAT S	Will suppress the SP-STATUS section of the WHAT verb.
WHAT LWS	Will yield only the system configuration section of the WHAT verb.
WHAT 'account-name'	Will display only those lines which have the account account-name logged onto them.

5.1.4.1 The WHAT, WHERE Message

The WHAT verb displays the state of the system status as shown below. (Numbers in brackets identify Notes listed below; they are not a part of display.)

```
CORE   LINES  PCB0  WSSTART  WSSIZE  SYSBASE/MOD/SEP  MAXFID  AVAIL.  OVERFLOW
 256K     11  1024     1376     127     5567  11   1   56423          49569
         [1]   [2]      [3]     [4]    ...[5].......    [6]            [7]

 ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ##
 ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ##    [8]
 ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ##

 ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ##    [9]
 ## ## ## ## ## ## ## ## ## ##

 00 0400 FD00      6.178   6.096 236.660
 02 0440 F30A      6.44A 233.578
 03 0460 FF00    478.31E 454.1C6 454.264 465.328
 04 0448 FF00    231.064
 05 04A0 EF00      6.44A 344.0D8 353.658 351.332
*06 04C0 FB00    121.000 121.1B4 166.626
 09 0520 BF00    170.060 170.0B4
 10 0540 BF00    170.060 170.14E
[10][11] [12][13]  [14]  [15] . . . . . . . . . .

 THE SPOOLER IS INACTIVE
 PRINTER # 0 IS SERIAL, INACTIVE, AND ON LINE.
 THE PRINTER IS RUNNING ON LINE 9.
 ASSIGNED OUTPUT QUEUES:  0
 THE NUMBER OF INTER-JOB PAGES TO EJECT IS 0.
```

NOTES for "WHAT":

[1] Number of communication lines (terminals) plus one (spooler) = number of processes on system.

[2] PCB-FID for channel zero; each following channels PCB-FID is displaced by 32 frames from PCB0.

[3] Extended workspace starting FID; WSSTART = PCB0 + 32*LINES (Including SPOOLER).

[4] Extended workspace size; number of frames per workspace is either 100 or 127 frames. There are three workspaces per line.

[5] System base-FID/modulo/separation; SYSBASE=WSSTART + WSSIZE*3*LINES.

[6] Maximum disk FID; (32MB - 56423).

[7] Available overflow space; linked frames + contiguous frames.

[8] BASIC locks (48); system reserved (15); spooler-linking-workspace (last bit). Bits start at 127.20.

[9] System lock bytes; ##=available; else has channel number as above.

LOCK #	LOC	USAGE
0	127.0	Lock-table lock.
1	127.1	Overflow table lock.
2	127.2	Group-lock table lock.
3	127.3	MESSAGE processor lock.
4-26		Reserved.

The sequence of channels is in the current priority chain sequence, except for those channels that have a PIB-status of "7D" (waiting for terminal input), which are not in the chain and, therefore, appear in numerical sequence. If the "S" option is used in the WHAT verb, all channels are in numerical sequence.

[10] Channel number; preceded by a "*" if your channel.
[11] PCB-FID (hex) of channel.
[12] PIB-status of channel;
7F/FF = Active, or ready to go.
7B/FB = Terminal output.
7D = Terminal input.
5F = Waiting for disk.
3F = Release Quantum/Sleeping.
Typically, spooler is "BF".
[13] PIB-status-2; 00=Normal, 40=in DEBUGGER.
[14] "T" = Tape or Cartridge Disk attached; "P" = Printer attached.
[15] Location counter (first address) and subroutine return = stack addresses.

Entry format = fff.lll where fff = decimal FID; lll = hex location.

Typical Locations:

6/9 = Terminal I/O	13-20 = EDITOR;	89 = DEBUGGER
225-248,275 = BASIC	53-64 = ACCESS Compiler	71-77 = LIST
290-298 RUNOFF	189-199 = BASIC Compiler	200-220 = SAVE-RESTORE
5 = TCL-I	161-183 = SPOOLER.	

5.1.5 LOADING AND USING SYSTEM SPACE

5.1.5.1 POVF

The POVF verb displays the contents of the system overflow table. The general form of the POVF verb is:

```
POVF {(P)}
```

The P option forces all printed output to the line printer. The first line of output is the FID of the first frame in linked overflow, followed by the number of frames in the linked chain.

The next lines (up to 16) describe blocks of contiguous overflow and have the following format:

```
m - n : p   m - n = p
```

where:

m is the first frame of a contiguous block.

n is the last frame of the block.

p is the number of frames in the block.

The total number of frames contained in all the contiguous overflow is then printed (using error message number 293):

```
TOTAL NUMBER OF CONTIGUOUS FRAMES  :number
```

For an example, see Section 5.1.2 of this manual.

5.2 SYSTEM DICTIONARY AND SYSTEM FILE

The remaining part of Section 5 describes the content and the use of the system dictionary and the system file. The system dictionary contains an identification item for each PICK user. This set of items defines the users who have access to the system. Such items are file definition or file synonym definition items.

The SYSTEM file is the highest level (level 0) file within PICK. It contains pointers to the appropriate data base accounts as well as pointers to system-level files.

5.2.1 USER IDENTIFICATION ITEMS

User identification items are initially created via the CREATE-ACCOUNT PROC. These items may subsequently be updated via the EDITOR. Entries in the System Dictionary should not be updated (from the SYSPROG account) when any other user is logged on to the system. This is because the system software maintains pointers to data in the System dictionary when users log on, and updating the System Dictionary will invalidate the pointers. An exception to this rule is creating a new account for a synonym to an existing account, which can be done at any time since new items are added to the end of the existing System Dictionary data, and thus do not disturb any pointers to it.

Attributes 5 through 8 of a user identification item contain data associated with the user's security (lock) codes, password, and system privileges.

Attribute	Use
5	Contains the set of retrieval lock-codes associated with the user. Multiple values (separated by value marks) are allowable. There is no restriction as to the format of individual lock-codes. This attribute may be null, indicating no lock-codes. (Lock-code usage is described in Section 7.2, SECURITY.)
6	Contains the set of update lock-codes associated with the user. (Same as described for retrieval lock-codes above.)
7	Contains the user's password, which is a single value. This attribute may be null. There is no restriction as to the format of the password.
8	Contains a code which indicates the level of "system privileges" (see below) assigned to the user.
9	May contain the code "U" to indicate that logon/logoff times should be logged by the system. May contain the code "R" to specify the RESTART option. May contain the code "L" which is equal to a null code.

Attributes one through four and attributes ten through thirteen are like those defined for regular file definition of file synonym definition items. A user identification item example:

```
Item SMITH in System Dictionary

001  D  <------------------------ D/CODE
002  2537  <--------------------- Base FID
003  13  <------------------------ Modulo
004  1  <------------------------- Separation
005  ABC  <---------------------- Retrieval lock code (L/RET)
006  1234  <--------------------- Update Lock Code (L/UPD)
007  A1234567  <------------------ Password
008  SYS2  <---------------------- System Privilege Level
009  U  <------------------------- Update Account File for this user
```

Three levels of system privileges are available; they are referred to as zero (lowest), one, and two (highest). Lower levels of system privileges restrict usage of certain facilities of the system, described as follows:

Facility	Lowest Privilege Level Allowed
Updating of MD	One
Use of tape or cartridge disk	One
Use of DEBUG (other than P, OFF, END and G commands)	Two
Use of DUMP processor	Two
Use of Assembler and Loader	Two
Use of FILE-SAVE and FILE-RESTORE processors.	Two

System privileges are assigned by the code in attribute eight of the user identification item. Leave this part null for level 0, specify SYS1 for level 1, and SYS2 for level 2.

Attribute 9 may contain the codes 'U' or 'R', or both. 'U' specifies that the accounting listing file is to be updated whenever the user logs on and off this account (see ACCOUNTING FILE). 'R' specifies that the Restart option is to be set. This causes the LOGON PROC to be reexecuted whenever an "END" is typed at the DEBUG level. This attribute may also contain the code "L" which is equal to a null code.

5.2.2 SYSTEM FILE AND SYSTEM-LEVEL FILES

Entries in the SYSTEM file define user MD's or special files necessary for PICK. The MD pointers are either D (file definition) or Q (file synonym) items. The item-ids of such items are the user-names that the user enters when the system requests him to LOGON. Such items are created by the CREATE-ACCOUNT processor, (for D items) or by use of the EDITOR or COPY processor for Q items. The format of user-identification items was discussed in the previous section. The SYSTEM file also contains file-pointers to system-level files that are necessary for overall operation of the system. These files were introduced in Section 1. The following matrix identifies in further detail the structure of these files.

Level 0	SYSTEM dictionary				
Level 1	ACC	BLOCK-CONVERT	ERRMSG	PROCLIB	SYSTEM-ERRORS
Level 2	ACC dictionary				SYSTEM-ERRORS dictionary
Level 3	ACC data				SYSTEM-ERRORS data

5.2.2.1 Acounting History File

The ACC file (Accounting history) has two types of items; those that indicate the actively logged-on users, and the accounting-history data items that keep track of the usage statistics of each user. The format of the items in this file is discussed in later sections.

The ACC files have a tri-level structure with an ACC account, an ACC dictionary, and an ACC data section.

5.2.2.2 Block Convert File

The BLOCK-CONVERT File is a three-level file that contains two unrelated types of items:

1. Items that define the format used in the characters displayed when the BLOCK-PRINT verb is used.

2. Items that are used to print a descriptive message when the "A" (assembly-code) option is used when compiling a BASIC program.

5.2.2.3 PROCLIB File

The PROCLIB file is a three-level file that contains commonly used PROCs, such as CT (Copy to Terminal), LISTU (List Active Users), etc.

5.2.2.4 SYSTEM-ERRORS File

The SYSTEM-ERRORS file is a three-level file reserved for logging system errors. Currently, its only use is to store disk errors.

System-level files are further described in the following section.

5.3 SYSTEM-LEVEL FILES

5.3.1 ACCOUNTING HISTORY FILE

The Accounting History file is one of the mandatory files in the PICK system. This file contains accounting history for the system, as well as entries that describe the currently active (logged-on) users.

The System dictionary (SYSTEM) contains the file definition item (item-id 'ACC') for the Accounting History file illustrated as follows:

Channel #	Item-id	Channel #	Item-id
0	0200	16	0400
1	0220	17	0420
2	0240	18	0440
3	0260	19	0460
4	0280	20	0480
5	02A0	21	04A0
6	02C0	22	04C0
7	02E0	23	04E0
8	0300	24	0500
9	0320	25	0520
10	0340	26	0540
11	0360	27	0560
12	0380	28	0580
13	03A0	29	05A0
14	03C0	30	05C0
15	03E0		

The 'ACC' dictionary is set up for examining and listing the data in the Accounting History file. There are two item types in the Accounting History file: those that represent active (logged-on) users, and those that keep track of accounting history.

5.3.1.1 Active User Items

The item-id of an active user item in the Accounting History file is the four-character hexadecimal FID of the PCB of the user's process. If the PCB's start at FID-512, (they proceed in steps of 32 frames from there on), we see that a user logged on to process zero will have an entry with an item-id '0200' (512), while a user logged on to process one will have an entry with an item-id '0220' (544), and so on. Attribute one of an active user item contains the name of the user (i.e., the item-id of the user identification item), attribute two the date logged on, and attribute three the time logged on). Active user items are created when a user logs on and deleted when he logs off.

5.3.1.2 Accounting History Items

The item-id of an accounting history item is the name of the user (i.e., the item-id of the user identification item), with the channel number concatenated by a "#". For example, if user 'SMITH' logs on to channel 12, when he logs off, the item whose item-id is 'SMITH#12' in the ACC file will be updated. This allows one to keep track of system usage by user-id as well as channel number.

Attributes one, two, and three are not used. The remainder of the attributes are described below:

Attribute	Use
4	Date(s) Logged on. Each unique date is stored. Value marks are tagged on to the value in this attribute if multiple logoffs occur on the same date (for LIST alignment purposes). Date is stored in PICK date format.
5	Time(s) Logged on. An entry is made for each logoff, representing the time at which the user logged on. Time is represented in seconds past midnight (24-hour clock).
6	Connect time(s). This entry represents the item in seconds between the logon and the logoff.
7	Charge-units. A number representing the CPU usage is added on each logoff.
8	Line-Printer pages. A number representing the number of pages routed to the line-printer for each session.

Attributes 4, 5, 6, 7, and 8 are stored as a "controlling-dependent" data set, with attribute 4 being the controlling value and the others the dependent ones. The "controlling-dependent" data set format is described in the ACCESS section of the PICK Reference Manual.

The accounting history file 'ACC' is not automatically updated every time a user logs off the system. The SYSTEM dictionary item for the user must have a 'U' in attribute 9 if the user is to have his Account file history items updated. The entries in the Account file contain the history of each session (logon to logoff). If the SYSTEM dictionary data has been changed since logon or the history file item to be updated is too large for the work-space, the message number 338 will be printed.

5.3.1.3 Accounting History File Summary

This section summarizes the formats of the active user items and the accounting history items in the Accounting History File. Also presented are sample entries for the Accounting History File.

A summary of the attributes for the active user items and the accounting history items is as follows:

Attribute Number	'ACC' Dictionary Name	Active User Item	Accounting History Item
	(item-id)	Four-character hexadecimal PCB-FID	user name#lineno
1	NAME	User name	Not used
2	DATE	Date logged on	Not used
3	TIME	Time logged on	Not used
4	DATES		Dates logged on
5	TIMES		Times logged on
6	CONN		Connect time
7	UNITS		Charge-units
8	PAGES		Number of printer pages generated.

An example of a LISTU sorted listing of the active users (users with a value for attribute 1) via an ACCESS SORT statement.

```
>LISTU  [CR]

CH# PCBF NAME.............  TIME...  DATE....  LOCATION.................

 00 0200 CM                 11:02AM  03/22/82
 01 0220 SYSPROG            12:10PM  03/22/82
 02 0240 EL-RDD             09:11AM  03/22/82
 03 0260 LC                 06:59AM  03/22/82
 05 02A0 HVE                09:55AM  03/22/82
*06 02C0 CM                 11:25AM  03/22/82
 07 02E0 BUGEYE             01:29PM  03/22/82
 10 0340 JT                 11:34AM  03/22/82
```

And finally, a sample listing of the accounting history item for user SMITH via an ACCESS LIST statement:

```
>LIST ACC = "SMITH]"          (selects items with item-ids starting with
                               the string "SMITH")
PAGE 1                                 12:17:22    22 MAR 1982

ACC..... DATE.  TIME...  CONN...  UNITS..  PAGES
                    *        *        *
SMITH#0     01/13    16:56    00:04       9
            01/14    10:13    00:00       5
                     10:15    00:01     343
            02/06    17:02    00:18      41
            02/09    10:21    00:17     690
            02/23    07:58    00:01      27
            03/09    11:35    01:57     378
                     16:05    00:22      94
SMITH#5     01/13    12:48    02:25     160       5
                     15:20    00:05      14
                     15:25    00:00       2
                     15:28    00:17     110
                     16:20    02:55    2575      16
                     19:15    00:00      13
            01/16    09:41    06:13    1853       6
                     15:55    00:12      15

2 ITEMS LISTED.
```

5.3.1.4 Accounting History File Clearance

To avoid overflowing the accounting history items in the Accounting History File for a specific user, the items should be periodically cleared. If the accounting history item for a user-account exceeds the available workspace, the user will be logged off, but the Accounting History File will not be updated. To recover from this situation, clear the accounting history items from the ACC file with the following steps:

1. Logon to the SYSPROG account.

2. Type the following (if you need a listing only):

   ```
   >SORT ACC WITH NAME LPTR  [CR]
   ```

3. Type the following:

   ```
   >SELECT ACC WITH 'user-name'  [CR]
   >DELETE ACC  [CR]
   ```

The point of overflow is determined by the activity of the user-account. Approximately 1000 logon/logoffs are allowed. This point can be calculated by taking the following steps:

1. Use the WHAT verb to determine the number of additional workspace frames allocated for the system (parameter WSSIZE in the WHAT display). Multiply this figure by 500 and add 3000.

2. To determine the current size, type:

   ```
   >STAT ACC 'user-name'  [CR]
   ```

   This will produce the following output:

   ```
   STATISTICS OF ACC:
   TOTAL = xxx  AVERAGE = yyy  COUNT = zzz
   ```

3. If the value displayed for TOTAL in step 2 (i.e., xxx) approaches the value calculated in step 1, then the user-account is approaching the overflow point.

5.3.2 BLOCK-CONVERT FILE

The BLOCK-CONVERT file contains data used by the BLOCK-PRINT verb to convert characters to a n-by-8 block format.

There are two types of entries in the BLOCK-CONVERT file; type I is the entry that forms the characters for the BLOCK-PRINT verb. Its format is:

```
Item-id
```

where "item-id" is the character to be formed; that is, the item whose item-id is "C" will form the character C; the item whose item-id is "{" will form the character {, etc. Each item must consist of exactly ten attributes.

The first attribute contains a code of the form:

```
n{c}
```

where "n" is the width of the character matrix (i.e., "n" of the n-by-8 block; the depth of all characters formed is fixed at 8); and the optional "c" is a character that will replace the item-id in the generation of the BLOCK-PRINT form.

There must be 8 succeeding attributes, each one specifying the format of a row in the generated form. Each attribute must begin with a "C" or a "B", specifying a character insertion or a blank insertion, respectively, followed by the number of such insertions needed; optionally, additional numbers may be specified, separated by commas. Each succeeding number switches the insertion from character to blank and vice-versa. The sum of all numbers must equal the character width specified in attribute 1. An example of BLOCK-CONVERT:

```
>COPY BLOCK-CONVERT S 8A (T)  [CR]

    S    Item-id; defines format for character "S"
001 7    Defines character width as 7
002 B1,5,1            Specifies string " SSSSS " (1 blank, 5 S, (1 blank)
003 C2,3,2            Specifies string "SS   SS" (2S, 3 blank, 2S)
004 C2,5              Specifies string "SS     "
005 B1,5,1            Specifies string " SSSSS "
006 B5,2                               "     SS"
007 C2,3,2                             "SS   SS"
008 B1,5,1                             " SSSSS "
009 B7
    8A   Item-id (BASIC object-code byte)
001 STOP Identifies object-code (STOP opcode).
```

The second type of data in the BLOCK-CONVERT file has a 2 hexadecimal digit item-id, corresponding to the BASIC opcode generated by the BASIC compiler; attribute 1 is the symbolic name for the opcode. These entries are used by the BASIC compiler "A" option to generate a listing of the BASIC object code.

An example of the use of the BLOCK-PRINT verb:

```
>BLOCK-PRINT S  [CR]

 SSSSS
SS   SS
SS
 SSSSS
     SS
SS   SS
 SSSSS
```

5.3.3 PROCLIB FILE

The PROCLIB file is used to contain all common PROCs (e.g., LISTU, CT, etc.). Each MD will contain a pointer to PROCLIB and items that transfer control to the corresponding PROCs in PROCLIB. For further information, refer to the PROC section of the PICK Reference Manual.

5.4 ACCOUNT FILE MAINTENANCE

5.4.1 CREATE-ACCOUNT PROC

The CREATE-ACCOUNT PROC generates a new account according to given specifications. It then copies the contents of the NEWAC file (the prototype MD) to the new user MD. The CREATE-ACCOUNT PROC is invoked by typing in the PROC name:

```
>CREATE-ACCOUNT  [CR]
```

The PROC then prompts the user for the required information as follows. Note that defaults may be overwritten.

```
ACCOUNT NAME:            MODULO,SEPARATION:  29,1
RETRIEVAL LOCKS:         (See Section 7.2.1)
   UPDATE LOCKS:
PRIVILEGE LEVEL (0-2):0
CONTROL CODE: L          (See Section 5.2.1)
 R--RESTART FLAG, U--UPDATE FLAG, L--DEFAULT
    PASSWORD:            (Optional)

SYSn VERBS ADDED TO THE ACCOUNT

 [901] 'A' account created!
```

The CREATE-ACCOUNT PROC should not be used to create a new synonym to an existent account; this should be done by using EDITOR to create the file synonym definition item (Q-item) in the SYSTEM dictionary.

5.4.2 DELETE-ACCOUNT PROC

DELETE-ACCOUNT deletes an account and all its files from the PICK system. DELETE-ACCOUNT runs the program DEL-ACC in SYSPROG-PL. The BASIC program removes the SYSTEM D-pointer for the account and puts it in SYSPROG's MD. Then it removes all D-pointers to data files from all the dictionaries on that account and places them in the account's MD. The program then calls on the DELETE-FILE verb which deletes the account's MD, plus all dictionary and data-level files for that account from SYSPROG's MD. Requirements to run DELETE-ACCOUNT:

1. You must be logged on to SYSPROG.
2. SYSPROG must have Q-pointers to the MD of the account and to SYSTEM.
3. D-items must exist in DICT SYSTEM for SYSPROG and the account name.
4. SYSPROG must have access to SYSTEM and all files on the account to be deleted.

All users should log off before running DEL-ACC because an item in the SYSTEM dictionary will be deleted. The DEL-ACC program produces a listing of all files being deleted. An example of DELETE-ACCOUNT usage:

```
>DELETE-ACCOUNT                          PROC name is typed at TCL.

ACCOUNT NAME ?SHERRY

FILES TO BE DELETED IN ACCOUNT SHERRY     02 APR 82      PAGE 1

FILE              BASE     MODE    SEP

MD                34593    37      1
GEN/LED           85344    1       1
GEN/LED           49911    231     1
BP                44319    17      5

DO YOU STILL WANT TO DELETE THE ACCOUNT ?YES    Must start with 'Y'
```

5.4.3 POINTER-FILES

File level dictionaries define the structure of data files, and they contain pointers to those data files. If a file level dictionary has no associated data file, it is a 'single level' file.

The POINTER-FILE contains the pointers to select-lists that are stored by the SAVE-LIST verb. These pointers may be examined, but, like file-pointers, should never be altered in any way by the user. The format of these pointers:

Item-id	item name	name is provided with SAVE-LIST verb.
001	CL	CL for lists.
002	FID	Base FID of the list.
003	n	Number of frames in the list.
004	m	Number of items in a list.
005	time & date	Time and date of generation.

The POINTER-FILE is referenced implicitly whenever the SAVE-LIST, GET-LIST, DELETE-LIST, COPY-LIST verbs are used. The POINTER-FILE is a single level file. The file-defining entry "POINTER-FILE" in the ACCOUNT MD must have the code "DC" in attribute 1. This indicates that the file contains non-standard items. When a new account is created, a POINTER-FILE is created for that account.

5.5 BASIC PROGRAM FILE

The BASIC program file must have a dictionary level and one or more data-level files and the master dictionary entry for the BASIC program file must contain a 'DC' in attribute 1. The source code must be in a data-level file and the dictionary will contain pointers to executable object code. This means that only source items will be in the program file. The symbol table information is associated with the object code and the object code, which is emitted by the compiler, does not need to be loaded before it can be run. It is equivalent to a CATALOGed program.

The BASIC file dictionary pointer item list:

item-id	item-name	
001	CC	Name given with BASIC program creation.
002	FID	CC for program.
003	n	Base FID of the program.
004	m	Number of frames in the program.
005	time and date	Null.
		Time and date of generation.

If there are multiple data files and if there is a program with the same name in more than one of them, the last one compiled is the one which will be run.

The CATALOG verb will include the name of the program in the master dictionary with a pointer to the file which contains the particular program.

The DECATALOG verb is available to delete the object code from the system. It does not require that the program has been CATALOGed.

5.6 SYSTEM MESSAGES FILE

5.6.1 ERRMSG FILE

This dictionary level file in the SYSPROG account contains the system messages (error and informative, see Appendix A). Each account's MD must have an item called ERRMSG which points to this file in the SYSPROG account. (This is automatically created by the CREATE-ACCOUNT PROC.)

Error messages generated by TCL, ACCESS, BASIC, PROC or any other system software are contained in the ERRMSG file. The user may change the error messages, add new error messages, or create another ERRMSG file for each account. This can be particularly useful when used in conjunction with the STOP and ABORT statements in BASIC, in which the user can specify an error message and pass parameters to the error message processor.

Items in the ERRMSG file must follow a certain format, in which the first character in each line of the item defines a special operation, as listed below:

Character	Meaning
A	Inserts the next parameter in the list of parameters which was passed to the error message processor with the error message. The parameters may be specified by the BASIC program (in the case of a BASIC STOP or ABORT statement), or by some system processor in the case of system-generated error messages.
A(n)	Inserts the next parameter, left-justified, in a field of n blanks.
D	Places the current date in the output buffer.
H	Causes the string following the "H" to be placed in the output buffer, with no carriage return or line feed. At the end of the error message item, the string "H+" will inhibit the final carriage-return/line-feed that is normally outpt.
L	Causes the output buffer to be printed with a carriage-return and line-feed.
L(n)	As above, and also causes n-1 blank lines to be printed.
R(n)	Inserts the next parameter right-justified in a field of n blanks.
S(n)	Sets the output buffer pointer to location "n".
T	Places the current time in the output buffer.
X	Skips a parameter in the parameter list.

5.6.1.1 Special ERRMSG File Items

The item "LOGON" in the System dictionary contains the request to logon to the system, typically:

```
LOGON TO THE GA ZEBRA 2500 AT 12:34:00
PLEASE ENTER ACCOUNT NAME>
```

When a user logs onto PICK, the error message specified by the item "LOGON" in the ERRMSG file is printed on the user's terminal. Therefore, any message which is to be received by all users on the system immediately upon logging on may be placed in this item. This item must exist on file even if there is to be no general system message.

The ERRMSG items "335" and "336" contain the connect time messages displayed when a user logs on or off the system.

Some examples of error message processing are:

In the BASIC program, the lines...

```
FILE = "BP" ; ID = "1006"
OPEN "",FILE ELSE STOP 201,FILE
READ ITEM FROM ID ELSE STOP 202,ID
```

could cause the program to stop with either of the following:

```
[201]  'BP' IS NOT A FILE NAME
'1006' NOT ON FILE.
```

If the item "LOGON" in the ERRMSG file for an account looked like:

```
HHello out there!
L
HIt's now
T
H and all's well!
```

then the user would see the following when he logged on:

```
Hello out there!
It's now 11:22:33 and all's well!
```

5.6.2 PRINT-ERR VERB

The PRINT-ERR verb allows the user to invoke the error message processor from TCL. The format is:

```
>PRINT-ERR file-name item-list
```

The error messages specified in the item-list will be processed with a parameter list of A,B,C,D... For example:

```
>PRINT-ERR ERRMSG 201  [CR]
[201] 'A' IS NOT A FILE NAME

>PRINT-ERR ERRMSG 289

             TERMINAL PRINTER
PAGE WIDTH:      A        B
PAGE DEPTH:      C        D
LINE SKIP:       E
LF DELAY:        F
FF DELAY:        G
BACKSPACE:       H
TERM TYPE:       I
```

dictionaries and files 6

As introduced in Section 1, files are organized in a "hierarchical" structure with files at each level pointing to multiple files at the next lower level. Four distinct file levels exist: System Dictionary, Master Dictionary, File Dictionary, and Data File.

The word "file" refers to a mechanism for logically maintaining a set of similar items. The data in a file must be accessed via the Dictionary associated with it. The dictionary is an index, or a directory, to a file. However, since the dictionary is also a file, it contains items like a data file. The items in a dictionary define the lower level dictionaries and data files.

6.1 FILE ACCESS

The file access system was designed to allow the access of a particular item (or a number of particular items) in a file, or to access all items in a file, consecutively.

A file is a logical structure which associates a set of items so that they can be accessed for both retrieval and update. Items may vary in length, but the maximum size of an item is 32,267 bytes. There is no limit to the number of items which may be contained in a file, nor any limit to the number of files in an account. Each item has a "name" which is called its item-id. An item-id is an identifier (key or name) and must be unique to the file which contains it.

Items are stored in the file in a "pseudo-random" sequence; this sequence is determined by the result of a computational "hashing" (randomizing) technique which is employed by the system for purposes of storage and retrieval of data on disk. This technique utilizes the item-id along with other predefined parameters for the file to produce the disk-address (frame-identifier or FID) that identifies the location of the item.

Items that are stored in a file may be accessed directly using the item-id as the key, or sequentially in the pseudo-random sequence. If items are to be accessed in any sorted sequence, a preliminary pass through the file to generate the sort sequence is needed (see SORT and SSELECT verbs in the ACCESS Manual). The result of the preliminary pass is a list of item-ids; this list may be saved for future use or used to access the items in the file in the required sorted sequence (see also SAVE-LIST and GET-LIST verbs in the ACCESS manual).

The direct file access technique, which uses the item-id to locate the item within the file, is an efficient method of locating data and lends itself to the on-line nature of the PICK system. The system overhead required to access an item using this technique is essentially independent of the actual size of the file.

Special reserved characters are used as delimiters for storing data within an item. These delimiters were described and illustrated in Section 1.4.1. The hexadecimal value of these delimiters is unique to PICK:

Attribute Marks (^)	X'FE'
Value Marks (])	X'FD'
Subvalue Marks (\)	X'FC'

This item structure allows each attribute (including values and subvalues) to be a variable length. This is discussed in Section 6.4.2, Item Structure, Physical.

In summary, the PICK system can accommodate:

- ANY NUMBER OF FILES, WHICH CONTAIN:
- ANY NUMBER OF ITEMS (RECORDS), WHICH CONTAIN:
- MULTIPLE ATTRIBUTES (FIELDS), WHICH MAY CONTAIN:
- MULTIPLE VALUES, WHICH MAY CONTAIN:
- MULTIPLE SUBVALUES.

All files, items, attributes, values, and subvalues are variable in length; and each item must be less than or equal to 32,267 characters in length.

6.2 THE DICTIONARIES

A dictionary defines and describes data within its associated file. The following dictionary levels exist within the system:

1. System Dictionary (one per system).

2. Master Dictionary (one per user account).

3. File Dictionary (one per file or files).

Since the dictionary itself is also a file, it contains items like a data file. The items in a dictionary serve as the actual definitions for lower level dictionaries or data files. The following types of items are stored in dictionaries:

1. File Definition Items (file-names/pointers)
   (also called "D" items).

2. File Synonym Definition Items (file-names/pointers)
   (also called "Q" items).

3. Attribute Definition Items (attribute names)
   (also called "A" items).

The File Definition Items and the File Synonym Definition Items are used to define files. The item-ids of these items are the file-names of the files they define or point to. File-names must start with a non-numeric character, may be of any length and may contain any character except a comma (,) or a semicolon (;). The Attribute Definition Items are used to define attributes within data file items.

For example, "INVENTORY", "TEST.FILE" and "Z1" are all legal file-names. It is common practice to use file-names that are descriptive of the type of data stored within the file. A file is said to be defined from the dictionary that contains the "D-item" that points to its dictionary. Therefore, according to the hierarchy of files in the system, all Master Dictionaries (or MDs) are defined from the SYSTEM dictionary. In turn, a user may define any number of user dictionaries (with associated file or files) from his Master Dictionary (see CREATE-FILE processor). Note that D-items are automatically created by the CREATE-FILE processor and should never be created by using the EDITOR.

In order to access a file in another user's account, the user must create a File Synonym Definition Item ("Q-item") using the EDITOR. Assuming that the system security structure permits it, such a synonym file definition allows access to any file within the system.

A synonym file-pointer may also be used for convenience. For example, the INVENTORY file may have a synonym file-name INV, which reduces the number of characters the user has to type in order to access a file.

The data in each dictionary item consists of attributes (and optional multivalues) just like the data in file items.

For ACCESS processors, special dictionary items (called Attribute Definition Items or A-items) define the nature of the data stored in their associated file. They contain additional information such as:

1. Conversion specifications which are used to perform table look-ups, masking functions, etc.

2. Correlative specifications which are used to describe inter-file and intra-file data relationships.

3. Justification (left or right) for output purposes.

A data file is referenced by its "file-name". The dictionary file which is associated with that data file is referenced by "DICT" followed by the data file-name. A dictionary file may have more than one data file associated with it. This relationship is explained in the following section.

In summary, a dictionary contains:

1. File Definitions, or "D-items" that define the physical extents of other, lower-level files.

2. File Synonym Definitions, or "Q-items" that point to files in a user's or another account.

3. Data Definition Items or "A-items" that are used by the ACCESS processor to define the structure of data in the data section of the file.

In addition, a Master Dictionary contains:

1. Verbs (see TCL, Introduction to PICK Manual).

2. PROCs (see PROC Manual).

3. Vocabulary elements of the ACCESS language (see ACCESS Manual).

6.2.1 THE SHARING OF DICTIONARIES

File-level dictionaries may define a unique data file or multiple data files. When a dictionary defines multiple data files, it is said to be "shared" by those data files. The characteristics of the data in these data files are typically similar.

For example, there may be sets of data relating to the various departments in a corporation. For ease of maintenance, these sets of data may share a dictionary, since the dictionary items that describe the data are identical for each department. These dictionary items, used by the ACCESS processor, apply to all of the data files defined by that dictionary. This structure has the advantage of requiring only one set of dictionary items for a set of similar files.

Any number of data files sharing a dictionary may be opened simultaneously. The general form for specification of a data file is:

```
dict-name{,data-name}
```

The first parameter, dict-name, always specifies the file dictionary. The second parameter, data-name, specifies the data file and is required only when multiple data files are using a common dictionary. If only one data file is using a dictionary, then the form:

```
file-name
```

specifies the dictionary and the data file of the same name.

For example, the inventory file may be called:

```
INVENTORY
```

but the departmental data files, whose shared dictionary is called "DEPT", require a further specification. For example:

```
DEPT,ACCOUNTING          or
DEPT,MAINTENANCE
```

The dictionary of a file contains a "D-item" which defines the associated data file. If the dictionary is not shared, the item-id of this pointer (file-name) is the same as that of the dictionary; this is the default case. Therefore, for example, the INVENTORY dictionary will contain an item, also called "INVENTORY", which is the pointer to the associated INVENTORY data file. The DEPT dictionary, on the other hand, will contain as many D-items as there are departments; the item-ids of these pointers may be the department names.

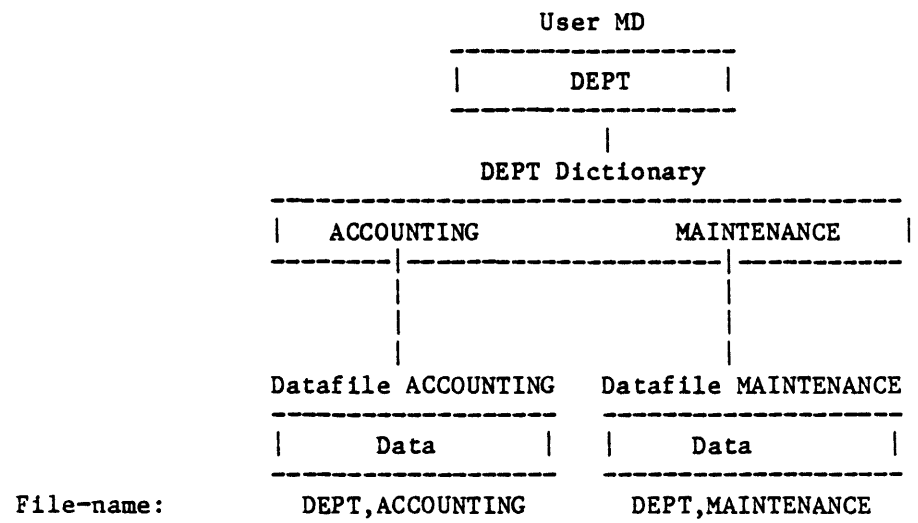
In the following example, the statements required to create a shared dictionary structure are:

1. To Create the dictionary of the file:

```
>CREATE-FILE DICT DEPT m1,s1  [CR]
```

2. To create the data section for each data file:

```
>CREATE-FILE DATA DEPT,ACCOUNTING m2,s2  [CR]
>CREATE-FILE DATA DEPT,MAINTENANCE m2,s2  [CR]
```



6.3 FILE STRUCTURE

6.3.1 BASE, MODULO, AND SEPARATION

The physical boundaries of the random-access file are defined by three parameters: the BASE, the MODULO, and the SEPARATION. The selection of a proper MODULO and SEPARATION is essential for an efficient file access method. An algorithm for optimum selection is presented in the next section.

The physical boundaries of a file are stored in the associated dictionary's File Definition Item. The item-id of this item is the file-name.

Files are defined at the time of creation by the following three parameters:

BASE — the physical disk address (frame-identifier or FID) of the start of a contiguous block of reserved disk space. Base is automatically selected by the system.

MODULO — the number of groups that the file space is logically divided into (sometimes called "buckets"). Modulo is selected by the user.

SEPARATION — the number of sequential frames per group. Separation is selected by the user.

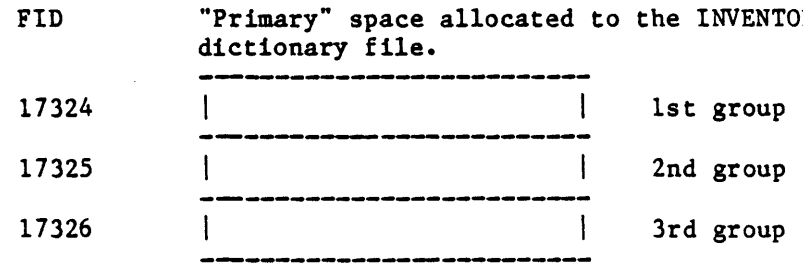
The BASE, MODULO, and SEPARATION of the file are stored by the CREATE-FILE processor (see Section 4.1) when the file is created. (These parameters should never be altered in any way by the user.)

At the time of file creation, a contiguous block of disk space is reserved. The size of this contiguous block is MODULO*SEPARATION, and is called the "Primary Space" allocated to the file. This does not, however, define the total space available for the file. As data is placed into each group, the group may overflow by linking on additional disk frames, as needed. There is no theoretical limit to this growth, other than the physical limit of disk space available. In practice, however, a group should be kept as small as possible. This may be achieved by the optimum selection of the file's MODULO.

An example showing a file's defined BASE, MODULO and SEPARATION:

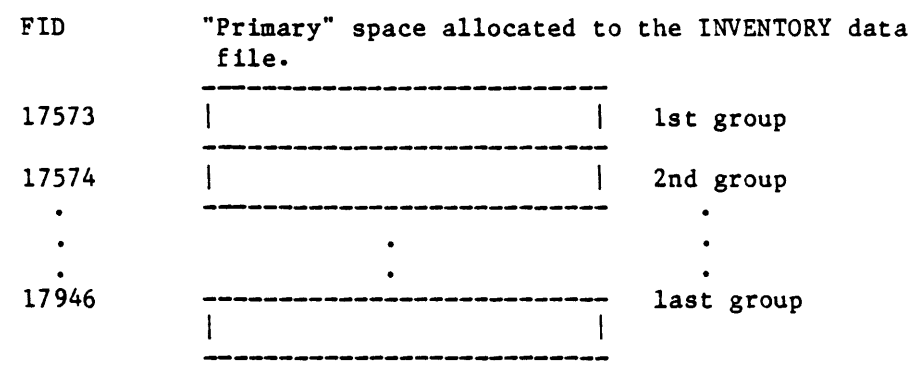
Item "INVENTORY" in the MD:

```
   INVENTORY

001 D
002 17324 ------- (base)
003 3 ----------- (modulo)
004 1 ----------- (separation)
    .
    .
```



Item "INVENTORY" in the dictionary INVENTORY

```
    INVENTORY

001 D
002 17573 ------- (base)
003 373 ---------- (modulo)
004 1 ----------- (separation)
    .
    .
    .
```



6.3.2 SELECTING MODULO AND SEPARATION

The effective file accessing and efficient disk utilization depend on proper selection of modulo and separation.

Recall that "modulo" is the number of groups in a file and "separation" the number of contiguous frames per group. A file is created by specifying its modulo and separation parameters; the frames allocated by the system (modulo*separation) are referred to as "primary" file-space. As data is placed into the file, any group may overflow by attaching frames from the available system space pool; this space is referred to as the "overflow" file-space.

When an item is to be added to a file, its item-id is "hashed" or converted into a large number by a multiplication process. This number is then divided by the modulo of the file. The remainder of this division is the "group" into which the item will be placed. This method will produce an even distribution of items in each group.

When an item must be retrieved or located, the same procedure is followed.

In the current file structure, a separation parameter of more than 1 should only be used if the average item-size is greater than 1000 bytes. In this case, a few disk reads are avoided when searching for (reading) an item in the primary file-space. In all other cases, particularly in a multi-user environment, the disk-head will almost certainly have moved during the time between the moment that a process requests one frame of a group and the next; therefore, whether the next linked frame of the group is contiguous (that is, if the separation is greater than 1) or not makes only a marginal difference.

Selecting a proper modulo is extremely important, since the number of groups directly affects the search and update time for an item in the group. The modulo separation process will attempt to make the average group length between 1 and 2 frames. If the item-size is 250 bytes or greater, this rule must be modified. Try to minimize as far as possible, the average number of frames in a group. The average number of items in a group should be selected with the average item-size in mind; the larger the item-size, the smaller the number of items in a group.

The number of disk reads, which is the factor that causes the most degradation of overall system response, increases dramatically as the number of frames per group increases. This is due to the fact that on the average, one-half of the frames in a group have to be written back to the disk after an item update. Thus, to update an item in a group, the system has to read every frame in the group, and write and verify one-half of them.

With this in mind, it is recommended that the following tables be used as a guide in selecting modulo, and that separation should be 1. Note that the discontinuities in the items/group columns occur because the selection of the number is such that the bytes/group figures are close to integral multiples of frames (500, 1000, 1500, etc.).

The last figure in Table 6-1, 0.8 Items/Group may be used for files with relatively few items that are very large, such as Assembly or BASIC program files. If the number of items in such a file is also very large, adjust the Items/Group figure upwards, since the lower figure will result in a lot of wasted disk space. Using the table, you can select an appropriate Items Group value; knowing the expected number of items in the file then gives the approximate modulo. The actual modulo must not be a multiple of 2 or 5; and should preferably be a prime number. (When the approximate modulo is a multiple of 2 or 5, you should round up; i.e., change 230 to 231.)

Table 6-1. Selecting Items/Group

If Average Item-Size is:		Then Average Items/Group Should Be:		And Average Bytes/ Group will be:
20	X	22.0	=	440
35	X	13.0	=	455
50	X	9.0	=	450
75	X	12.0	=	900
100	X	9.0	=	900
125	X	7.5	=	937
150	X	6.0	=	900
175	X	8.0	=	1400
200	X	7.0	=	1400
250	X	5.8	=	1450
300	X	6.4	=	1920
350	X	5.5	=	1925
400	X	4.8	=	1920
500	X	3.8	=	1900
1000	X	3.0	=	3000
5000	X	0.8	=	4000

Table 6-2. Examples of Computing Modulo

Average Item Size		Approximate # of Items		Items/Group From Table 6-1		Approximate Modulo
20		850	/	22.0	=	39
40		8000	/	11.0	=	727
210		1800	/	7.0	=	257
4000		230	/	1.0	=	230

6.4 ITEM STRUCTURE

6.4.1 PHYSICAL

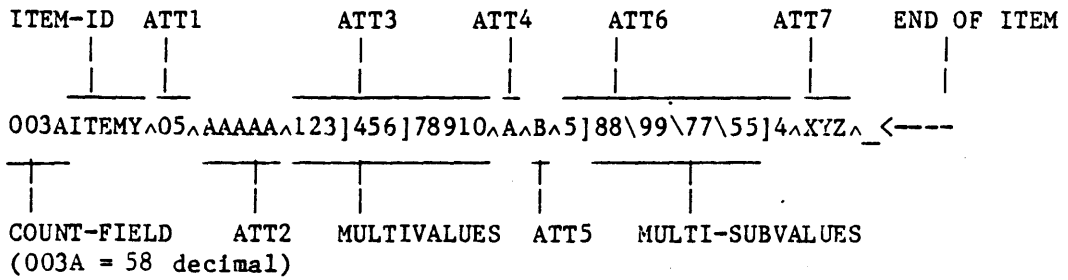
Data within an item is stored as attributes, values, and subvalues, all of which provide variable length storage. This section further describes the physical item format as stored on disk.

As briefly described previously (Sections 1.4.1 and 6.1), an item consists of one or more variable length attributes, separated by attribute-marks. An attribute mark is a character with a special PICK value of X'FE' (hexadecimal), which prints out as '^'. The first attribute in an item (attribute 0) is the item-id. The item-id is preceded by a four-character hexadecimal count field which specifies the total number of characters in the item including the count field itself. For example, consider the following stored item:

```
002EITEMX^LINE1^SMITH, JOHN^1234 MAIN STREET^
```

Attribute 0 is the item-id "ITEMX". It is preceded by "002E" which specifies that there are X'002E' (decimal 46) bytes in the item. Attribute 1 of "ITEMX" is "LINE 1". Attribute 2 is "SMITH, JOHN". Attribute 3 is "1234 MAIN STREET".

An attribute, in turn, may consist of any number of variable length values separated by value marks. A value mark has an eight bit PICK value of X'FD', which prints as "]". Finally, a value may consist of any number of variable length subvalues (also known as secondary values) separated by secondary value marks. A secondary value mark has an eight bit value of X'FC', which prints as "\". For example, consider the following item:



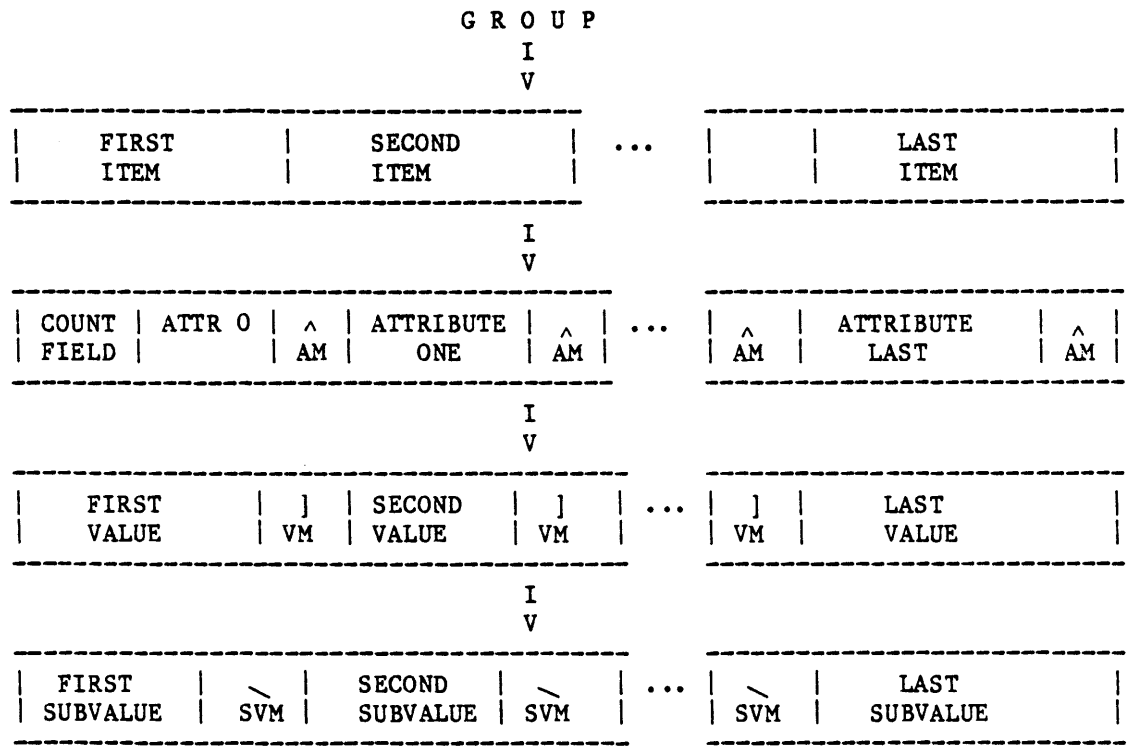
The absence of an attribute value is specified by placing an attribute mark immediately following the attribute mark that indicates the end of the previous attribute (i.e., '^^'). This maintains the correct attribute sequence. The "null" between two adjacent attribute marks may be thought of as representing the absent attribute. Depending on the data structure, two adjacent value marks may be permissible, indicating the absence of a multivalue. Two adjacent sub-multivalues are normally not permitted.

The mnemonics AM, VM, and SVM will be used hereafter to denote attribute mark, value mark, and subvalue mark, respectively.

Within a group, there may be zero or more items whose item-id's hash to that group. Such items are stored sequentially in the group, the sequence being solely dependent on the order in which the items are created. Once created, the order of the individual items in the group does not alter; as the item size changes due to updates; the remaining data in the group is shifted left or right to accommodate the new item.

After determining the group to which an item-id hashes, a linear search is conducted to find the particular item-id that is being retrieved. The count field is used to skip from one item to the next during this search. The presense of an AM where the count field of the next item should be indicates the END-OF-GROUP condition. An empty group therefore has an AM in the very first data position, which is also the condition set up by the CREATE-FILE and CLEAR-FILE processors (Sections 4.1 and 4.2).

In summary, the physical file item structure:



6.4.2 LOGICAL

This section describes the item structure at the logical level. While it is important to understand the physical item structure, in normal system usage items are always accessed at a more abstract or higher level. Files are identified by a file-name. Within a file, items are referenced by their item-id. Attributes are referred to as lines (e.g., Attribute 1 is called "line 1"). The first example in this section shows a sample COPY operation where the item with the item-id ITEMX (in the file SAMPLE-FILE) is being copied to the terminal. The item has three attributes (lines) of sample data.

Utility processors like COPY and the EDITOR deal at the file-item line level. They make no logical distinction in definition between various lines in an item other than their implied line numbers.

ACCESS processors, however, add an additional dimension through the use of the dictionary. The dictionary informs ACCESS as to the nature of the information stored for each of the attributes.

An item may be thought of as a "record" in computer terminology. It is more effective to think of and use an item as a group of related records, however. One tends to see a record as a collection of fields distributed horizontally, having meaning by virtue of their offsets from the initial byte of the record.

In the PICK system, a data-string has meaning by virtue of its attribute number. Therefore, if you think of an item (record) as a vertical list of attributes (fields) with attribute 1 on the first line, attribute 2 on the second, etc., you get a clearer picture of the system's storage structure.

Further, the basic intent of the value mark is to delimit the contents within each attribute, and the intent of the subvalue mark to delimit multiple entries within each value.

It is therefore effective to store transaction records relating, for example, to a single vendor, within one item. Within a single attribute, the fields from different records may be separated by value marks. Attributes used in this manner are referred to as multivalued. Continuing the chain, a value within an attribute may itself contain several values. These are called subvalues and represent multiple subrecords within a given transaction record, as in the case of a purchase order specifying several different parts. The individual records remain identifiable because of the ordinal relationship of the delimiting value marks. The ACCESS processor is then able to generate reports from this storage structure.

The logical item format is identical for all processors. It is the responsibility of the user to ascertain the further qualifications, if any, of the various attributes. For example, the following item listing by COPY is compared with the next listing produced by the LIST processor.

```
        >COPY SAMPLE-FILE  ITEMX (T)  [CR]

          ITEMX  <-------------------------------- item-id
    001   5207  <-------------------------------- Attribute 1
    002   SMITH, JOHN  <--------------------------- Attribute 2
    003   1234 MAIN STREET  <---------------------- Attribute 3

        >LIST SAMPLE-FILE "ITEMX" ATTRIBUTE-1  NAME  ADDRESS  [CR]

        PAGE 1                              09:28:32   12 NOV 1982

        SAMPLE-FILE..ATTRIBUTE-1..NAME.............ADDRESS.........

        ITEMX         04/03/82      SMITH, JOHN      1234 MAIN STREET
```

Here, the SAMPLE-FILE dictionary "defines" attribute 2 (line 2) as NAME and attribute 3 (line 3), as ADDRESS. This permits the user to reference his data symbolically (through dictionaries). However, the actual data stored on file is the same regardless of the processor accessing it.

Also note that the COPY of the item displays a value of 5207 for attribute 1 of the item, whereas, the ACCESS listing displays it as "04/03/82", which is the same data after conversion using the standard system date code.

6.5 ITEM STORAGE AND THE HASHING ALGORITHM

The PICK system employs a computational group hashing technique which utilizes the item-id and the file parameters that are defined at the time of file creation. This technique generates the disk address (FID) of the group in which the item is stored. The hashing formula used by the system to store or retrieve items is:

```
X = 0
FOR J = 1 TO LEN(ITEMID)
    X = X*10 + SEQ(ITEMID[J,1])
NEXT J
GROUP = REM(X,MODULO)
FID   = GROUP*SEPARATION + BASE
```

where:

ITEMID	contains the sequence of characters in the item-id.
LEN	function returns the number of characters in the item-id.
ITEMID[J,1]	extracts the j-th character of the item-id.
SEQ	function converts the above character to binary for addition.
REM	function returns the remainder of the division of X by MODULO.
FID	is the resulting disk address where the item may be found.

The item-id is treated as a variable length string of binary bytes; these bytes are accumulated sequentially with each partial sum being multiplied by 10. Dividing this value by the positive integer MODULO yields an unsigned integer remainder within the range:

```
0 <= Remainder < MODULO
```

This is then the group number (i.e., 0, 1, 2, ..., up to MODULO - 1) where the item is to be stored. Multiplying by the SEPARATION and adding the BASE yields the actual FID of the first frame in the group.

After computing a FID to locate the specific group in which the item resides, each item's item-id in the group must be compared for a "match". The frames comprising a group are linked both forward and backward. This system facility makes the group appear as a physically sequential string where items are stored one immediately after another. In fact, any portion of an item may spill across a physically frame boundary.

When a file is created, it is allocated a primary area of frames, the number of frames being: MODULO * SEPARATION. Thus, this amount of contiguous disk space is permanently allocated to the file. As the file grows, individual groups may fill up. When this happens, an additional frame is added to the group from a pool of available space. This additional frame is linked into the group to increase the length of the logically sequential group. Additionally, if a delete or update causes the group to shrink, any unused frames outside the primary area are returned to the pool of available space.

6.6 FILE ITEM STRUCTURE

To summarize, all files consist of one or more items. Each is identified by a keyword, the item-id, that can be located quickly through the hashing algorithm. An entire item is, in reality, a string of attributes delimited by the character " ". The first attribute, attribute zero, serves as the key and functions as the item-id.

An important concept to remember is that dictionaries are also files. Their structure follows the same pattern of item-id followed by attributes, like any other file. Dictionaries achieve special significance by following a relatively rigid structure, unlike data files which follow whatever structure the data is suited to.

Dictionary files reserve certain characters for attribute 1. If one of these reserved characters appear as attribute 1 in the item, then the item-id and the following attributes in that item take on special significance in the definition of the item and its purpose.

There are three classes of items that can appear in a dictionary file. These classes are discussed in the following sections, with emphasis and lengthy description of attribute definition items 7 and 8, conversion and correlative factors.

All classes of items are listed in the Table 6-3.

Table 6-3. Summary of File and Attribute Definition Items

Attribute Number	Name	File Definition Item	Synonym Definition Item	Attribute Definition Item
0	Item Identification	Item-id		
1	D/CODE	D,DX,DY,DC, DCX,DCY	Q	A,S,X
2	F/BASE or A/ACCOUNT or A/AMC	Base FID of file	Account-name	Attribute Mark Count, AMC
3	F/MOD or S/FILE or S/NAME	Modulo of file	Synonym file-name	Tag or Heading
4	F/SEP or V/STRUC	Separation of file	Not used	Controlling/Dependent (C/D) structure codes
5	L/RET	Retrieval lock code(s)		Reserved
6	L/UPD	Update lock code(s)		Reserved
7	V/CON	Reserved		Conversion specification
8	V/CORR	Reserved		Correlative specification
9	V/TYP	Justification code		
10	V/MAX	Maximum field length for item-id		
11	Reserved			
12	Reserved			
13	F/REALLOC	Reallocation during file-restore	Reserved	

6.6.1 FILE DEFINITION ITEMS (D)

File definition items are used to define lower-level dictionary files or data files. They are created automatically by the CREATE-FILE verb (Section 4.1).

At the System Dictionary level, file definition items are used to define the Accounting File and each user's Master Dictionary. File definition items in the MD are used to define the file level dictionaries, which in turn may contain one or more file definition items which define the associated data file(s). The item-id and each attribute of the file definition item contains required and optional information which describes (and 'points to') the lower level dictionary file or data file:

Attribute 0, Item-id — The item-id of a file definition item is the file name of the dictionary or data file being pointed to. If the item is pointing to a data level file, then the item-id must be the same as the name of the data level file.

Attribute 1 — This is the D/CODE attribute. It must contain one of the following: D, DC, DX, DY, DCX, or DCY. where:

- D identifies the file as a lower level dictionary file or a data file. When the file is created, the CREATE-FILE processor will place a D in this attribute. One or two letters may follow the D to indicate:
  - C the file contains binary data. (Used only by the system POINTER-FILE and BASIC files.)
  - X do not save this file on filesave tapes. (The file will not exist after a file restore.)
  - Y do not save the data in this file on filesave tapes. (On a file restore, the file will be recreated in an empty state.)

Attribute 2 — F/BASE is the base frame identification (FID). It must contain the base FID (as a decimal number) of the defined file.

Attribute 3 — F/MOD is the modulo of the file being defined. The modulo must be a decimal number.

Attribute 4 — F/SEP contains the separation (SEP) of the file being defined. The separation must be a decimal number.

CAUTION: Attributes 2, 3, and 4 must never be altered. Doing so will result in file corruption and possibly the system will crash.

Attribute 5	L/RET is the retrieval lock code, a form of password protection discussed in Section 7.2, System Security.
Attribute 6	L/UPD is the update lock code, a form of password protection discussed in System Security.
Attribute 7	Reserved.
Attribute 8	Reserved.
Attribute 9	The V/TYP attribute contains the justification code for values in the attribute ('L' or 'R' for flush left or flush right).
Attribute 10	The V/MAX attribute contains the maximum length for values in this attribute in decimal.
Attribute 11, 12	Reserved.
Attribute 13	Attribute 13 contains the new modulo and separation to be used when the file is reallocated during the system restore process, called file-restore. If the modulo and/or separation of a file needs to be changed, the user puts the new values in this attribute. The next file-restore will examine these values and reallocate the file accordingly.

The following file definition item example defines the file level dictionary for an INVENTORY data file.

This item has the item-id INVENTORY and is stored in the user's MD. The example also shows the file definition item which defines the data area of the INVENTORY file. This item also has an item-id of INVENTORY, but is stored in the dictionary level file and points to the data level file.

The user should note that in a single level (dictionary) file, the file definition item (e.g., INVENTORY) may be absent from the file dictionary or it may be present and point to the dictionary itself.

Item-Id	INVENTORY (in MD)	INVENTORY (in DICT INVENTORY)
D/CODE	001 D	001 D
F/BASE	002 17324	002 17573
F/MOD	003 3	003 373
F/SEP	004 1	004 1
L/RET	005	005
L/UPD	006	006
V/CONV	007	007
V/CORR	008	008
V/TYP	009 L	009 R
V/MAX	010 10	010 7

Note that the item "INVENTORY" in the Master Dictionary has definitions relating to the items in the dictionary of the INVENTORY file (such as V/TYP of "L" and V/MAX of "10"; the item "INVENTORY" in the INVENTORY dictionary has definitions relating to the items in the data section, such as V/TYP of "R" and V/MAX of "7".

6.6.2 FILE SYNONYM DEFINITION ITEMS (Q)

When an item in a dictionary file is used to define another "lower level" file, the item-id, or attribute 0, of the item becomes the name of the file being defined. It is sometimes convenient to give a file more than one name, for example, giving the file "INVENTORY" the name "INV" for short. File synonyms can also be useful in a Master Dictionary to alter the command language terminology and/or create abbreviations. The operating system makes this possible with the File Synonym Definition item. Aside from giving a file an alternate name within the same user account, the file synonym definition item can also point "outside" its account and reference files in other accounts, provided security restrictions are met. The item-id of the synonym definition item is the new version of the name. The "real" name is placed in attribute 3. The following is a summary of the attributes in a file synonym definition item.

Attribute 0, The Item-Id	The item-id becomes the synonym name of the file being defined. If the synonym item is being used to point to a file in another account, the item-id is the "real" name of the file.
Attribute 1	This is the D/CODE attribute. It must contain a "Q".
Attribute 2	The S/ACCOUNT attribute contains the account name in which the file "pointed to" by the Q-Item is located. This can be used to set up access to a file in another account. In this case, the name for both the original file, the D-Item, and the new synonym, the Q-Item are the same. Thus, the ability of a Q-Item to become a synonym is not used, only its ability to "point" to a file in another account. Access is still controlled by the security system. If attribute 2 is null, the file being pointed to is assumed to be in the same account.
Attribute 3	The S/FILE attribute contains the name of the file as defined by the D-Item. When referring to a file by its Q-Item name, the system will search back to the original file definition item for the base frame, modulo, separation and other attributes for purposes of file access. If this attribute is null, the synonym file is the MD.
Attribute 4	Not used.
Attribute 5	The L/RET attribute may contain retrieval lock codes.
Attribute 6	The L/UPD attribute may contain update lock codes.
Attribute 7	Reserved.
Attribute 8	Reserved.

Attribute 9	This is the V/TYP attribute. It contains the justification code. V/TYP defaults to 'L' if it is null.
Attribute 10	This is the V/MAX attribute. It contains the maximum length in decimal for values in the attribute being defined. V/MAX defaults to 10 if it is null.
Attribute 11	Reserved.
Attribute 12	Reserved.
Attribute 13	Reserved.

A synonym file definition item is required in order to access a file in another account. In addition, there are many cases where it is convenient to reference a file within the same account by more than one name. In this case also, a Q-Item must be created with attribute 2 of the Q-Item null. This is a better method of creating a synonym file pointer than by creating duplicate D-pointers.

A Q-Item to another user's Master Dictionary should have the user's account-name in attribute 2, and a null attribute 3. This allows faster access than if the file-name "MD" is placed in attribute 3.

Q-Items are created by using the EDITOR. The items are edited into the Master Dictionary using the form:

```
EDIT MD item-id
```

where item-id is the synonym name being defined.

There is also a standard PROC (SET-FILE) that creates a temporary Q-Item called QFILE, which may be used to set up a pointer quickly. This PROC is described in the Utilities manual.

The following example illustrates a sample INVENTORY file synonym definition item which allows the user access to the file in the account named SMITH; the user can reference this file via the synonym file name INV. It also shows sample Q-Items that point to another user's Master Dictionary and to a file within the same account.

```
(Item-Id)          MD        INV              USER3        SAMPLE

D/CODE             001 Q     001 Q            001 Q        001 Q
S/ACCOUNT          002       002 SMITH        002 SMITH    002
S/FILE             003       003 INVENTORY    003          003 SAMPLE-FILE
F/SEP              004       004              004          004
                     ...           ...            ...
```

These Q-items are in the Master Dictionary of user "JONES", Item "INV" is a synonym pointer to the file "INVENTORY", which is defined as a file in the Master Dictionary of user "SMITH". Note that the item MD must be ´001 Q´ and that the Q-pointers to other MDs do not have ´MD´ in attribute 3. Item "USER3" refers to the Master Dictionary of user "SMITH", since attribute 3 is null. Item "SAMPLE" is a synonym to the file "SAMPLE-FILE", defined in the Master Dictionary of JONES, since attribute 2 is null.

An example, using the EDITOR to create the Q-Item called "INV":

```
>EDIT MD INV  [CR]
NEW ITEM
TOP
.I  [CR]
001 Q  [CR]
002 SMITH  [CR]
003 INVENTORY  [CR]
004 .  [CR]
005 .  [CR]
006 .  [CR]
007 .  [CR]
008 .  [CR]
009 L  [CR]
010 15 [CR]
011  [CR]
TOP
.R 99 /.//  [CR]
004
005
006
007
008
EOI 011
.FI  [CR]

´INV´ FILED.
```

Note that null attributes may not be filled with blanks. Since a null input (carriage return or line feed only) will cause the EDITOR to exit the input environment, it is necessary to place a temporary character in the "null" attributes and later replace that character with a null via the REPLACE (R) command. (See the EDITOR manual for further detail.)

6.6.2.1 Q-Pointer Flexibility

If attributes 2 and 3 of the file synonym definition item are null, the Q-pointer is a pointer to the file in which it is stored. This case has two applications. If you type ED MD MD, you will find that the MD item contains only a Q in attribute 1. This is sufficient and any other definition is less efficient. Specifically, the MD entry should not be a D-pointer. The same follows for MD or the account name entry.

The second use is in the definition of a dictionary-only file. If you want to reference the file without typing 'MD' each time, an entry with the same name as the D-pointer to the dictionary in the Master Dictionary is inserted in the file dictionary whose only content is a Q.

An example of the uses of Q as the only attribute:

In the Master Dictionary

```
     MD          File reference to MD.
001  Q           Reference back to 'where you are now'.
```

In the dictionary of the file FILENAME

```
     FILENAME    The name referenced by the name FILENAME
                 in the Master Dictionary.
     001 Q       Reference back to the dictionary itself.
```

The name of the Q-pointer is discarded as soon as the first D-pointer is encountered. That is, a reference to QFILENAME which points to the file FILENAME will look for the D-pointer FILENAME in the dictionary of FILENAME. It will not look for a pointer by the name of QFILENAME. A partial exception to this is in ACCESS which will attempt to obtain the conversion, length, and justification from the Q-pointer.

If the Q-pointer does not contain them, then the ACCESS compiler will search the D-pointer for them. If the D-pointer does not contain them, then the conversion will default to null, the justification to 'L', and the field length to 9 bytes.

It is, therefore, possible to specify various formats for the item-id field for purposes of sorting and listing.

6.6.2.2 Account Specification

Attribute 2 of any Q-pointer references an account name. If attribute 2 is null, then the Q-pointer references a file in the account onto which you are logged. If attribute 2 is not null, the file-open processor will search the system dictionary for a definition of the account name.

If the processor does not find a D-pointer in the system dictionary, the system will respond with the following message:

[201] 'Contents of attribute 2 of QFILENAME' IS NOT A FILE NAME.

It is possible to reference files in the account onto which you are logged by putting the name of the D-pointer to the account in attribute 2 of the Q-pointer definition, but this will cause the system unnecessary work.

To reference the Master Dictionary of another account, the name of the D-pointer to the account (account-name) is placed in attribute 2 and a null in attribute 3.

A file or an account may be protected from access. (See Section 7.2, System Security.) If a user creates a Q-pointer to a file or an account which he is not authorized to access, the system will deny him that capability. An error message stating that the file or account is protected will be returned.

6.6.2.3 File Specification

Attribute 3 contains the name of the file referenced by the Q-pointer. If attribute 3 is null, then the default is to the Master Dictionary specified by attribute 2.

In general, the file name referenced in attribute 3 of the Q-pointer definition must be a D-pointer in the Master Dictionary of the account referenced in attribute 2.

6.6.2.4 Extensions to File-name Reference

The contents of attribute 3 of the Q-pointer definition may contain file-name, data-name. In this case, the Q-pointer will reference the data in data-name only, and will ignore the other data files referenced in the dictionary of file-name. The result is a considerable simplification of the BASIC programs and PROCs which reference the various data sets in a multiple data file structure.

The following example shows a Q-pointer that will reference the data file SCREW-DRIVERS in the dictionary of HARDWARE in the account INVENTORY:

```
QFILENAME
001 Q
002 INVENTORY
003 HARDWARE,SCREW-DRIVERS
```

A file dictionary does not need to reference a data file with the same name as the dictionary. It may be convenient to create the file as a dictionary-only file which contains a Q-pointer that points to itself. The dictionary may then be referenced without typing 'MD', which is useful if there is to be extensive development of data definition items. It is also convenient if there is to be several data-level files, since each may be given a recognizable name.

6.6.3 ATTRIBUTE DEFINITION ITEMS (A)

A PICK data file consists of items identified by item-id, followed by 1-n (or zero) attributes. The purpose of the Attribute Definition Item in the file dictionary is to define the nature of the data within a particular attribute for use by ACCESS processors. Each Attribute Definition Item has an Attribute Mark Count (AMC) which is a pointer to the data field (data item attribute) it is defining. For example, an AMC of 5 means the Attribute Definition Item "defines" atribute 5 of the data items. An attribute definition item defines the attribute specified by AMC, for all items in the related data file(s). In addition, the Attribute Definition Item provides a name for an attribute. For example, the user could refer to "LIST-PRICE" rather than "Attribute 14". Attribute Definition Items are constructed as follows:

Attribute 0, Item-Id	The item-id is the name desired for the defined attribute. attribute. This name would be used in ACCESS input statements to reference the defined attribute.
Attribute 1 (D/CODE)	Within a dictionary file, an item that has the character A in attribute 1 indicates attribute definition. It means that the following attributes define the nature of the data in the file or files the dictionary points to. An item with S in attribute 1 instead of an A indicates synonym attribute definition. This means the attribute is a duplicate of another, but is using a different symbolic name. An item with X in attribute 1 instead of an A means to skip this attribute for the time being. The number position of this attribute will be held open so that an attribute may be inserted here at a later time. Meanwhile, the number continuity of attributes beyond this attribute will be kept so that they may be listed in ACCESS retrievals. However, when you do not wish to skip this attribute, the X must be changed to A.
Attribute 2 (A/AMC)	Attribute 2 is the attribute mark count, or AMC attribute. It refers to the number of the attribute in the data file to which the attribute definition item refers. Remember, the item-id gives the mnemonic label the user will refer to the attribute by; this is the actual numerical pointer the system will use to identify the attribute. Since the item-id is attribute 0 for every item, attributes 1-n are logically referred to by the AMC of 1-n. If the AMC is 0, then the item-id is being defined. A "pseudo" AMC which is higher than the actual number of attributes in the file being referenced can be used to manage data that is computed, but not really stored in the file.

Attribute 2 (Cont)

For instance, the attribute "DEALER PRICE" might not be stored in the file, but would be computed by taking the real attribute "AVERAGE COST" and multiplying it by the real attribute "DEALER-MARKUP". Once defined, this "virtual data," which is never really stored, becomes as "real" and useful for the user as any stored data. In addition, an AMC of 9998 is used to access the current item counter (item sequence number), and an AMC of 9999 is used to access the size or count field of the item.

Attribute 3 (S/NAME)

Attribute 3 contains an optional tag or heading label that is used for LIST and SORT statement printouts. To save keystrokes, the item-id of the attribute definition item, which is normally the label, might be "PN". Attribute 3 could contain "INTERNAL INVENTORY PART NUMBER CODE" to make the printout more readable for the occasional user. The heading can be defined as having more than one line. If this attribute is null, the item-id will be used for the heading.

To specify tags for multiple column listings, multiple headings separated by a value mark, Control-], should be stored in this attribute.

Attribute 4 (V/STRUC)

Attribute 4 contains the associative structure code. This code can be used to identify "controlling" and "dependent" attributes. This relationship is used primarily in printout formatting. For instance, if the controlling attribute is suppressed during printout, the dependent attributes will be suppressed too. This relationship extends to attributes containing multiple values. If the 28th value of a controlling attribute is suppressed, the 28th values of each dependent attribute will be suppressed. Subvalues contained within values are controlled according to the value in which they are contained. A controlling attribute can control many dependents; a dependent attribute can only have one controller.

When defining a controlling attribute and the one or more dependent attributes it controls, the format is "C" followed by the attribute mark count or "AMC" of each of the dependent attributes (e.g., C;amc;amc; etc.).

When defining dependent attributes, the code D;amc is used in attribute 4 of the definition item of each dependent attribute to identify the one attribute that it is controlled by. (The AMC is the attribute mark count of the controlling attribute.) See the ACCESS manual for more detailed information.

Attributes 5 and 6 — Unused.

Attributes 7 and 8 (V/CONV and V/CORR) — The conversion and correlative attributes are described in Section 6.7, which follows.

Attribute 9 (V/TYP) — Attribute 9 contains the output justification specifications for the line printer or terminal screen (flush left or right) which are also taken into consideration during sorting. This defines how the data for the attribute being defined will print in reports and listings that are generated. This may be either "L", "R", "T" or "U". V/TYP defaults to "L" if it is null.

- L — is used to specify left-justified text. If the data for this attribute exceeds the length specified (in attribute 10), then the data will be folded at the end of the field (column) and excess characters will be printed on the next line. L also means that all sorts will be performed in left-to-right sequence.
- R — is used to specify right-justified output and right-justified numeric sorts (even for alphanumeric fields).
- T — will cause output to be left-justified. If the data exceeds the specified maximum length, it will be folded at the last blank space in the field, and the remaining characters will be placed on the next line.
- U — will cause output to be left-justified, but there will be no folding on overflow. If overflow occurs, excess characters will run into the next column.

Attribute 10 (V/MAX) — Attribute 10 contains the maximum length of values for the attribute in decimal. This maximum is for columnar printout purposes and does not represent a limitation on the length of the stored data. If you wish to suppress the listing of a control-break field on detail lines, a zero should be entered. V/MAX defaults to 10 if it is null.

Attributes 11-14 — Reserved for system use.

Examples of attribute definition items which define different fields for an INVENTORY file:

(Item-Id)		QUANTITY		LIST-PRICE		EXTENDED-PRICE
D/CODE	001	A	001	A	001	A
A/AMC	002	4	002	5	002	300
S/NAME	003		003	LIST PRICE	003	
V/STRUC	004		004		004	
	005		005		005	
	006		006		006	
V/CONV	007		007	MR2$,	007	MR2$,
V/CORR	008		008		008	A;4*5
V/TYP	009	R	009	R	009	R
V/MAX	010	7	010	8	010	10

6.7 CONVERSION AND CORRELATIVE CODES

Within the PICK operating system, data can be stored on disk in a variety of ways. For instance, a date is stored as a four-byte code; time is stored as the number of seconds since midnight. Also, data can be stored in one format, but can be converted in a variety of ways for output. A check register might store check amounts in decimal form, but a printout would include a leading dollar sign. The actual check might require a number of leading asterisks. A summary financial report might use that data base, but round up or down to even the dollar amounts. Both attributes 7 and 8 provide the ability to convert from a stored format to a processing format.

Using these features, data can be altered from a stored format to an intermediate format for computation, and then to another format for output. Since these functions operate on an attribute-by-attribute basis, different data within one file can be manipulated with complete flexibility.

Lines 7 and 8 of dictionary attribute definition items define the conversion and correlative codes. Conversion codes appear on line 7, correlative codes appear on line 8. Processing codes may be specified as either Correlative codes or Conversion codes depending upon when the user wants the codes to be applied to the data.

During execution of an ACCESS sentence, the data in the items being listed is represented in three different formats. The first is the "stored" format, which is the format of the attributes exactly as they appear in the items in the file. Whenever a piece of data is retrieved from a file it is picked up in stored format.

The Correlative code, if any, may then be applied to the data, converting it to "intermediate" format.

The intermediate format is used whenever:

1. The attribute name is part of a sort key.

2. The data is compared to a selection criterion.

3. The attribute name has a print limiter.

4. The attribute name has a TOTAL or GRAND-TOTAL connective.

5. The data produces a control-break.

6. The data is printed (except on break lines).

Conversion codes are applied as output conversions to the intermediate format data whenever the data is printed, including break lines. This transforms the data from "intermediate" format to "external" format. The data is printed in external format.

If a Conversion code is specified for an attribute name which is followed by a selection criterion, the conversion is applied as an input conversion to the values in the ACCESS sentence to form the selection criterion value.

There may be more than one conversion or correlative code. If so, they are stored as multivalues separated by value marks. A value mark is a right bracket (]) which may be initiated by Control] (ASCII 253).

Conversions may be written in an attribute definition item for the purpose of acting upon the data in that item. (Date conversions, for example.) They may be written in another attribute definition item which has the same AMC as an attribute which contains data to be used in the conversion or correlative. This is typical of translate conversions that use the value of the attribute as the item-id in another file.

A conversion may also be used as part of an F or A correlative in an attribute definition item which has no value of its own. In this case, the values of other attributes are used in calculations by referencing their AMCs. Also, when this is done, the attribute definition item with the correlative should have a dummy AMC on line 2. (99 in the example below.)

```
5                SEC              CSZ
001 A            001 A            001 A
002 5            002 5            002 99
003 ZIP          003 SEC          003 CSZ
004              004              004
005              005              005
006              006              006
007              007              007
008              008 T1,3         008 F3;", ";:;4;" ";5
009 R            009 R            009 L
010 5            010 3            010 30
```

In the above example, the ZIP code is stored in attribute 5. Another attribute, SEC, is a synonym of 5 (has the same AMC) and extracts the first three digits of attribute 5 for the postal sectional center. Yet another attribute, CSZ, concatenates attribute 3, 4, and 5 with a comma and space between 3 and 4 and a space between 4 and 5. (3 is city, 4 is state and 5 is zip.)

Note that CSZ has a "dummy" attribute number of "99". Although CSZ is an attribute definition item, it never has a value of its own. Its value is derived from other attributes (3, 4, 5) which do have values.

General guidelines in the use of conversion, correlative codes:

1. If there are both a correlative and a conversion, the correlative is processed first.

2. Correlatives convert the stored data to a processing format and conversions convert processing format to output, or external format. In some cases, the internal format and the processing format is the same (dates for example), in which case, the conversions convert the internal format directly to the external format. Likewise, sometimes the processing format is the same as the output format and no conversion is required.

3. Be aware that SELECTS and SSELECTS will ignore line 7, but will act upon the results of conversions and correlatives on line 8.

4. As a general rule, date and time conversions should be on line 7 and all other processing codes on line 8.

5. An F or A correlative may include other conversions or correlatives as elements.

The following is an overview of the correlative and conversion codes with a brief description of each.

Code	Description
A	ARITHMETIC or ALGEBRAIC. Allows required mathematical steps such as addition, subtraction, multiplication and division to be performed in algebraic form. For example: A("10"+5)*25 adds 10 to value of attribute 5 then multiplies that total by the value of attribute 25.
C	CONCATENATE. Used to join attribute values and/or constants into a string. For example: C10;" ";11 takes the value of attribute 10, 2 spaces, and the value of attribute 11 and makes them into a string.
D	DATE. Converts dates from internal format to external format.
F	FUNCTION or STACK PROCESSOR. Invokes the function processor which allows manipulation and arithmetical processing of attribute values and constants.
G	GROUP. Used as a correlative to extract one or more fields from a multiple field attribute value. For example: G2*1 would extract one field of data following the second * contained in the value of the attribute. A G correlative may also be used as part of a F correlative to extract fields from another attribute. F;6(G1*2) would extract two fields following the first * in attribute 6 even though the correlative itself was in an attribute other than 6.

Code	Description
L	LENGTH. Used to place constraints on what kind of data will be returned, based on data length. For example: L0 returns the number of characters in the attribute value. L25 returns the attribute value if it is 25 characters long. L15,25 returns the attribute value if it is between 15 and 25 (inclusive) characters in length.
MC	MASK CHARACTER. Used to convert an attribute value to all upper and lowercase letters or to extract all numeric or all alphabetic or all non-numeric characters from an attribute value.
ML	MASK DECIMAL. Used to format and scale numbers and dollar amounts; left-justified.
MR	MASK DECIMAL. Same as ML only right-justified.
MT	MASK TIME. Used to convert internal time of day to external format.
MX	MASK HEXADECIMAL. Causes ASCII data in attribute to be printed as its hexadecimal equivalent.
P	PATTERN MATCH. Used to restrict return to only those values which match the specified pattern.
R	RANGE. Returns only items whose attribute values fall within a specified range.
S	SUBSTITUTION. Can be used to substitute the value of another attribute if the value is not null or zero, and can substitute a specified value if the value is equal to null or zero.

Code	Description
T	TEXT EXTRACTION. Used to extract only characters in a certain position in a value. For example: T3,5 will return 5 characters starting with the third.
Tfile	TRANSLATE. Used to convert data by translating it from another file. For example: TINV;C;;4 If the above were used as a conversion or correlative in attribute 6, it would use the value of attribute 6 as the item-id in the INV file and return the value of attribute 4 in that item. If attribute 6 were multivalued, it would look in INV for item-ids with those values and return the value of attribute 4 from each of them.
U	USER-DEFINED. Causes an assembly language routine to be performed.

6.7.1 THE ARITHMETIC CODE (A)

The Arithmetic code 'A' is designed to perform the same function as the F-code, but its format is both simpler to write and easier to understand than the format of the F-code. The general form of the A-code is:

A(expression)

where an expression is made up of operands, functions and operators, as described below.

6.7.1.1 Operands

1. AMC Numbers - An attribute Mark Count (AMC) is specified by putting the number in the A-code, just like in the F-code. An AMC of 0 (zero) will indicate the Item-Id. The special AMCs 9999 and 9998 retain their original functions and can be legally inserted into an A-code. An AMC can optionally be followed by the letter "R", which indicates repetition of the value just like in F-codes.

2. AMC Names - An attribute name can be used instead of an AMC in an A-code, as long as the name exists in the dictionary of the file being listed. The dictionary name is used as an argument to the "N" function of the A-code. The format of the "N" function is N(attribute-name). For example, if the name INVOICE-AMOUNT exists in the dictionary, the corresponding "N" function would be N(INVOICE-AMOUNT).

The operation of the "N" function is:

The attribute-name is referenced in the dictionary of the file and an error message is printed if it is not found. The AMC of the dictionary item (attribute 2) is used as the AMC in the A-code.

Any correlatives existing in attribute 8 of the dictionary item, whether F-codes or A-codes, will be used in the A-code. If an A-code or F-code exists in attribute 8 of the dictionary item, the AMC from attribute 2 is ignored.

Note that an A-code can call another A-code by name and that the second A-code can specify a third A-code, and so on. However, no attempt is made to assure that an A-code does not call itself. If this is attempted, or any time that "nested" calls are made more than seven levels deep, the ACCESS compiler will abort with a RTN STACK FULL message.

3. Literal Numbers - A number is specified by enclosing the number in quotes, either single (') or double ("). For example, the number 10 could be specified by "10" or '10'. Any integer, whether positive, negative or zero is legal inside quotes.

4. Literal Strings - Any literal string, enclosed in single quotes (') or double quotes (") is a legal operand.

5. Special Operands - The A-code has several special system operands which are the same as special F-code operands (Section 6.7.4.1).

NI	is the item counter
NV	is the value counter
NS	is the subvalue counter
ND	is the detail-line counter
NB	is the break level counter
LPV	is the load previous value
D	is the system date (in internal format)
T	is the system time (in internal format)

These special operands can be used exactly like an AMC, attribute name or literal. Also, any of the above legal operands preceded by a minus sign, (-) is a legal operand.

6.7.1.2 Functions

1. Remainder Function: "R" - The remainder function takes two expressions as operands and returns the remainder of the first operand divided by the second. The format of the "R" function is R(expression,expression). For example, R(2,"5") returns the remainder when the value of attribute 2 is divided by 5.

2. Summation Function: "S" - The summation function takes one expression as an operand and works the same way as the S operator in the F-codes. For example, S(4) will sum any multivalues of attribute 4. The summation operator may appear anywhere in an A-code. A maximum of two summations is allowed.

3. Substring Function - A subtring may be specified by using square brackets. The numbers inside the brackets may be literal numbers, AMCs, or entire expressions. For example, 1["2",'3'] means the 3-character long string starting at position 2 of attribute 1. The expression 1["1",'99'*(2=4)] will evaluate to the value of attribute one, unless attributes two and four are different, in which case the expression evaluates to a null string.

6.7.1.3 Operators

1. Arithmetic Operators - The operators +, -, * and/ denote addition, subtraction, multiplication and division, respectively. All of the arithmetic operators take two expressions as operands, and return the sum, difference, product or quotient of the two operands. It is important to note that division in an A-code always returns an integer result, just like in F-codes, so that "3"/"2" evaluates to 1, not 1.5.

2. Relational Operators - The relational operators >, <, >=, <=, =, and # denote the logical relations greater than, less than, greater than or equal to, less than or equal to, equal and not equal, respectively. Each of the relational operators takes two expressions as operands, and evaluates to 1 (TRUE) or 0 (FALSE) depending whether or not the indicated relation holds between the two operands. For instance, "1">="2" evaluates to 0 (FALSE) because the number 1 is not greater than or equal to the number 2. Expressions involving these and other A-code operators are written much like BASIC expressions.

NOTE: The precedence of the operators is important to keep in mind when writing an A-code. In the absence of parentheses to indicate the order in which operators are to be applied, multiplication and division have greater precedence than addition and subtraction, which in turn have greater precedence than the relational operators. If two operators have the same precedence, they are applied from left to right. For example, 1*2+3<4 will evaluate as ((1*2)+3)<4, but 1>=2-3/4 will evaluate as 1>=(2-(3/4)). Also, 1+2-3 will evaluate as (1+2)-3, and 4/5*6 will evaluate as (4/5)*6. 20 levels of parentheses nesting are allowed in A-codes.

A-Code operator examples:

A-Code	Meaning
A1+2	Adds attributes 1 and 2.
A"10"*3	Multiplies the value of attribute 3 by 10.
AS(4+"25")	Adds 25 to each value of attribute 4, then sums the multivalues.
AN(INV-AMT)-N(BAL.DUE)	Subtracts the value of the attribute defined by BAL.DUE in the dictionary from the value of the attribute defined by INV-AMT.
AN(SS-NUM)['4','2']	Returns the 4th and 5th digits of the attribute specified by SS-NUM.

6.7.2 THE CONCATENATION CODE (C)

The concatenation code 'C' provides the facility to join two or more strings of data into one string. For example, names are often stored with last name in one attribute and first name in another attribute so that sorting may be done on the last name. However, to address a letter, you would want the name to appear as John Doe and this is done with concatenation.

The general form of the 'C' code is:

```
C x op x {op x op ...}    or    C op x op {x op x ...}
```

where:

C	is the code name.
x	is the character to be inserted between the concatenated attributes and/or literals. A semicolon (;) is a reserved character that means no separation character is to be used. Any non-numeric character (except a system delimiter or an asterisk) is valid, including a blank.
op	is an attribute mark count (amc) or any string enclosed in single quotes ('), double quotes (") or backslashes (\), or is an asterisk (*) which specifies the last generated value (from a previous conversion or correlative operation).

Concatenation may be accomplished with the 'F' code, which will be described later, or with the 'C' code. The C code may be used either as a correlative or as a conversion. Data will sort correctly if it is a correlative and used as the sort key, but will be ignored as a sort key if in line 7.

Attributes which create a string or value by concatenating with the C code must have an attribute mark count (AMC) of 0 (zero). Attributes containing a C code have no value of their own, but are given a value as a result of the C code action.

The strings, or elements, which are joined together may be values from attributes specified by using the attribute numbers, or specified literals enclosed in single quotes ('), double quotes ("), or backslashes (\).

The elements in the resulting string may be separated by any non-numeric character including a blank, but excluding a semicolon (;) an asterisk (*), or a system delimiter. The semicolon is used to show tha there is to be no separation. If a semicolon is to be in the resulting string, it must be enclosed in quotes as a literal.

The asterisk is reserved to indicate an element consisting of the value generated by a preceding conversion or correlative.

Examples of 'C' code use:

1. ATTRIBUTE VALUES

```
014  SMITH
015  JOHN H.
```

C CODE	RESULTING OUTPUT
C;"NAME":14,15	NAME:SMITH,JOHN H.

2. ATTRIBUTE VALUES

```
001  DIME
002  DOZEN
```

C CODE	RESULTING OUTPUT
C;1/2	DIME/DOZEN

3. ATTRIBUTE VALUES

```
001  DOE
002  MARY JANE
003  121 MAIN ST
004  TULSA
005  OK
006  74101
```

C CODE	RESULTING OUTPUT
C2 1	MARY JANE DOE
C1,2	DOE,MARY JANE
C1;',' 2	DOE, MARY JANE
C1;', ';2	DOE, MARY JANE
C2 1]C* 1	MARY JANE DOE DOE
C2 1 1	MARY JANE DOE DOE
C;"NAME:";1,2	NAME:DOE, MARY JANE
C"NAME: ";1;", ";2	NAME: DOE, MARY JANE

6.7.3 THE DATE CODE (D)

The date code 'D' provides the facility for converting dates to or from a compact internal format. Dates may be stored in items as numbers, and may be printed in any of several different formats in a listing, by using the 'D' (Date conversion) code. This allows you to store dates with fewer bytes of disk space and to perform mathematical calculations involving stored dates.

The internal format of any date is the integer number of days between that date and the zero date, December 31, 1967. Dates before 12/31/67 are stored as negative numbers; dates after 12/31/67 are stored as positive numbers. The following table of dates compares both internal and external (listing) format:

INTERNAL FORMAT	EXTERNAL (LISTING) FORMAT
-100	22 SEP 1967
-10	21 DEC 1967
-1	30 DEC 1967
0	31 DEC 1967
1	01 JAN 1968
10	10 JAN 1968
100	09 APR 1968
1000	26 SEP 1970
10000	18 MAY 1995

Some external formats which may be specified for the 'D' code are given below:

D CODE	INTERNAL FORMAT	EXTERNAL (LISTING) FORMAT
D	5744	22 SEP 1983
D/	5744	09/22/1983
D-	5744	09-22-1983
DO	5744	22 SEP
DO/	5744	09/22
D2*	5744	09*22*83
D%1	DUE%5744	DUE%22 SEP 1983
D%1/	BAL%5744	BAL%9/22/1983
D#1-	CHG#5744	CHG#09-22-1983
DO$1	SALE$5744	SALES$22 SEP
DO	SALE$5744	SALE$5744
D4-	5744	09-22-1983
DY	5744	1983 (4-digit year)
D2Y	5744	83 (2-digit year)
DQ	5744	3 (3rd quarter)
DD	5744	22 (22nd day of month)
DM	5744	9 (9th month)
DMA	5744	SEPTEMBER
DJ	5744	265 (265th day of year)
DW	5744	4 (4th day of week)
DWA	5744	THURSDAY
DI	9/22/83	5744 (reverse conversion)
DI]D2	9/22/82	22 SEP 83 (multivalue conversion)

The 'D' code has the following general form:

D{n}{*m}{s}

where:

D is the code name

n is an optional single digit number which specifies the number of digits to occur in the year on output. If 'n' is 0, no year will appear in the date. n = 0, 1, 2, 3, or 4 is valid; if 'n' is omitted, n = 4 is assumed.

* Stands for any single non-numeric character which specifies the delimiter between fields for group extraction. (* may not be a system delimiter.)

m is a single digit number that must accompany * (if * is specified). m specifies the number of fields to skip for group extraction.

s stands for either any single non-numeric character that may be specified to separate the day, month and year on output, or a special sub-code, either M, D, Y or Q. If 's' is specified, the date will be in the format 12-31-1967. If 's' is not specified, the date will be in the format 31 DEC 1967. If a sub-code is specified, then only the number of the quarter, year, month or day is listed.

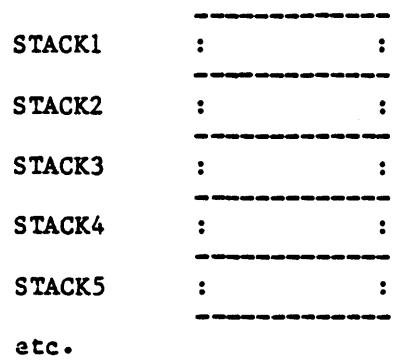
Note the use of the '*m' option which will perform a Group Extration before applying the date conversion. (See GROUP EXTRACTION CODE 'G'.) A special note for using the 'D' code on input: if no year is specified, the current year will be used. If only two digits are specified for the year, then years entered as 30 to 99 will be stored as 1930 through 1999, and years entered as 00 to 29 will be stored as 2000 through 2029. Additional formats for the D code may be used as shown in the following Date Conversion chart.

D Code	Conversion Format
D	DD MMM YYYY
D-	DD-MMM-YYYY
D0 (D zero)	DD MMM
D0- (D zero -)	MM-DD
DD	Day of month
DI	Internal
DJ	Julian
DM	Month (numeric)
DMA	Month (alpha)
DQ	Quarter (numeric)
DW	Day of week (numeric)
DWA	Day of week (alpha)
DY	Four digit year

6.7.4 THE FUNCTION CODE (F)

The function code 'F' is used to perform mathematical operations on the attribute values of an item, or to manipulate strings.

All operations specified by an 'F' code operate on the last two entries in a pushdown stack. This pushdown stack may be visualized as follows:



STACK1 is the top position in the stack, STACK2 is the next position, etc. As a value is pushed onto the stack, it is pushed into position STACK1; the original value of STACK1 is pushed down to STACK2; and so on.

An 'F' code is comprised of any number of operands or operators in reverse Polish format separated by semicolons. When an operand specification (a numeric attribute number or constant, or a string) is encountered, the value is "pushed" onto the top of the stack. When an operator is encountered, the specified operation is carried out on the top one or two entries in the stack, depending upon the operator. When the entire 'F' code has been processed, the value printed is the value in the top of the stack.

The general form of the 'F' code is as follows:

```
Felement;element;element...
```

An "element" may be any of the following: a numeric AMC specifying an attribute value to be pushed onto the stack (optionally followed by an "R" to specify that the first value or subvalue of an attribute is to be used repeatedly when using it against a multivalued value or subvalue).

An element may also be of the form 'Cn' where 'n' is a numeric constant to be pushed onto the stack; a 'D' which specifies that the current date is to be pushed onto the stack; a 'T' which specifies that the current time is to be pushed onto the stack; a special 2-character operand; or an operator which specifies an operation to be performed on the top two entries in the stack. The operators are listed in Table 6-4.

Table 6-4. ´F´ Code Operators

Operator	Operation
*{n}	Multiplication of the top two entries in the stack. If the optional "n" is used, the result is "descaled" by n, that is, it is divided by 10**n.
/	Division of STACK2 by STACK1, result to STACK1.
R	Same as "/" but remainder is returned to top of stack (instead of quotient).
+	Addition of the top two entries in the stack, result to STACK1.
-	Subtraction of STACK1 from STACK2, result to STACK1.
:	Concatenate; the string value from STACK1 is concatenated onto the end of the string value from STACK2.
[]	Substring; a subset of the string value from STACK3 is extracted using STACK2 as the starting character position, and STACK1 as the number of characters to extract; the result is placed on top of the stack. This is equivalent to the BASIC [m,n] operator, where "m" is in STACK2 and ´n´ in STACK1.
S	A total sum of all previous computations is placed at the top of the stack.
_	(Underline.) Exchanges top two positions in stack.
P	Pushes the top stack value back onto the stack; that is, it duplicates the top stack value.
(...)	Conversion operator; a standard conversion operator such as D (date), G (group), etc. may be specified within parentheses and will operate on the top stack value; the result will replace the original top stack value.

The following operators operate on the top 2 stack entries, and a result of zero or one is placed on the top of the stack, depending on whether the condition is not or is satisfied.

=	"Equal" relational operator.
<	"Less than" relational operator.
>	"Greater than" relational operator.
#	"Not equal" relational operator.
[	"Equal to or less than" relational operator.
]	"Equal to or greater than" relational operator.

The relational operators compare STACK1 to STACK2; after the operation, STACK1 will contain either a 1 or 0, depending upon whether the result is true or false, respectively (e.g., if the 'F' code were F;C3;C3;= then STACK 1 would contain a 1). This comparison is often followed by a multiplication of the new contents of STACK 1, (1 or 0), and the contents of STACK 2 (previously STACK 3). If the comparison was true, the result is a 1, so after the multiplication the contents of STACK 2 moves up to STACK 1 and is available for further processing. If the result was false, the result was 0 and the multiplication result would leave a zero in STACK 1. Examples of F are:

F2;3;4;*;+ is equivalent to:

(attribute3 * attribute4) + attribute2

F23;24;*;C100;+;P;C0;<;* is equivalent to:

(attribute23 * attribute24) + 100 ;
If this result is < zero, final result is zero; else the above value is returned as the result. (The above value is generated, and is repeated in the stack via the P operator; it is then compared to zero, which gives a result of 0 or 1 depending on whether it was less than zero or not; this is muliplied by the original value, giving a zero or the original value.)

F3;4;*;6R;+;S is equivalent to:

(attribute3 * attribute4) + attribute6
If attributes 3 and 4 are multivalued, and 6 is not, the single value in attribute 6 will be used repeatedly in the addition (if the "R" is not present, it will be used only once, and zeros will be used for other computations); a sum of such computations for all multivalues in attribute3 or attribute4 (whichever has the greater number of multivalues) is returned.

F3;" ";:;4;:;5;C1;C10;[];: is equivalent to:

attribute3:" ":attribute4:attribute5[1,10]
That is, the value from attribute 3 is concatenated to that from attribute 4, with a space between them; the first through the 10th characters from attribute 5 are then concatenated to the end of that result.

6.7.4.1 Special 'F' Code Operands

The 'F' code operands may be multivalued, may contain conversion specifications, or may be a special 2-character operand specifying one of several counters. Different interpretations are given to an 'F' correlative versus an 'F' conversion for an attribute with a TOTAL modifier.

Multivalued F code operands. When arithmetic operations are performed on two multivalued lists (vectors), the answer will also be multivalued and will have as many values as the longer of the two lists. Zeros will be substituted for the null values in the shorter list. For example, suppose the attribute with AMC=10 had a value of "5]10]15" and the attribute with AMC=15 had a value of "20]30]40]50"; if the correlative F;10;15;+ were processed, the result in STACK1 would be "25]40]55]50". If a single valued attribute is to be repetitively added (or subtracted, etc.) with a multivalued attribute, then the single letter R should immediately follow the AMC in the 'F' code (e.g., F;10;25R;+).

Any conversion may be specified in the body of a Function correlative. The conversion specification must be enclosed by parentheses. Examples of F correlation with conversions:

```
F;10;11;(TDICT SALES;X;3;3);*
```

Places the data from attribute 10 in the stack; translates the data from attribute 11 through the dictionary of file-name SALES and stacks it; then multiplies the two numbers together.

```
F;D;(DY);3;(DY);-
```

Computes the difference in years between the current date and the date in attribute 3.

```
F;1;(ML#10);2;:
```

Concatenates the data in attribute 2 with the result of applying the format string "L#10" to attribute 1.

Special 2-character operands may be used as 'F' code elements as listed below:

Operand	Description
NI	Current item counter (number of items listed or selected).
ND	Number of Detail lines since last BREAK on a Break line. On a detail line it has a value of 1. On a grand-total line, it equals the item counter. (Used to generate averages in conjunction with control breaks.)
NV	Current multivalue counter for columnar listing only.
NS	Current submultivalue counter for columnar listing only.
NB	Current Break level number; 1 = lowest level break; this has a value of 255 on the grand-total line and a value of zero at detail time. The lowest level control-break, the one on the right in the sentence, will have a value of 1.
LPV	The Load Previous Value operand (see following section) will load the result of the last conversion onto the stack.

For example:

```
F;ND;3;/
```

On every detail line, this returns the value from attribute 3; on every Break line (including the grand-total line), the average value of the data in attribute 3 is returned. (This must be specified as a conversion in line 7.)

The Function code operates in two different fashions on an attribute with a TOTAL modifier, depending on whether it is specified as a correlative or as a conversion. As a correlative, the function is applied before the accumulation of the total, and is ignored on the Break data line; therefore, the total of the functioned value is computed. As a conversion, the function is ignored on detail lines, and is applied only on the Break data line and other subtotal fields in the output. Therefore, the function of other totaled values is obtained. If the function is specified as a conversion, the numeric operand (AMC) in the Function code must correspond to an attribute that is being totaled within the statement. If such an attribute does not exist, a value of zero is returned. Note that the numeric operators may be dummy AMCs in that they may reference other attributes within the statement that have function correlatives.

6.7.4.2 The Load Previous Value (LPV) Operator

The function processor commences operation with no prior data. If attribute 8 commences with an F-correlative, there is no prior data set up by any processor in the system. Entering attribute 7 there is the result of attribute 8 or at least the data retrieved from the item according to the specification in attribute 2 of the data definition item. It is possible to load this data into the function correlative stack using the LPV instruction. Since a conversion may call a function correlative, you may also load the last result of a series of conversions within a given attribute definition line into a function which follows the conversion in the line. For instance:

```
    DATA DEFINITION ITEM
001 A                              Data definition item mark.
002 3                              Specifies data attribute 3.
    .
    .
007 F;LPV;"100";/                  Will divide the result of
                                   attribute 8 by 100.
008 F;2;3;*                        Contents of data attribute 2
                                   times the contents of data
                                   attribute 3.
```

If this data definition item is totaled, the total generated will be loaded into attribute 7 and divided by 100 prior to output on the break line.

```
002 5                              Data attribute 5.
    .
    .
008 G*1]MR%8]F;LPV;"52";R;"*C";:]TFILE;C;;3
```

This has rather less motivation, since it is equivalent to:

```
008 F;5(G*1]MR%8);"52";R;"*C";:;(TFILE;C;;3)
```

The LPV should not be used as the first operator in a F-correlative, because it has the effect of loading the contents of the temporary data area into the stack. If the LPV is used at other points in an F-correlative, strange things will happen.

The LPV operator is available for use in A-correlatives.

6.7.4.3 Summary of F Code Stack Operations

The following operands are pushed onto the stack, and all other stack elements are pushed down one level.

Operand	Action
n{R}	The attribute value for the corresponding attribute number (where 'n' is a decimal number) is stacked. The optional R specifies that the value is to be repeated if it is a single value and there are multiple values in other attributes which are part of the stack operation.
Cn	The integer constant 'n' (where 'n' is a decimal number) is stacked.
"String"	The literal string enclosed in double quotes is stacked.
'String'	The literal string enclosed in single quotes is stacked.
D	The current date (in internal format) is stacked.
T	The current time (in internal format) is stacked.
P	The top stack element is pushed back onto the stack.
NI	The current item counter is stacked.
ND	The number of detail lines since the last control break is stacked. (This number is 1 on detail lines, and is the same as the item counter on grand-total lines.)
NS	The current submultivalue counter is stacked.
NB	The current control-break number is stacked.

The following operator pops three entries off the stack, computes a result, and pushes it onto the top of the stack.

[]	The substring from the string value in the first stack element is extracted, starting from the character position defined in the second element, and ending at the character position defined in the third element. For example, the operation F4;C5;C8;[] extracts the values in the 5th through 12th character positions from the string in attribute 4.

The following operators pop the top two operands from the stack, compute a result, and push the result onto the stack. All other stack elements are popped up one level.

+	The top two elements are added together and the sum is stacked.
-	The first stack element is subtracted from the second and the difference is stacked.
*{N}	The top two elements are multiplied and the result is stacked. If n is specified, the result is divided by 10**n before it is stacked.
/	The second stack element is divided by the top stack element and the dividend is stacked.
R	The second stack element is divided by the top stack element and the remainder is stacked.
:	The top stack element is concatenated onto the end of the second stack element and the resultant string is stacked.

The following relational operators compare the two top elements of the stack, pop them both off the stack and then push a 1 (TRUE) or 0 (FALSE) onto the stack.

=	The top two elements of the stack are compared, a 1 is stacked if they are equal, a 0 is stacked if they are not equal.
#	Stacks a 1 if the top two stack elements are unequal; stacks a 0 if they are equal.
>	Stacks a 1 if the top stack element is greater than the second stack element, stacks 0 otherwise.
<	Stacks a 1 if the top stack element is less than the second, a 0 otherwise.
[	Stacks 1 if the top element is greater than or equal to the second element, 0 otherwise.
]	Stacks 1 if the top element is less than or equal to the second element, 0 otherwise.

The following operators function on just the top one or two stack entries and have no effect on the rest of the stack.

S	Sums the multiple values (if any) of the top stack element.
_	(Underline.) Exchanges the first and second stack elements.

6.7.5 THE GROUP EXTRACTION CODE 'G'

The group extraction code 'G' is used to extract one or more fields from an attribute value.

If an attribute value consists of multiple fields separated by a delimiter, the 'G' code can be used to extract one or more contiguous fields. The general form of the 'G' (Group Extraction) code is:

G{m}*n

where:

G is the capital letter G (the Group Extraction code).

m optionally specifies the number of fields to skip. If m is not specified, zero is assumed and no fields are skipped.

* represents any single non-numeric character, except a minus sign (-), which is the field separator, or any system delimiter.

n is a decimal number which is the number of contiguous fields to be extracted.

Sample usage of 'G' code:

G Code	Attribute Value	Output Value
G/1	04/02/1956	04
G1/1	04/02/1956	02
G2/1	04/02/1956	1956
G/2	04/02/1956	04/02
G1/2	04/02/1956	02/1956
G0*1	123*888*444	123
G1*2	123*888*444	888*444
G2*1	123*888*444	444
G3*1	123*888*444	(null)
G1*1	*WRITTEN 21 DEC 1977	WRITTEN 21 DEC 1977

6.7.6 THE LENGTH AND RANGE CODES 'L', 'R'

The length code 'L' and the range code 'R' place restrictions on output based on the length and the range of a value.

The Length code places length constraints on what kind of data will be returned, or returns the length of the data. The user may select a fixed length Ln (number of characters) necessary to meet output criteria, or a length range Ln,m.

The L code may be used as follows:

L0	Returns the length of the data string submitted to the length processor.
Ln	Returns the data value if it is less than or equal to n characters long. If the data does not meet the criteria then null is returned.
Ln,m	Returns the data value if it is greater than or equal to n characters long or less than or equal to m, where n<m.

The R code returns data values which fall within the specified ranges, where multiple ranges are allowed.

The format of the R code is:

Rn,m{;n,m...}	Returns the data value if it falls within the range of n to m. Multiples of ranges may be specified as shown, in ascending order. Note that when negative range sets are used, the more negative number must be stated first.

Note that any delimiter (except system delimiters) may be used to separate the numbers in a range. However, for the sake of clarity, a minus sign should not be used, as the minus sign may refer to the number(s) in the range.

In all cases, if the range(s) specifications are not met, null is returned.

6.7.7 THE MASK CHARACTER CODE 'MC'

The mask character code 'MC' allows the user to change attribute data to upper or lowercase or to select certain classes of characters.

The following is a list of the forms of the MC code and their function:

MCU — Converts data to uppercase. Will change all lowercase letters to uppercase; has no effect on uppercase letters or non-alphabetic characters.

MCL — Converts data to lowercase. Will change all uppercase letters to lowercase; has no effect on lowercase letters or non-alphabetic characters.

MCT — Converts uppercase characters to lowercase, starting with the second character in each word.

MCP — Converts all nonprintable characters (X'00' - '1F', '7F' - 'FF') to periods.

MCA — Extracts and prints all alphabetic characters, either uppercase or lowercase; non-alphabetic characters will be deleted from the data.

MCN — Extracts all numeric characters (0-9) from the data; deletes all other characters.

MC/A — Extracts all non-alphabetic characters from the data; deletes all alphabetic characters.

MC/N — Extracts all non-numeric characters; deletes all numeric characters.

Sample usage of the MC (Mask Character) code:

Value	MC Code	Output Value
JOHN SMITH 1234	MCL	john smith 1234
John Smith 1234	MCU	JOHN SMITH 1234
John Smith 1234	MCA	JohnSmith
John Smith 1234	MCN	1234
572-08-2394	MCN	572082394
(714) 552-4275	MCN	7145524275
abc123$%XYZ	MC/A	123$%
abc123$%XYZ	MC/N	abc$%XYZ

6.7.8 THE MASK LEFT AND MASK RIGHT CODES ´ML´, ´MR´

The mask left ´ML´ and mask right ´MR´ codes allow special processing and formatting for numbers or dollar amounts. For best results, these codes should be used in the conversion attribute (attribute 7) of the item dictionary.

The ACCESS MR and ML codes function exactly like the ´R´ and ´L´ format strings in BASIC. The general form of the MR and ML codes is:

```
ML
MR{n{m}}{Z}{,}{C/D/M/E/N}{$}{(format-mask)}
```

where:

M is the code name. MR specifies the number to be right justified; ML specifies left justification. Note that the L or R justification specification in attribute 9 of the dictionary item will override the ML or MR except when a format-mask is also used.

n is a single decimal digit (0-9) which specifies the number of digits to be printed to the right of the decimal point. If ´n´ is not specified, 0 is assumed. If 0 is assumed or specified, no decimal point will be printed.

m is a single digit number (0-9) which specifies that the number on file is to be ´descaled´ (divided) by that power of ten. (That is, if m=2, the number is divided by 100, if m=3, the number is divided by 1000, and so on.) The number ´m´ is the number of implied digits to the right of the decimal point for the number as it is stored in the file. If m>n, then the number will be rounded off, either up or down, to ´n´ digits.

Z is the optional zero-suppress parameter. If ´Z´ is specified, the 0 (zero) numbers will be printed as blanks.

, specifies insertion of commas every three digits to the left of the decimal point.

C causes negative values to be followed by the letters CR.

D causes positive values to be followed by the letters DB.

M causes negative numbers to be followed by a minus sign (-).

E causes negative numbers to be enclosed inside angle brackets (< and >).

N causes the minus sign on negative numbers to be suppressed.

$ appends a dollar sign ($) to the number before justification.

The format mask specification, which is enclosed in parentheses, consists of format codes and literal data. A format code is one of the characters #, *, or %, optionally followed by a number to indicate that number of repetitions of the characters.

Format Mask	
#n	specifies data to be justified in a field of 'n' blanks.
*n	specifies data to be justified in a field of 'n' asterisks (*).
%n	specifies justification in a field of 'n' zeros (0).

Any amount of alphabetic data may also be specified inside the parentheses in the format mask specification. The data will be printed exactly as specified in the format mask, with the number being processed appearing either right or left justified in the place of the #'s, *'s, or %'s. Any #'s, *'s, or %'s not overwritten by the number will appear either to the left or right of the number as specified.

Note the $ option apends a dollar sign to the number and then justifies the number, so the dollar sign will always appear just before the first digit of the number on output.

Sample usage of the MR and ML codes:

Data	Conversion	Result
1234	MR2	12.34
1234	MR	1234
1234	MR(%10)	0000001234
1234	MR(*10)	******1234
12345678	MR2,E	123,456.78
-12345678	MR2,E	<123,456.78>
-12345678	MR2,C%	$123,456.78CR
572082394	MR24	57208.24 (Note rounding)
572082394	MR2,$(#20)	$5,720,823.94
572082394	MR2,($#20)	$ 5,720,823.94
572082394	MR2,$(*20)	*******$5,720,823.94
572082394	ML(###-##-####)	572-08-2394
572082394	MR(#3-#2-#4)	572-08-2394
572082394	ML(#3-#4 EXT.#2)	572-0823 EXT.94

6.7.9 THE MASK TIME CODE 'MT'

The mask time code 'MT' provides the facility for converting time to or from a compact internal format suitable for arithmetic processing. Additional date formats are also described in this section.

The internal time format is the number of seconds from midnight. The external time is 24 hour military format (e.g., 23:25:59) or 12 hour format (e.g., 11:25:59PM). The general form of the MT code is as follows:

MT{H}{S}

where:

MT is the Mask Time code specification

H is the capital letter H which optionally specifies 12 hour format. If 'H' is omitted, 24 hour (military) format is assumed.

S is the capital letter S which optionally specifies seconds on output. If 'S' is omitted, seconds are not listed on output.

When codes MTH and MTHS are used, 12 hour external format is specified. For input conversion, then, the time is entered with AM or PM immediately following the numeric time (AM is optional). On output, AM or PM is always printed immediately following the numeric time.

NOTE: 12:00 AM is considered midnight, and 12:00 PM is considered noon. AM and PM will be ignored on input if code MT is specified. Illegal values are converted to null on input. Sample usage of MT:

MT CODE	INPUT VALUE	STORED VALUE	OUTPUT VALUE
MT	12:	43200	12:00
MTH	12AM	0	12:00AM
MTS	12AM	0	00:00:00
MTHS	12AM	0	12:00:00AM
MT	12:15AM	900	00:15
MTH	12:15AM	900	12:15AM
MT	1AM	3600	01:00
MTH	1AM	3600	01:00AM
MT	6AM	21600	06:00
MTH	6AM	21600	06:00AM
MT	1PM	46800	13:00
MTH	1PM	46800	01:00PM
MT	13:	46800	13:00
MTH	13:	46800	01:00PM
MT	XYZ	0	00:00:00

6.7.10 THE MASK HEXADECIMAL CODE 'MX'

The mask hexadecimal code 'MX' is used to convert strings to or from their hexadecimal equivalents.

The MX code specifies that character strings are to be converted, one character (byte) at a time, into their hexadecimal (base sixteen) representations. Each character will be converted to a 2-digit (one byte) hexadecimal number.

This feature is useful in finding non-printable characters in data strings. The general form of the MX (Mask Hexadecimal) code is:

```
MX
```

The mask decimal code may be combined with the mask character code as shown below:

Code	Action
MCDX	Converts decimal data value to its hexadecimal equivalent.
MCXD	Converts hexadecimal data value to its decimal equivalent.

Sample usage of the MX (Mask Hexadecimal) code:

INPUT VALUE	CONVERTED VALUE
ABC	414243
JOHN	4A4F484E
john	6A6F686E
HI THERE	4849205448455245

6.7.11 THE PATTERN AND SUBSTITUTE CODES 'P', 'S'

The pattern match 'P' and substitute 'S' codes place restrictions on output based on P(attern) matching and S(ubstitution) of the value of an attribute. The Pattern Match code returns data values which match the specified pattern. If the data does not match the specified pattern, then null is returned. The general form of the Pattern Match code is:

```
P(matchstring){;(matchstring)}
```

The Pattern Match consists of any combination of the following:

P(nN)	An integer number followed by the letter 'N', which tests for that number of numeric characters.
P(nA)	An integer number followed by the letter 'A', which tests for that number of alpha characters.
P(nX)	An integer number followed by the letter 'X', which tests for that number of alphanumeric characters.
P('literal')	A literal string, which tests for that literal string.

For example, if the user wished to have only social security numbers returned, the following Pattern Matching code could be used:

```
P(3N-2N-4N);(9N)
```

This specifies that only those data values comprised of 9 numeric or a string match of 3 numeric, a hyphen, 2 numeric, a hyphen, 4 numeric are returned. The general form of the Substitute code is:

```
S;attribute-number'text'
```

The Substitution code substitutes the data value of the referenced attribute number if the current data value is not null or zero. If the data value is null or zero, it will be substituted with a literal string specified by the user. For example:

```
S;4;'XXX'
```

This specifies that if the data value is not equal to zero or null then it will be replaced by the contents of attribute 4. If it is equal to zero or null, it will be replaced by the string 'XXX'. An additional feature of the Substitution code is that if it is used in junction with the Function Processor, it may serve to test for null or zero and take different actions according to what kind of data it encounters. For example:

```
F1(S;*;'NORMAL VALUE')
```

This specifies that if attribute 1 is null or zero, the string "NORMAL VALUE" will be used, otherwise the contents of attribute 1 will be used. Note that the S code should only be used in attribute 8.

6.7.12 THE TEXT EXTRACTION CODE 'T'

The text extraction code 'T' is used to extract a fixed number of contiguous characters from an attribute value. This is useful for fixed field data, or for truncating data when you wish to prevent folding. Note that the attribute value that contains the characters to be extracted may not exceed 500 bytes.

An attribute with a T code has no value of its own but gets it value by extracting characters from the value of another attribute.

Used by itself, it will extract characters from the attribute of which it is a synonym.

Used in an F correlative, the T code can extract characters from any attribute in the item.

The general form of the 'T' (Text Extraction) code is:

T{m,}n

where:

T is the code name

m is the optional starting column number

, is the separator necessary between 'm' and 'n' if 'm' is specified

n is the number of characters to extract

If the form 'Tn' is specified, 'n' characters will be extracted, either from the left or the right, depending upon the attribute definition item's V/TYP (line 009). If the V/TYP is 'L', the 'Tn' form of the Text code will extract the first 'n' characters of the attribute value. If the V/TYP is 'R', the 'Tn' code will extract the 'n' rightmost characters of the attribute value.

If the form 'Tm,n is specified, then 'n' characters, starting at column 'm', will be extracted. The 'Tm,n' form always counts columns and extracts characters from left to right. This means that the extraction will be from left to right, regardless of the attribute definition item's V/TYP.

Examples of T code:

T CODE	ATTRIBUTE VALUE	JUSTIFICATION	VALUE OUTPUT
T3	ABCDEF	L	ABC
T3	ABCDEF	R	DEF
T3,5	HELLO OUT THERE	L	LLO O
T3,5	HELLO OUT THERE	R	LLO O
T1,11	THIS IS A LONG STRING	L	THIS IS A L
T2,9	*12 DEC 77	L	12 DEC 77
T4,7	123SMITH CR	L	SMITHbb *
T4,7	848JOHNSONDB	L	JOHNSON
T3	123SMITH CR	L	123
T3	848JOHNSONDB	L	848
T2	123SMITH CR	R	CR
T2	848JOHNSONDB	R	DB

(* The bb represents two blanks)

Another example of the T code:

```
1                   ZIP
001 A               001 A
002 1               002 1
003 CITY, ST ZIP    003 ZIP
008                 008 T5
009 L               009 R
010 30              010 5
```

A listing of a file with the above attributes would give:

```
CITY, ST ZIP                  ZIP

DALLAS, TX 75230              75230
BARTOW, FL 33830              33830
LOS ANGELES, CA 90061         90061
```

Thus, using the T code, you could store the city, state and zip in the normal fashion and still be able to sort it by ZIP.

It is also possible to select or sort on state in the above example using the T code if you combine it with the G and F codes.

```
F;1(G1,1);(T2,2)
```

The G code extracts " CA 90061" and the T code extracts two characters from the group starting with the second (blank is first) to give CA.

6.7.13 THE TRANSLATE FILE CODE 'Tfile'

The translate file code 'Tfile' code provides a facility for converting a value by translating through a file. The value to be translated is used as an item-id for retrieving an item from the defined translation file. The translation value is retrieved from the specified attribute of the item. The general form of the Tfile (Translate) code is:

```
T{DICT}file;c{n};i-amc;o-amc{;b-amc}
```

where:

T — is the code name.

file — is the name of the file through which translation takes place, The file name may be preceded by "DICT" to indicate a dictionary.

c — is the translate subcode, which must be one of the following:

- V Conversion item must exist on file, and the specified attribute must have a value. Aborts with an error message if translation is impossible.
- C Convert if possible; use original value if item in translate file does not exist or has null conversion attribute.
- I Input verify only (functions like 'V' for input and like 'C' for output).
- O Output verify only (functions like 'C' for input and like 'V' for output).
- X Convert if possible; otherwise return a null value.

n — is an optional value mark count specification. If the c element is followed by a number, the translate will return only the value in VMC n, instead of the complete collection of values concatenated together with blanks. Subvalues will be returned with included blanks.

i-amc — is the decimal attribute number for input conversion (in BASIC or BATCH). The input value is used as an item-id in the specified file, and the translated value is retrieved from the attribute specified by the i-amc. (If the i-amc is omitted, no input translation takes place.)

o-amc — is the attribute mark count for output translation. When ACCESS creates a listing, the attribute values will be looked up in the specified file, and the attribute specified by the o-amc will be listed instead of the original value.

b-amc — if specified, will be used instead of o-amc during the listing of break-on and total lines.

Sample usage of the Tfile (Translate) code:

Item CARDS in DICT MUNCH is:

```
001  S
002  4
003  CREDIT CARDS
004
005
006
007  TCARD-FILE;C;;1
008
009  L
010  20
```

Data section of MUNCH file is:

```
Item-id:    COCOS              CARLS-JR              MEYERHOFS
    001:    HAMBURGERS         HAMBURGERS            SANDWICHES
    002:    MAC ARTHUR BLVD.   BRISTOL STREET        S.COAST VILLAGE
    003:    5583233            9792231               8830002
    004:    MC]V               NONE                  MC]V]BA
```

Data section of CARD-FILE is:

```
Item-id:    MC                 V                     BA
    001:    MASTER CHARGE      VISA                  BANKAMERICARD
```

```
>LIST MUNCH "COCOS""CARLS-JR""MEYERHOFS" CARDS HDR-SUPP  [CR]

COCOS       MASTER CHARGE
            VISA
CARLS-JR    NONE
MEYERHOFS   MASTER CHARGE
            VISA
            BANKAMERICARD
```

6.7.14 THE USER-DEFINED CONVERSION CODE "U"

The "U" code specifies an entry point into a user-written piece of software. The general form of the "U" (user-defined) code is:

```
Unxxx
```

where:

U — is the code name.

n — is the entry point number.

xxx — is the hexadecimal FID (Frame ID) of the frame containing the user's assembly code.

WARNING: Do not use the U code unless you fully understand its action at the assembly-code level.

support processors 7

This is the final section of the PICK Operator Guide. Last, but not least, the subjects covered are utility verbs and processors, system security levels and verbs which can be used to obtain statistics reports.

7.1 UTILITY PROCESSORS

7.1.1 CT PROC

This PROC is invoked by typing:

```
CT file-name item-list {(options)}
```

The item(s) specified will be copied to the terminal. Options recognized by the copy verb may be added.

7.1.2 LISTACC PROC

The LISTACC PROC is invoked by typing:

```
LISTACC {account-name}...
```

This PROC lists accounting data for the account-name(s) specified. If no account-name(s) are specified, accounting data for all users will be listed.

7.1.3 LISTCONN PROC

This LISTCONN PROC is invoked by typing:

```
LISTCONN {(P)}
```

This PROC sorts all connectives in any dictionary and lists them on the terminal (or the line printer if (P) is specified).

7.1.4 LISTDICT PROC

The LISTDICT PROC sorts all attribute definition items in any dictionary and lists them on the terminal (or the line printer if (P) is specified). If file-name is not specified, Master Dictionary items are listed. The general form is:

```
LISTDICT {file-name} {(P)}
```

7.1.5 LISTFILES PROC

The LISTFILES PROC sorts all file or file synonym definition items in any dictionary and lists them on the terminal (or on the line printer if (P) is specified). If file-name is not specified, Master Dictionary files are listed. The general form is:

```
LISTFILES {file-name} {(P)}
```

7.1.6 LISTPROCS PROC

The LISTPROCS PROC sorts all PROC's in any file or dictionary and lists them along with a brief abstract on the terminal (or the line printer if (P) is specified). The general form is:

```
LISTPROCS {file-name} {(P)}
```

7.1.7 LISTU PROC

The LISTU PROC lists the account name of the users currently active on the system, along with their logon time and channel number. The general form is:

```
LISTU  or  LISTUSERS
```

7.1.8 LISTVERBS PROC

The LISTVERBS PROC sorts all verbs (not PROC's) in any dictionary and lists them on the terminal (or on the line printer if (P) is specified). The general form is:

```
LISTVERBS {file-name} {(P)}
```

7.2 SYSTEM SECURITY

Security codes may optionally be placed in the L/RET and L/UPD attributes of a dictionary item to restrict access and update. At logon time, each user is assigned the set of security codes which are in his user identification item. During the session, whenever an L/RET or L/UPD code is encountered, a search is made of the user assigned codes for a match; if no match is found, the user is denied access. A security code may consist of any combination of legal ACSII characters.

7.2.1 L/RET AND L/UPD

Both file definition ("D" code) and synonym file definition ("Q" code) items have L/RET (retrieval locks) and L/UPD (update locks) attributes. When these attributes have values stored, they are known as security codes. Although there is no prohibition against multiple values for these attributes, only the first attribute value is used for matching against the user assigned codes. Since each file may be individually locked for both update and retrieval, a user must be assigned multiple codes to that set of data he is allowed to access. Using this feature, a complex "mask" can be constructed for each user, giving each user a different subset of files which he may access.

Security at the file level is invoked at the processor level. The following processors are assumed to be updating processors and, therefore, require a match on the L/UPD atttribute in the file definition item: COPY, EDIT, BASIC if writing a file, RUN and the Assembler. Other processors are assumed to be retrieval processors and require a match on the L/RET attribute in the file definition item. BASIC requires a match against L/RET code when the file is opened, and a match against the L/UPD if data is changed in the file. Failure to match one of the user security codes with either the L/RET or L/UPD attribute value will generate the following message and return control to TCL:

```
[210] FILE xxx IS ACCESS PROTECTED
```

7.2.2 USER ASSIGNED CODES

Each user identification item in the System Dictionary (see Section 5.2.1, USER IDENTIFICATION ITEMS) contains the list of security codes assigned for that particular user. These codes are values for the attributes L/RET and L/UPD. The lock code in the user-identification item and the lock code in the file being verified must match.

Security codes may be assigned initially when an account is created via the use of the CREATE-ACCOUNT (PROC Security codes may be added or deleted by updating the appropriate security codes); however, updates to the user identification item should only be performed when no one else is logged onto the system.

7.2.3 SECURITY CODE COMPARISON

Security codes are verified comparing the value in the file dictionary against the corresponding string of values in the user identification item. Characters are compared from left to right. An equal (verified) compare occurs when the value in the file dictionary is exhausted and all characters match up to that point. Sample security code comparisons:

FILE DICTIONARY CODE	USER IDENTIFICATION CODE	RESULT
123	123	Match
12	123	Match
123	12	No Match
XYZ	XYZ5	Match
AQZ	AQ	No Match

When referencing a file using a Q synonym, a security code match is made at all levels (i.e., SYSTEM, MD and file dictionary) and, therefore, a correspondence must be maintained at all levels in order to process the Q synonym files. Since the user identification item for the account containing the primary file is verified for security codes, the user referencing the Q synonym must have a code defined in this user identification item which will verify with the first code in the equated account's user identification item. Therefore, in a user identification item, only the first code is used to protect the account from Q synonym accesses, while all the codes in the item are assigned to the user when he logs on.

7.3 FILE STATISTICS REPORT

The File Statistics Report is a valuable tool for data base management. This Report is automatically generated by running a FILE-SAVE, or may be generated at any time by using the LIST-FILE-STATS processor.

The FILE-SAVE process creates one item in the STAT-FILE for each D-pointer saved on the FILE-SAVE tape or cartridge disk. A listing of the STAT-FILE is created at the end of every FILE-SAVE. The same listing can be generated from TCL by the LIST-FILE-STATS PROC.

The statistics report adds data security by providing a list of file Base, Modulo and Separation parameters and by recording the order of files on a FILE-SAVE tape or cartridge disk.

The report is broken down by account with a line of information generated for each line in the account that includes:

1. Total and average item size.

2. Total and average number of items per group.

3. Utilization of file-space.

4. Actual data stored and "pad" space used in the file.

A total line is generated for each account showing the total:

- items
- bytes (characters)
- frames (includes linked)
- group format errors

Creation of the STAT-FILE dictionary and data areas is part of the SYS-GEN procedure. STAT-FILE is contained on the System Programmer (SYSPROG) Account. As it is normally updated from this account, there is no need for STAT-FILE on any other account.

Alternatively, the file may be created via the following:

```
CREATE-FILE STAT-FILE 1,3 29,1  [CR]
```

When a FILE-SAVE is started, the STAT-FILE data area is cleared and the current file statistics information is written into the data area.

The STAT-FILE data area will also be empty after a file restore is done, because attribute 1 of the file definition is a DY. This is desirable since the statistics are no longer applicable.

It is helpful to make synonym accounts in the System Dictionary Q-pointers so that there is only one D-pointer for each account. This way, the data for the account will be saved under the one account name that is a D item in SYSTEM.

The item-id in the STAT-FILE is of the form:

```
t:n
```

where "t" is the tape reel or cartridge number where the file was dumped (this will be 0 if the SAVE was run without dumping data to the tape or cartridge disk) and "n" is the FILE-NUMBER. This file-number is used in the selective restoration of files using "SEL-RESTORE".

Note that files in the listing that have a SIZE field of zero are synonym D-pointer files; that is, a previously found D-pointer caused the data in the file to be dumped.

The NAME field of the items in the STAT-FILE contains data in the form:

```
account-name*dict-name*data-name
```

where one, two, or all three of the fields may be present depending on whether the file is an account, a dictionary, or a data file.

7.4 FILE CHANGE VERIFICATION (CHECK-SUM)

The CHECK-SUM command generates a checksum for file items, thus providing a means to determine if data in a file has been changed. A checksum is the arithmetic total, disregarding overflow, of all bytes in the selected items. The general form is:

```
CHECK-SUM {DICT} file-name {item-list} {attribute} {selection-criteria}
```

A checksum is generated for items in the specified file, or subset of items if the optional "item-list" and/or "selection-criteria" appear. Furthermore, the checksum may be calculated for one specified attribute.

If no attribute is specified, the first default attribute will be used. If there is no default attribute, or if the AMC is 9999, the entire item will be included. The checksum will include the binary value of each character times a positional value.

This yields a checksum which has a high probability of being unique for a given character string. The dictionary portion is checksummed if the "DICT" option appears.

A message is output giving checksum statistics in the following form:

```
BYTE STATISTICS FOR file-name (or attribute name):
TOTAL = t AVERAGE = a ITEMS = i CKSUM = c BITS = b
```

where:

t is the total number of bytes in the attribute (or item) included

a is the average number of bytes

i is the number of items

c is the checksum

b is a bit count

The attribute mark trailing the specified attribute (or item) will be included in the statistics.

To use checksums, the user should issue CHECK-SUM commands for all files or portions of files to be verified and keep the output statistics. Subsequently, the CHECK-SUM commands can be reissued to verify that the checksum statistics have not changed. The checksum must be recalculated whenever the user updates the file.

7.5 FILE STRUCTURE INQUIRY

File structure inquiries can be made under form commands. The ITEM and GROUP commands provide information about the item and group structure of PICK files. Output can be displayed at the terminal or optionally directed to the line printer.

ISTAT and HASH-TEST commands are verbs that produce file hashing histograms; ISTAT for specified file items and HASH-TEST on the basic of a user specified test modulo.

7.5.1 ITEM COMMAND

The ITEM command has the following general form:

```
ITEM file-name item-id {(options)}
```

This command displays the base FID of the group into which the specified item-id hashes. If the item is not already on file, the message "ITEM NOT FOUND" is displayed. In addition, every item-id in that group is listed along with a character count of the item (in hex). At the end of the list, the following message is displayed:

```
n ITEMS m BYTES p/q FRAMES
```

where:

n is the number of items in the group

m is the total number of bytes used in the group

p is the number of full frames in the group

q is the number of bytes used in the last frame of the group

Valid options for this command are as follows:

P Direct output to line printer.

S Suppress item list.

An example of the use of the ITEM command:

```
>ITEM MD A  [CR]

27121
0022 FILE-DOC
001C bd
0009 A
0011 T-ATT
000F DUMP
0018 B/ADD
000F DIVX
0014 EDIT-LIST
0028 V/CONV
0022 LISTU
0019 V/MIN
0018 ACCOUNT-RESTORE
001D D/CODE
0028 SL
0023 INST-INDEX
0047 SAL
0072 TB
000E SAVE
18 ITEMS 591 BYTES 1/91 FRAMES
```

7.5.2 GROUP COMMAND

The GROUP command has the following general form:

```
GROUP file-name {(options)}
```

This command displays the base FID of each group in the specified file. In addition, every item-id in the group is listed along with a character count of the item (in hex). At the end of the list for each group, the following message is displayed:

```
n ITEMS m BYTES p/q FRAMES
```

where:

- n is the number of items in the group
- m is the total number bytes used in the group
- p is the number of full frames in the group
- q is the number of bytes used in the last frame of the group.

Valid options for this command are as follows:

- I Suppresses output of null groups.
- P Direct output to line printer.
- S Suppress item list.

7.5.3 ISTAT COMMAND

ISTAT is an ACCESS verb. Its general format is:

```
ISTAT {DICT} file-name {item-list} {selection-criteria}
      {modifiers} {(options)}
```

An ACCESS sentence using the ISTAT command is:

```
>ISTAT PARCEL  [CR]
```

The ISTAT command provides a file hashing histogram for the selected items in the selected file as illustrated by the following example:

```
FILE= PARCEL MODULO= 3 SEPAR= 1                 13:50:42   22 MAR 1982
FRAMES BYTES ITEMS
000002 00757 027  *>>>>>>>>>>>>>>>>>>>>>
000002 00836 030  *>>>>>>>>>>>>>>>>>>>>>>>>
000002 00785 028  *>>>>>>>>>>>>>>>>>>>>>

ITEM COUNT=          85, BYTE COUNT=     2378, AVG. BYTES/ITEM=      27.9
AVG. ITEMS/GROUP=   28.3, STD. DEVIATION=  1.5, AVG. BYTES/GROUP=    792.6.
```

If you wish to suppress the histogram for the selected items, include an S in the options parameter.

7.5.4 HASH-TEST COMMAND

HASH-TEST produces a file hashing histogram as a result of a user-specified test modulo. The general form of this ACCESS verb:

```
HASH-TEST {DICT} file-name {item-list} {selection-criteria}
          {modifiers} {(options)}
```

An example of HASH-TEST:

```
>HASH-TEST PARCEL  [CR]

TEST MODULO:  9  [CR]
FILE= PARCEL MODULO= 9 SEPAR= 1                 13:50:55  22 MAR 1982
FRAMES BYTES ITEMS
000001 00256 009  *>>>>>>>>>
000001 00281 010  *>>>>>>>>>>
000001 00255 009  *>>>>>>>>>
000001 00229 008  *>>>>>>>>
000001 00248 009  *>>>>>>>>>
000001 00251 009  *>>>>>>>>>
000001 00272 010  *>>>>>>>>>>
000001 00307 011  *>>>>>>>>>>>
000001 00279 010  *>>>>>>>>>>

ITEM COUNT=          85, BYTE COUNT=      2378, AVG. BYTES/ITEM=     27.9
AVG. ITEMS/GROUP=    9.4, STD. DEVIATION=    .8, AVG. BYTES/GROUP=   264.2
```

7.6 PICK SYSTEM VERIFICATION

The VERIFY-SYSTEM verb checks the system software.

The general form is:

```
VERIFY-SYSTEM {n{-m}}
```

where n-m represents a range of frame numbers to verify. If VERIFY-SYSTEM is entered without frame number(s), it generates a check-sum for every frame of software, from 1 to 399. These check-sums are compared with those in the ERRMSG file, in an item named "CHECK-SUM". This item contains the correct check-sum for all system software frames. Each line in the item contains a check-sum for one frame of code, optionally followed by one or more hexadecimal limits. If the limits are present, the check-sum is generated only for bytes within the limits. If no limits are present, the check-sum is generated for bytes 0-2047. This is done because some frames contain tables which change from time to time, such as the system overflow table. Note that only assembly code is check-summed; table entries are not.

If all the program frames verify, message 341 is printed:

```
[341] ZEBRA PICK REV m.n SYSTEM VERIFIED.
```

If a frame generates a check-sum which does not match the check-sum for that frame in the "CHECK-SUM" item, the FID of the frame, the generated check-sum and the stored check-sum from the item are printed, and message 342 is printed at the end of the check run:

```
[342] ***ZEBRA PICK 'A' SYSTEM DOES NOT VERIFY***
THERE ARE 'B' FRAMES WITH MISMATCHES
```

where 'A' is the revision number and 'B' is the number of frames whose check-sums do not match.

If frame number(s) are specified and they verify, message 343 is printed:

```
'A' SYSTEM.
FRAMES(S) 'B' VERIFIED
```

If the frame(s) do not verify, message 344 is printed:

```
FRAME(S) 'B' DO NOT VERIFY!!
THERE ARE 'C' FRAMES WITH MISMATCHES
```

A VERIFY-SYSTEM should be run whenever it is suspected that the system software is in error.

If a mismatch is found, the software can be restored by mounting the SYS-GEN tape and using the A option after pressing RESET.

ERRMSG (error messages) A

PICK ERROR MESSAGES

ID No.	Message
2	UNEVEN NUMBER OF DELIMITERS (´ ").
3	VERB?
4	´A´ IS NOT AN ACCOUNT NAME.
5	THE WORD "A" IS ILLEGAL.
7	A VALUE MUST FOLLOW THE HEADING, FOOTING, TAG OR GRAND-TOTAL CONNECTIVE.
9	SYSTEM DL/ID MISSING.
10	FILE NAME MISSING.
11	FRAME LOCKED AT LOCATION X´A´.
13	DATA LEVEL DESCRIPTOR (FILE-NAME IN DICTIONARY) IS MISSING.
15	THE FILE-NAME IS PRECEDED BY AN ILLEGAL CONNECTIVE.
16	CURRENT SOFTWARE RELEASE IS: ABS:A MONITOR: B.
17	"WITHIN" IS VALID ONLY IN COUNT/LIST/SUM OR STAT STATEMENTS.
18	THE LAST WORD MAY NOT BE A CONNECTIVE.
19	A VALUE WITHOUT AN ATTRIBUTE NAME IS ILLEGAL.
20	ERROR IN THE "USING" SYNTAX.
21	MEANINGLESS ITEM-ID IN STATEMENT.
22	"TO" BEFORE ITEM-ID VALID ONLY IN A CHANGE STATEMENT.
24	THE WORD "A" CANNOT BE IDENTIFIED.
25	THE CONNECTIVE "WITH" MAY NOT IMMEDIATELY PRECEDE A VALUE.
26	ATTRIBUTE VALUES MAY NOT BOTH PRECEDE AND FOLLOW AN ATTRIBUTE NAME.
29	AT LEAST ONE ITEM-ID MUST BE SPECIFIED FOR A "WITHIN"-TYPE STATEMENT.
30	FORMAT ERROR IN M/DICT ENTRY DEFINING VERB.
32	MAG TAPE ATTACHED TO LINE ´A´.
34	DISK CARTRIDGE ATTACHED TO LINE ´A´.
39	YOU MUST BE ON SYSPROG TO CHANGE THIS PASSWORD.
40	PASSWORDS CAN ONLY BE DELETED ON SYSPROG.
41	STRANGE ACCOUNT POINTER. PASSWORD CANNOT BE CHANGED.
45	YOU MUST BE ON THE SYSPROG ACCOUNT FOR THIS FUNCTION!!!
47	YOU MUST BE ON LINE 0 FOR THIS FUNCTION!!!
53	SET IN MD7/8/18: ITEM-ID LIST REQUIRED FOR WITHIN
60	ACCOUNT ´A´ CANNOT BE DELETED.
61	´A´ IS NOT AN ACCOUNT.
62	A ´Q´ TYPE ACCOUNT CANNOT BE DELETED.
63	ACCOUNTS CAN ONLY BE DELETED ON SYSPROG.

ID No.	Message
71	AN ILLEGAL CONNECTIVE MODIFIES THE WORD "A".
72	THE VALUE "A" IS MEANINGLESS.
79	THE NUMBER OF SEPARATE AND CLAUSE SETS CANNOT EXCEED 9.
80	A SYSTEM ERROR HAS OCCURRED IN MODE: 'A' THIS MAY BE DUE TO SORT-KEY(S) PRECEDING SELECTION CRITERIA.
82	YOUR SYSTEM PRIVILEGE LEVEL IS NOT SUFFICIENT FOR THIS STATEMENT.
84	VERB IS ONLY VALID FOR 1/2" TAPE.
85	AT BEGINNING OF 1/4" TAPE BUFFER.
87	TAPE TRANSFER NOT COMPLETED. ERROR CODE: A
89	VERB IS NOT A VALID 1/2" TAPE OR CARTRIDGE DISK VERB.
91	END TAPE CHECK - 'A' FILE(S).
92	END OF RECORDED DATA - 'A' FILE(S).
93	ATTACH THE TAPE UNIT.
94	END OF FILE.
95	TAPE ATTACHED TO LINE 'A'.
96	BOT
97	END OF TAPE
98	PARITY ERROR!
102	MONITOR AND ABS DUMP COMPLETED.
103	TAPE ERROR.
104	DEI CARTRIDGE TAPE ASSIGNED.
105	ARCHIVE CARTRIDGE TAPE ASSIGNED.
118	THE FORM "WITH ATTRIBUTE AND ATTRIBUTE" IS UNDEFINED.
120	'A' NEGATIVE BALANCE NOT PERMITTED.
158	AN ILLEGAL CONNECTIVE OF THE FORM "A" MODIFIES "B".
163	AN A-, F-, OR T-CORRELATIVE HAS FAILED COMPILATION.
164	TOTAL OR CONTROL-BREAK CONNECTIVE NOT SUCCEEDED BY ATTRIBUTE DEFINITION.
166	THE A-CORRELATIVE ATTRIBUTE NAME 'A' IS ILLEGAL.
167	MISSING TERMINAL QUOTE ("/") IN A-CORRELATIVE: 'A'.
168	ILLEGAL A-CORRELATIVE: 'A'.
169	MISSING LEFT PAREN IN A-CORRELATIVE: 'A'.
170	MISSING RIGHT PAREN IN A-CORRELATIVE: 'A'.
173	MISSING SEMI-COLON IN A-CORRELATIVE.
192	THIS IS AN ERRONEOUS OBJECT STRING.
193	BASIC OBJECT CODE CANNOT BE COPIED TO AN ITEM.
194	DON'T COPY TO THE SAME ITEM IN THE SAME FILE.
196	'A' IS TOO LARGE TO BE AN ITEM.
197	THIS GET-LIST OR SAVE-LIST SPECIFICATION IS INCORRECT.
198	THE LIST POINTED TO BY POINTER ITEM 'A' IS DEFECTIVE.
199	INSUFFICIENT WORK SPACE FOR ITEM 'A'.
200	FILE NAME?
201	'A' IS NOT A FILE NAME.
202	'A' NOT ON FILE.
203	ITEM NAME?

ID No.	Message
204	FILE DEFINITION 'A' IS MISSING.
205	NO STATEMENTS TO BE ASSEMBLED.
208	ERROR IN ITEM-ID LIST.
210	FILE 'A' IS ACCESS PROTECTED.
211	'A' NO ASSEMBLED CODE CAN BE FOUND.
212	'A' HAS NO FRAME STATEMENT
213	'A' LOCATION COUNTER ERROR AT LINE NO. 'E'.
214	'A' OVERFLOWS FRAME A AT LINE NO. 'E'.
215	'A' HEX ERROR AT LINE NO. 'E'.
216	'A' LOADED; FRAME = B SIZE = C CKSUM = D.
217	'A' VERIFIED; FRAME = B SIZE = C CKSUM = D.
218	'A' FRAME = B HAS F MISMATCHES.
219	ILLEGAL COMMAND: 'A'.
220	'A' EXITED.
221	'A' FILED.
222	'A' DELETED.
224	'A' IS A POINTER ITEM. IT MAY NOT BE LOADED INTO A NON-POINTER FILE.
225	TSYM MUST BE DICT & DATA MOD/SEP 1,1 37,1 SORRY FOR INCONVENIENCE.
226	TAPE FORMAT ERROR.
234	ITEM SIZE EXCEEDS 32,000 BYTES.
235	ATTEMPT TO WRITE INTO UPDATE PROTECTED FILE!
239	'A' LIST SAVED. 'B' FRAME(S) USED.
240	A "SELECT" OR "SSELECT" MUST BE USED PRECEDING A 'SAVE-LIST' STATEMENT!
241	PROGRAM 'A' COMPILED; 'B' FRAMES USED.
242	'A' DECATALOGED.
243	'A' LIST SAVED - 'B' FRAME(S) USED.
244	'A' CATALOGUED!
245	'A' LIST DELETED.
246	'A' IS NO LONGER A LIST OR NO LONGER ON FILE.
247	'A' IS BASIC OBJECT CODE.
249	AN 'F' OPTION IS NOT VALID WITH THIS VERB.
260	INVALID BATCH LOCK COMMAND: 'A'.
265	PROC STACK OVERFLOW.
267	PROC TRANSFER TO 'A' CANNOT BE COMPLETED.
268	DESTINATION OF PROC "GO" STATEMENT: 'A'. CANNOT BE FOUND.
269	PRIMARY INPUT BUFFER HAS EXCEEDED 300 BYTES.
270	FORMAT ERROR IN THE PROC STATEMENT: 'A'.
272	A VALUE EXISTS FOR THE ATTRIBUTE REFERENCED BY THE ELEMENT: A.
273	ERROR IN COLUMN-NUMBER/FIELD-WIDTH OR FORMAT SPECIFICATION AT: 'A'.
274	UNRECOGNIZABLE BATCH-STRING ELEMENT: 'A'.
275	Y OR F SUB-ELEMENT ERROR AT BATCH-STRING ELEMENT: 'A'.
276	D-2 UPDATE WITHOUT D-1 BEING SPECIFIED, AT BATCH-STRING ELEMENT: 'A'.
277	J ELEMENT MISSING AT BATCH-STRING ELEMENT: 'A'.
278	ERROR IN PROCESSING SECONDARY BATCH-STRING ELEMENT: 'A'.
279	INCORRECT SCALING FACTOR IN F* BATCH-STRING ELEMENT: 'A'.
280	FILE-DEFINITION BATCH ELEMENT ERROR AT: 'A'.

ID No.	Message
281	D1 MUST HAVE Y11 STORAGE CORRELATIVE ... ERROR AT: ´A´.
282	INVALID PARAMETER FOR SELECT LIST.
289	TERMINAL PRINTER: PAGE WIDTH: a b PAGE DEPTH: c d LINE SKIP: e LF DELAY: f FF DELAY: g BACKSPACE: h TERM TYPE: i
290	THE RANGE OF THE PARAMETER "A" IS NOT ACCEPTABLE.
293	TOTAL NUMBER OF CONTIGUOUS FRAMES: ´A´.
298	FORMAT ERROR IN SPECIFICATIONS.
316	NUMERIC PARAMETER MISSING.
331	THE ACCOUNT FILE IS MISSING.
335	<<<R80 GENERAL AUTOMATION REV. 3.0>>> <<<time ZEBRA date>>>
336	< LOGGED OFF AT time ON date >
337	USER IS NOT LOGGED ON.
338	ACCOUNT FILE STATISTICS WERE NOT UPDATED DUE TO EITHER: 1. INSUFFICIENT WORK-SPACE TO CONTAIN THE ACCOUNT FILE ITEM, OR 2. SYSTEM DICTIONARY CHANGED WHILE YOU WERE LOGGED ON.
340	< CONNECT TIME = A MINS.; CPU = B UNITS; LPTR PAGES = C >.
341	ZEBRA PICK ´A´ SYSTEM VERIFIED!!!
342	*** ZEBRA PICK ´A´ SYSTEM DOES NOT VERIFY! *** THERE ARE ´B´ FRAMES WITH MISMATCHES.
343	´A´ SYSTEM. FRAME(S) ´B´ VERIFIED!!
344	´A´ SYSTEM ´B´ FRAMES DO NOT VERIFY!! THERE ARE ´C´ FRAMES WITH MISMATCHES.
390	´A´ LOADED.
391	´A´ VERIFIED.
398	THE MAXIMUM OF 20 LEVELS FOR A "WITHIN"-TYPE STATEMENT HAS BEEN EXCEEDED.
399	THE FILE OR DL/ID REQUIRES A V(ERTICAL) CORRELATIVE FOR THIS STATEMENT.
401	NO ITEMS PRESENT.
402	ITEM ´A´ IS A ´D´ POINTER IN THE SECONDARY FILE.
403	END OF LIST.
404	´A´ ITEMS SELECTED.
405	´A´ ITEMS LISTED.
406	ITEM COUNT = a, BYTE COUNT = b, AVG. BYTES/ITEM = c AVG. ITEMS/GROUP = d STD. DEVIATION = e, AVG. BYTES/GROUP = f.
407	´A´ ITEMS COUNTED.
408	ONE ITEM COUNTED.
410	A SYNONYM (Q-TYPE) FILE CANNOT BE SPECIFIED IN THIS STATEMENT.
411	´DICT´ OR ´DATA´ MUST BE SPECIFIED IN A CLEAR-FILE STATEMENT.

ID No.	Message
412	INSUFFICIENT DISK SPACE AVAILABLE FOR THE FILE.
413	THE FILE NAME ALREADY EXISTS IN THE MASTER DICTIONARY.
414	ILLEGAL OR MISSING MODIFIER USED IN DEFINING THE FILE AREA(S).
415	'A' EXISTS ON FILE.
416	RANGE ERROR IN MODULO OR SEPARATION PARAMETER.
417	FILE 'a' CREATED; BASE = b, MODULO = c, SEPAR = d.
418	FILE DEFINITION ITEM 'A' NOT COPIED.
419	THE SPECIFIED FILE CANNOT BE CLEARED OR DELETED.
420	DICTIONARY FILE DELETION CANNOT BE DONE WITHOUT DELETION OF DATA-SECTION(S) FIRST.
421	STATISTICS OF a: TOTAL = b AVERAGE = c COUNT = d.
422	BYTE STATISTICS FOR: 'A' TOTAL = b AVERAGE = c ITEMS = d CKSUM = e BITS = f.
423	TOTAL OF 'A' = 'B'.
424	FRAME ID: a DEVICE ADDRESS b UNIT NUMBER c CYLINDER d HEAD e SECTOR f.
425	INVALID FRAME-ID REQUEST.
426	DATA FILE ALREADY EXISTS.
427	THERE IS NO DATA SECTION FOR THIS FILE.
428	'A' OBJECT POINTER NOT FOUND.
429	'A' PROGRAM OBJECT VERIFIES.
430	'A' PROGRAM OBJECT DOES NOT VERIFY.
432	A' ITEM-NAME USED IN PROGRAM-FILE DICTIONARY, COMPILE ABORTED.
522	BLOCK-CONVERT FILE MISSING OR IMPROPERLY DEFINED.
524	THE LETTER 'A' IS NOT IN THE BLOCK-CONVERT FILE.
525	INPUT CHARACTER 'A' IS IMPROPERLY FORMATTED IN BLOCK-CONVERT FILE.
530	ALREADY LOGGED ON.
531	PROCESS ROADBLOCKED.
532	ILLEGAL USER ID.
533	LOGON SUCCESSFUL.
534	LOGOFF SUCCESSFUL.
535	ILLEGAL LINE NUMBER.
536	ALREADY LOGGED OFF.
550	A REQUIRED NUMERIC PARAMETER IS MISSING OR INVALID.
552	ITEM 'A' HAS INVALID FORMAT.
654	ILLEGAL OUTPUT STRING LENGTH!
700	RUN-TIME F-CORRELATIVE ABORT.
701	INVALID FUNCTION CORRELATIVE DEFINITION: 'A'.
705	ILLEGAL CONVERSION CODE: 'A'.
706	TRANSLATE CONVERSION-CODE: 'A' IS ILLEGAL.
708	'A' CANNOT BE CONVERTED.
781	'A' ADDED.
782	'A' UPDATED.
783	'A' DELETED.
802	'A' ITEMS DUMPED.

ID No.	Message
803	'A' ITEM(S) LOADED.
805	'A' ITEMS COPIED.
806	'A' ITEMS UPDATED.
807	'A' ITEM(S) DELETED.
850	NODE NOT AVAILABLE.
851	SENDING ACCOUNT MISSING %%SELP%% VERB.
852	INVALID NODE ID.
853	UNEXPECTED MESSAGE RECEIVED.
854	NO LAN PROCESSOR ON THIS MACHINE.
861	NO LAN CONTROLLERS PRESENT.
900	PROBLEM HAS OCCURRED IN THE CREATE-ACCOUNT PROCESS. PLEASE START OVER.
901	'A' ACCOUNT CREATED.
911	UNABLE TO RUN BINARY SAVE - OTHER USERS STILL LOGGED ON.
990	ERROR IN ABS-DUMP FRAME LIMITS SPECIFICATIONS.
992	'A' ITEM(S) HAVE BEEN RESTORED.
993	ACCOUNT NAME MUST BE SPECIFIED.
999	time date CORE LINES PCBO WSSTART WSSIZE SYSBASE/MOD/SEP MAXFID AVAIL. OVERFLOW AK B C D E F G H I
1001	COUPLED SET SPECIFICATION ERROR.
1002	WINDOW AND TAG CONNECTIVES NOT ALLOWED.
1011	SYSTEM ERROR IN THE SELECTION PROCESSOR. TRY TO GENERATE A REPRODUCIBLE CASE. CONTACT YOUR SYSTEM SUPPORT ANALYST.
1050	THAT IS AN OBSOLETE BASIC VERB (RUN OR CATALOG). PLEASE OBTAIN THE CURRENT DEFINITION.
1051	THAT IS A PROGRAM CATALOGED UNDER PRE-R80 PROTOCOLS. PLEASE RECATALOG THE PROGRAM.
1100	START CODE LOCKED.
1101	NULL PRINTER NUMBER.
1102	PRINTER NUMBER TOO BIG.
1103	NO FORM NUMBER.
1104	ILLEGAL CHARACTER.
1105	PRINTER MUST BE STOPPED.
1106	FORM NUMBER TOO BIG - EXCEEDS 125.
1107	TOO MANY PAGES IN THE PAGE SKIP - EXCEEDS 9.
1108	NEGATIVE NUMBER.
1109	TOO MANY OUTPUT QUEUES.
1110	ILLEGAL PRINTER TYPE - MUST BE S.
1111	ILLEGAL LINE NUMBER.
1113	ILLEGAL SERIAL PRINTER NUMBER.
1114	THE LINE WHICH YOU SPECIFIED IS BEING USED AS ANOTHER PRINTER ON THE SYSTEM.
1115	ALLOCATION ATTEMPTED ON UNINITIALIZED PRINTER.
1116	THERE IS NO JOB ENQUEUED FOR OUTPUT ON THE FORMS YOU SPECIFIED, THEREFORE ALIGNMENT IS IMPOSSIBLE.
1117	YOUR ALIGN WAS JUST ABORTED BY SOMEONE. YOU MUST START THE ALIGN PROCESS OVER.

ID No.	Message
1118	THE PRINTER CONTROL BLOCK HAS BEEN INITIALIZED. CHECK FOR CORRECT PRINTER FORM AND LPI.
1119	YOU ARE ATTEMPTING TO START PRINTER 'A' ON LINE 'B', WHICH IS NOT STOPPED.
1121	YOU ARE IMPROPERLY LOGGED ON.
1123	LINE # 'A' IS ALREADY LOGGED ON.
1127	ILLEGAL PRINTER NUMBER. MUST BE 0-19, INCLUSIVE.
1129	'A' FORM QUEUE ELEMENTS UNLINKED.
1130	PRINTER LIST ELEMENTS: date time STAT LK LN CURPOS BEGFID CP FO FRMS DATE TIME ACCT
1131	PRINTER LIST ELEMENTS: date time # STAT LK LN STATUSES CP FO FRMS DATE TIME ACCT
1132	'A' QUEUE ELEMENTS.
1133	'A' FRAMES IN USE.
1134	PRINTER ASSIGNMENTS: PRINTER OUTPUT QUEUES PAGE DEV OR STATUS TYPE NUMBER SKIP LINE #
1135	FORM QUEUE 'A'.
1140	YOUR OPEN FILES WERE CLOSED.
1141	LINE STATUS COP FORM # IES #
1143	ALIGN TERMINATED; PRINTER STOPPED.
1144	TAPE NOT AVAILABLE AT THIS TIME.
1145	ILLEGAL SPECIFICATION NUMBER 'A'.
1147	TAPE NOT ATTACHED.
1148	TAPEOUT TERMINATED BECAUSE OF ASSIGN T OR NULL ASSIGNMENT.
1150	THERE IS SOMETHING WRONG WITH THE SYNTAX OF YOUR VERB'S OPTIONS.
1160	YOUR OUTPUT SPECIFICATION IS NO OUTPUT. REASSIGN YOUR LINE IF YOU WISH TO OUTPUT A HOLDFILE.
1161	END OF REQUESTED PRINT FILES.
1162	END OF PRINT FILE CONTROL BLOCK.
1169	ILLEGAL PRINTER NUMBER. MUST BE BETWEEN 0 AND 7 INCLUSIVE.
1170	PRINTER # 'A' SET TO STOP +.
1171	AND IS INACTIVE.
1172	BUT IS STILL ACTIVE.
1173	PRINTER # 'A' CONTROL BLOCK HAMMERED, CLEARED TO NULL.
1174	IS UNALLOCATED.
1175	PARALLEL PRINTER # 'A' HAS BEEN DELETED.
1176	SERIAL PRINTER # 'A' HAS BEEN DELETED, AND ITS PROCESS SENT TO LOGON.
1177	PRINTER # 'A' IS INACTIVE.
1178	JOB BEING OUTPUT ON PRINTER # 'A' IS NOT YOUR PRINT FILE.
1179	JOB ABORTED ON PRINTER # 'A'.

ID No.	Message
1180	PRINT FILE # 'A' WAS NOT UNLINKED BECAUSE IT IS BEING OUTPUT.
1181	PRINT FILE # 'A' WAS NOT UNLINKED BECAUSE IT IS UNUSED.
1182	PRINT FILE # 'A' WAS NOT UNLINKED BECAUSE IT IS NOT SPOOLED.
1183	PRINT FILE # 'A' WAS NOT CREATED ON THE ACCOUNT ONTO WHICH YOU ARE NOW LOGGED.
1184	PRINT FILE # 'A' WAS UNLINKED AND IS AVAILABLE AS A HOLD FILE.
1200	THE SPOOLER IS INACTIVE.
1201	THE SPOOLER IS ACTIVE.
1202	NEEDS TO START PRINTERS.
1203	NEEDS TO LOG DISK ERRORS.
1204	
1209	CONTROL BLOCK FOR PRINTER # 'A' IS IN AN AMBIGUOUS STATE. DELETE THE PRINTER FROM THE SPOOLER SYSTEM.
1210	PRINTER # 'A' IS +.
1211	UNALLOCATED.
1212	SERIAL +
1213	PARALLEL +
1214	, INACTIVE +
1215	, ACTIVE +
1216	, STOPPED +
1217	, AND ON LINE
1218	, AND OFF LINE
1219	*** THE PRINTER CABLE IS OFF. ***
1220	*** THERE IS NO CONTROLLER FOR THIS PRINTER . ***
1222	THE PRINTER IS RUNNING ON LINE 'A'.
1229	PRINT FILE BEING OUTPUT IS ELEMENT 'A, A' +.
1230	AN OPEN FILE FOR LINE # 'A'.
1231	CLOSED FILE FOR LINE # 'A'.
1232	GENERATED ON ACCOUNT 'A' +.
1233	, WHICH IS a FRAMES LONG.
1234	AND THE OUTPUT IS CHOKED.
1235	.
1239	NO OUTPUT QUEUES ASSIGNED TO PRINTER.
1240	ASSIGNED OUTPUT QUEUES: 'A'+.
1241	,'A' +.
1242	,'A' +.
1243	NUMBER OF INTER-JOB PAGES TO EJECT IS 'A'.
2003	ON!
2004	OFF!
2401	EXECUTE INITIALIZED. 'A' WORKSPACES PRE-ALLOCATED.
2402	THE NUMBER OF WORKSPACES TO BE ALLOCATED MUST BE IN THE RANGE 0-250.
2403	THE NUMBER OF NESTED LEVELS MUST BE IN THE RANGE 0-15.
2404	THE EXECUTE-CONTROL FILE (IN SYSTEM) MUST HAVE A D/CODE OF "DCY".
2405	THE EXECUTE-CONTROL FILE (IN SYSTEM) CANNOT BE OPENED.
2406	THE MAXIMUM NUMBER OF NESTED EXECUTE STATEMENTS HAS BEEN EXCEEDED.

PICK/BASIC ERROR MESSAGES

ID No.	Message
B1	RUN-TIME ABORT AT LINE 'A'.
B10	LINE 'A' VARIABLE HAS NOT BEEN ASSIGNED A VALUE; ZERO USED.
B11	LINE 'A' TAPE RECORD TRUNCATED TO TAPE RECORD LENGTH.
B12	LINE 'A' FILE HAS NOT BEEN OPENED.
B13	LINE 'A' NULL CONVERSION CODE IS ILLEGAL; NO CONVERSION DONE.
B14	LINE 'A' BAD STACK DESCRIPTOR.
B15	LINE 'A' ILLEGAL OPCODE: 'C'.
B16	LINE 'A' NON-NUMERIC DATA WHEN NUMERIC REQUIRED; ZERO USED.
B17	LINE 'A' ARRAY SUBSCRIPT OUT-OF-RANGE.
B18	LINE 'A' ATTRIBUTE NUMBER LESS THEN -1 IS ILLEGAL.
B19	LINE 'A' ILLEGAL PATTERN.
B20	LINE 'A' COL1 OR COL2 USED PRIOR TO EXECUTING A FIELD STATEMENT; ZERO USED.
B21	LINE 'A' MATREAD: NUMBER OF ATTRIBUTES EXCEEDS VECTOR SIZE.
B24	LINE 'A' DIVIDE BY ZERO ILLEGAL; ZERO USED.
B25	PROGRAM 'C' HAS NOT BEEN CATALOGED.
B27	LINE 'A' RETURN EXECUTED WITH NO GOSUB.
B28	LINE 'A' NOT ENOUGH WORK SPACE.
B29	CALLING PROGRAM MUST BE CATALOGED.
B30	LINE 'A' ARRAY SIZE MISMATCH.
B31	LINE 'A' STACK OVERFLOW.
B32	LINE 'A' PAGE HEADING EXCEEDS MAXIMUM OF 1400 CHARACTERS.
B33	LINE 'A' PRECISION DECLARED IN SUBPROGRAM 'C' IS DIFFERENT FROM THAT DECLARED IN THE MAINLINE PROGRAM.
B34	LINE 'A' FILE VARIABLE USED WHERE STRING EXPRESSION EXPECTED.
B35	'MD' INVALID OBJECT OF 'CLEARFILE'; IGNORED.
B36	SYSTEM DICT ILLEGAL OBJECT OF 'CLEARFILE'; ABORT.
B42	NOT ENOUGH DESCRIPTOR SPACE.
B100	LINE 'A' COMPILATION ABORTED: NO OBJECT CODE PRODUCED.
B101	LINE 'A' MISSING "END", "NEXT", "WHILE", "UNTIL", "REPEAT" OR "ELSE"; COMPILATION ABORTED, NO OBJECT CODE PRODUCED.
B102	LINE 'A' BAD STATEMENT.
B103	LINE 'A' LABEL 'C' IS MISSING.
B104	LINE 'A' LABEL 'C' IS DOUBLY DEFINED.
B105	LINE 'A' 'C' HAS NOT BEEN DIMENSIONED.
B106	LINE 'A' 'C' HAS BEEN DIMENSIONED AND USED WITHOUT SUBSCRIPTS.

ID No.	Message
B107	LINE 'A' "ELSE" CLAUSE MISSING.
B108	LINE 'A' "NEXT" STATEMENT MISSING.
B109	LINE 'A' VARIABLE MISSING IN "NEXT" STATEMENT.
B110	LINE 'A' 'END' STATEMENT MISSING.
B111	LINE 'A' "UNTIL" OR "WHILE" MISSING IN "LOOP" STATEMENT.
B112	LINE 'A' "REPEAT" MISSING IN "LOOP" STATEMENT.
B113	LINE 'A' TERMINATOR MISSING.
B114	LINE 'A' MAXIMUM NUMBER OF VARIABLES EXCEEDED.
B115	LINE 'A' LABEL 'C' IS USED BEFORE THE EQUATE STATEMENT.
B116	LINE 'A' LABEL 'C' IS USED BEFORE THE COMMON STATEMENT.
B117	LINE 'A' LABEL 'C' IS MISSING A SUBSCRIPT LIST.
B118	LINE 'A' LABEL 'C' IS THE OBJECT OF AN EQUATE STATEMENT AND IS MISSING.
B119	LINE 'A' WARNING - PRECISION VALUE OUT OF RANGE (IGNORED).
B120	LINE 'A' WARNING - MULTIPLE PRECISION STATEMENTS (IGNORED).
B121	LINE 'A' LABEL 'C' IS A CONSTANT AND CAN NOT BE WRITTEN INTO.
B122	LINE 'A' LABEL 'C' IS IMPROPER TYPE.
B123	PROGRAM CONTAINS AN EQUATE WHICH CANNOT BE RATIONALIZED.
B124	LINE 'A' LABEL 'C' HAS LITERAL SUBSCRIPTS OUT OF RANGE.
B125	LINE 'A' LABEL 'C' HAS A JUMP GREATER THAN 32K BYTES.
B126	OBJECT CODE EXCEEDS 65K.
B127	OBJECT CODE AND SYMBOL TABLE EXCEED 65K.
B128	LINE 'A' LABEL 'C' EQUATED ARRAY SUBSCRIPT OUT OF RANGE.
B154	FOR STATEMENT WITH NO NEXT STATEMENT.
B199	FORMAT ERROR IN SOURCE FILE DEFINITION.
B209	LINE 'A' FILE IS UPDATE PROTECTED.
B210	LINE 'A' FILE IS ACCESS PROTECTED.
B850	NODE NOT AVAILABLE.
B853	UNEXPECTED MESSAGE RECEIVED.
B854	NO LAN PROCESSOR ON THIS MACHINE.

PICK/ACCU-PLOT ERROR MESSAGES

ID No.	Message
G90	THE ACCUPLOT ACCOUNT HAS BEEN INITIALIZED. THE FRAME ASSIGNMENTS NEED TO BE VERIFIED. THE MODES NEED TO BE LINKED. THE DEVICES NEED TO BE LINKED. THE MODES NEED TO BE LOADED.
G91	THE ACCUPLOT FRAMES HAVE BEEN ASSIGNED. THE MODES NEED TO BE LINKED. THE DEVICES NEED TO BE LINKED. THE MODES NEED TO BE LOADED.
G92	THE ACCUPLOT MODES HAVE BEEN LINKED. THE DEVICES NEED TO BE LINKED. THE MODES NEED TO BE LOADED.
G93	THE ACCUPLOT MODES AND DEVICES HAVE BEEN LINKED. THE MODES NEED TO BE LOADED.
G94	ACCUPLOT IS OPERATIONAL.

ASCII codes B

The ASCII codes used by the Pick System are:

DEC	Hex	Character	DEC	Hex	Character
0	0	NULL	36	24	$
1	1	SOH	37	25	%
2	2	STX	38	26	&
3	3	ETX	39	27	'
4	4	EOT	40	28	(
5	5	ENQ	41	29	)
6	6	ACK	42	2A	*
7	7	BEL	43	2B	+
8	8	BS[1]	44	2C	,
9	9	HT[1]	45	2D	-
10	A	LF[1]	46	2E	.
11	B	VT[1]	47	2F	/
12	C	FF[1]	48	30	0
13	D	CR[1]	49	31	1
14	E	SO	50	32	2
15	F	SI	51	33	3
16	10	DLE	52	34	4
17	11	DC1	53	35	5
18	12	DC2	54	36	6
19	13	DC3	55	37	7
20	14	DC4	56	38	8
21	15	NAK	57	39	9
22	16	SYN	58	3A	:
23	17	ETB	59	3B	;
24	18	CAN	60	3C	<
25	19	EM	61	3D	=
26	1A	SUB	62	3E	>
27	1B	ESC	63	3F	?
28	1C	FS	64	40	@
29	1D	GS	65	41	A
30	1E	RS[1]	66	42	B
31	1F	US[1]	67	43	C
32	20	SPACE	68	44	D
33	21	!	69	45	E
34	22	"	70	46	F
35	23	#	71	47	G

DEC	Hex	Character	DEC	Hex	Character
72	48	H	104	68	h
73	49	I	105	69	i
74	4A	J	106	6A	j
75	4B	K	107	6B	k
76	4C	L	108	6C	l
77	4D	M	109	6D	m
78	4E	N	110	6E	n
79	4F	O	111	6F	o
80	50	P	112	70	p
81	51	Q	113	71	q
82	52	R	114	72	r
83	53	S	115	73	s
84	54	T	116	74	t
85	55	U	117	75	u
86	56	V	118	76	v
87	57	W	119	77	w
88	58	X	120	78	x
89	59	Y	121	79	y
90	5A	Z	122	7A	z
91	5B	[	123	7B	{
92	5C	\	124	7C	:
93	5D	]	125	7D	}
94	5E	^	126	7E	~
95	5F	_	127	7F	DEL
96	60	`	.		
97	61	a	.		
98	62	b	.		
99	63	c	251	FB	SB[2]
100	64	d	252	FC	SVM[2]
101	65	e	253	FD	VM[2]
102	66	f	254	FE	AM[2]
103	67	g	255	FF	SM[2]

[1]For special use on LSI-11 and -12 terminals:

BS	Cursor Backspace	FF	Cursor Forward
HT	Cursor Tab	CR	Cursor Carriage Return
LF	Cursor Down	RS	Cursor Home
VT	Cursor UP	US	Cursor New Line

[2]For special use by PICK:

SB	Start buffer
SVM	Secondary value mark (displays \)
VM	Value mark (displays])
AM	Attribute mark (displays ^)
SM	Segment mark (displays _)

ZEBRA series firmware executives C

C.1 UTILITIES AND DIAGNOSTICS

This section describes the use of the General Automation ZEBRA Utilities/ Diagnostics package in EPROM2, model numbers 70-01447A02 and 70-01447A04. The invocation and operation of Utility/Diagnostic functions, and procedures for the use of extended, non-resident functions are described.

C.1.1 OVERVIEW

The utilities and diagnostics programs are based upon a menu-driven system with formatted input/output to the console terminal port A (CPU board, Figure C-1). The resident portion resides in EPROM2 on the CPU board, starting address $400000. Only the EPROM utilities will be covered in this section. Extended non-resident portions may be loaded into memory, via serial port B (CPU board), from cartridge tape or disk. The resident portion may operate with reduced capabilities without the non-resident portion in memory. This resident portion contains utilities necessary to load the non-resident portions, boot the operating system, and save and restore the disk(s) on tape.

In addition, default responses are available in almost all cases and are indicated by bracketing them with greater-than and less-than signs (i.e., <default value>).

Additionally, valid ranges of responses are given if not self-evident, generally assuming a minimum value of zero or one as appropriate (i.e., <=value).

The type of response may be a single character, a string of characters (leading blanks ignored, blank delimiter or carriage return terminator), or a value in either decimal format (16 bits maximum) or hexadecimal format (indicated in the prompt by the dollar sign, which is not entered as part of the user response).

The format control characters are expressly designed for the Dialogue 80 terminal. Other terminals may be used; however, the display formatting for other terminals is not provided at present.

C.1.2 OPERATION

The minimum hardware requirements for the operation of this package are powered Multi-Bus backplane, a fully operational CPU board, and a terminal (Dialogue 80 or equivalent) with the necessary cables. The assumption is made that the onboard monitor (version A only) has been run and that the memory maps and device initialization remain as set by the basic utility system. This minimum system provides the resident features of the package. Extended or non-resident features must be loaded into memory by a utility in the resident package before they may be used (see Menu items #5, #6, and #7, Table C-1).

The extended features provide comprehensive status messages (Section C.1.3) as well as the bulk of the functions. If an attempt to access these functions is made before they are loaded, the user is notified. Since the program is still under development, some functions listed in the menus may not be available. The system detects this and notifies the user. The resident functions should allow troubleshooting of at least one of the possible sources for extended features loading.

When RESET is pressed, the sign-on message is sent to the console:

'GENERAL AUTOMATION ZEBRA RESIDENT UTILITIES - VERSION X.X, xxxx Kbytes of RAM'

The ROM resident version number is displayed, followed by the decimal number of kilobytes of RAM memory. Note that the ROM version number must match the version number of the extensions in the portion preceding the decimal point.

After the sign-on message is displayed, bit 0 (pins 1 and 2) of connector J2 on the ZEBRA CPU board (Figure C-1) is tested. If no jumper is installed, then the utilities are entered. If a jumper is installed between pins 1 and 2, then the automatic boot takes place, first displaying the message:

'Beginning automatic boot of operating system'

This reads in the PICK or XENIX operating system loader and executes it. (Refer to Section 2.0 of this document.) If the boot fails, the disk status is displayed and the utilities are entered, so that diagnostics can be selected for anaylsis of the failure.

The Utilities Diagnostics package is invoked via the RESET button. Exit from the Diagnostics package to the basic utility system is either by 'QUIT'ing from the first (top) level menu or by generating a break from the terminal (control BREAK on the Dialogue 80 terminal). Return from the basic utility system to the Diagnostics package can be accomplished via the 'G400000' command.

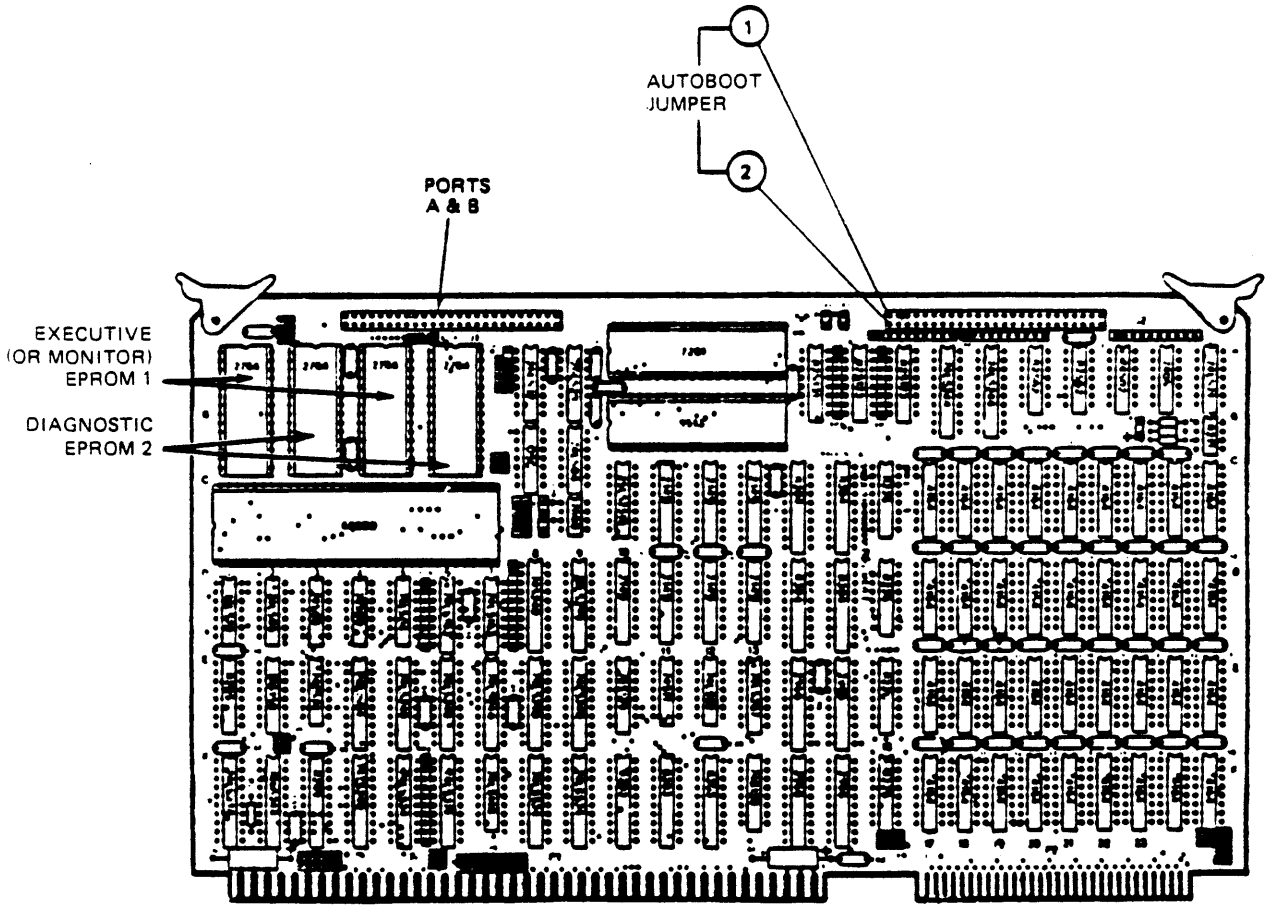


Figure C-1. ZEBRA CPU Board

When the package is first entered, the user is prompted for the terminal type:

```
Dialogue 80 terminal or equiv.  (<y>/n):  _
```

The Main Menu is then displayed (Table C-1):

Table C-1. Main Menu Items

SYSTEM UTILITIES

1. QUIT
2. BOOT OPERATING SYSTEM
3. BACK-UP DISK TO TAPE
4. RESTORE TAPE TO DISK
5. LOAD EXTENSIONS FROM SERIAL PORT B
6. LOAD EXTENSIONS FROM DISK
7. LOAD EXTENSIONS FROM TAPE
8. SAVE EXTENSIONS TO DISK
9. SAVE EXTENSIONS TO TAPE
10. CPU TESTS
11. CPU DIAGNOSTICS
12. ADES TESTS
13. ADES DIAGNOSTICS
14. COMMUNICATION TESTS
15. COMMUNICATION DIAGNOSTICS
16. CAPRO TESTS
17. CAPRO DIAGNOSTICS

The extended features (Menu item #8, #9, and #14 through #17) are automatically disabled. Loading these features is always required when the package is first entered, even if they already reside in memory. The message "PROGRAM NOT LOADED" will appear if the extensions are not loaded and the feature is selected.

The user is prompted for selection of a system utility as follows:

```
Enter program number (<1>):  _
```

Note that the default is utility number 1 (QUIT). Defaults for all prompts are indicated by bracketing with greater-than and less-than signs. All prompts will terminate with a colon (:), and user input will be ignored until this character appears. Note that one character is buffered, but will not be echoed until the prompt is displayed. The following subsections detail the operation of each function available via the main menu. These functions may have submenus; when this is the case, they are described. (Note: In some cases the only submenu item may be "QUIT". This is temporary; the expansion of these menus is planned.)

C.1.2.1 Quit (Menu Item 1)

Returns the user to the basic utility system.

C.1.2.2 Boot Operating System (Menu Item 2)

Provides the initial loader for the PICK or XENIX operating system. One sector is read from sector 0, cylinder 0, head 0 of disk unit 0 into memory. If this operation is successful, program execution continues at the start of the freshly loaded memory, which contains the operating system specific loader.

If unsuccessful, disk status is displayed and the user is given the opportunity to retry the boot, otherwise, the main menu reappears. The ADES Diagnostics may be used to determine and rectify the problem (Section C.1.2.13).

C.1.2.3 Back-Up Disk to Tape (Menu Item 3)

Provides the means to make multi-volume (unlabeled) cartridge (1/4-inch) tape backups from any of the four disks which may be connected to the ADES controller. The user is prompted for the disk unit number and the size of the disk is then determined. The number of 2K blocks to backup and approximate number of tapes required are displayed, followed by a prompt for the tape unit number. (You may use fewer than the number of tapes displayed to suit system requirements.) The user is prompted to insert the next tape and hit the return key, otherwise, the main menu reappears. The tape is retensioned and backup continues until end of tape or completion of the backup. Tape is then rewound causing a file mark to be written. The process continues until the backup is complete.

If unsuccessful, disk and tape status are displayed and the user is given the opportunity to retry the backup, otherwise, the main menu reappears. The ADES Diagnostics may be used to determine and rectify the problems.

C.1.2.4 Restore Tape to Disk (Menu Item 4)

Provides the means to make multi-volume (unlabeled) cartridge tape restores to any disk connected to the ADES controller. The user is prompted for the disk unit number and the size of the disk is then determined. The number of 2K blocks to restore and the approximate number of tapes required are displayed, followed by a prompt for the tape unit number. Then, as required, the user is prompted to insert the next tape and hit the return key, otherwise, the main menu reappears. The tape is retensioned and then restoration continues until a file mark or the completion of the restore. Tape is then rewound. The process continues until the restore is complete.

If unsuccessful, disk and tape status are displayed and the user is given the opportunity to retry the restore, otherwise, the main menu reappears. The ADES Diagnostics may be used to determine and rectify the problem.

C.1.2.5 Load Extensions from Serial Port B (Menu Item 5)

This utility is intended for use only by programmers who are developing new diagnostic extensions. It performs the most basic form of non-resident program loading since serial port B is used, which is available on the CPU board itself, and requires no additional hardware for the load other than a source of serial data in Motorola 'S' record format. No hardware or software handshaking is required up to the maximum rate of 9600 baud. The user is given the option to set the baud rate (default 9600) before the transfer. Other characteristics are set as originally initialized by the basic utility system. The user may exit after setting the baud rate but before the transfer is initiated, which provides an alternate means of setting port B baud rate. If the user continues with the download, type 1 through 3 records are input until a type 7 through 9 terminator record is received, or an input conversion error is detected. In any case, the actual number of bytes downloaded will be displayed prior to return to the main menu.

C.1.2.6 Load Extensions from Disk (Menu Item 6)

Provides the fastest means of non-resident program loading from any disk connected to the ADES controller. The user is prompted for the disk unit number and the first sector, starting at sector 1, cylinder 0, head 0 is read into memory. The revision number is verified, and if correct, the extensions are loaded.

If unsuccessful, disk status is displayed and the user is given the opportunity to retry the load or return to the main menu. The ADES Diagnostics may be used to determine and rectify the problem.

C.1.2.7 Load Extensions from Tape (Menu Item 7)

Provides the backup means of non-resident program loading from any tape connected to the ADES controller. Used to load new extensions from a distribution or backup tape into memory and from there to disk via the SAVE EXTENSIONS TO DISK utility. The user is prompted for the tape unit number, the tape is rewound and the first block is read into memory. The revision number is verified, and if correct, the remainder of the extensions are loaded.

If unsuccessful, tape status is displayed and the user is given the opportunity to retry the load or return to the main menu. The ADES Diagnostics may be used to determine and rectify the problem.

C.1.2.8 Save Extensions to Disk (Menu Item 8)

Saves extensions loaded into memory from another source onto any disk connected to the ADES controller. The user is prompted for the disk unit number.

If unsuccessful, disk status is displayed and the user is given the opportunity to retry the save or return to the main menu. The ADES Diagnostics may be used to determine and rectify the problem.

C.1.2.9 Save Extensions to Tape (Menu Item 9)

Saves extensions loaded into memory from another source onto any tape connected to the ADES controller. The user is prompted for the tape unit number and the tape is retensioned. When the save is complete, tape is rewound.

If unsuccessful, tape status is displayed and the user is given the opportunity to retry the save or return to the main menu. The ADES Diagnostics may be used to determine and rectify the problem.

C.1.2.10 CPU Tests (Menu Item 10)

Provides access to a submenu of CPU tests. These will continuously test various CPU hardware blocks to verify the hardware and determine its reliability over an extended period of time. The submenu for this feature is:

```
CPU TESTS

1. QUIT
2. RAM MEMORY
3. SEGMENT MAP
4. PAGE MAP
5. ALL
```

1. Quit - Returns the user to the main menu.

2. RAM Memory - Continuously and non-destructively exercises RAM by writing a user-selected test pattern (default $FFFF) and its complement, in word increments, throughout memory. The amount of memory tested is displayed. The user is given the option of halting on an error. Entering any character from the keyboard will terminate the test at the completion of the current pass.

3. Segment Map - Continuously and non-destructively maps two megabytes of logical address space into two megabytes of physical address space for each context proceeding from context 0 to context 15, and within each context from segment map entry 0 to segment map entry 63. The user is given the option of halting on an error. Entering any character from the keyboard will terminate the test at the completion of the current pass.

4. Page Map - Continuously and non-destructively maps all pages as non-accessed, non-used local memory. The user is given the option of halting on an error. Entering any character from the keyboard will terminate the test at the completion of the current pass.

5. All - Continously runs all the above tests in sequence until terminated by any character from the keyboard or by halting on an error (if that option has been selected).

C.1.2.11 CPU Diagnostics (Menu Item 11)

Provides access to a submenu of diagnostic functions that will test various CPU hardware blocks for the purpose of troubleshooting. The submenu for this feature is:

CPU DIAGNOSTICS

1. QUIT
2. RAM MEMORY
3. SEGMENT MAP
4. PAGE MAP
5. ALL

1. Quit - Returns the user to the main menu.

2. RAM Memory - Non-destructively verifies RAM by writing a user-selected pattern (default $FFFF) and its complement, in word increments, throughout memory. The amount of memory tested is displayed upon completion of the diagnostic.

3. Segment Map - Non-destructively verifies segment mapping capability by mapping two megabytes of logical address space into two megabytes of physical address space for each context proceeding from context 0 to context 15, and within each context from segment map entry 0 to segment map entry 63.

4. Page Map - Performs a rudimentary non-destructive verification of the page map by mapping all pages as non-accessed, non-used local memory.

5. All - Performs all the above diagnostic functions in sequence.

C.1.2.12 ADES Tests (Menu Item 12)

Provides access to a submenu of test functions that will continuously test various ADES hardware blocks to verify the hardware and determine its reliability over an extended period of time. The submenu for this feature is:

ADES TESTS

1. QUIT
2. DISK/TAPE EXERCISE

1. Quit - Returns the user to the main menu.

2. Disk/Tape Exercise - Exercises the ADES interface, controller, and the specified disk and tape drive(s). This test is destructive of data on the media and should only be perfomed after the disk is backed up and a scratch tape inserted. Each tape drive must have a corresponding disk drive (same unit number), but disk drives may be tested without tapes. The user is prompted for the fill byte, ID byte, and maximum number of retries. Then the user is prompted for disk unit 0. (Default is <no> meaning tape only.) If the user responds 'yes', the disk is to be tested and the user is prompted to test the corresponding tape unit 0. No tape unit is prompted for if the disk unit is not to be tested. This process is repeated for disk units 1, 2, and 3.

   Note that if the utility extensions are not loaded, the message "UNRECOGNIZED COMPLETION CODE" will appear. The message should be ignored.

The following is the test sequence:

1. Format disk pack(s).
2. Rewind tape(s).
3. Burst backup 255 2K blocks on tape(s).
4. Format track 0, head 0 with complement of user fill byte on disk(s).
5. Rewind tape(s).
6. Burst restore 255 2K blocks on disk(s).
7. Read the first four sectors on track 0, head 0 and compare with user fill byte used to format disk(s).

Each operation is completed on each unit before the next operation is begun. The unit number and operation are displayed before the operation is begun.

The completion code text is displayed after each operation. The program may be halted at any time by pressing any key on the terminal keyboard. This termination occurs only after the present operation is complete, thus in the case of a format pack operation, several minutes may pass before the termination is recognized. The character typed is otherwise ignored.

The total of each non-zero completion code is then displayed for each unit along with the last defect map. The total number of transfer errors (transfer count mismatch from format pack and read disk) and read compare errors (item 7) are also displayed. Disk units not enabled will show no completion codes and zero defects.

After the data on all units is displayed, the user has the option of continuing or exiting the test.

C.1.2.13 ADES Diagnostics (Menu Item 13)

Provides access to a submenu of diagnostic functions that will test various ADES hardware blocks for the purpose of troubleshooting or disk maintenance operations. Refer to ADES/GYPSY Interface (Appendix E) for additional information on ADES Diagnostics. The submenu for this feature is:

ADES DIAGNOSTICS

1. QUIT
2. WRITE COMMAND
3. WRITE COMMAND QUICK
4. WRITE DATA
5. READ DATA
6. READ STATUS
7. READ STATUS QUICK
8. INITIALIZE DEFECT MAP
9. FORMAT PACK
10. VERIFY PACK

1. Quit - Returns the user to the main menu.

2. Write Command - Allows the user to issue any command to the ADES controller. This command function is usually used for convenience, but it may be used when no explicit functions are available. The user is prompted for each command byte (default value is zero) and is given the opportunity to abort the command before it is actually output to the ADES controller. Since this function waits for all direct and transparent operations to complete before issuing the new command, it may be classified as a synchronous function. (All diagnostics are synchronous, except where indicated.) Note that a READ DATA or WRITE DATA function may be required to follow this function if the command requires data transfer. The READ STATUS or READ STATUS QUICK function may be used to properly terminate the operation.

3. Write Command Quick - This function is identical to the WRITE COMMAND function described above except the command is issued regardless of the present state of the ADES controller, and thus may be classified as an asynchronous function. When used with READ STATUS QUICK, it may be used to start a foreground (direct) task after issuing a background (transparent) task.

4. Write Data - This function is used after issuing a WRITE COMMAND or WRITE COMMAND QUICK, in order to fill a buffer with user data and output it to the ADES controller. The user is prompted for the buffer size (maximum and default is 4096 bytes), which must be an even hexadecimal value. Either data presently in the buffer (from the last read or write) or new data may be specified. New data is entered by specifying the number of words in a pattern (default buffer size) and supplying that number of hexadecimal words via the keyboard. This word pattern is then used to fill the remainder of the buffer. This function will write data to the ADES controller until 'busy' is removed, satisfying all data transfer requests even if more or less than the buffer size. If the transfer count does not match the buffer size, a warning is displayed indicating the actual number of transfers. For this reason, this function may not be used for bi-directional transfers (i.e,., FORMAT TRACK WITH FIFO DATA AND CRC VERIFY).

5. Read Data - This function is used after issuing a WRITE COMMAND or WRITE COMMAND QUICK, in order to read data from the ADES controller into a buffer. The number of bytes input (hexadecimal) is displayed followed by eight lines of eight hexadecimal words each, which display the contents of the buffer. The next block of 64 words may be displayed by pressing the carriage return. Once all the valid data has been displayed, blanks will be displayed. The user has the option to review the data as many times as required before leaving this function.

6. Read Status - This function allows the user to read the 15 ADES status bytes. It is used in conjunction with the previously defined diagnostic functions. If extended functions have been loaded, status byte 0 is decoded and messages are displayed for each active bit, and the completion code (status byte 1) is displayed as a message. Since this function waits for all direct and transparent operations to complete before reading status, it may be classified as a synchronous function. Note that a READ DATA or WRITE DATA function must be used before this function if the command requires data transfer.

7. Read Status Quick - This function is identical to the READ STATUS function described above except that status is read regardless of the present state of the ADES controller. It may thus be classified as an asynchronous function.

8. Initialize Defect Map - This function initializes the last few cylinders on the selected disk connected to the ADES controller. These cylinders are used by the ADES controller for the disk defect map. After the user enters the ID byte to be used, the command is issued and the defect map is read into the buffer. This takes approximately 10 seconds for a 35M byte disk after the disk has been sequenced up. Status is then read and the applicable registers are displayed. If the completion code indicates a fatal error, the function terminates at this point with an error message. Otherwise, the user is prompted to continue and the total defect count is displayed. If there were no defects, the function then terminates. If defects were found, the number of defects per head is displayed followed by a ten line listing of the defect head, cylinder, sector (displays ALL for track defects), and the alternate head, cylinder and sector. If more than ten defects were found, pressing the return key causes the next ten defects to be displayed, and so on. When all defects have been displayed, the user has the option to review the list from the start. Before exiting, the ADES controller is reset.

9. Format Pack - This function formats all but the last few cylinders on the selected disk. The user supplys the fill and ID bytes and the maximum number of retries, after which the command is issued and the defect map is read into the buffer. This takes approximately 7 minutes for a 35M byte disk after the disk has been sequenced up. The remainder of this function is identical to the INITIALIZE DEFECT MAP function.

10. Verify Pack - This function verifies all cylinders on the selected disk. This takes approximately 35 seconds for a 35M byte disk after the disk has been sequenced up. The remainder of this function is identical to the INITIALIZE DEFECT MAP function.

DISK TIMING

Submenu Item	20MB	35MB	70MB	158MB
8. INITIALIZE	5 sec.	10 sec.	13 sec.	70 sec.
9. FORMAT	4 min.	7 min.	12 min.	23 min.
10. VERIFY	30 sec.	35 sec.	70 sec.	90 sec.

C.1.2.14 Communication Tests (Menu Item 14)

This extended function provides access to a submenu of test functions that will continuously test various multi-port communications controller hardware blocks in order to verify the hardware and test its reliability over an extended period of time. The submenu for this feature is:

1. QUIT

1. Quit - Returns the user to the main menu.

C.1.2.15 Communication Diagnostics (Menu Item 15)

This extended function provides access to a submenu of diagnostic functions that will test various multi-port communications controller hardware blocks for the purpose of troubleshooting. The submenu for this feature is:

1. QUIT

1. Quit - Returns the user to the main menu.

C.1.2.16 CAPRO Tests (Menu Item 16)

This extended function provides access to a submenu of test functions that will continuously test various CAPRO magnetic tape controllers and drive hardware blocks in order to verify the hardware, and test its reliability over an extended period of time. The submenu for this feature is:

1. QUIT

1. Quit - Returns the user to the main menu.

C.1.2.17 CAPRO Diagnostics (Menu Item 17)

This extended function provides access to a submenu of diagnostic functions that will test various CAPRO magnetic tape controllers and drive hardware blocks for the purpose of troubleshooting. The submenu for this feature is:

1. QUIT

1. Quit - Returns the user to the main menu.

C.1.3 PROCEDURE

To follow this procedure, you must have model 70-01447A ROM which requires revision H or higher of the PIO GYPSY interface adapter board.

C.1.3.1 Loading Disk from 1/4" Tape

The following steps should be followed when loading a new operating system from tape:

1. Initialize Defect Map - Main Menu = 13, Submenu = 8.

2. Format Pack - Main Menu = 13, Submenu = 9.

3. Restore Tape to Disk - Main Menu = 4.

4. Verify Pack - Main Menu = 13, Submenu = 10.

C.1.3.2 Utility Extensions

The utility extensions can be provided on a separate tape or on the operating system tape.

The user must load the extensions into memory from the main menu (item 6 or 7). If the extensions are provided on a separate tape, the user may load extensions from tape (main menu = 7), and then save extensions to disk (main menu = 8).

At this time, the only feature in the extensions is a description of the "status byte" and "completion code" (see Sections C.1.4.1.1 and C.1.4.1.2), with the ADES messages. The Communication and CAPRO tests and diagnostics functions now only have one submenu item, "QUIT".

C.1.3.3 ZEBRA 1500 Winchester Disk (RODIME) Initialization

This procedure is necessary to record the characteristics of a 5-1/4" Winchester disk prior to any other operation. This procedure is necessary only when the drive is first placed in service and will normally be performed by General Automation when the system is assembled.

1. POWER UP the machine.

2. DEPRESS RESET

   (Jumper J2 Pin 1-2 not installed on CPU board).

3. After that, CRT will display:

   "GENERAL AUTOMATION ZEBRA UTILITIES-VERSION 3.0 256K BYTES OF RAM

   DIALOGUE 80 TERMINAl or equivalent (Y or N)

   TYPE: Y [CR]

   (The terminal will display a menu and request for entry.)

4. TYPE: 13 [CR] (ADES DIAGNOSTICS)

   (Now, a new menu (Diagnostics Menu) will be displayed.)

5. TYPE: 7 [CR] (READ STATUS QUICK)

   (You must read STATUS=80, COMPLETION=80 which means that a controller reset has occurred.)

   TYPE: N [CR]

6. TYPE: 2 [CR] (WRITE COMMAND)

   TYPE: (SEQ) A1 [CR]
   0 [CR]
   . .
   . .
   . .

   TYPE: [CR] until "ABORT COMMAND" message appears.

7. TYPE: 7 [CR] (READ STATUS QUICK)

   (Must read STATUS=90, COMPLETION=00).

   TYPE: N [CR]

8. TYPE: 2 [CR] (WRITE COMMAND)

```
TYPE: A8  [CR]
       0  [CR]   (0 = first drive; 1 = second drive)
       3  [CR]
       2  [CR]
      80  [CR]
       0  [CR]
       2  [CR]
      81  [CR]
       2  [CR]
      81  [CR]
       3  [CR]
      20  [CR]
       1  [CR]
       0  [CR]
       0  [CR]
          [CR]
```

9. TYPE: 7 [CR] (READ STATUS QUICK)

(Must read STATUS=90, COMPLETION=00)

TYPE: N [CR]

After the preceding steps have been completed, the operating system should be loaded as described in Section C.1.3.1.

C.1.4 EXTENDED STATUS MESSAGES

This section summarizes messages which can result from extended features loaded by main menu items #5, #6, and #7.

C.1.4.1 ADES Messages

C.1.4.1.1 SR 0 (Status Byte)

Bit	Description
0	Transparent command overrun
1	Direct command overrun
2	Transparent command complete
3	Transparent busy
4	Direct command complete
5	GYPSY busy
6	Not used
7	Data transfer request (active low)

C.1.4.1.2 SR 1 (Completion Code)

Code	Description
00	Good execution
01	Good execution; seek retry after seek fault error
02	Good execution; seek retry after cylinder mismatch error
04	Good execution; data retry after ID checksum error
05	Good execution; data retry after CRC error
08	Good execution; tape read operation complete, terminated by file mark
13	Drive seeking
18	Rewind; position command in progress
19	Erase tape; position command in progress
1A	Retention; position command in progress
1B	Advance file mark command in progress
1C	Transparent backup in progress
1D	Transparent restore in progress
20	Disk drive not present
21	Seek fault
22	Cylinder mismatch error
23	Sector not found
24	ID checksum error
25	Data CRC error
28	Disk write protected

Code	Description
30	Invalid GYPSY command
31	Invalid disk drive
32	Invalid head
33	Invalid cylinder
34	Invalid sector
36	Invalid drive ID
37	Invalid sector size
38	Disk (cylinder maximum) overrun
39	Disk fault
40	Bad sector detected by CRC verify routine
41	Defect exists
42	Defect map full
43	No defect map
44	No space for alternate
45	Defect not found
60	Tape drive not present
61	Tape cartridge write protected
62	File mark detected
63	Block in error not found
65	Hard data error
68	Cartridge not in place
69	End of tape
6A	Tape not online
6B	Reset occurred
6C	Bottom of tape
6D	No data detected
6E	Illegal command (to tape drive from GYPSY)
70	Invalid command with tape write mode set
71	Invalid command with tape read mode set
73	Tape not available (currently being used in transparent command)
80	Controller reset has occurred
F0	No command awaiting acknowledgement
F4	Transparent command not complete
F8	No transparent command in progress

C.1.4.1.3 SR 9 (Tape Exception Byte 0)

Code	Description
0	File mark detected
1	Block in error not found
2	Hard data error
3	End of tape
4	Tape write protected
5	Drive not online
6	Cartridge not in place
7	Exception flag (see other bits)

C.1.4.1.4 SR A (Tape Exception Byte 1)

Code	Description
0	Tape reset has occurred
1	Not used
2	Not used
3	Beginning of tape
4	Not used
5	No data detected
6	Illegal command
7	Exception flag (see other bits)

C.2 1500 - 5500 FIRMWARE EXECUTIVE

The ZEBRA 1500 through 5500 Series CPU board is equipped with EPROMs (see Figure C-1) containing the Firmware Executive. The Firmware Executive has a set of commands to aid the user in general operation, programming, system test, and diagnostics.

NOTE

Command descriptions and display examples in the following text are based upon July 1984 release of EPROMs 70A01561A01, A03, and 70A01562A02, A04.

Command descriptions will contain an underlined portion of keywords (Section C.2.2), indicating the minimum abbreviation that will be recognized. Any remaining portion of the keyword may be entered for clarity. Keywords are displayed in uppercase, but may be entered in upper or lowercase.

Bracketed portions of command formats indicate optional arguments. In most cases, these arguments may be entered in any order. A qualifier (byte, word, or long word) is indicated with a ´.q´ which will default to word size. Only long-word size is valid for address registers or register-addressed memory. Spaces shown must be entered, but the number of spaces does not matter as long as it is at least one. Multiple commands may be entered on one line when separated by a semicolon (;). Interactive commands may only appear at the end of a multiple command line. The backspace key performs a destructive backspace. The console bell will ring if an attempt is made to backspace past the beginning of input. The rubout/delete key performs a destructive backspace to the beginning of the input line. Commands will be listed in the order the command processor searches for them.

C.2.1 EXECUTIVE INITIALIZATION

Following a RESET, the Executive establishes the initial operating environment. During this process, configuration information is displayed, allowing the user to verify that all system hardware has been detected and properly initialized. The following example illustrates the display for a ZEBRA consisting of the 68000 CPU, one Winchester disk drive, one 1/4" cartridge tape drive and ten CPU system ports.

```
GENERAL AUTOMATION EXECUTIVE - VER n.n, P/N 1561-X

    CPU Type: 68000
    Local RAM: 256 KBytes
    Disk Drive 0: 138,852 KBytes*
    Disk Drive 1: No Drive Present*
    Cartridge Tape Drive 0: Drive Present
    Cartridge Tape Drive 1: No Drive Present
    Ports Present: 10

Enter BOOT, BACKUP or RESTORE

Ok,
```

In this example, the AUTO BOOT jumper (CPU board connector J2, pins 1 and 2) (see Figure C-1) is installed, causing automatic startup of the operating system, which will generate additional display information. If the jumper was not installed, or Disk Drive 0 was not detected, the Executive is entered and the 'Ok,' prompt is displayed. If no key is depressed within 10 to 15 seconds, AUTO BOOT will occur.

*Each disk must be ready before this message is written, and only one drive at a time is sequenced up. On a system with 128MB drive, this could take up to 2 minutes per drive. Doing so would prevent a boot, backup, restore, load, etc.

C.2.2 1500 - 5500 EXECUTIVE COMMANDS

The Executive commands are:

BACKUP
BOOT
COMMANDS or HELP or ?
CONNECT
CONTEXT
DIAGNOSTIC
DUMP
ERASE
GOTO
LOAD
MEMORY
PAGE
REGISTER
RESET
RESTORE
RETENSION
REWIND
SAVE
SEGMENT
SRECORD
SYSTEM

C.2.2.1 BACKUP Command

The BACKUP command provides for image backup of disk to 1/4" cartridge tapes (CT) or 1/2" magnetic tapes (MT). The user will be prompted with the following message:

```
Mount Tape N (y/n): _
```

N is the tape number to mount. If there is a problem with the tape, the user will be prompted with an error message and asked again. For cartridge tape, the completion code is displayed as a part of the error message. If there is a disk error, it is fatal and the completion code will be displayed and the monitor reentered. Note that the backup will not exceed the size of the disk, no matter what the FOR count specified is.

C.2.2.1.1 1/4" Cartridge Tape BACKUP

The BACKUP command format for cartridge tape is as follows:

BACKUP [unit] CT [unit] [FROM sector] [FOR blocks]

The unit has a default value of zero. The FROM sector address (relative sector number) has a default of zero (beginning of disk to backup). The FOR blocks is the number of 2K byte (4 sector) blocks to backup. The default number of blocks is the size of the disk. Note that the backup will become truncated to an integral multiple of the 4. Do not use this command to back up multiple disk systems that do not meet this requirement. It is not possible to write a partial block to the cartridge tape. Note that tapes are not automatically rewound or erased. The user should "ERASE" all tapes before beginning this procedure. Also, multiple disks may be backed up on one or more tapes.

C.2.2.1.2 1/2" Tape BACKUP

The BACKUP command format for 1/2" tape is as follows:

BACKUP [unit] MT [unit] [FROM sector] [FOR sectors] [BLOCK bytes]

The unit has a default value of zero. The FROM sector address (relative sector number) has a default of zero (beginning of disk to backup). The FOR sectors is the number of sectors to backup. The default number of sectors is the size of the disk. The BLOCK bytes is the number of bytes per tape block which may not be greater than 8000 (hex) and must be a multiple of sector size. The default is 4000 (hex) (16K bytes). A partial end block will be written for the residual. If the tape is write protected, offline, or not ready, an error message will be displayed and the user will be allowed to continue after rectifying the problem.

Each tape begins with a standard 80-character PICK label, which contains the block size, user label, and tape number. Note that the first tape is not automatically rewound at the start nor the last tape at the end, in order to allow multiple disk backups on one or more tapes. The user must REWIND the first tape.

All tapes are terminated with two filemarks. Note that this program uses RAM space for buffering and also relocates itself to RAM for maximum speed. This means you cannot depend on the contents of RAM after the operation. The number of recoverable write errors (retries) will be displayed after each tape has been rewound. If after eight retries, the tape block cannot be successfully written, an error message will be displayed with the contents of tape register 1. Note that 1/2" tape backup is two to four times faster than cartridge tape and much more reliable.

C.2.2.2 BOOT Command

The BOOT command is similar to the load command, except BOOT begins execution of the data after the load. The command format is as follows:

```
BOOT [DISK] [unit]
BOOT CT [unit]
BOOT MT [unit]
```

Additional text will appear as the operating system starts up. Note that the boot-up default occurs from the first sector of disk 0 and reads in a single sector operating system loader.

C.2.2.3 COMMANDS (?, HELP) Command

The COMMANDS (?, HELP) command displays the valid commands for a particular version of the Executive, as the command set is selected as assembly time and may vary from one application to another. The full command names are listed in uppercase from left to right, top to bottom in the order they are searched for. This allows the determination of the minimum abbreviation for a command. The command format is as follows:

```
COMMANDS
?
HELP
```

The following is an example of the listing of the full standard command set. Additional commands would probably appear at the end for a particular application.

```
? BOOT BACKUP COMMANDS CONNECT CONTEXT DIAGNOSTIC DUMP ERASE
GOTO HELP LOAD MEMORY PAGE REGISTER RESE RESET RESTORE
RETENSION REWIND SAVE SEGMENT SRECORD SYSTEM
```

Note the RESE command. This command would result in a ´Not Found" ´ message. This is because this command was included to force the full ´RESET´ command to be entered with no abbreviation. There might be other such instances of dummy commands, depending on the application.

C.2.2.4 CONNECT Command

The CONNECT command provides for transparent communication between the console and host ports. An optional termination string may be specified. Termination will occur when the termination string or an enabled break condition is received from the console or the host. The entire termination string is sent to the destination before termination. The command format is as follows:

CONNECT [terminator]

C.2.2.5 CONTEXT Command

The CONTEXT command allows examination and modification of the Context Register. The Context Register is a four-bit register that selects 1 of 16 unique sections of the Segment Map. If no hexadecimal context is entered, the current context is displayed and a new context may be entered or, if nothing is entered (return only), no change is made. If a context is entered when the command is entered, the context is changed but not displayed. The command format is as follows:

CONTEXT [number]

C.2.2.6 DIAGNOSTIC Command

The DIAGNOSTIC command invokes the diagnostics package. Refer to ZEBRA Hardware Reference Manual, 88A00775A, for menu and descriptions of diagnostics. The command format is as follows:

DIAGNOSTIC

Entry of the top menu item (QUIT) of Diagnostics returns control to the Executive.

C.2.2.7 DUMP Command

The DUMP command will display in hex and ASCII format 16 bytes per line. The hex format may be either byte, word, or long word (default word). If the terminating condition is not entered, 16 lines (256 bytes) are displayed followed by a colon (:) each time the return key is entered. Any other character followed by a return will terminate the command. The terminating condition may be either FOR a count or UNTIL an address. The command format is as follows:

```
DUMP[.q]address
DUMP[.q]address FOR count
DUMP[.q]address UNTIL address
```

An example of a word display line format follows:

```
00001000  4455 4D50 2057 4F52 4420 4C49 4E45 0D0C  DUMP WORD LINE...
```

Periods are displayed for the non-printing ASCII codes. The display always consists of 16 bytes of data per line.

C.2.2.8 ERASE Command

The ERASE command will erase data from a 1/4" cartridge tape. The tape will then be positioned at BOT. The command format is:

```
ERASE [CT]
```

This command is valid only for cartridge tape.

C.2.2.9 GOTO Command

The GOTO command is used to initiate a user program under Executive control. If the program has not already been run and terminated by the Executive normally, the address must be specified. Otherwise, the state at the time of the program break is restored. After reset, the Executive initializes the user registers to all ones (1's), the supervisor and user stack pointers to the top of local memory, and the status register with 2700 hex. The command format is as follows:

```
GOTO [address]
```

C.2.2.10 LOAD Command

The LOAD command is used to read data into memory from disk, 1/4" cartridge tape (CT), or 1/2" magnetic tape (MT). The command format is as follows:

```
LOAD [DISK] [unit] [TO address] [FROM address] [FOR sectors]
LOAD CT [unit] [TO address] [FOR blocks]
LOAD MT [unit] [TO address] [FOR bytes] [BLOCK bytes]
```

The unit has a default value of zero and the default device is the disk. The TO address has a default of 31000 (hex). The address is set in the user's 'PC' register so that the start of the loaded memory may be jumped to with a simple GOTO command. The FROM sector address has a default of zero and is only applicable to disk. The FOR sectors/blocks/bytes is the number of sectors/ blocks/bytes to load. Block size is defined as the block size of the source device, which is one disk sector (512 bytes) or CT block (2048 bytes) or MT bytes. It is possible to read a partial block into memory only from MT. The default FOR count is one (1) except for MT, whose default is to the next filemark. Magnetic tapes must have standard PICK labels. Tapes must be rewound or retensioned before use.

C.2.2.11 MEMORY Command

The MEMORY command is used to display and modify memory data anywhere in the address space in byte, word, or long-word format. If the address is not entered, zero (0) is assumed. The default size is word. The command format is as follows:

```
MEMORY[.q] [address]
```

This is an interactive command. The address and value are displayed and the processor waits for user input. If no value is entered, the location is not changed and the next address and value are displayed. If a slash (/) is entered without a value or after a value, the previous address and value are displayed. If a comma (,) is entered without a value or after a value, the present address and value are redisplayed. If a period (.) is entered without a value or after a value, the command terminates. If an at sign (@) is entered without a value or after a value, the long word at the present address is the next address. A qualifier may follow the at sign. An example of the display format follows:

```
Ok, MEMORY.L 2000
  00002000=001F0800: 001F0000
  00002004=142135F9: /
  00002000=001F0000: ,
  00002000=001F0000: @.B
  001F0000=17: .
Ok,
```

C.2.2.12 PAGE Command

The PAGE command allows examination and modification of the Page Map. If the page number is not part of the command, page zero (0) is assumed. Each page is displayed and the processor then waits for console input. If a value is entered, that value is stored and the next page is displayed. If nothing is entered (return only), no change is made to the data. If a period (.) is entered, the processor is exited. If a period is entered after a value, the processor is exited after the new value is stored. If a slash (/) is entered, the previous page is displayed. If a slash is entered after a value, the previous page is displayed after the value is stored. If a comma (,) is entered, the same page is displayed. If a comma is entered after a value, the page is displayed after the value is entered. An example of the command and its effects follows:

```
Ok, PAGE 3DE
  Page 3DE=21DE, L=1EF000, XX, Multibus Mem, P=0EF000:
  Page 3DF=21DF, L=1EF800, XX, Multibus Mem, P=0EF800:
  Page 3E0=F000, L=1F0000, UD, Multibus I/O, P=000000:
Ok,
```

In the above example, the address and contents of the page are displayed. The 'L=' displays the associated logical memory address. Note that this logical address actually corresponds to the segment map physical address. In the first two lines, the 'XX' indicates that the 'Used' and 'Dirty' bits are false, but in the third line they are true (UD), indicating that that page has been written to. Next, the type of memory assignment is displayed. Finally, the 'P=' displays the physical address of that memory assignment.

C.2.2.13 REGISTER Command

The REGISTER command is used to display or alter the contents of the user registers. If no register is entered, all registers and exception information are displayed and the command terminates. If a register name is entered, that is the first register displayed for modification. The command format is as follows:

```
REGISTER [register]
```

This is an interactive command. The register and value are displayed and the processor waits for user input. If no value is entered, the register is not changed and the next register and value are displayed. If a slash (/) is entered without a value or after a value, the previous register and value are displayed. If a period (.) is entered without a value or after a value, the command terminates. If an at sign (@) is entered without a value or after a value, the long-word value in the register is used as the next address in the MEMORY mode. A qualifier may follow the at sign. Exception information cannot be altered. If the last register (PC) or first register (D0) is reached, wraparound is performed.

An example of the display format follows:

```
Ok, REGISTER D0
  D0=FFFFFFFF: 0/
  PC=00002000: @.W
  00002000=4E71: .
Ok,
```

The registers displayed when no register name is entered include any special registers as a result of an exception. The registers displayed also depend on the processor type (68000 or 68010). The following is a list of all register mnemonics:

D0-D7	32-Bit Data Registers
A0-A6	32-Bit Address Registers
A7	32-Bit Supervisor Stack Pointer
US	32-Bit User Stack Pointer
SR	16-Bit Status Register
PC	32-Bit Program Counter
V0	16-Bit Vector Offset (68010 only)
SS	16-Bit Special Status Register (Bus/Address Exception only)
AA	32-Bit Access Address(Bus/Address Exception only)
OB	16-Bit Output Buffer (68010 Bus/Address Exception only)
IB	16-Bit Input Buffer (68010 Bus/Address Exception only)
IR	16-Bit Instruction Register (Bus/Address Exception only)

C.2.2.14 RESET Command

The RESET command reinitializes the Executive. This is different from a hardware reset. The purpose of this command is to reset the serial ports and Executive variables without destroying the contents of memory. The command may not be abbreviated. The command format is as follows:

RESET [DT] [PORTS] [VECTORS]

The DT option resets only the disk/tape (not 1/2" tape) subsystem. The PORTS option resets the Executive serial ports. The VECTORS option reinitializes the exception vector table. If no options are specified, a complete reset is performed.

C.2.2.15 RESTORE Command

The RESTORE command provides for image restore of disk from 1/4" cartridge tapes (CT) or 1/2" tapes (MT). The user will be prompted with the following message:

```
Mount Tape N (y/n): _
```

N is the tape number to mount. After entry, the tape label will be displayed. If there is a problem with the tape, the user will be prompted with an error message and asked again. For cartridge tape, the completion code is displayed as a part of the error message. If there is a disk error, it is fatal and the completion code will be displayed and the Executive reentered. Note that the restore will not exceed the size of the disk, no matter what the FOR count specified is.

C.2.2.15.1 1/4" Cartridge Tape RESTORE

The RESTORE command format for cartridge tape is as follows:

RESTORE [unit] CT [unit] [TO sector] [FOR blocks]

The unit has a default value of zero. The TO sector address (relative sector number) has a default of zero (beginning of disk to restore). The FOR blocks is the number of 2K byte (4 sector) blocks to restore. The default number of blocks is the size of the disk.

C.2.2.15.2 1/2" Tape RESTORE

The RESTORE command format for 1/2" tape is as follows:

RESTORE [unit] MT [unit] [TO sector] [FOR blocks] [BLOCK bytes]

The unit has a default value of zero. The TO sector address (relative sector number) has a default of zero (beginning of disk to restore). The FOR blocks is the number of blocks to restore. The default number of blocks is the size of the disk. The BLOCK bytes is the maximum tape block size which may not be greater than 8000 (hex) and must be a multiple of sector size. The default is 4000 (hex) (16K bytes). A partial end block will be read for the residual. If the tape is offline or not ready, an error message will be displayed and the user will be allowed to continue after rectifying the problem.

Each tape begins with a standard 80-character PICK label, which contains the block size, user label, and tape number. Note that the first tape is not automatically rewound at the start nor the last tape at the end, in order to allow multiple disk backups on one or more tapes. The user must REWIND the first tape.

If the tape number is incorrect or the block size too large, an error message will be displayed and the Executive reentered. Note that this program uses RAM space for buffering and also relocates itself to RAM for maximum speed. This means you cannot depend on the contents of RAM after the operation. The number of recoverable read errors (retries) will be displayed after each tape has been rewound. If, after eight retries, the tape block cannot be successfully read, an error message will be displayed with the contents of tape register 1.

C.2.2.16 RETENSION Command

The RETENSION command will retension the 1/4" cartridge tape. The tape will then be positioned at BOT. The command format is:

RETENSION [unit] CT

This command is valid only for cartridge tape.

C.2.2.17 REWIND Command

The REWIND command will rewind the 1/4" cartridge tape or 1/4" mag tape. The tape will then be positioned at BOT. The command format is:

REWIND [unit] CT or MT

C.2.2.18 SAVE Command

The SAVE command is used to write out to disk or cartridge tape data from memory. The command format is as follows:

SAVE [DISK] [unit] [FROM address] [TO sector] [FOR sectors]
SAVE CT [unit] [FROM address] [FOR blocks]
SAVE MT [unit] [FROM address] [FOR bytes] [BLOCK bytes]

The unit has a default value of zero and the default device is the disk. The FROM address has a default of 31000 (hex). The TO sector address has a default of zero and is only applicable to disk. The FOR sectors/blocks/bytes is the number of sectors/blocks/bytes to save. Block size is defined as the block size of the destination device, which is one disk sector (512 bytes) or a CT block (2048 bytes) or MT bytes. It is possible to write a partial block from memory only to MT.

C.2.2.19 SEGMENT Command

The SEGMENT command allows examination and modification of the Segment Map. If the segment number is not part of the command, segment zero (0) is assumed. Each segment is displayed and the processor then waits for console input. If a value is entered, that value is stored and the next segment is displayed. If nothing is entered (return only), no change is made to the data. If a period (.) is entered, the processor is exited. If a period is entered after a value, the processor is exited after the new value is stored. If a slash (/) is entered, the previous segment is displayed. If a slash is entered after a value, the previous segment is displayed. If a slash is entered after a value, the previous segment is displayed after the value is stored. If a comma (,) is entered, the same segment is displayed after the value is entered. An example of the command and its effects follows:

```
Ok, SEGMENT 20
  Segment 20=F20, L=100000, S=rwx, U=rwx, P=100000:
  Segment 21=F21, L=108000, S=rwx, U=rwx, P=108000:
  Segment 22=F22, L=110000, S=rwx, U=rwx, P=110000:
Ok,
```

In the above example, the address and contents of the segment are displayed. Note that the address and contents are for the present context. The 'L=' displays the associated logical memory address. The 'S=' and 'U=' display the protection mode for the supervisor and user mode respectively (r = read access, w = write access, and x = execution access). Finally, the 'P=' displays the physical address that drives the page map.

C.2.2.20 SRECORD Command

The SRECORD command is used to load, via the host unit, standard Motorola 'S' record format absolute load modules. All Motorola record types are recognized. An error will cause an error message to be displayed, but loading will continue until terminated by the end record or a user break. It might be desirable to copy host input to the console to see the records received. This is done with the 'HOST INPUT=console' command. If the rest of the command line is not null, the remainder of the line is sent to the host. This might be used to initiate the transfer. The command format is as follows:

SRECORD [text]

All three address size formats (16, 24, and 32) are recognized. Further, the head record, record count record, and end record cause a message to be printed to the console as shown in the following example:

```
Ok, SRECORD cat main.sav
  Header Record Received:  HDR
  Records Read:  625
  End Record, Bytes Read:  12542, Start Address:  00020000
Ok,
```

Note that the string 'cat main.sav' is sent to the host in order to initiate the transfer. This particular example is typical of a XENIX host. The start address is stored in the user's program counter (PC).

C.2.2.21 SYSTEM Command

The SYSTEM command works the same as the REGISTER command with no arguments. The register values displayed are those at the last exception when in the Executive mode (user program not running). If there has been no exception, nothing is displayed. Registers may not be altered. This function is mainly used to determine the cause of Executive faults. The command format is as follows:

SYSTEM

C.3 ZEBRA 700/750 FIRMWARE EXECUTIVE

The ZEBRA motherboard is equipped with EPROMs containing Executive and Diagnostics firmware. The Firmware Executive has a set of commands to aid the user in general operation, programming, system test. Diagnostics, which is called by the Executive, invokes the testing of selected ZEBRA subsystem.

NOTE

Command descriptions and display examples in this section are based upon Executive EPROMs 70A01563A01, 02 and Diagnostic EPROMs 70A01304A01, 02 shown in Figure C-2.

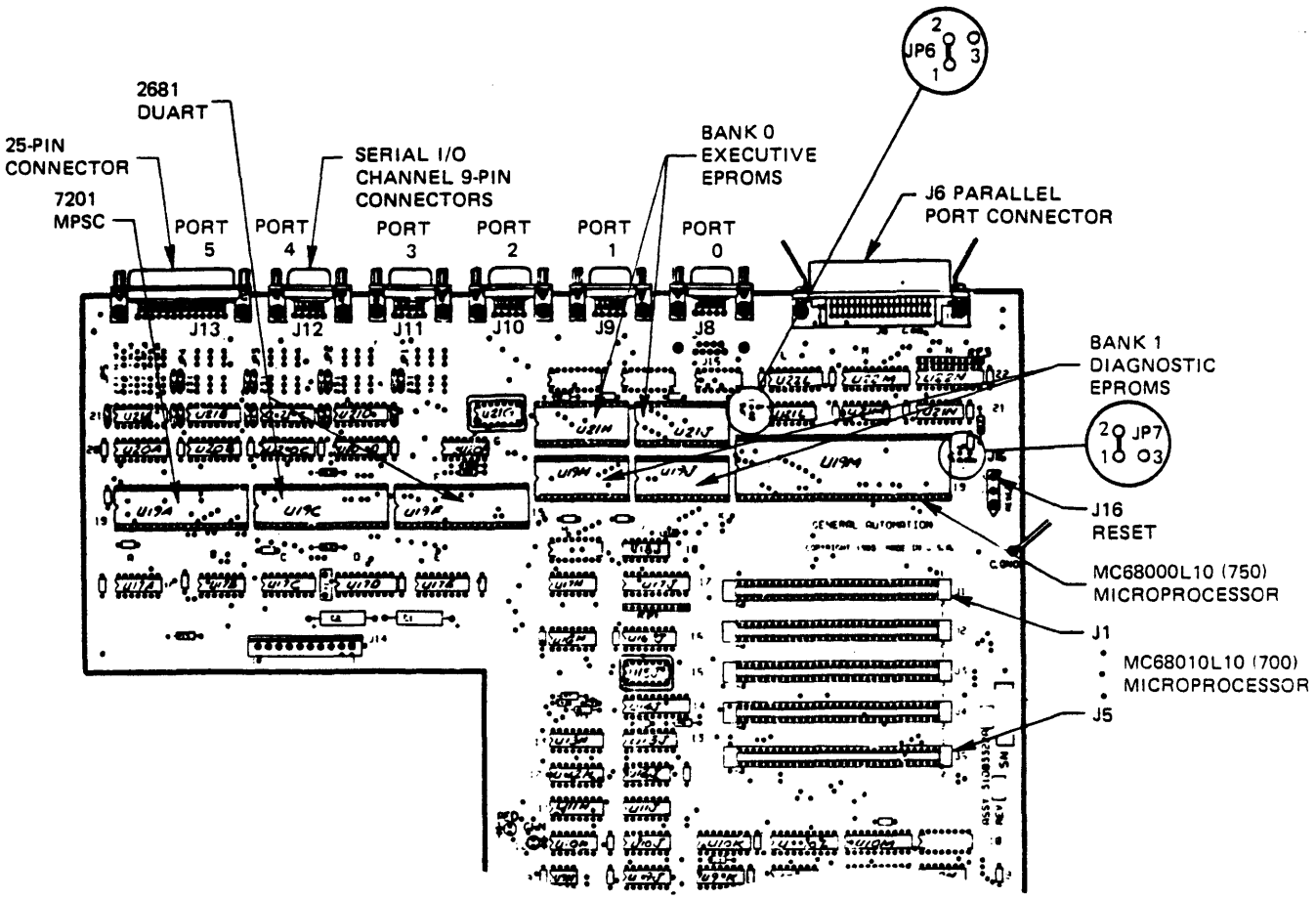


Figure C-2. 700/750 Motherboard - 31P03322A03

Underlined text, shown in interactive command examples, is generated by the Executive. Keywords are displayed in uppercase, but may be entered in upper or lowercase.

Bracketed portions of command formats indicate optional arguments. In some cases, these arguments may be entered in any order. A qualifier (byte, word, or long word) is indicated with a ´.q´ which will default to word size. Only long-word size is valid for address registers or register-addressed memory. Spaces shown must be entered, but the number of spaces does not matter as long as it is at least one. Multiple commands may be entered on one line when separated by a semicolon (;). Interactive commands may only appear at the end of a multiple command line. The backspace key performs a destructive backspace. The console bell will ring if an attempt is made to backspace past the beginning of input. The rubout/delete key performs a destructive backspace to the beginning of the input line.

In commands referencing a SASI device, the DU (Device/Unit) specification is in the form ´device,unit´. The device is the address of the particular SASI controller and the unit is the Logical Unit Number (LUN) of the device attached to that controller. Devices may have a number from 0 to 7 (default 0). The range of logical unit numbers depends upon the controller. If only one number is entered, it is assumed to be the unit. The DU specification must follow the device specification (e.g., DISK, CD, CT).

C.3.1 EXECUTIVE INITIALIZATION

Following a RESET, the Executive establishes the initial operating environment. A sign-on message will then be displayed:

```
GENERAL AUTOMATION EXECUTIVE - VER n.n., P/N 1563-X

Enter BOOT, BACKUP or RESTORE

Ok, _
```

"VER" identifies the latest release of the Executive; "nnnn" will be the amount of RAM existing in the system. If no keyboard input is entered within 30 seconds following Ok, automatic boot (AUTO BOOT) will take place.

C.3.2 EXECUTIVE COMMANDS

Commands may be entered with an underlined portion of keywords, indicating the minimum abbreviation that will be recognized. Any remaining portion of the keyword may be entered for clarity. A summary of the Executive commands is:

BACKUP
BOOT
COMMANDS or HELP or ?
CONNECT
CONTEXT
DIAGNOSTICS
DUMP
ERASE
FORMAT
GOTO
LOAD
MEMORY
PAGE
REGISTER
RESET
RESTORE
RETENSION
REWIND
SAVE
SEGMENT
SRECORD
SYSTEM

C.3.2.1 BACKUP Command

The BACKUP command provides for image backup of disk to another disk (DISK), 1/4" cartridge tape (CT) or cartridge disk (CD). If removable media is used, the user will be prompted with the following message:

```
Mount Cartridge N (y/n): _
```

N is the cartridge number to mount. If there is an error, the error code and logical address will be displayed and the Executive reentered. Note that the backup will not exceed the size of the disk, no matter what the FOR count specified is. For either fixed or cartridge disk, the first eight sectors will be skipped as they contain information particular to that disk.

C.3.2.1.1 1/4" Cartridge Tape BACKUP

The BACKUP command format for cartridge tape is as follows:

BACKUP [unit] CT [unit] [DU device, lun] [FROM sector] [FOR sectors]

The unit has a default value of 0 for the source (disk) and 0 for the destination (tape). The FROM sector address (relative sector number) has a default of 8 (beginning of disk to backup). The FOR sectors is the number of sectors to backup. The default number of sectors is the size of the disk.

C.3.2.1.2 Fixed or Cartridge Disk BACKUP

The BACKUP command format for fixed disk or cartridge disk is as follows:

BACKUP [unit] CD [unit] [DU device, lun] [FROM sector] [TO sector] [FOR sectors]
BACKUP [unit] DISK [unit] [DU device, lun] [FROM sector] [TO sector] [FOR sectors]

The unit has a default value of 0 for the source (disk) and 0 for the destination (disk or cartridge disk). The FROM sector address (relative sector number) has a default of 8 (beginning of disk to backup). The FOR sectors is the number of sectors to backup. The default number of sectors is the size of the disk.

C.3.2.2 BOOT Command

BOOT uses the same format as LOAD (Section C.3.2.11), but will immediately execute the loaded code upon completion.

C.3.2.3 COMMANDS (?, HELP) Command

The COMMANDS (?, HELP) command displays the valid commands for a particular version of the Executive, as the command set is selected at assembly time and may vary from one application to another. The full command names are listed in uppercase from left to right, top to bottom in the order they are searched for. This allows the determination of the minimum abbreviation for a command. The command format is as follows:

COMMANDS
?
HELP

The following is an example of the listing of the full standard command set. Additional commands would probably appear at the end for a particular application.

```
? BACKUP BOOT COMMANDS CONNECT CONTEXT DIAGNOSTIC DUMP ERASE
FORMAT GOTO HELP LOAD MEMORY PAGE REGISTER RESE RESET RESTORE
RETENSION REWIND SAVE SEGMENT SRECORD SYSTEM
```

Note the RESE command. This command would result in a 'Not Found" ' message. This command was included to force the full 'RESET' command to be entered with no abbreviation, THUS AVOIDING AN ACCIDENTAL SYSTEM RESET. There may be other such instances of dummy commands, depending on the application.

C.3.2.4 CONNECT Command

The CONNECT command provides for transparent communication between the console and host ports. An optional termination string may be specified. Termination will occur when the termination string or an enabled break condition is received from the console or the host. The entire termination string is sent to the destination before termination. The command format is as follows:

CONNECT [terminator]

C.3.2.5 CONTEXT Command

The CONTEXT command allows examination and modification of the Context Register. It is valid only for ZEBRA 700 systems containing an MMU board. The Context Register is a four-bit register that selects 1 of 16 unique sections of the Segment Map. If no hexadecimal context is entered, the current context is displayed and a new context may be entered or, if nothing is entered (return only), no change is made. If a context is entered when the command is entered, the context is changed but not displayed. The command format is as follows:

CONTEXT [number]

C.3.2.6 DIAGNOSTIC Command

The DIAGNOSTIC command invokes the diagnostics package. Refer to ZEBRA Hardware Referance Manual, 88A00775A, for menu and descriptions of diagnostics. The command format is as follows:

DIAGNOSTIC

C.3.2.7 DUMP Command

The DUMP command will display in hex and ASCII format 16 bytes per line. The hex format may be either byte, word, or long word (default word). If the terminating condition is not entered, 16 lines (256 bytes) are displayed followed by a colon (:) each time the return key is entered. Any other character followed by a return will terminate the command. The terminating condition may be either FOR a count or UNTIL an address. The command format is as follows:

DUMP[.q]address
DUMP[.q]address FOR count
DUMP[.q]address UNTIL address

An example of a word display line format follows:

00001000 4455 4D50 2057 4F52 4420 4C49 4E45 0D0C DUMP WORD LINE..

Periods are displayed for the non-printing ASCII codes. The display always consists of 16 bytes of data per line.

C.3.2.8 ERASE Command

The ERASE command will erase data from a 1/4" cartridge tape. The tape will then be positioned at BOT. The command format is:

ERASE [CT]

This command is valid only for cartridge tape.

C.3.2.9 FORMAT Command

The FORMAT command is used to configure and format the selected disk drive connected to the OMTI disk controller. The command has two formats, one for new disk types and one for known disk types.

```
FORMAT device [DISK CD] [unit] [WIDTH number] [PERIOD number]
  [MODE number] HEADS number CYLINDERS number [REDUCE number]
  TYPE number SECTORS number [INTERLEAVE number]

FORMAT device unit MODEL number [modifiers]
```

The default unit is 0. The step pulse width may be from 0 to FF. The default step pulse width is 2 (microseconds). The step period may be from 0 to FF. The default step period is 1 (50 microseconds). The step mode may be from 0 to 2. The default is 0 (buffer stepping). The number of heads may be from 1 to 7F. The number of cylinders may be from 1 to 7FFF. The cylinder at which to reduce write current may be from 0 to FF. The default cylinder at which to reduce write current is 0 (no reduced current). The type may be from 0 to 7. The number of sectors may be from 1 to 7F. The default interleave is 2. The interleave must be less than the number of sectors.

Standard configurations for different manufacturers' disk models have the following assignments:

Model	Manufacturer	Sectors	Heads	Cylinders	Capacity
0	IOMEGA BETA-5	52	1	394(7)	5,151,744
1	IMI 5018	32	6	306(6)	14,745,600
2	RODIME 202	32	4	320(6)	10,289,152
3	RODIME 202E	32	4	640(12)	20,578,304
4	RODIME 204E	32	8	640(12)	41,156,608

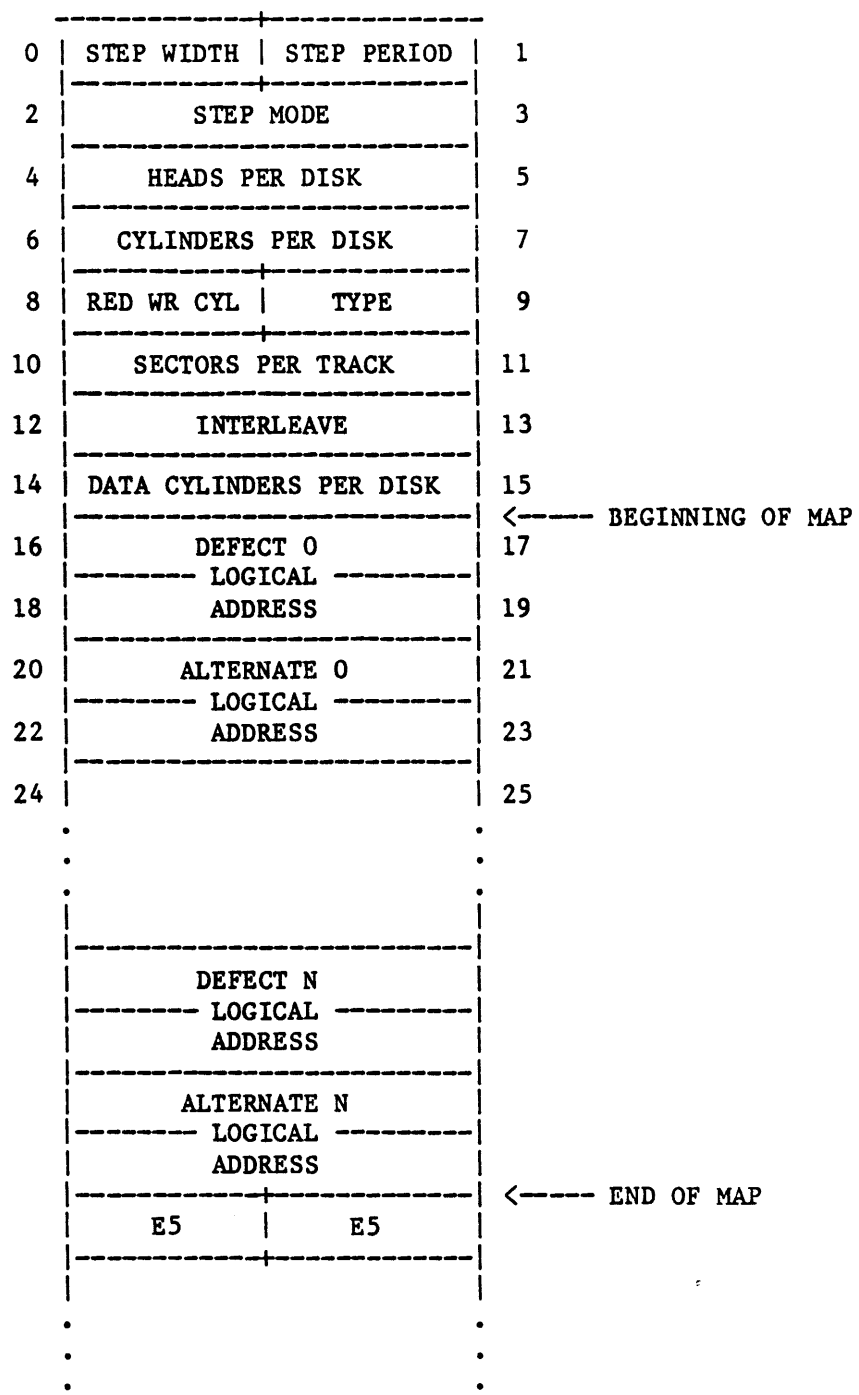
Defective track alternates are mapped to the last cylinders on the disk. the number of reserved cylinders appears in parentheses after the number of cylinders. The maximum number of data cylinders is the total cylinders minus the reserved cylinders. The number of reserved cylinders is 2% of the number of cylinders (rounded down) or 254 tracks, whichever is less.

If the model is specified first, its characteristics may be used for defaults and new values substituted.

During formatting, the defective head and track numbers are displayed. When formatting is completed, the number of defective tracks is displayed and the user is prompted to enter additional tracks from the manufacturer's defect list. The format of the response is defined as follows:

```
Add Defect (Head, Cylinder):  [!] head cylinder
```

The user responds with the head and cylinder number of the additional defective track. If the numbers are preceded with an exclamation point (!), they are taken to be decimal. A null input will terminate the defect list and all defective tracks are then remapped. After mapping, a 2048-byte configuration table/defect map are written into the first sector on track 0, head 0. The format of the disk record is defined on the following page.


0
STEP WIDTH
STEP PERIOD
1
2
STEP MODE
3
4
HEADS PER DISK
5
6
CYLINDERS PER DISK
7
8
RED WR CYL
TYPE
9
10
SECTORS PER TRACK
11
12
INTERLEAVE
13
14
DATA CYLINDERS PER DISK
15
<----- BEGINNING OF MAP
16
DEFECT 0
LOGICAL
17
18
ADDRESS
19
20
ALTERNATE 0
LOGICAL
21
22
ADDRESS
23
24
25
DEFECT N
LOGICAL
ADDRESS
ALTERNATE N
LOGICAL
ADDRESS
<----- END OF MAP
E5
E5

C.3.2.10 GOTO Command

The GOTO command is used to initiate a user program under Executive control. If the program has not already been run and terminated by the Executive normally, the address must be specified. Otherwise, the state at the time of the program break is restored. After reset, the Executive initializes the user registers to all ones (1's), the supervisor and user stack pointers to the top of local memory, and the status register with 2700 hex. The command format is as follows:

GOTO [address]

C.3.2.11 LOAD Command

The LOAD command is used to read in from disk or cartridge tape data into memory. The command format is as follows:

LOAD [DISK] [unit] [DU device, lun] [TO address] [FROM address]
[FOR sectors]
LOAD CD [unit] [DU device, lun] [TO address] [FROM address] [FOR sectors]
LOAD CT [unit] [DU device, lun] [TO address] [FOR sectors]

The default load device is the disk (default unit 0). The default cartridge tape or disk unit is 0. The TO address has a default of 8000 (hex). The address is set in the user's "PC" register so that the start of the loaded memory may be jumped to with a simple GOTO command. The FROM sector address has a default of 8 and is only applicable to disk. The FOR sectors is the number of sectors to load with default of 2.

NOTE: The BOOT command format is identical to the above, with display of "BOOT" rather than "LOAD". BOOT will immediately execute following completion of the LOAD.

C.3.2.12 MEMORY Command

The MEMORY command is used to display and modify memory data anywhere in the address space in byte, word, or long word format. If the address is not entered, zero (0) is assumed. The default size is word. The command format is as follows:

```
MEMORY[.q] [address]
```

This is an interactive command. The address and value are displayed and the processor waits for user input. If no value is entered, the location is not changed and the next address and value are displayed. If a slash (/) is entered without a value or after a value, the previous address and value are displayed. If a comma (,) is entered without a value or after a value, the present address and value are redisplayed. If a period (.) is entered without a value or after a value, the command terminates. If an at sign (@) is entered without a value or after a value, the long word at the present address is the next address. A qualifier may follow the at sign. An example of the display format follows:

```
Ok, MEMORY.L 2000
  00002000=001F0800: 001F0000
  00002004=142135F9: /
  00002000=001F0000: ,
  00002000=001F0000: @.B
  001F0000=17: .
Ok,
```

C.3.2.13 PAGE Command

The PAGE command allows examination and modification of the Page Map. It is valid only for ZEBRA 700 system with MMU board. If the page number is not part of the command, page zero (0) is assumed. Each page is displayed and the processor then waits for console input. If a value is entered, that value is stored and the next page is displayed. If nothing is entered (return only), no change is made to the data. If a period (.) is entered, the processor is exited. If a period is entered after a value, the processor is exited after the new value is stored. If a slash (/) is entered, the previous page is displayed. If a slash is entered after a value, the previous page is displayed after the value is stored. If a comma (,) is entered, the same page is displayed. If a comma is entered after a value, the page is displayed after the value is entered. An example of the command and its effects follows:

```
Ok, Page 3DE
  Page 03F=003F, L=01F800, XX, Multibus Memory, P=01F800:
  Page 040-1000, L=020000, XX, Invalid Memory, P=000000:
Ok,
```

In the above example, the address and contents of the page are displayed. The 'L=' displays the associated logical memory address. Note that this logical address actually corresponds to the segment map physical address. In the first two lines, the 'XX' indicates that the 'Used' and 'Dirty' bits are false, but in the third line they are true (UD), indicating that that page has been written to. Next, the type of memory assignment is displayed. Finally, the 'P=' displays the physical address of that memory assignment.

C.3.2.14 REGISTER Command

The REGISTER command is used to display or alter the contents of the user registers. If no register is entered, all registers and exception information are displayed and the command terminates. If a register name is entered, that is the first register displayed for modification. The command format is as follows:

REGISTER [register]

This is an interactive command. The register and value are displayed and the processor waits for user input. If no value is entered, the register is not changed and the next register and value are displayed. If a slash (/) is entered without a value or after a value, the previous register and value are displayed. If a period (.) is entered without a value or after a value, the command terminates. If an at sign (@) is entered without a value or after a value, the long word value in the register is used as the next address in the MEMORY mode. A qualifier may follow the at sign. Exception information cannot be altered. If the last register (PC) or first register (D0) is reached, wraparound is performed. An example of the display format follows:

```
Ok, REGISTER D0
  D0=FFFFFFFF: 0/
  PC=00002000: @.W
  00002000=4E71: .
Ok,
```

The registers displayed when no register name is entered include any special registers as a result of an exception. The registers displayed also depend on the processor type (68000 or 68010). The following is a list of all register mnemonics:

D0-D7	32 Bit Data Registers
A0-A6	32 Bit Address Registers
A7	32 Bit Supervisor Stack Pointer
US	32 Bit User Stack Pointer
SR	16 Bit Status Register
PC	32 Bit Program Counter
V0	16 Bit Vector Offset (68010 only)
SS	16 Bit Special Status Register (Bus/Address Exception only)
AA	32 Bit Access Address(Bus/Address Exception only)
OB	16 Bit Output Buffer (68010 Bus/Address Exception only)
IB	16 Bit Input Buffer (68010 Bus/Address Exception only)
IR	16 Bit Instruction Register (Bus/Address Exception only)

C.3.2.15 RESET Command

The RESET command reinitializes the Executive. This is different from a hardware reset. The purpose of this command is to reset the serial ports and Executive variables without destroying the contents of memory. The command may not be abbreviated. The command format is as follows:

RESET [DT] [PORTS] [VECTORS]

The DT option resets only the disk/tape (not 1/2" tape) subsystem. The PORTS option resets the Executive serial ports. The VECTORS option reinitializes the exception vector table. If no options are specified, a complete reset is performed.

C.3.2.16 RESTORE Command

The RESTORE command provides for image restore of disk from another disk (DISK), 1/4" cartridge tape (CT) or cartridge disk (CD). If removable media is used, the user will be prompted with the following message:

Mount Cartridge N (y/n): _

N is the cartridge number to mount. If there is an error, the error code and logical address will be displayed and the Executive reentered. Note that the restore will not exceed the size of the disk, no matter what the FOR count specified is. For either fixed or cartridge disk, the first 8 sectors will be skipped, as they contain information particular to that disk.

C.3.2.16.1 1/4" Cartridge Tape RESTORE

The RESTORE command format for cartridge tape is as follows:

RESTORE [unit] CT [unit] [DU device, lun] [TO sector] [FOR sectors]

The unit has a default value of 0 for the destination (disk) and 0 for the source (tape). The TO sector address (relative sector number) has a default of 8 (beginning of disk to restore). The FOR sectors is the number of sectors to restore. The default number of sectors is the size of the disk.

C.3.2.16.2 Fixed and Cartridge Disk RESTORE

The RESTORE command format for fixed or cartridge disk is as follows:

```
RESTORE [unit] CD [unit] [DU device, lun] [FROM sector] [TO sector]
  [FOR sectors]
RESTORE [unit] DISK [unit] [DU device, lun] [FROM sector] [TO sector]
  [FOR sectors]
```

The unit has a default value of 0 for the destination (disk) and 0 for the source. The TO sector address (relative sector number) has a default of 8 (beginning of disk to restore). The FOR sectors is the number of sectors to restore. The default number of sectors is the size of the disk.

C.3.2.17 RETENSION Command

The RETENSION command will retension the 1/4" cartridge tape. The tape will then be positioned at BOT. The command format is:

```
RETENSION [CT]
```

This command is valid only for cartridge tape.

C.3.2.18 REWIND Command

The REWIND command will rewind the 1/4" cartridge tape. The tape will then be positioned at BOT. The command format is:

```
REWIND [CT]
```

This command is valid only for cartridge tape.

C.3.2.19 SAVE Command

The SAVE command is used to write out to disk or cartridge tape data from memory. The command format is as follows:

```
SAVE [DISK] [unit] [DU device, lun] [FROM address] [TO sector]
  [FOR sectors]
SAVE CD [unit] [DU device, lun] [FROM address] [TO sector] [FOR sectors]
SAVE CT [unit] [DU device, lun] [FROM address] [FOR sectors]
```

The default load device is the disk (default unit 0). The default cartridge tape or disk unit is 0. The FROM address has a default of 8000 (hex). The TO sector address has a default of zero and is only applicable to disk. The FOR sectors is the number of sectors to save.

C.3.2.20 SEGMENT Command

The SEGMENT command allows examination and modification of the Segment Map. It is valid only for ZEBRA 700 system with MMU board. If the segment number is not part of the command, segment zero (0) is assumed. Each segment is displayed and the processor then waits for console input. If a value is entered, that value is stored and the next segment is displayed. If nothing is entered (return only), no change is made to the data. If a period (.) is entered, the processor is exited. If a period is entered after a value, the processor is exited after the new value is stored. If a slash (/) is entered, the previous segment is displayed. If a slash is entered after a value, the previous segment is displayed. If a slash is entered after a value, the previous segment is displayed after the value is stored. If a comma (,) is entered, the same segment is displayed after the value is entered. An example of the command and its effects follows:

```
Ok, SEGMENT 20
  Segment 20=F20, L=100000, S=rwx, U=rwx, P=100000:
  Segment 21=F21, L=108000, S=rwx, U=rwx, P=108000:
  Segment 22=F22, L=110000, S=rwx, U=rwx, P=110000:
Ok,
```

In the above example, the address and contents of the segment are displayed. Note that the address and contents are for the present context. The 'L=' displays the associated logical memory address. The 'S=' and 'U=' display the protection mode for the supervisor and user mode respectively (r = read access, w = write access, and x = execution access). Finally, the 'P=' displays the physical address that drives the page map.

C.3.2.21 SRECORD Command

The SRECORD command is used to load, via the host unit, standard Motorola 'S' record format absolute load modules. All Motorola record types are recognized. An error will cause an error message to be displayed, but loading will continue until terminated by the end record or a user break. The command format is as follows:

SRECORD [text]

All three address size formats (16, 24, and 32) are recognized. Further, the head record, record count record, and end record cause a message to be printed to the console as shown in the following example:

```
Ok, SRECORD cat main.sav
  Header Record Received:  HDR
  Records Read:  625
  End Record, Bytes Read:  12542, Start Address:  00020000
Ok,
```

Note that the string 'cat main.sav' is sent to the host in order to initiate the transfer. This particular example is typical of a XENIX host. The start address is stored in the user's program counter (PC).

C.3.2.22 SYSTEM Command

The SYSTEM command works the same as the REGISTER command with no arguments. The register values displayed are those at the last exception when in the Executive mode (user program not running). If there has been no exception, nothing is displayed. Registers may not be altered. This function is mainly used to determine the cause of Executive faults. The command format is as follows:

SYSTEM

creating a SYSGEN cartridge tape or disk for ZEBRA 750 D

D.1 USING THE SYSGEN PROGRAM

SYSGEN is a program that allows on-line creation of a formatted operating system cartridge tape or disk. This tape or disk may then be used for both operating system load and FILE-RESTORE.

The SYSGEN account will be supplied on a separate cartridge disk for cartridge disk 750s and will be included on the 3.0 release tape for 750s with 1/4-inch tape. The account consists of the following four files:

1. STRAP.SO, which contains the bootstrap object code,

2. MON.SO, which contains the monitor object code,

3. SYS-OBJ, which contains the ABS object code, and

4. PROCS, which contains the program that creates the SYSGEN cartridge tape or disk.

The item-ids of the SYS-OBJ file are in the format FRMxxx where xxx is a decimal frame number. The ABS frames are loaded from a list which resides in the dictionary of the SYS-OBJ file and is called FRMN.LIST. You may add to or replace this list to tailor the ABS frames to your own applications. The item-ids you use may be in other than FRMxxx format. Just add them to FRMN.LIST using the editor. Note that FRMN.LIST will dump the ABS section in numerical order, although this is not a requirement. If you have item-ids with duplicate frame numbers, the last one on the SYSGEN media will overwrite all previous versions of that frame on disk.

If you have changed an ABS frame that will cause the CHECK-SUM item to differ from the one supplied on the SYSGEN account, you must edit the item CHECK-SUM in the dictionary of the SYS-OBJ with the correct checksum for that frame.

The format of the SYSGEN cartridge tape or disk is as follows:

```
BOT --->        LOADER
                MONITOR
                ABS
                SYSTEM ACCOUNTS
```

The SYSGEN program performs the SAVE of the system accounts in the same way as a FILE-SAVE except that there is no T-DUMP of the STAT-FILE because this file is not generated. Therefore, if you wish to select which accounts are to be placed on the SYSGEN tape, you must edit the SYSTEM file and change the code for the account pointers in attribute 1 to 'DX' for those accounts which you do not want on the tape. Be sure that all accounts you wish on the tape do not have 'DX' pointers.

When you log on to the SYSGEN account or type in 'SYS-GEN' from TCL, you will be given the following prompt:

(C)artridge tape or (D)isk cartridge (C,D,X)?

If you respond with a 'C', a SYSGEN tape cartridge will be created.

If you respond with a 'D', a SYSGEN disk cartridge will be created.

If you respond with an 'X', you will terminate the remainder of the SYSGEN PROC and return to TCL.

If you responded with a 'C' or 'D', you will receive the following prompt:

Do you wish to update the CHECK-SUM item in the ERRMSG file (Y,N,X)?

Answer Y(es) if the ABS on your system will cause the CHECK-SUM item to differ from the one supplied on the SYSGEN account.

An 'X' response will place you at TCL.

D.2 USING A SYSGEN CARTRIDGE TAPE OR DISK

The procedure for use of a SYSGEN cartridge tape or disk is exactly the same as described under System Startup, starting with Section 2.1.2, "ZEBRA 750."

system-cursor definition utility E

The System-Cursor Definition Utility provides the user a means to customize the System-Cursor functions for his particular needs. The utility includes an editor, selection process, and "compiler." A menu-driven BASIC program creates, maintains, compiles, and selects up to 26 different terminals for inclusion in the System Cursor. Although only 26 terminals may be selected for inclusion in the System Cursor at one time, any number of terminals may be defined by this utility.

E.1 USER SEQUENCE OF OPERATION

Enter the command "DEFINE-CURSOR" at TCL. A display and menu will be presented on the terminal. Figure E-1 shows a sample display and the menu.

```
System Cursor Definition Utility

The following terminals are defined.  Terminals marked
with an asterisk (*) are selected to be included in
your System Cursor Definition.

*A ADDS           *H HONEYWELL      *O VT100        *U TV920
*B BEEHIVE        *I IBM3010        *P MIME         *V VIEWPOINT
*C DTC            *K VT52           *Q TEC           V VIEWPOINT2
*D DATAMEDIA      *L LSI            *R REGENT       *W WYSE50
*E EMULOG200      *M AMPEX          *S SOROC        *X DATAGRAPHIX
*G GTC            *N ENVISION       *T TV950        *Y WYSE100

    1)  Create Terminal Definition
    2)  Modify Terminal Definition
    3)  Delete Terminal Definition
    4)  Add Terminal to Selected Definitions
    5)  Delete Terminal from Selected Definitions
    EX  Exit without updating System-Cursor
    FI  Update System-Cursor to selected terminals

        Enter Selection (1-5) or EX or FI:
```

Figure E-1. Display and Menu

The following choices may be made from the menu (see Figure E-1).

1) Create Terminal Definition

Selection 1 will allow the creation of a new terminal definition. A terminal definition consists of a series of parameters that control a particular terminal (type, size, control codes, etc.). After you enter the menu selection 1, the routine will prompt for the terminal name to be defined. It will then check if that name already exists. If so, you may opt to modify the existing definition, or enter another name. If you opt to modify the existing definition, the routine will proceed as in Selection 2.

If the name is new, you will be asked if you want to use a copy of an existing terminal definition for the initial values for the new definition. If so, you will be prompted for the name of the existing terminal to be used as a "template." This is useful for defining terminals which are similar to another existing terminal. If you do not choose to use an existing terminal definition as a "template," the routine proceeds to prompt for each of the parameters for the new definition. (See Section E.2, Defining the Terminal Tables, for a description of these parameters.) Otherwise, the routine proceeds as in Selection 2.

2) Modify Terminal Definition

Selection 2 will allow you to modify existing terminal definitions. After you enter menu selection 2, the routine will prompt for the name of the terminal definition to be modified. If the name does not exist, you may opt to create it. If you opt to create a new definition, the routine proceeds as in Selection 1. Otherwise, the routine proceeds as described below.

In the terminal definition modification mode, the set of parameters for the terminal is broken into page-size blocks for display and modification. First, a section of the existing definition is displayed, then the prompt "Modify Lines?" is issued. You may answer YES, NO (default), or give a list of line numbers to modify. If you answer YES, you will be prompted for the list of line numbers; if you answer NO, the next section of the existing definition is displayed and the process repeats until the entire definition has been reviewed. Otherwise, you will be prompted to enter new data for each of the selected lines.

At each of these prompts, the following special entries may be made. First, a carriage return (null value) will cause the data for the line to be unchanged. Second, an entry of any number of spaces will cause the data for the line to be changed to null. Third, an entry of a single question mark (?) will display a brief explanation of the contents of the line and then a prompt for input.

Once all sections have been reviewed, you will be asked if the terminal definition is correct. If not, the review and modify process will be repeated or you may exit without saving any modifications. If the definition is correct, an attempt will be made to "compile" it. If the compilation detects errors, you may have to correct the errors via the modification process. Otherwise, you may select the terminal to be included in the list of terminals for your System Cursor. Figures E-2, E-3, and E-4 show sample display portions of the modification mode.

After all the lines have been prompted for, the routine returns to the selection menu.

3) Delete Terminal Definition

Selection 3 allows you to delete terminal definitions. After you enter menu selection 3, you will be prompted for the name of the terminal to be deleted. If the terminal definition is on file, it will be deleted.

4) Add Terminal to Selected Definitions

Selection 4 allows you to add a terminal to the list of terminals to be included in your System Cursor. After you enter menu selection 4, you will be prompted for the name of the terminal to be added to the list of selected terminals. If the name exists, the routine will check if that type of terminal has already been selected. If not, the desired terminal will be selected.

If the type of the desired terminal has already been selected for another terminal, you will be asked if you want to replace the previous selection with the new selection. If so, the previous selection will be deleted from the list of selected terminals and the new selection added.

5) Delete Terminal from Selected Definitions

Selection 5 allows you to delete a terminal from the list of terminals to be included in your System Cursor. After you enter menu selection 5, you will be prompted for the name of the terminal to be deleted from the list of selected terminals. If the name you enter is in the list, it will be deleted.

EX Exit without updating System-Cursor

This choice (EX) quits the definition process without updating the operating System Cursor. However, all your modifications and selections are preserved, so that when you re-execute the System Cursor Definition Utility, all definitions and selections will be displayed just as they were when the routine was EXited.

FI Update System-Cursor to selected terminals

This choice (FI) updates the operating System Cursor with the new selections and then quits the definition process.

```
Terminal = TV920

    1.  TYPE.............................  U
    2.  SCREEN SIZE......................  80,24
    3.  CURSOR ADDRESS CODE..............  L
    4.  @(X) CURSOR POSITIONING..........  CR STR(CHAR(12),X)
    5.  @(X,Y) CURSOR ADDRESSING.........  ESC "=" Y X
    6.  @(-1) CLEAR SCREEN & HOME........  CHAR(26)
    7.  @(-2) CURSOR HOME................  CHAR(30)
    8.  @(-3) CLEAR TO END OF PAGE.......  ESC "Y"
    9.  @(-4) CLEAR TO END OF LINE.......  ESC "T"
   10.  @(-5) START BLINK................  ESC "^"
   11.  @(-6) STOP BLINK.................  ESC "q"
   12.  @(-7) START PROTECT..............  ESC ")"
   13.  @(-8) STOP PROTECT...............  ESC "("
   14.  @(-9) CURSOR BACK................  BS
   15.  @(-10) CURSOR UP.................  VT

Modify lines?  NO


Terminal = TV920

   16.  @(-11) SLAVE ON..................
   17.  @(-12) SLAVE OFF.................
   18.  @(-13) START REVERSE VIDEO.......  ESC "j"
   19.  @(-14) STOP REVERSE VIDEO........  ESC "k"
   20.  @(-15) START UNDERLINE...........  ESC "l"
   21.  @(-16) STOP UNDERLINE............  ESC "m"
   22.  @(-17) ENABLE PROTECT MODE.......  ESC "&"
   23.  @(-18) DISABLE PROTECT MODE......  ESC "'"
   24.  @(-19) CURSOR FORWARD............  CHAR(12)
   25.  @(-20) CURSOR DOWN...............  CHAR(18)
   26.  @(-99) EMBEDDED VISUAL ATTRIBUTES?  YES

Modify lines?  NO

Is table for terminal TV920 correct?  YES
```

Figure E-2. Sample Portion of Modification Mode Display (TV920)

```
Terminal = AMPEX

    1.  TYPE.............................  M
    2.  SCREEN SIZE......................  80,24
    3.  CURSOR ADDRESS CODE..............  L
    4.  @(X) CURSOR POSITIONING..........  CR STR(CHAR(12),X)
    5.  @(X,Y) CURSOR ADDRESSING.........  ESC "=" Y X
    6.  @(-1) CLEAR SCREEN & HOME........  ESC "*"
    7.  @(-2) CURSOR HOME................  CHAR(30)
    8.  @(-3) CLEAR TO END OF PAGE.......  ESC "Y"
    9.  @(-4) CLEAR TO END OF LINE.......  ESC "T"
   10.  @(-5) START BLINK................  ESC "A" ESC "n"
   11.  @(-6) STOP BLINK.................  ESC "o" ESC "a"
   12.  @(-7) START PROTECT..............  ESC ")"
   13.  @(-8) STOP PROTECT...............  ESC "("
   14.  @(-9) CURSOR BACK................  BS
   15.  @(-10) CURSOR UP.................  VT

Modify lines? NO

Terminal = AMPEX

   16.  @(-11) SLAVE ON..................  HEX(1B4A)
   17.  @(-12) SLAVE OFF.................  HEX(1B4B)
   18.  @(-13) START REVERSE VIDEO.......  ESC "A" ESC "j"
   19.  @(-14) STOP REVERSE VIDEO........  ESC "k"
   20.  @(-15) START UNDERLINE...........  ESC "l"
   21.  @(-16) STOP UNDERLINE............  ESC "m"
   22.  @(-17) ENABLE PROTECT MODE.......  ESC "&"
   23.  @(-18) DISABLE PROTECT MODE......  ESC "'"
   24.  @(-19) CURSOR FORWARD............  FF
   25.  @(-20) CURSOR DOWN...............  LF
   26.  @(-99) EMBEDDED VISUAL ATTRIBUTES?

Modify lines? NO

Is table for terminal AMPEX correct? YES
```

Figure E-3. Sample Portion of Modification Mode Display (AMPEX)

```
Terminal = VIEWPOINT2

    1.  TYPE.............................  V
    2.  SCREEN SIZE......................  80,24
    3.  CURSOR ADDRESS CODE..............  A
    4.  @(X) CURSOR POSITIONING..........  CHAR(16) X
    5.  @(X,Y) CURSOR ADDRESSING.........  CHAR(11) Y CHAR(16) X
    6.  @(-1) CLEAR SCREEN & HOME........  FF
    7.  @(-2) CURSOR HOME................  CHAR(11) "" CHAR(16) NUL
    8.  @(-3) CLEAR TO END OF PAGE.......  ESC "k"
    9.  @(-4) CLEAR TO END OF LINE.......  ESC "K"
   10.  @(-5) START BLINK................  SO ESC "OB"
   11.  @(-6) STOP BLINK.................  SI
   12.  @(-7) START PROTECT..............
   13.  @(-8) STOP PROTECT..............
   14.  @(-9) CURSOR BACK................  CHAR(21)
   15.  @(-10) CURSOR UP.................  CHAR(26)

Modify lines?  NO


Terminal = VIEWPOINT2

   16.  @(-11) SLAVE ON..................  HEX(1B33)
   17.  @(-12) SLAVE OFF.................  HESC(1B34)
   18.  @(-13) START REVERSE VIDEO.......  CHAR(14) ESC "OP"
   19.  @(-14) STOP REVERSE VIDEO........  CHAR(15)
   20.  @(-15) START UNDERLINE...........  CHAR(14) ESC "O'"
   21.  @(-16) STOP UNDERLINE............  CHAR(15)
   22.  @(-17) ENABLE PROTECT MODE.......
   23.  @(-18) DISABLE PROTECT MODE......
   24.  @(-19) CURSOR FORWARD............  CHAR(6)
   25.  @(-20) CURSOR DOWN...............  LF
   26.  @(-99) EMBEDDED VISUAL ATTRIBUTES?

Modify lines?  NO

Is table for terminal VIEWPOINT2 correct?  YES
```

Figure E-4. Sample Portion of Modification Mode Display (VIEWPOINT2)

E.2 DEFINING THE TERMINAL TABLES

The System-Cursor Definition Utility is used to define the Terminal Tables (menu choices 1 and 2). Some of the required fields in the Terminal Table are explained in the following sections.

E.2.1 TERMINAL TYPE

The Terminal Type is a single uppercase letter (A to Z) which identifies the terminal to the system. Usually, it is set up at logon with the TERM command. The Terminal Type field in the terminal table corresponds to the type as set with the TERM command. Up to 26 different types are available.

E.2.2 SIZE

The Size field defines the screen size in columns and rows. The size is entered as two numbers separated by a comma; the first number represents column, the second number represents row (e.g., 80,24). The size is used to limit the range of cursor addresses which may be produced by the System-Cursor routine. If a value exceeds the size, the maximum size is substituted.

E.2.3 CURSOR ADDRESSING TYPE

The Cursor Addressing Type is usually a single letter (A, L, T, H, D). The types are "A" for Adds type addressing, "L" for Lear-Siegler type addressing, "T" for TEC type addressing, "H" for Hazeltine type addressing, and "D" for decimal type addressing. All types except "D" produce binary column and row addresses, with a single byte used for each. "D" type addressing produces one to three digits for column and row addresses. If "D" type addressing is used, the code may be followed by two digits (22, 23, 32, 33) to force padding to the desired number of digits (e.g., "D32" will produce decimal addressing with three digits used for the column and two digits for the row (leading zeros added to force the length)). "D" alone will use "floating" decimal numbers of one to three digits.

All cursor addressing codes may be followed by a plus sign (+) to add 1 to the column and row addresses before generating the address codes. This allows for terminals that define the upperleft corner of the screen as "1,1" instead of "0,0". Thus, decimal addressing for a three-digit row and column address numbered from "1,1" is "D33+".

To determine the proper binary cursor addressing type (A, L, T, H), use Table E-1. This table shows the column or row address and the associated code (control code or ASCII character) for each of the addressing types. The formula for each binary type follows:

(A)DDS	COL = CHAR((INT(X/10)*6)+X) ROW = CHAR(Y+64)	(L)SI	COL = CHAR(X+32) ROW = CHAR(Y+32)
(H)AZE	COL = CHAR(X) ROW = CHAR(Y)	(T)EC	COL = CHAR(-(1+X)) ROW = CHAR(-(1+Y))

Table E-1. Binary Cursor Addressing

X	Y	ADDS COL	ADDS ROW	LSI	TEC	HAZE	X	Y	ADDS COL	ADDS ROW	LSI	TEC	HAZE
0	0	nul	@	space	del	nul	40		@		H	W	(
1	1	soh	A	!	~	soh	41		A		I	V	)
2	2	stx	B	"	}	stx	42		B		J	U	*
3	3	etx	C	#	|	etx	43		C		K	T	+
4	4	eot	D	$	{	eot	44		D		L	S	,
5	5	enq	E	%	z	enq	45		E		M	R	-
6	6	ack	F	&	y	ack	46		F		N	Q	.
7	7	bel	G	´	x	bel	47		G		O	P	/
8	8	bs	H	(	w	bs	48		H		P	O	0
9	9	ht	I	)	v	ht	49		I		Q	N	1
10	10	dle	J	*	u	lf	50		P		R	M	2
11	11	dc1	K	+	t	vt	51		Q		S	L	3
12	12	dc2	L	,	s	ff	52		R		T	K	4
13	13	dc3	M	-	r	cr	53		S		U	J	5
14	14	dc4	N	.	q	so	54		T		V	I	6
15	15	nak	O	/	p	si	55		U		W	H	7
16	16	syn	P	0	o	dle	56		V		X	G	8
17	17	etb	Q	1	n	dc1	57		W		Y	F	9
18	18	can	R	2	m	dc2	58		X		Z	E	:
19	19	em	S	3	l	dc3	59		Y		[	D	;
20	20	space	T	4	k	dc4	60		`		\	C	<
21	21	!	U	5	j	nak	61		a		]	B	=
22	22	"	V	6	i	syn	62		b		^	A	>
23	23	#	W	7	h	etb	63		c		_	@	?
24		$		8	g	can	64		d		´	?	@
25		%		9	f	em	65		e		a	>	A
26		&		:	e	sub	66		f		b	=	B
27		´		;	d	esc	67		g		c	<	C
28		(		<	c	fs	68		h		d	;	D
29		)		=	b	gs	69		i		e	:	E
30		0		>	a	rs	70		p		f	9	F
31		1		?	`	us	71		q		g	8	G
32		2		@	_	space	72		r		h	7	H
33		3		A	^	!	73		S		i	6	I
34		4		B	]	"	74		t		j	5	J
35		5		C	\	#	75		u		k	4	K
36		6		D	[	$	76		v		l	3	L
37		7		E	Z	%	77		w		m	2	M
38		8		F	Y	&	78		x		n	1	N
39		9		G	X	´	79		y		o	0	O

E.2.4 CURSOR CODE STRINGS

Cursor Code Strings are expressions that produce the control and escape sequences used by the terminal defined. The syntax of the expressions is similar to BASIC syntax, except that a blank may be used between elements in these expressions as well as a colon. Cursor code strings may consist of the following fields, each separated by blanks or colons:

1. Defined control character (e.g., ESC, BS, DEL, NUL, etc.)
2. String literal in quotes (e.g., "A", "[0", etc.)
3. Character function (e.g., CHAR(21))
4. Hexadecimal string (e.g., HEX(1B41))
5. String function (e.g., STR(NUL,5) or STR(CHAR(12),X))
6. Cursor address variable (e.g., X, Y, or Z)

The cursor address variables (X, Y, Z) cause the specified address (byte or decimal string) to be inserted into the control string at the specified position. The variable X contains the column, Y contains the row, and Z contains the row previously referenced in an @(X,Y) code (or zero, if the last reference was @(-1) or @(-2)).

The symbolic name for the control codes and their decimal and hexadecimal equivalents are shown in Table E-2. Any of these codes may be included in the cursor code string. Note that it is often easier to reference the backspace character as BS instead of CHAR(8), or NUL instead of CHAR(0).

Table E-2. Control Codes with Decimal and Hexadecimal Equivalents

CODE	DEC	HEX	CODE	DEC	HEX
NUL	0	00	DC1	17	11
SOH	1	01	DC2	18	12
STX	2	02	DC3	19	13
ETX	3	03	DC4	20	14
EOT	4	04	NAK	21	15
ENQ	5	05	SYN	22	16
ACK	6	06	ETB	23	17
BEL	7	07	CAN	24	18
BS	8	08	EM	25	19
HT	9	09	SUB	26	1A
LF	10	0A	ESC	27	1B
VT	11	0B	FS	28	1C
FF	12	0C	GS	29	1D
CR	13	0D	RS	30	1E
SO	14	0E	US	31	1F
SI	15	0F	SP	32	20
DLE	16	10	DEL	127	21

E.2.5 SPECIAL CURSOR CODE STRINGS

Most of the cursor code strings are self-explanatory and consist of control characters, escape sequences, and other obvious codes. Some of the code strings are not as obvious and require further explanation.

E.2.5.1 Column-Only Cursor Positioning

Many terminals do not support the column-only cursor positioning. Terminals which do support column-only positioning (e.g., ADDS) may use the terminal's normal control sequence (e.g., (CHAR(16) X)). For terminals without column-only positioning, the function may be simulated in two ways. Using the first method, the cursor can be positioned to column zero of the current line (carriage return), followed by a cursor-right code for the number of columns required (e.g., CR STR(CHAR(12),X)). VT-100 type terminals may use a sequence like: CR ESC "[" X "C" BS, where the decimal value of X is part of the cursor-right escape sequence.

The second method of simulating column-only positioning is less desirable, but may be effective in some instances. It uses the dummy cursor address variable Z in place of the Y address in a normal X-Y cursor address code (e.g., ESC "=" Z X).

E.2.5.2 Clear Screen & Home @(-1)

The Clear Screen & Home code may consist of two different terminal control sequences (one for clear screen and one for home). This is the case for VT-100 type terminals. Many other terminals combine these two control sequences into one.

E.2.5.3 Embedded Visual Attributes @(-99)

The special cursor function @(-99) normally disables pagination and clears the system's line counter. It retains this function, but also passes a value back to the user program that defines whether the terminal uses embedded (1) or non-embedded (0) visual attributes. This is important for application programs that require precise screen layout. If a terminal has embedded attributes (i.e., a screen location is required to turn the attribute on and to turn the attribute off), answer YES to the prompt for this field.

NOTE: If the @(-99) function is used to turn off pagination, it should be used in a variable assignment statement (DUMMY = @(-99)) instead of a print statement (PRINT @(-99)). The reason for this is that if the print statement is used and if the embedded visual attribute field is answered with a YES, PRINT @(-99) will turn off pagination, but will also print out a 1; if the embedded visual attribute field is answered with a NO, PRINT @(-99) will turn off pagination and print out a 0.

E.3 CUSTOMIZING THE CURSOR DEFINITIONS

Besides defining terminal types and the control sequences used for each, you may also define new cursor control functions. Functions @(-21) to @(-98) are available for user definition.

To define your own function, create an attribute definition item in the dictionary of the CURSOR file with a numeric ITEM-ID, and a D/CODE of S or A. Normally, determine the ITEM-ID by adding 5 to the function desired (e.g., for @(-30), the ITEM-ID is 35). Special definitions are assigned to attributes 15, 19, and 20 of the dictionary item. Attribute 15 may contain a multisubvalued list of pattern match specifications. Attribute 19 may contain a multisubvalued description (which is printed in response to '?' during the definition process). Attribute 20 may contain the value '1' to cause a "page break" in the definition process (e.g., a dictionary definition item with a 1 in AMC 20 will be the last field prompted for before the "Modify lines?" query and display of additional blocks).

E.4 INITIALIZATION DURING COLD-START

The COLD-START proc in SYSPROG-PL should be modified to reinitialize the System-Cursor routine during the COLD-START procedure. To do this, add the command "INIT-CURSOR CURSOR *" to the COLD-START proc.

E.5 RESETTING TERMINAL TYPE TABLE FOR BINARY BACKUP

It is beneficial to use the RESET-CURSOR verb before making a binary system backup or ABS tape. It ensures that, on a restore or ABS load, the terminal type pointers will not point to an invalid location. This procedure will not have any effect if the command "INIT-CURSOR CURSOR *" is done at COLD-START time.

TCL stacker F

The TCL Stacker allows you to display, modify, correct, and execute TCL commands that you have previously executed or entered. All of the TCL stacker commands are preceded by a period and may be entered in uppercase or lowercase. The TCL stacker will stack up to the last 40 TCL commands entered, and, unless otherwise specified, display only the last 20 commands. However, it will not stack a command that is the same as the previous command entered.

Also, the TCL stacker includes a Help screen that describes all the different TCL stacker commands. Each port has its own stack from which to operate from and the Help screens that list executable TCL stacker commands are dependent upon system privilege level.

F.1 TCL STACKER COMMANDS

The following sections describe the TCL stacker commands and give instructions on how to use them.

F.1.1 DISPLAYING HELP SCREEN: ? OR .H{ELP} COMMAND

The general format for this command is:

```
>.?H{P}
>.H{elp}{P}
```

This command will display the Help screen on the terminal (or printer, if P is specified).

There are three different Help screens, one for each of the system's three different privilege levels (0, 1, or 2). This command functions at all three system privilege levels.

F.1.2 LISTING THE TCL STACK TO THE TERMINAL: .L COMMAND

The .L command will list the last 20 TCL commands entered by the user. The general format for this command is:

```
>.L
>.L{n}{,m}{(p)}
```

If parameter n is used, then only the last n number of commands will be listed. If parameter m is entered, then the last n (default = 20) commands will be listed starting at command m. The p parameter is used for displaying line p's last TCL commands. Use of the p parameter requires a system privilege level of 2. With the exception of the p parameter, the .L command functions under all three system privilege levels.

F.1.3 EXECUTING A TCL STACKED COMMAND: .X COMMAND

The .X command is used to execute a TCL stacked command. The general format for this command is:

```
>.X
>.X{n}
```

When n is omitted, the default is command 1, or the last TCL command entered. Otherwise, n will specify which command to execute. The .X command will allow execution of SELECTed items and works with SELECTed lists. The .X command functions under all three system privilege levels.

F.1.4 DISPLAYING THE STATUS OF THE TCL STACKER: .Q COMMAND

The .Q command will display the status of the TCL stacker for your port. It will display the following:

```
 P# STK U/C
=== === ===
002  On Off
```

where:

P#	gives the port number
STK	tells whether your stack is on or off
U/C	tells whether uppercasing of TCL commands is in effect.

The general format for this command is:

```
>.Q
>.Q(A)
```

The 'A' option will display the status of all the lines, but this is a system privilege level 2 function only.

F.1.5 TOGGLING THE TCL STACKER ON OR OFF: .O COMMAND

The .O command will toggle the TCL stacker on or off depending on its present state. It will respond with ON! or OFF! to let the user know the new status. When the TCL stacker is off, the only TCL stacker commands that will function are: .? (Help), .O (TCL stacker on or off), .U (toggle uppercasing on/off), and .Q (display TCL stacker status). The general format for this command is:

```
>.O
>.O(n)
```

The n parameter willl allow toggling of another port's TCL stacker on or off, but you must have a system privilege level of 2 to use this function.

F.1.6 TOGGLING THE UPPERCASING FUNCTION ON OR OFF: .U COMMAND

The .U command will toggle the automatic uppercasing of TCL commands on or off depending on its present state. It will respond with ON! or OFF! to let the user know the new status. When the uppercasing function is off, everything runs as normal. But when the uppercasing function is on, all TCL commands that are entered will be uppercased before being executed. This allows TCL commands to be entered in either uppercase, lowercase, or a combination of uppercase and lowercase. The general format for this command is:

```
>.U
>.U(n)
```

The n parameter allows toggling of another port's TCL stacker on or off, but you must have a system privilege level of 2 to use this parameter. The .U command requires a minimum system privilege level of 1.

F.1.7 DELETING THE TCL STACK: .K COMMAND

The .K command is used to delete your entire stack. When this is executed, all of your last stacked commands (up to 40) will be deleted. The .K command requires a minimum system privilege level of 1. The general format for this command is:

```
>.K
```

F.1.8 BRINGING TCL STACKED COMMAND TO TOP-OF-STACK: .T COMMAND

The .T command will put command line n on the top of the stack for ease in executing the same command. The general format for this command is:

```
>.Tn
```

Command line n must be specified and greater than 1 or no change will take place. The .T command requires a minimum system privilege level of 1.

F.1.9 REPLACING OR MODIFYING TCL STACKED COMMAND: .R COMMAND

The .R command allows you to replace and modify TCL stacked commands. The general format for this command is:

```
>.R{n}/old text/new text
>.R{n}U{m}/old text/new text
>.R{n}{,m}/old text/new text
```

The .R command functions like the Editor replace command. Parameter n gives the starting line number for modification. If parameter n is omitted, it will default to line 1. Parameter m gives the number of lines to modify. If parameter m is omitted, it will default to 1 line. The U parameter is used to modify all cases of old text, with new text just like the Editor Replace Universal (RU) function. The .R command requires a minimum system privilege level of 1.

F.1.10 DELETING A TCL STACKED COMMAND: .D COMMAND

The .D command allows you to delete TCL commands from the TCL stacker. The general format for this command is:

```
>.D{E}
>.D{E}{n}{,m}(A)
```

Parameter n gives the number of lines to delete. If parameter n is omitted, it will default to 1 line. Parameter m gives the starting line number to begin deleting. If parameter m is omitted, it will default to line 1. The .D command requires a minimum system privilege level of 1.

F.1.11 EDITING THE TCL STACKER: .E COMMAND

The .E command allows you to edit the TCL command stacker using all the functions of the PICK Editor. The general format of this command is:

```
>.E
>.E{n}
```

If parameter n is specified, the editing will begin from line n. If omitted, parameter n will default to line 1. The .E command requires a minimum system privilege level of 1.

index