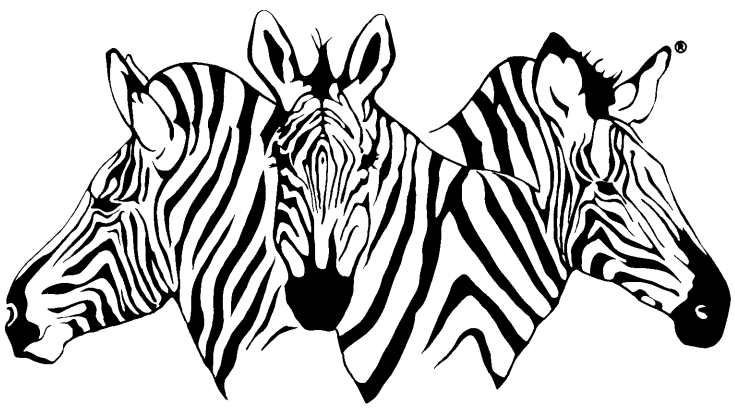
PROC
reference manual

88A00780A02



ZEBRA® BY GA

RECORD OF REVISIONS

Title: PROC Reference Manual

Document No. 88A00780A02

Date	Revision Record
Mar 84	Original Issue
Feb 85	Revision A02 - Change Package (85A00518A01)

PROC
reference manual

88A00780A02

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: PROC Reference Manual

Document No. 88A00780A02

Date	Revision Record
Mar 84	Original Issue
Feb 85	Revision A02 - Change Package (85A00518A01)

FOREWORD

This document is one of a family of ZEBRA reference manuals devoted to PICK processors that are on call within the PICK operating system. Before reading this document and using the processor described, it is recommended that you first become familiar with the PICK terminal control language and file structure. These subjects are thoroughly covered in 88A00782A, listed below with other documents covering PICK processors.

Document No.	Title
88A00757A	PICK Operator Guide
88A00758A	ACCU-PLOT Operator Guide
88A00759A	COMPU-SHEET Operator Guide
88A00760A	Quick Guide for the PICK Operating System
88A00774A	PICK Utilities Guide
88A00776A	PICK ACCESS Reference Manual
88A00777A	PICK SPOOLER Reference Manual
88A00778A	PICK BASIC Reference Manual
88A00779A	PICK EDITOR Reference Manual
88A00781A	PICK RUNOFF Reference Manual
88A00782A	Introduction to PICK TCL and FILE STRUCTURE
88A00783A	PICK JET Word Processor Guide

TABLE OF CONTENTS

proc 1

1.1 THE PROC PROCESSOR

PROC allows you to prestore a complex sequence of Terminal Control Language (TCL) operations (and associated processor operations) which can then be invoked by a single-word command. Any sequence of operations which can be executed at the TCL level can also be prestored via the PROC processor. This prestored sequence of operations (called PROC) is executed interpretively by the PROC processor and therefore requires no compilation phase.

The PROC processor has the following features:

1. Four variable length Input/Output (I/O) buffers
2. Parameter passing between buffers
3. Interactive terminal prompting
4. Extensive I/O and buffer control commands
5. Conditional and unconditional branching
6. Relational character testing
7. Pattern matching
8. Free-field and fixed-field character manipulation
9. Optional command labels
10. User-defined subroutine linkage
11. Inter-PROC linkage

1.2 PROC OVERVIEW

A PROC provides the means to catalogue a sequence of operations which can then be invoked from the terminal by a one-word command. Any operation that can be executed by the TCL can be performed in a PROC. This use of PROC is similar to the use of Job Control Language (JCL) in some computer systems. The PROC language in PICK, however, is more powerful since it has conditional capabilities and can be used to interactively prompt the terminal user. Additionally, a PROC can test and verify input data as it is entered from the terminal keyboard.

A PROC is executed interpretively by the PROC processor and therefore requires no compilation phase. A PROC stored as an item in the user's Master Dictionary (MD) is executed in the TCL environment by typing the item-id of the PROC, any optional arguments, and a carriage return.

While a PROC can exist in the MD, the actual body of the PROC may be within the same item, or it may be stored as an item in any dictionary or data file. The first attribute (first line) of a PROC is always the code PQ. This specifies to the system that what follows is to be executed by the PROC processor. All subsequent attribute values contain PROC statements that serve to generate TCL commands or insert parameters into a buffer for the interactive processors, such as the EDITOR. PROC statements consist of an optional numeric label, usually a one- or two-character command, and optional command arguments. PROCs can be created using the EDITOR.

Execution of a PROC is shown in the following example where the sample PROC named LISTU is invoked.

```
>LISTU [CR]

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

  00 0200 SP          08:00AM 01/01/78 Channel 0
  02 0240 CM          09:10AM 01/01/78 Channel 2
  03 0260 LC          07:30AM 01/01/78 Channel 3
  04 0280 JP          10:14AM 01/01/78 Channel 4
 *06 02C0 SAL         08:35AM 01/01/78 Channel 6
  10 0340 JET         09:00AM 01/01/78 Channel 10
```

The ability to pass arguments to a TCL level process via a PROC is illustrated as follows:

```
>LISTDICTS POLICY [CR]

POLICY..... D/CODE.. A/AMC.. V/CONV..... V/TYP  V/MAX

AUDIT-PERIOD        A        01                 L      4
POLICY-PERIOD-FROM  A        02       D         L      10
POLICY-PERIOD-TO    A        03       D         L      11
EXPIRES             A        04       D         L      12
```

Here LISTDICTS is the name of the prestored PROC, while POLICY is the argument being passed.

The ability to interactively prompt input data from the user (and subsequently verify this data) is illustrated as follows:

```
>ENTER-DATA [CR]

PART-NUMBER             = 3215-19 [CR]
DESCRIPTION             = TRANSISTOR [CR]
QUANTITY                = FIFTY [CR]

ERROR:NUMERIC DATA ONLY!!

QUANTITY                = 50 [CR]
```

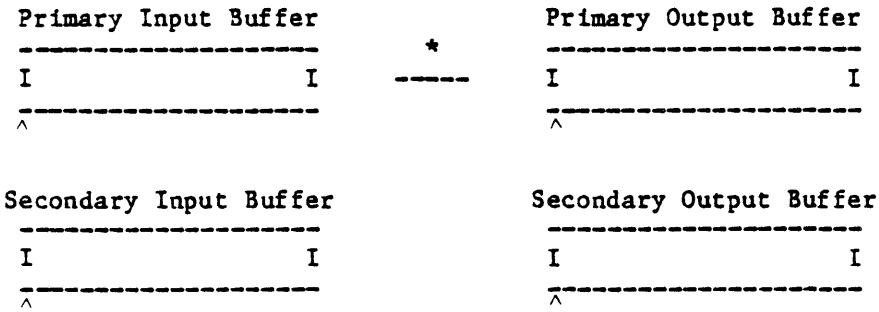
The PROC then prompts the user for the required data. The PROC could then, for example, store this data in a buffer which could then be passed to the BASIC processor to update the file.

Once a PROC is invoked, it remains in control until it terminates. When the PROC temporarily relinquishes control to a processor such as the EDITOR, BASIC, etc., or to a user-supplied subroutine, it functionally remains in control since an exit from the called processor returns control to the PROC. TCL only regains control when the PROC is terminated explicitly, or when all of the lines in the PROC have been exhausted.

1.3 INPUT/OUTPUT BUFFER OPERATION

Operations specified within a PROC involve the movement of data from either of two input buffers (data storage areas) to either of two output buffers.

PROC utilizes four input/output buffers: 1) the primary input buffer, 2) the secondary input buffer, 3) the primary output buffer, and 4) the secondary output buffer (called the stack). The general interrelationship of these buffers is as follows:



*Path taken depends on "currently active" buffers.

Essentially, the function of a PROC is to move data from either input buffer to either output buffer, thus forming the desired TCL and processor commands. At any given time, one of the input buffers is specified as the "currently active" input buffer, while one of the output buffers is specified as the "currently active" output buffer. Buffers are selected as "currently active" via certain PROC commands (these commands are discussed in detail in this section). Thus, when moving data between the buffers, the source of transfer is the currently active input buffer, while the destination of the transfer is the currently active output buffer.

The primary input buffer contains the PROC name and any optional arguments, exactly as they were entered when the PROC was invoked. The contents of this buffer remains the same throughout execution of the PROC unless explicitly modified by an IP, IT, IH, RI, Plus or Minus command.

The primary output buffer builds the single command which ultimately is submitted at the TCL level for processing. Any command which can be executed via the terminal at the TCL level can also be constructed and executed via a PROC.

The secondary input buffer contains data subsequently input by the user in response to an IS command. The data in this buffer is volatile: It is overwritten by subsequent IS commands, and with ERRMSG numbers by Error Message generating processes.

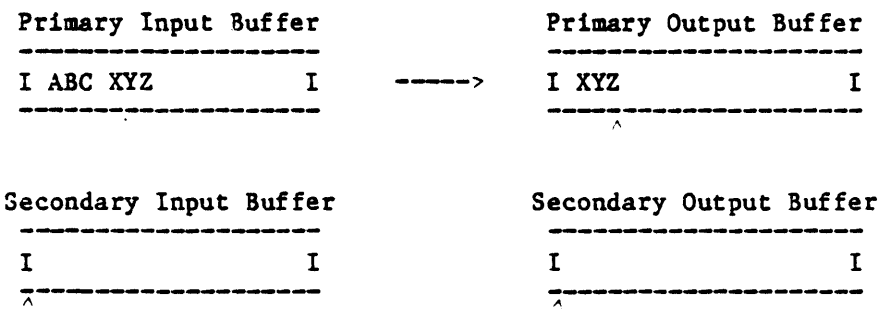
The secondary input buffer is loaded with data from several system processors, most notably the spooler. Information such as last hold file entry number is placed into this buffer. More information on this can be found in the PICK SPOOLER manual. Note that the secondary input buffer is a very temporary entity; if its contents is to be used, it should be used immediately after the execution of the processor which loaded the buffer.

The secondary output buffer (the stack) contains data that is to be used by the processor called by the PROC-generated TCL statement. Zero or more lines may be stored in the stack. Each request for terminal input by the called process, or, for example, each INPUT statement in BASIC, will be satisfied with a line of data from the stack. In the event that the called processor requests more data than exists in the stack, data will be requested from the terminal from that point onwards.

Note that each line of data in the secondary output buffer must be terminated by a carriage return which is explicitly placed in the stack via an H command (see Section 1.11). This is not the case with the primary output buffer; a carriage return is automatically placed at the end of the TCL command in the primary output buffer upon execution of that buffer via the P, PW, PH, PX, or PP command.

When all desired data has been moved to the output buffers, control is passed to TCL via a P, PH, PX, PW, or PP command. The command which resides in the primary output buffer is executed at the TCL level and the data in the secondary output buffer (if any) is used to feed processors such as BASIC or the EDITOR. When the process is completed, control returns to the PROC, at which time new data may be moved to the output buffers.

Moving data between the buffers is done by using "parameters." A parameter is defined as a string of characters (residing in one of the buffers) terminated by a space. If the character string includes spaces, the string is surrounded by single quotes. (The single quotes are considered part of the parameter and will be placed in the output buffer along with the parameter.) To keep track of the parameters, each buffer has a pointer which points to the "current" position in that buffer. These pointers are depicted in the buffer diagrams as small arrows placed beneath the buffer. As a general illustration of this concept, consider the following sample situation:



Here the PROC has been invoked by the characters ABC XYZ, which are then automatically placed in the primary input buffer. PROC commands have then been processed which position the input pointer of the primary input buffer to the second parameter (XYZ), and then subsequently move that parameter to the primary output buffer as illustrated by the path between the buffers (i.e., the currently active buffers are the primary input buffer and the primary output buffer).

1.4 AN OVERVIEW OF PROC COMMANDS

A PROC may consist of any number of PROC commands, one command per line.

The first line (attribute) of a PROC must contain the code PQ. This identifies the item as a PROC. The remaining lines in the PROC may contain any valid PROC commands. There is no limit to the number of lines in a PROC. However, each line may contain only one command and each command must begin in column position one of the line.

PROC commands are summarized in Table 1-1. A complete description of each command is presented in the following sections.

Any PROC command may optionally be preceded by a numeric label. Such a label serves to uniquely identify its associated PROC command for purposes of branching or looping within the PROC. Labels may consist of any number of numeric characters (e.g., 5, 999, 72, etc.). When a label is used, the PROC command must begin exactly one blank beyond the label. For example:

```
1 GO 5
23 A
99 IF A = ABC GO 3
2 ST ON
```

Only the first occurrence of the label is used as the destination of any control transfers (i.e., no check is made for erroneous duplicate labels).

As an introductory example to PROC commands, consider the following PROC stored as item 'DISPLAY' in the user's MD:

```
001  PQ
002  HLIST ONLY
003  A2
004  P
```

Assume that the user types in the following:

```
>DISPLAY INVENTORY  [CR]
```

This input invokes the above PROC and places the words DISPLAY INVENTORY in the primary input buffer. The second line of the above PROC is an H command which causes the text LIST ONLY to be placed in the primary output buffer. The third line is an A command which picks up the second word (parameter) in the primary input buffer and places it in the primary output buffer. Thus, the primary output buffer contains the words LIST ONLY INVENTORY. The last line of the PROC is a P command which submits the content of the primary output buffer to TCL for processing (LIST ONLY INVENTORY is an ACCESS sentence which causes the item-ids of the INVENTORY file to be listed; refer to the ACCESS Manual).

Table 1-1. Summary of PROC Commands

Command	Description
A	Moves data from input to output buffers.
B	Backs up input pointer.
BO	Backs up output pointer.
C	Specifies comment.
D	Outputs from either input buffer to terminal.
F	Moves input pointer forward.
G or GO	Unconditionally transfers control.
H	Moves text string to either output buffer.
IF	Conditionally executes specified command.
IH	Moves text string to either input buffer.
IP	Inputs from terminal to either input buffer.
IS	Inputs from terminal to secondary input buffer.
IT	Inputs from tape label to primary input buffer.
O	Outputs text string to terminal.
P	Causes execution of a PROC.
PP	Displays contents of output buffers and executes PROC.
PW	As above, waits for user response before proceeding.
PH	As above, but suppresses all terminal output for the verb.
PX	Like P, but returns to TCL after processing, not to PROC.
RI	Clears (resets) input buffer.
RO	Clears (resets) output buffer.
S	Positions input pointer.
SP	Selects primary input buffer.
SS	Selects secondary input buffer.
STON	Selects secondary output buffer (stack).
STOFF	Selects primary output buffer.
T	Provides formatted terminal output (cursor control).
U	Exits to user-defined subroutine.
X	Exits back to TCL level to calling PROC, or returns from subroutine.
+,-	Adds, subtracts decimal number to parameter in input buffer.
()	Links to another PROC.
[]	Subroutine call, local or to another PROC.

1.5 BUFFER SELECTION (SP, SS, AND ST)

The SP and SS commands select the primary or secondary input buffer, respectively, and set the input pointer at the beginning of the buffer. The STON will turn the stack on while the STOFF will turn the stack off.

The input buffers receive data from the terminal and store it so that it may be transferred to the output buffers. Only one of the two input buffers is "currently active." The SP and SS commands are used to select one or the other input buffer.

At the initiation of a PROC, the primary input buffer is automatically selected, and the buffer-pointer is set to the start of the input buffer, which contains the name by which the PROC was called from TCL. After the execute-primary-output-buffer command (P, PH, PX, PP, or PW), the primary input buffer is selected and the pointer set to the beginning of the buffer on return of control to the PROC from TCL. The contents of the primary input buffer is not disturbed, however.

The general form of the SP command is:

SP

It selects the primary input buffer and sets the input pointer at the beginning of the buffer.

The general form of the SS command is:

SS

It selects the secondary input buffer and sets the input pointer at the beginning of the buffer.

Note that the IS command will also select the secondary input buffer.

The primary output buffer is used to store one TCL statement that is eventually executed by a P, PH, PX, PP, or PW command. The secondary output buffer (stack) is used to store zero or more lines of data to satisfy terminal input requests by the processor invoked by the above mentioned TCL statement. Note that the "stack" is a first-in, first-out queue.

Only one of the two output buffers is "currently active." The STON or STOFF commands are used to select one or the other output buffer. Upon initial entry to a PROC, the stack is off.

The STON command selects the secondary output buffer (the stack) as the currently active output buffer (i.e., turns the stack on). Its general form is:

STON or ST ON

The STOFF command selects the primary output buffer as the currently active output buffer (i.e., turns the stack off). Its general form is:

STOFF or ST OFF

When the stack is on, all data picked up by the A command is moved to the secondary output buffer. When the stack is off, this data is moved to the primary output buffer. The stack may be turned on or off at any point within the PROC.

Figure 1-1 shows the results of these instructions. The pointers indicate currently active buffers in each case.

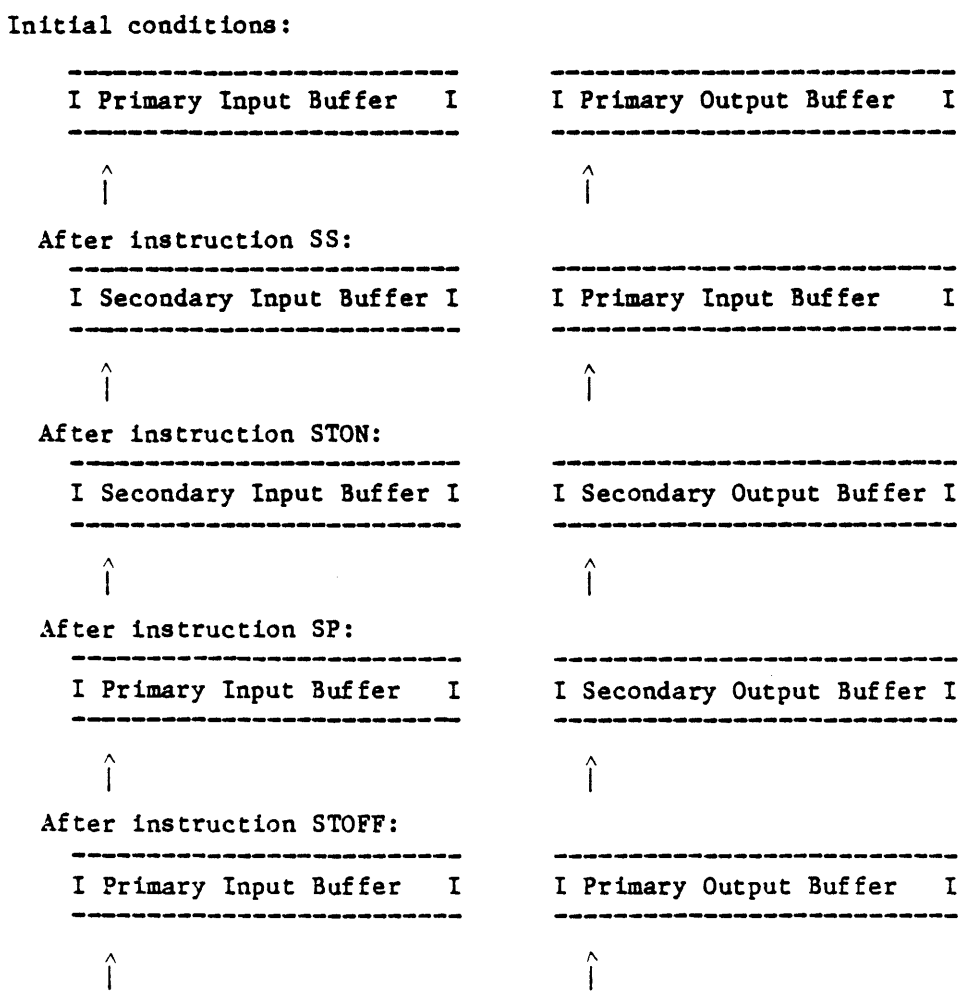


Figure 1-1. Sample Usage of SS, SP, STON, and STOFF Commands

1.6 POINTER POSITION (S, F, B, AND BO)

The S command positions the input pointer in the currently active input buffer. The F and B commands move the input pointer forward or backward one parameter, respectively. The BO command moves the output pointer backward one parameter. The general form of these commands:

General Form	Description
Sp	Moves input pointer to the p^{th} parameter.
S0 or S1	Moves input pointer to beginning of buffer.
F	Moves input pointer forward one parameter.
B	Moves current input pointer back one parameter.
BO	Backs up current output pointer one parameter.

Sp moves the input pointer to the p^{th} parameter of the currently active input buffer, where the parameters are separated by blanks. If there is no p^{th} parameter, then a backslash (\) will be inserted as a place holder for each null parameter until the p^{th} parameters is reached. S0 or S1 will set the pointer to the beginning of the buffer.

The F command causes the input pointer for the currently active input buffer to move forward one parameter. If the input buffer pointer is currently at the end of the buffer, this command has no effect.

The B command causes the input pointer for the currently active input buffer to move backward one parameter. If the input buffer pointer is currently at the beginning of the buffer, this command has no effect.

The BO command causes the output pointer for the current output buffer to move backward one parameter. If the output buffer pointer is currently at the beginning of the buffer, this command has no effect.

Examples of pointer commands:

Before	Command	After
Secondary Input Buffer I ABC DE FGHIJ I ^	S3 *	Secondary Input Buffer I ABC DE FGHIJ I ^
Secondary Input Buffer I ABC 123 DEF 456 I ^	F *	Secondary Input Buffer I ABC 123 DEF 456 I ^
Secondary Input Buffer I ABC 123 DEF 456 I ^	B *	Secondary Input Buffer I ABC 123 DEF 456 I ^
Primary Output Buffer I XXX YYY ZZZ I ^	BO **	Primary Output Buffer I XXX YYY ZZZ I ^

Active Buffer Prior to Command Execution

* Primary or secondary input buffer.
** Primary output buffer.
^ Pointer

1.7 MOVING PARAMETERS (A)

The A command is used to move a parameter from the input buffer to the output buffer. Either the primary or secondary input buffer may be used as the source, and either the primary or secondary output buffer may be used as the destination; the buffers used depend on commands executed prior to the A command. The general form of the A command:

A{c}{p}{,m}

where:

- c is the surround character (must be non-numeric)
- p is the number of the parameter to be moved
- ,m is the number of characters to be moved (up to the first blank character)

The function parameters c, p, and m are mutually independent and may be used in any combination to achieve the desired result.

p is a decimal ordinal number which indicates the parameter to be moved from the input buffer. If specified, the input-buffer pointer will be reset to the first character of the p^{th} parameter in the input buffer, and the p^{th} parameter which is surrounded by blanks will be moved. If p is not specified, the input-buffer pointer remains pointing to the character after the end of the last character moved or to the first character of a parameter, if the pointer was previously set by an S or Sp command or an F or B command.

Leading blanks are deleted from the parameter. The end of the parameter is designated by the first blank which is encountered, unless the entire parameter is enclosed in single quotes, in which case, the entire string including the surrounding quotes is moved.

If the primary output buffer is active (i.e., the stack is OFF), the parameter is copied with surrounding blanks if c is missing. If the character c is a backslash ( ), the parameter is copied without any surrounding blanks. When the form with c is used (where c is any non-numeric character except a left parenthesis), the character c surrounds the parameter. This feature is useful for picking up item-ids and values (which require double quotes) for processing by the ACCESS language processor. Note that c is inactive when the stack is ON (i.e., parameters are always copied to the stack as they are).

Multiple parameters may be moved to the primary output buffer via a single A command if these parameters are separated by semicolons in the input buffer. The parameters will be moved to the primary output buffer with the semicolons deleted and surrounded by blanks or by the character c, if c is specified. This function has no effect in the secondary output buffer.

After the execution of an A command, the input-buffer pointer points to the very next character after the string that was moved. Normally, this means the next blank or surround character following the last parameter in the buffer, if any. If there is no parameter, the A command causes no operation at all.

If the optional m is used, where m is a decimal number, only "m" characters of the parameter are moved to the output buffer. Examples of the A command are shown below. Each assumes that the output pointer is at the beginning of the buffer prior to the illustrated operation.

Buffer	Command	Buffer
Primary Input Buffer I AB CD EF GHI JK I ^	A *	Primary Output Buffer I CD I ^
Primary Input Buffer I AB CD EF GHI JK I ^	A5,2 **	Secondary Output Buffer I JK I ^
Primary Input Buffer I AAA BBB CCC I ^	A\2 *	Primary Output Buffer IBBB I ^
Primary Input Buffer I ABC DEF GHIJK I ^	A',2 *	Primary Output Buffer I 'DE' I ^
Secondary Input Buffer I ABC;DEF;GH JKL I ^	A" ***	Primary Output Buffer I "ABC""DEF""GH" I ^
Secondary Input Buffer I AAAA BB CCC D I ^	A2 ***	Primary Output Buffer I BB I ^

Active buffers prior to command execution:

* Primary or secondary input; primary output.
** Primary or secondary input; secondary output.
*** Secondary input; primary output.
^ Pointer

1.8 DATA INPUT (IS, IP, AND IT)

The IS command selects the secondary input buffer and accepts input from the terminal. The IP command accepts input from the terminal to the currently active input buffer. The IT command inputs the next tape label from tape. The general form of these commands:

Command	Description
IS{r}	Input operation from terminal to secondary input buffer; "r" is the prompt character.
IP{r}	Input operation from terminal to currently active input buffer; "r" is the prompt character.
IT	Input operation from tape to primary input buffer. Reads next tape label from attached tape unit.

The IS command selects the secondary input buffer as the currently active input buffer and inputs data from the terminal into the buffer. If the r specification is used, then that character is a prompt character at the terminal (r may be any character, including a blank). The prompt character will remain in effect until a new IS or IP command with a new r specification is executed. If r is omitted, then a colon (:) is used as a prompt. Data input by the user in response to the prompt is placed into the secondary input buffer. Subsequently, the data may be moved to an output buffer by using the A command. Any time the IS command is executed, input from the terminal overwrites all previous data in the secondary input buffer.

The IP command inputs data from the terminal into the currently active input buffer. Data input at the terminal in response to an IP command replaces the current parameter (i.e., as pointed to by the input pointer) of the currently active input buffer. If several parameters are input at the terminal, then they will all replace the current parameters in the buffer. If the input pointer is at the end of the data in the input buffer, then the new input data will be appended to the end. The r specification is identical to the r specification for the IS command.

The IT command inputs the tape label from the tape currently attached and copies that label into a cleared currently active input buffer. The IT command will first clear the currently active input buffer and then input the tape label into that buffer. If no tape label exists, then the command has the same effect as the RI command (i.e., it will reset the input buffer).

Examples of the use of these commands:

Command	Description
IS	Selects secondary input buffer and inputs data from terminal. Prompt character is a colon (:).
IS=	Selects secondary input buffer and inputs data from terminal. Prompt character is an equal sign (=).
IP?	Replaces current parameter in currently active input buffer with data from terminal. Prompt character is a question mark (?).
IT	Inputs tape label to primary input buffer. If no label, then input buffer is cleared.

1.9 DATA OUTPUT (O AND D)

The O command (capital letter O) is used to output a specified text string to the terminal. The D command is used to output parameters from either input buffer to the terminal. The general form of these commands:

Command	Description
O{text}{+}	'text' is output to terminal; '+' suppresses carriage return.
D{p}{,n}{+}	p'th or current (if p omitted) parameter of active input buffer is output to terminal; '+' suppresses carriage return.

The O command causes the text which immediately follows the O to be output to the terminal. If the last character of the text is a plus sign (+), then a carriage return will not be executed at the end of the text output. This feature is useful when using the O command in conjunction with an input command. For example, consider the following commands:

```
OPART-NUMBER+
IS=
```

These commands produce the following output on the terminal:

```
PART-NUMBER=
```

The specified prompt character (=) is displayed adjacent to the output text since the O command ended with a plus sign (+). The user then enters the input data right after the prompt character. For example:

```
PART NUMBER=115020
```

Further examples of the O command:

Command	Output to Terminal
OTHIS IS AN EXAMPLE	THIS IS AN EXAMPLE [CR]
OTHIS IS AN EXAMPLE+	THIS IS AN EXAMPLE

The D command is used to output parameters from either input buffer to the terminal. If the form Dp is used, then the p^{th} parameter of the currently active input buffer is displayed on the terminal. If the form D is used, then the current parameter (i.e., as pointed to by the input pointer) of the currently active input buffer is displayed on the terminal. If the form D0 (D followed by the number zero) is used, the complete currently active input buffer is displayed. If the forms Dp,n or D,n are used, then the n characters starting at the p^{th} or current parameter (up to the first blank character encountered) are displayed.

A plus sign (+) may be appended to the end of the D command, to specify the suppression of a carriage return. The D command does not affect the input pointer.

Examples of the D command:

Buffer	Command	Output to Terminal
Primary Input Buffer I AA BBB CC DDD I ^	D *	BBB [CR]
Secondary Input Buffer I AA BBB CC DDD I ^	D4+ **	DDD
Primary Input Buffer I ABC XYZ 123 I ^	D,2 ***	XY [CR]

Active buffer prior to command execution:

* Primary input buffer.
** Secondary input buffer.
*** Primary or secondary input buffer.
^ Pointer

1.10 TERMINAL OUTPUT AND CURSOR CONTROL (T)

The T command is used to specify terminal cursor positioning, to output literals, or to output non-keyable character codes. The cursor functions are terminal independent. The special terminal function codes are also available. The T command has the following general form:

T {function},{function}, ...

where:

{function} is any of the following:

"Text"	Causes the literal text to be output at the current position.
B	Causes a BELL code to be output.
C	Causes a Clear Screen code to be output.
Inn	Causes the integer character nn to be output.
Xnn	Causes the hex character nn to be output.
(X,Y)	Causes the terminal cursor to position to X,Y. This is controlled by the term type code. The special function codes (-1 through -10) are also supported.

The T command allows the user to create formatted screens in PROCs. The prompting and positioning of formatted screens generally appears cleaner and more acceptable to terminal operators. Note that this command does use the SYSTEM-CURSOR mode and so can be controlled terminal by terminal with the term type code. It is strongly recommended that the user employ the terminal independent control codes -1 through -10 in place of 'hard coding' these functions for a single terminal type. These codes are:

(-1)	Generates the clear-screen character; clears the screen and positions the cursor at 'home' (upper left corner of the screen).
(-2)	Positions the cursor at 'home' (upper left corner).
(-3)	Clears from cursor position to the end of the screen.
(-4)	Clears from cursor position to the end of the line.
(-5)	Starts blinking on subsequently printed data.
(-6)	Stops blinking.
(-7)	Initiates 'protect' field. All printed data will be 'protected' (i.e., cannot be written over).
(-8)	Stops protect field.
(-9)	Backspaces the cursor one character.
(-10)	Moves the cursor up one line.

The T command may be continued onto multiple lines by ending the preceding line with a comma. Also, comments may be added after the critical command letters. Thus, the code to clear the screen (C) could also be spelled out as "CLEAR"; the code for a bell (B) could be "BELL", etc. The T command never automatically adds a carriage return or line feed. Examples of T command:

Command	Output to Terminal
T C,B,(10,5),"TITLE"	This sequence first clears the screen. It outputs a bell code to the terminal. The cursor is positioned to column 10, row 5. The text "TITLE" is output.
T (0,8),(-4)	This positions the cursor at column 0, row 8. It then clears the entire line assuming that the terminal used supports that function.
T (-5),"twinkle",(-6)	This starts a blinking field, prints the word "twinkle" and ends the blinking field. This assumes the terminal supports blinking.
T CLEAR,"TITLE", (5,5) Comment,"TEXT"	This illustrates the continuation of a command over a line boundary and the insertion of a comment in the line.

1.11 SPECIFYING TEXT STRINGS AND CLEARING BUFFERS (IH, H, RI, AND RO)

The IH and H commands are used to place a specified text string in the currently active input or output buffer, respectively. The RI and RO commands are used to reset the input and output buffers (respectively) to the empty (null) condition. The general form of these commands:

Command	Description
IH{text}	'text' replaces current parameter of active input buffer
H{text}{<}	'text' is placed in active output buffer; carriage return specification included.
RI{p}	primary input buffer (from p^{th} parameter and on, if p is used) and secondary input buffer are reset.
RO	both output buffers are reset.

The IH command causes the text (including any blanks) immediately following the IH to replace the current parameter (as specified by the input pointer) in the currently active input buffer. The input buffer pointer will remain pointing to the beginning of the inserted string.

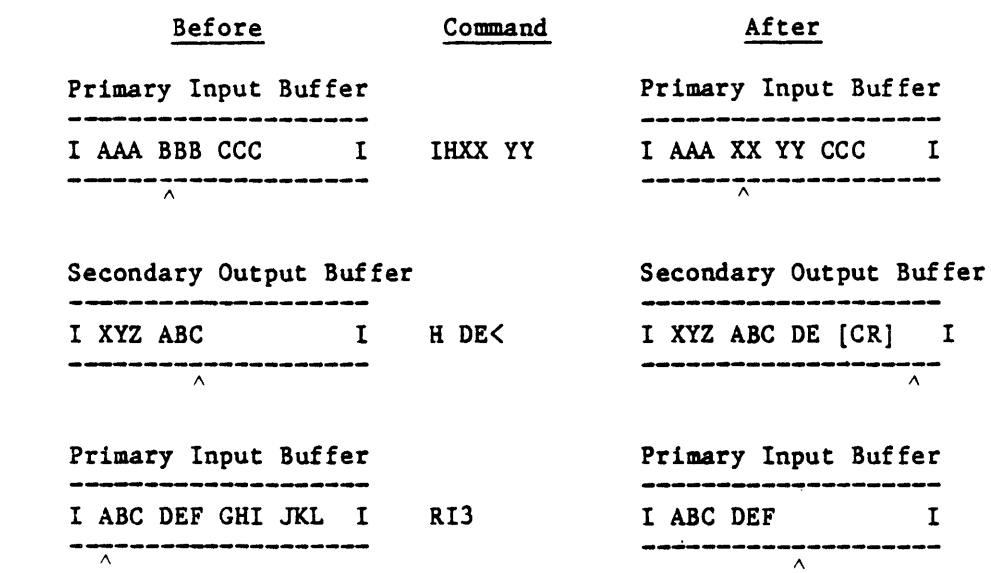
The H command causes the text (including any blanks) which immediately follows the H to be placed in the currently active output buffer at the position pointed to by the output pointer.

When the last parameter of a desired output line has been moved to the secondary output buffer (the stack), a carriage return specification (<) must be placed in the stack. For example, the command HXYZ< would be used to place in the stack the text XYZ followed by a carriage return, while the command H< would place a carriage return (only) in the stack. If there is no carriage return in the stack, the system will place one after the last character in the stack.

If the form RI is used, then both input buffers are reset to the empty (null) condition. If the form RIp is used, then the primary input buffer from the p'th parameter to the end of the buffer (as well as the entire secondary input buffer) are reset to the empty (null) condition. The RI command also selects the primary input buffer as if an SP command has been executed.

The RO command (capital letters R and O) resets both output buffers to the empty (null) condition. The RO command also selects the primary output buffer as though a STOFF command had been executed.

Examples of the use of these commands:



1.12 TRANSFERRING CONTROL (GO AND SIMPLE IF)

Transfer of control (i.e., branching) may be specified within a PROC via use of the GO and IF commands. The GO command provides an unconditional branch capability, while the IF command provides a conditional capability. The general form of these commands:

General Form	Description
GO n or G n	Control is transferred to the statement with 'n' as the numeric label.
IF {#}a-cmnd proc-cmnd	The IF command provides for the conditional execution of a specified PROC command.

The GO command causes control to transfer to the PROC command which begins with the label n. The user should note that several PROC commands may begin with the same label. If this is the case, the GO command transfers control to the first PROC command which begins with the specified label (scanning from the top).

The PROC branch command, G or GO, allows for variable branching. Specifically, the user may use commands of the form GO A or GO An, where "A" and "An" reference specific parameters in the primary input buffer. These commands will cause a branch to the label referred to by the contents of these buffer parameters. Note that if the label referenced does not exist, the PROC will simply continue with the next statement following the branch instruction. For example:

```
001 PQ                          Define PROC
002 RI                          Clear input buffers
003 O ENTER MENU NUMBER +
004 S1                          Point to parameter position
005 IP:                         Get response from CRT
006 GO A1                       Branch based on response
007 X-INVALID RESPONSE!         Missing label number
.
.
.
```

The IF command takes on three forms in the general form, where a-cmnd is any legal form of the A command (refer to Section 1.7), except for the form using the character surround feature (i.e., Ac), and where proc-cmnd is any legal PROC command. If the optional # is not used, the IF command simply tests for the existence of a parameter in the input buffer as specified by the A command. If a parameter exists, the specified PROC command is executed; otherwise, control passes to the next sequential PROC command. For example:

```
IF A2 GO 15
```

This command tests for the existence of a second parameter in the currently active input buffer. If a parameter exists, control passes to the PROC command beginning with label 15; otherwise control passes to the next sequential PROC command. If the # option is used, the test is reversed. For example:

```
IF #A2 GO 15
```

This command causes control to transfer to the command with label 15 if a second parameter does not exist. Note that when using an A command as a test condition of an IF command, parameters are not moved to an output buffer as they would be if the A command were used alone. Rather, the A command is used simply to specify which parameter in the input buffer is to be tested. However, the input pointer will be repositioned as specified by the A command.

A number of examples illustrating the simple form of the IF command are shown below. For a discussion of the other two forms of the IF command, refer to Sections 1.13, Relational Testing, and 1.14, Pattern Testing.

NOTE

The following examples assume that the primary input buffer is the currently active input buffer and contains the following parameters:

```
--------------------
I ABC AAA XYZ       I
--------------------
  ^
```

Command	Explanation
IF A GO 27	Control is transferred to the command with label 27.
IF A3 OHELLO	Message HELLO is output to terminal; control then continues with next sequential command.
IF A4 OHELLO	Message is not output; control continues with next sequential command.
IF A1,1 G 2	Control is transferred to the command with label 2.

1.13 RELATIONAL TESTING (IF)

The relational form of the IF command allows parameters in the input buffers to be tested relationally. The relational form of the IF command is an extended version of the simple IF form (see Section 1.12, Transferring Control) The relational form is:

IF a-cmnd op string proc-cmnd

where:

a-cmnd and proc-cmnd	are as defined for the simple IF form.
op	is one of the relational operators listed in Table 1-2.
string	is a literal string of characters which the parameter is to be compared against.

For example:

IF A,3 = YES GO 5

Here the PROC would transfer control to the command with the label 5 if the first 3 characters of the current parameter in the currently active input buffer are YES.

To resolve a relational condition, character pairs (one from the selected parameter and one from the literal string) are compared one at a time from leftmost characters to rightmost. If no unequal character pairs are found, the strings are considered to be equal. If an unequal pair of characters are found, the characters are ranked according to their numeric ASCII code equivalents. The character string contributing the higher numeric ASCII code equivalent is considered to be greater than the other string. For example, AAB is considered to be greater than AAAA, and 02 is considered greater than 005.

If the selected parameter and the literal string are not the same length, but the shorter of the two is otherwise identical to the beginning of the longer one, then the longer string is considered greater than the shorter string. For example, the string WXYZ is considered to be greater than the string WXY.

Further examples illustrating the relational IF command are:

NOTE

The following examples assume that the primary input buffer is the currently active input buffer and contains the following parameters:

```
--------------------
I ABC AAA XYZ       I
--------------------
  ^
```

Command	Explanation
If A = ABC GO 3	Control is transferred to the command with label 3.
IF A3 > XYX HTEST	The text string TEST is placed in the currently active output buffer; control then continues with next sequential command.
IF A2 > XYX HTEST	Text string TEST is not placed in output buffer; control continues with next sequential command.
IF A1,2 = AB GO 7	Control is transferred to the command with label 7.

Table 1-2. Relational Operators

Operator Symbol	Operation
=	Test for equal.
#	Test for not equal.
<	Test if parameter less than literal string.
>	Test if parameter greater than literal string.
[	Test if parameter less than or equal to literal string.
]	Test if parameter greater than or equal to literal string.

1.14 THE PATTERN MATCHING IF COMMAND

The pattern matching form of the IF command allows parameters in the input buffers to be tested for a specific pattern match. The pattern matching form of the IF command is an extended version of the simple IF form (see Section 1.12, Transferring Control). The pattern matching form is:

```
If a-cmnd op (pattern) proc-cmnd
```

where:

a-cmnd and proc-cmnd	are as defined for the simple IF form.
op	is one of the relational operators described for the relational IF form.
pattern	is used to test a parameter for a specified combination of numeric characters, alpha characters, alphanumeric characters, or literals.

The pattern specification in an IF statement consists of any combination of the following:

- An integer number followed by the letter N (which tests for that number of numeric characters).
- An integer number followed by the letter A (which tests for that number of alpha characters).
- An integer number followed by the letter X (which tests for that number of alphanumeric characters).
- A literal string (which tests for that literal string of characters).

As an example, consider the following command:

```
If A = (3NABC) G 3
```

This command causes a transfer of control to the command with label 3 when the current parameter of the currently active input buffer consists of three numerals followed by the characters ABC (e.g., 123ABC).

If the integer number used in the pattern is 0, the test is true only if all the characters in the parameter conform to character type. The following command, for example, outputs the message OK if the characters of the current parameter are all alpha characters:

```
IF A = (0A) OOK
```

Further examples of the pattern matching form of the IF command:

NOTE

The following examples assume that the primary input buffer is the currently active input buffer and contains the following parameters:

```
----------------------------
I ABC 10/09/77 XYZ B123C 33  I
----------------------------
  ^
```

Command	Explantion
IF A = (3A) G 7	Control is transferred to the command with label 7.
IF A2 = (2N/2N/2N) G 5	Control is transferred to the command with label 5.
IF A4 = (0N) G 9	Control continues with next sequential command.
IF A5 = (0N) GO 2	Control is transferred to the command with label 2.
IF A4 = (1A3NC) OGOOD	The message GOOD is output to the terminal; control continues with next sequential command.
IF A1 = (3X) IF A1 > ABB G 9	Control is transferred to the command with label 9.

Note that for any of the three IF command forms, the PROC statement which is conditionally executed may, in turn, be another IF command (i.e., IF commands may be nested). The following command, for example, transfers control to label 99 if the current parameter consists of two numerals in the range 10 through 19 (inclusive):

```
IF A = (2N) IF A ] 10 IF A [ 19 GO 99
```

You may wish to visualize nested IF commands as though implied AND operators were placed between them.

1.15 FURTHER FORMS OF THE IF COMMAND (IF E AND IF S)

The IF E form of the IF command may be used to test for errors generated by a preceding PROC-generated statement. The IF S form of the IF command may be used to test whether a LIST, as generated by a SELECT, SSELECT, QSELECT, or GET-LIST statement, is in effect. The general form of IF E and IF S:

```
IF {#}E {op string} proc-cmnd

IF {#}S proc-cmnd
```

The IF E command allows PROCs to test for system-generated errors (as specified in the ERRMSG file). The E command is valid only after a P type command (i.e., when a PROC-generated statement has completed execution and control is returned to the PROC). The E command uses the secondary input buffer and, therefore, is valid only until an IS command is executed.

The errors tested for may be unspecified (i.e., any error) or they may be specified by the error number. The relational operators "=", ">", "<". "[", "]" may also be used to test for errors in specified ranges. Thus, the error command may be used in two ways. An example of the first would be:

```
015 IF E X ENCOUNTERED AN ERROR AT LINE 15
```

whereby control will transfer to TCL and the text "ENCOUNTERED AN ERROR AT LINE 15" will be printed if any error were encountered.

An example of a statement that tests for an error range is:

```
015 IF E > 91 IF E < 99 X TAPE ERROR!
```

in which case control will transfer to TCL and the text will be printed if an error in the range 92-98 has been encountered.

There are certain TCL statements that select lists of item-ids or values, such as SELECT, SSELECT, QSELECT, and GET-LIST. Refer to the ACCESS manual for details regarding these statements. There is an important interaction between these statements and a PROC. A selected list must be used by the TCL statement immediately following it or else it will be lost. If the select-type statement has been executed by a PROC, the TCL statement that uses it is normally placed in the stack prior to execution of the select statement. This second TCL statement will automatically execute after the select is complete; the PROC will not gain control in between. If there is a null line in the stack, the PROC will then regain control. The PROC may then test if the select statement executed correctly.

The IF S command will test for the presence of a selected list; the selected list will be present only if a select-type TCL statement has already been executed at the time that the IF S command is encountered.

If the select statement has generated an error, such as "NO ITEMS PRESENT" or "ITEM NOT ON FILE," the select list will not exist and the IF S may be used to check on this condition.

Examples of the IF S usage:

```
    TEST1
001 PQ
002 HGET-LIST
003 OENTER LIST-NAME+
004 IP?
005 A
006 STON
007 H<
008 P
009 IF #S XILLEGAL LIST-NAME!
010 HLIST INVENTORY LPTR
011 P
012 next statement
```

```
    TEST2
001 PQ
002 HGET-LIST
003 ENTER LIST-NAME+
004 IP?
005 A
006 STON
007 HLIST INVENTORY LPTR
008 P
009 next statement
```

The PROC's TEST1 and TEST2 will operate identically if the GET-LIST statement executes without an error (i.e., if the list exists on file). However, TEST2 will continue with PROC execution even if the list is not on file, since there cannot be an IF S test after the stacked LIST statement executes. TEST1, on the other hand, has a null line in the stack when the GET-LIST executes; therefore, control is returned to the PROC, which can test to see if it executed properly. If the list is not on file, the PROC will terminate on line 9.

1.16 ADDITIONAL COMMANDS (PLUS (+), MINUS (-), U, AND C)

The Plus and Minus commands are used to add or subtract (respectively) a specified decimal number to/from the current parameter of the currently active input buffer. An exit to a user-defined subroutine may be accomplished via the U command. The C command is used to place comments within the body of the PROC. The general form of these commands:

General Form	Description
+n	Decimal number is added to current parameter of active input buffer.
-n	Decimal number is subtracted from current parameter of active input buffer.
Umode-id	'Umode-id' exits to user-defined subroutine.
C{text}	Comment is ignored by PROC processor.

The Plus (+) command causes the decimal number n to be added to the current parameter (as pointed to by the input pointer) of the currently active input buffer. The current parameter must be numeric.

The Minus (-) command causes the decimal number n to be subtracted from the current parameter (as pointed to by the input pointer) of the currently active input buffer. The current parameter must be numeric.

The Plus or Minus commands will have no effect if the input pointer is currently at the end of the buffer. Also, the user must take care that the updated value of the parameter is the same length as the original parameter, since no automatic check for this is made.

The U command is used to provide an exit to a user-defined subroutine. The format for this command is identical to the P command using the mode-id option; however, the U command is meant to be used for a simple subroutine call. Upon return from the subroutine, control is passed to the command immediately following the U command.

WARNING

Do not use the U command unless you fully understand its action at the system assembly level.

The C command is used to place comments within the body of the PROC. The general form of this command is as follows:

```
C{text}
```

All the text following the C command will be ignored by the PROC processor. For example:

```
013 C THIS IS A COMMENT
```

The C command may be used freely throughout the PROC for purposes of clarity and documentation; however, note that making a PROC excessively long will slow its execution.

Sample use of the Plus and Minus commands:

Before	Command	After
Primary Input Buffer I ABC 001 XYZ I ^	+99 *	Primary Input Buffer I ABC 100 XYZ I ^
Secondary Input Buffer I XXXX YY 39 I ^	-5 **	Secondary Input Buffer I XXXX YY 34 I ^
Primary Input Buffer I ABC 001 XYZ I ^	+99 *	Primary Input Buffer I ABC 001 XYZ I ^

Active buffer prior to command execution:

* Primary input buffer.
** Secondary input buffer.
^ Pointer

1.17 PROC EXECUTION AND TERMINATION (P, PH, PP, PW, PX, AND X)

The P command causes the PROC to execute via TCL. The PH command is similar to the P, but causes terminal output to be suppressed. The PX command acts like the P, but turns off any further interaction with the PROC. The PP and PW commands are identical to the P command, except that the content of both output buffers are displayed at the terminal prior to execution. The X command is used to exit from the PROC. The general form of these commands:

General Form	Description
P	Causes PROC to execute the primary output buffer area.
PX	As above, PROC control is terminated after the statement has executed.
PH	Causes the PROC to execute with terminal output suppressed totally.
PP	Causes PROC to execute after displaying content of both output buffers.
PW	Causes PROC to display both output buffers and wait for command from the terminal.
X{text}	Causes termination of PROC and display of optional text message.

The P command causes the PROC to execute by submitting the content of the primary output buffer to TCL for processing; the contents of the stack (if any) is used to feed interactive processors such as BASIC or EDITOR. After execution via TCL, the PROC regains control at the statement immediately following the P command.

The PX command acts just like the P does, with the exception that control is not returned to the PROC after the TCL statement has been executed.

The PP command causes execution of the contents of the buffer just like the P command, except that the content of both output buffers is displayed on the terminal.

The PW command acts in the same way as the PP command, except that after the data is displayed, terminal input is then requested via a question mark (?) prompt character. If the user enters an S, the current PROC-generated command is skipped and PROC execution continues at the command following the PW. If an X is entered, PROC execution is aborted and an exit is taken to TCL.

The PH command executes the contents of the buffer, but suppresses any output from the executed process.

Any other character will cause PROC action to continue. The PW command is normally used as a debugging tool and may be replaced by a P command once the user has determined that the PROC is functioning properly.

The X command is used to exit from the PROC. Normally, PROC control is terminated with execution of the final PROC statement, in which case an X command is not needed. However, the X command may be used at intermediate points in the PROC coding to cause termination of the PROC. Any text following the X will be output as a message upon termination of the PROC. For example:

```
X***EXIT TO TCL***
```

If the PROC was called as a subroutine, the X command will cause a return to the calling PROC.

Examples of the use of these commands:

Command	Explanation
X	PROC is terminated. However, If the PROC was called as a subroutine, then the X command will return control to the calling PROC and continue at the next command.
XHURRY BACK	PROC is terminated and the message "HURRY BACK" is displayed on the terminal.
XHURRY BACK+	As above; the message is printed without a carriage return/line feed appended.

1.18 LINKING TO OTHER PROCS

A Link command in one PROC causes control to transfer to the first command of another PROC, which may reside in any dictionary or data file. This allows the storage of PROCs (except for the LOGON PROC) outside of the MD. Also, large PROCs can be broken into smaller PROCs to minimize processing time.

The general form of the Link command is:

```
({DICT} file-name {item-id}) {n}
```

where:

DICT	If used, specifies the dictionary portion of the file.
file-name	Specifies the file.
item-id	If used, specifies the name of the PROC invoked. If omitted, the current parameter (as specified by the input pointer) of the currently active input buffer is retrieved and used as the item-id.
n	If used, control is transferred to the line whose label is 'n'.

The first line of the linked-to PROC is skipped, since it is assumed that this line contains the PQ code.

As an example of the Link command, consider the situation where a PROC named EXECUTE is used to execute any one of a series of PROCs in a file named PROC-FILE. The specific PROC executed is specified by a single-character alphabetic code input by the user. This sample PROC is shown below:

Item EXECUTE

```
001 PQ
002 OPLEASE INPUT CODE+
003 IS?
004 IF A = (1A)  (PROC-FILE)
005 XILLEGAL RESPONSE
```

If, for example, the user's response to the IS command of line 3 is the character D, then line 4 of the PROC (which contains a Link command as part of the IF command) transfers control to the PROC stored in item 'D' of file PROC-FILE.

Consider next the situation where the PROC named LISTU previously was present in each user's MD. Assume that LISTU was then moved to the dictionary section of a file named PROCLIB and the PROC shown below was then placed in each user's MD. The LISTU PROC which was moved to the PROCLIB file will now be invoked by the Link command in the PROC shown below and, thus, the LISTU PROC need not be duplicated in each user's MD.

Item 'LISTU' in MD

```
001 PQ
002 (DICT PROCLIB LISTU)
```

Note that the PROC buffers remain unchanged when a linkage occurs. Also, the first line of the linked-to item is always skipped, since it is assumed that this line contains the PQ code.

1.19 SUBROUTINE CALL COMMANDS

One PROC can call another as a subroutine, or a local subroutine call can be invoked using the call command. The subroutine call commands have the following general form:

General Form	Description
[] n	Local subroutine call to line "n"; returns on first X command following the label "n".
[{DICT} file-name {item-id}] {n}	External subroutine call; transfers control to label n of the called subroutine. Returns on first X command encountered. If item-id is not specified, the current parameter of the currently active input buffer will be used.

The local subroutine call command will store the location of the next PROC command in the PROC subroutine stack and transfer control to the command whose label is n. Excecution of PROC commands continues from that point (including P, PP, and PW commands), until an X command is executed, which will return control to the PROC command following the call command.

In the external subroutine call, "DICT", "file-name", and "item-id" are identical to the Link command described in Section 1.18. If item-id is not specified, the name of the called subroutine is taken from the current parameter (as specified by the input pointer) of the currently active input buffer.

The optional n indicates that subroutine execution is to begin at label n, rather than at the second line of the subroutine PROC.

As with local subroutine calls, an X command will return control to the calling PROC.

In both forms of the subroutine call, none of the input or output buffers are affected by the call itself.

Examples of subroutine calls:

Local Subroutine Call

```
001 PQ
002 [] 3                    NOTE:  '+' suppresses carriage
003 OFIRST                         return after X returns.
004 3 OSECOND
005 X+
```

Output on Terminal

```
SECOND
FIRST
SECOND
```

External Subroutine Call

```
001 PQ
002 [MD LISTU]
003 ODONE WITH LISTU
```

Output to Terminal

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

 00 0200 SP        08:00AM 01/01/78 Channel 0
 02 0240 CM        09:10AM 01/01/78 Channel 2
 03 0260 LC        07:30AM 01/01/78 Channel 3
 04 0280 JP        10:14AM 01/01/78 Channel 4
*06 02C0 SAL       08:35AM 01/01/78 Channel 6
 10 0340 JET       09:00AM 01/01/78 Channel 10
 DONE WITH LISTU
```

1.19.1 PROC EXAMPLES

A sample EDITOR operation which changes attibute 3 of item 11115 of file ACCOUNT to the value ABC is as follows:

```
>EDIT ACCOUNT 11115 [CR]
TOP
.G3 [CR]
003 100 AVOCADO
.R [CR]
003 ABC
"11115" FILED
```

A PROC named CHANGE which will perform the same operation is as follows:

Item CHANGE IN MD

```
001 PQ
002 HEDIT
003 A2
004 A3
005 STON
006 HG
007 A4
008 H<
009 HR<
010 A5
011 H<
012 HFI<
013 P
```

Note that the PROC has been written in such a manner that it updates any specified attribute in any specified item in any specified file. The format used to invoke this PROC is as follows:

```
CHANGE file item attribute-no new-value
```

If, for example, the user wishes to perform the same operation shown in the first example, the PROC must be invoked as follows:

```
>CHANGE ACCOUNT 11115 3 ABC  [CR]
```

The user should note that the normal messages output by the EDITOR (e.g., TOP, 11115, FILED, etc.) are output when the PROC is executed. These messages may be suppressed, however, by preceding each EDITOR command by a period (.); for further information regarding these features, refer to the EDITOR manual.

1.19.1.1 SSELECT and COPY Verbs

A sample operation at the TCL level using the SSELECT verb and then the COPY verb is as follows:

```
>SSELECT INVENTORY WITH QOH > "900" BY-DSND QOH [CR]
19 ITEMS SELECTED
>COPY INVENTORY [CR]
TO: (HOLD-FILE) [CR]
19 ITEMS COPIED
```

An identical operation is performed by the PROC named TEST shown as follows:

Item 'TEST' in MD

```
001 PQ
002 HSSELECT INVENTORY WITH QOH > "900" BY-DSND QOH
003 STON
004 HCOPY INVENTORY<
005 H(HOLD-FILE)<
006 P
```

Upon execution of the TEST PROC, the output buffers contain the data shown as follows:

Primary Output Buffer

```
--------------------------------------------------
I SSELECT INVENTORY WITH QOH >"900" BY-DSND QOH [CR]   I
--------------------------------------------------
```

Secondary Output Buffer

```
--------------------------------------------------
I COPY INVENTORY [CR] (HOLD-FILE) [CR]                 I
--------------------------------------------------
```

Note that SSELECT sentence is contained in the primary output buffer, while the secondary output buffer contains both input elements of the copy operation, each terminated by a carriage return.

For further information regarding the SSELECT verb, refer to the ACCESS Manual. For further information regarding the COPY verb, refer to Introduction to PICK TCL and FILE STRUCTURE Manual, or to the Utilities Manual.

1.19.1.2 PROC Use of Variable Testing (GO and D Commands)

A sample tape positioning PROC is:

```
      T-SPACE
001 PQ
002 4 IF #A2 GO 3
003 IF A2 = (ON) G 7
004 IH000
005 7 S3
006 IH00
007 HT-ATT
008 P
009 IF E = 95 X
010 IF E = 93 X
011 2 HT-RDLBL
012 P
013 IF E = 94 GO 9
014 HP (I)
015 P
016 HT-FWD
017 P
018 HP (L)
019 P
020 S3
021 +1
022 S2
023 -1
024 IF A = 0 X
025 IF A = 00 X
026 IF A = 000 X
027 GO 2
028 3 ONO. OF FILES+
029 IP?
030 GO 4
031 9 OEND OF RECORDED DATA - (+
023 D3+
033 X FILES)
```

It differs from previous examples in that it uses the arithmetic command. It has practical value in that the user does not have to enter T-FWD at the TCL level for every file that is positioned over.

Note that PROC may be executed by entering "T-SPACE" or "T-SPACE n" at the TCL level; if no parameter is entered, the PROC will request one.

The input buffer contains the following data:

Parameter#	1	2	3	4
	T-SPACE	n	xx	

where:

n	is the number of files to be spaced over
xx	is the count of such files, initialized to "00" at line 6

The PROC will attach the tape unit (T-ATT); check for errors 95 and 93, terminating execution if either error occurs. Then it will execute a T-RDLBL to read the tape label and print it on the terminal; if error 94 (EOF) occurs on this statement, the end-of-tape data has been reached; the message on line 31 will be printed along with the file-count from parameter 3.

If T-RDLBL executes successfully, the tape file is spaced over by executing a T-FWD (the print is turned off and on around this command by the command P (I) and P (L), to inhibit spurious messages).

This is repeated until parameter 2 goes to 0; note the multiple tests required to test for 0, 00, or 000 on lines 24-26, since the - command doesn't change the parameter size.

1.20 PROC TAPE AND CARTRIDGE DISK CONTROL

The tape and cartridge disk drive units are an exceptional part of the computer system, since, unlike essentially all the other devices and structures in the system, they cannot be shared. Therefore, their availability and condition are of great importance to any sequence of processes running under PROC control which use the tape or cartridge disk drive, since PROCs can not mount tape or cartridge disk, and they can not continue when the tape or cartridge disk is inoperative.

Note that the tape and cartridge disk control error message numbers are 84 through 107, and are to be found in the ERRMSG file.

Also, note that certain tape and cartridge disk difficulties are not amenable to PROC control because they interrupt the process and speak only to the operator of the terminal. In these cases, human intervention and judgment is necessary.

Because of the nature of the tape and cartridge disk, it is advisable to specifically attach the tape or cartridge disk each time it is used, by means of the T-ATT verb. The T-ATT verb will return one of two possible patterns. If the device is available or is already attached, it will return:

```
90 nmnn
```

in the PROC secondary input buffer. The number 90 is the ERRMSG number which specifies that the device is attached. The string 'nmnn' is the tape or cartridge disk block size as it is displayed on the screen. When using tape, it is in general advisable to use an explicit block size with the T-ATT verb, because there may be cases when the prior state of the process is unanticipated. (With cartridge disk, the block size is always 1024 bytes.)

If the tape or cartridge disk is not available, the T-ATT verb will return:

```
95 nn
```

in the PROC secondary input buffer, where 95 is the ERRMSG number indicating that the device is not available, and the string 'nn' is the number of the line which has the device attached.

Another problem with the tape and cartridge disk concerns the modification of the spooler which removes the tape and cartridge disk drive from spooler control. If you wish to send the print file to tape or cartridge disk on completion under PROC control, but it is inconvenient to have the drive attached to the print-file generating process because several different lines are executing the same function and all wish to send their current print file to the tape or cartridge disk, then it is convenient to send the print-file to the tape or cartridge disk from the SP-EDIT process at the completion of generation, using at least the T and W options of the SP-EDIT verb.

The T option will force the output to tape or cartridge disk under a SPOOL Y condition, and the W option will cause the process to wait until the drive becomes available.

If a fully automatic process is desired, the MS options may be added to the SP-EDIT verb. For example:

```
verb which generates a print file
PQ                Execute the verb.
SS                Get the print file entry number.
B
5 IF # A XNO DATA.
IF A = 1099 G 15
F
G 5
15 F
IF A # (ON) XBAD DATA.
HSP-EDIT          Go SP-EDIT the print file.
A                 Move the print file number to the primary output buffer.
H MSTW            Spool to tape or cartridge disk; wait until drive is
                  available.
PP                Yields SP-EDIT n MSTW.

Process next or exit
```

index