TABLE OF CONTENTS

1.1 AN OVERVIEW OF CHANGES IN R80.

R80 is composed of a group of incremental enhancements to Pick & Associates R77 release. The primary emphasis is on enhancement of software speed and reliability through the clarification, refinement, and standardization of the system code. This has occurred in essentially all of the processors which compose the system. Most of the modifications are transparent to the user, and will not be discussed below.

1.2 CHANGES IN THE TERMINAL INTERFACE.

1.2.1 The terminal handler activates on each character.

The R77 release automatically activated the process receiving data from a terminal after 11 bytes or the occurrence of a carriage-return, whichever occurred first. Now the process will activate on each character input from the terminal. It is thus possible to obtain one character and activate without a carriage-return, which allows certain features to be added. If you are sending a long report with pagination to the terminal, it may be terminated by executing a CONTROL-X, rather than with a BREAK and END sequence. The feature is used similarly in the SP-EDITor with respect to the T response to the SPOOL prompt, and in the file save processor at mount-new-reel time.

1.2.2 Avoidance of BREAK and END.

There has in general been an attempt to remove the need for BREAK and END from standard system use. The three most common and unnecessary uses have been when displaying long listings to the terminal, using the form EDIT FILENAME *, and under SPOOL T in the SP-EDITor. All of these now allow an escape before the end of execution of the stated task without the use of BREAK and END. The BREAK and END are as they were, and are available as needed.

1.2.3 Changes to the handling of the ESCAPE character.

The ESCAPE character (X'1B') now passes into the system as a start buffer mark (X'FB') for use with the prestore facility in the EDITOR. The result of this is that the ESCAPE is no longer echoed as an ESCAPE. It is echoed to the terminal as a X'5B' ([). This means that command sequences issued from the keyboard to the terminal which are implemented as an escape sequence will not work because the ESCAPE will not be echoed to the terminal. The easiest way around this is to put the terminal into local mode, execute the escape sequence, take the terminal back out of local mode, execute a CONTROL-A for good measure, and press on. Local mode is usually entered and exited by depressing a 'local' key, which will toggle a flag in the terminal. Should there be no way to get into local mode with a given terminal, then a series of one-line BASIC programs can be written to send the escape sequence to the terminal. In general, there may be command sequences which have meaning to a particular terminal which also have meaning to the system input processor. The special input control characters are CONTROL-P, CONTROL-R, and CONTROL-W. They must be implemented with BASIC programs if they are necessary for terminal control. For instance, the BASIC program 'PRINT CHAR(18)' will send a

CONTROL-R to the terminal.

Several of the verbs which do not reference files or items have been modified so that they do not need the left parenthesis before any options which might be desired. See the relevant section in the documentation on the spooler.

1.3 CHANGES TO THE FILE HANDLER.

The file-open routine has been extensively changed (and moved, if you use assembly programs). The effect is transparent, but allows better system security and more extensive use of Q pointers. The following table defines the level referenced by each field.

	FILENAME	The name of the file to be used in file open calls.
001	Q	Specifies this as a Q-pointer.
002	ACCTNAME	The account name
003	FILENAME	The name of the file referenced.
004		Optional, as are all subsequent.

Basic Q-pointer form.

1.3.1 The reflexive form of the Q-pointer.

If attributes two and three are null, the Q-pointer is a pointer to the file in which it is stored. This case has two applications. If, on an R80 system you type ED MD MD, you will find that the only contents of the MD item is a Q in attribute 1. This is in general sufficient, and any other definition is less efficient. Specifically, the MD entry should not be a D-pointer. The same follows for M/DICT or the account name entry

The second use is in the case of a dictionary-only file. If it is desired to reference the file without typing 'DICT' 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 contents is a Q. It is legal, but less efficient, to have an R77-style D-pointer in the dictionary which is identical to the D-pointer in the master dictionary.

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.

Uses of Q as the only attribute.

Note that 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 find the D-pointer FILENAME in the dictionary of FILENAME. It will not find a pointer by the name of QFILENAME. A partial exception to this is in ENGLISH, which will attempt to obtain the conversion, length and justification from the Q-pointer. If the Q-pointer does not contain them, then the ENGLISH 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 different formats for the item-id field for purposes of sorting and listing.

1.3.2 Account specification.

The second attribute in 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 error 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. The effect is to cause the system unnecessary work.

Reference to the master dictionary of another account is done with the name of the D-pointer to the account in attribute 2 and a null attibute 3.

1.3.3 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. The R77 version with MD in attribute 3 will not work, since MD is now a Q-pointer. The following error message will be issued:

[201] 'MD' IS NOT A FILE NAME.

This occurs because MD is now a Q-pointer. Note that backward conversions from R80 to R77 will require that MD be redefined as a D-pointer.

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.

1.3.4 Extensions to the file name reference.

R77 introduced the facility of multiple data files referenced from a single dictionary. These were, and are pointed to by D-pointers in the dictionary of the file whose names are the names of the various data files. The technique used to reference a specific data file is by using the string

FILENAME,DATAFILENAME.

Under R80 the contents of attribute 3 of the Q-pointer definition may

contain FILENAME,DATAFILENAME. In this case the Q-pointer will reference the data in DATAFILENAME only, and will ignore the other data files referenced in the dictionary of FILENAME. The result is a considerable simplification of the BASIC programs and PROCS which reference the various data sets in a multiple-data-file structure.

Therefore the following Q-pointer will reference the data file DATAFILENAME in the dictionary of FILENAME in the account ACCOUNTNAME.

```
QFILENAME
001 Q
002 ACCOUNTNAME
003 FILENAME,DATAFILENAME
```

Referencing a data file with a Q-pointer.

Note that in the multiple data file case there may or may not be a data file with the same name as the dictionary of the file. It may be convenient to create the file as a dictionary-only file and insert a Q-pointer in the dictionary with the same name as the file. The dictionary may then be referenced without typing 'DICT'. This may be useful if there is to be extensive development of many data definition items. In general, there does not need to be a data file with the same name as the dictionary.

1.3.5 Modifications to the file structure.

1.3.5.1 BASIC program files.

Two changes in file protocols have occurred. The first has to do with BASIC program files, and is treated in the section on BASIC. Simply, the program file must have a dictionary level and one or more data level files. The source code must be in the data level file or files. If there are multiple data files, and a program of the same name in more than one of them, strange things may happen. The dictionary level of the file will contain pointers to contiguous strings which contain run modules emitted by the BASIC compiler. The master dictionary entry for the BASIC program file must contain a 'DC' in attribute 1.

1.3.5.2 The POINTER-FILE.

Secondly, the common POINTER-FILE has been done away with. Each account or group of accounts may be allocated a POINTER-FILE by the system manager. The account name of the account generating a list is no longer stored as part of the list name. As such it is possible to use the default list name '' (null), but it is not advisable to do so.

Every pointer-file must contain a 'DC' in attribute 1 of its definition, as before. It must be two-level, but it is convenient to make the pointer to the data level in the dictionary of the pointer-file a Q-pointer. The name POINTER-FILE is reserved and known to the list handler. It is therefore possible, and may be convenient, to call the actual pointer-file or files by names different than POINTER-FILE, and construct POINTER-FILE as a Q-pointer to whatever pointer-file is desired at the moment. In this

way any one pointer-file may be referenced.

1.3.6 Modifications to CREATE-FILE et alia.

The CREATE-FILE, DELETE-FILE, AND CLEAR-FILE verbs used to require a left parenthesis, (, prior to the file name, or the key words DICT or DATA. The change is minor, and the old form still works, but it is consistent with Pick & Associates' commitment to less typing. Note that Q-pointers may be used to reference files which are to be cleared, but that the D-pointer name is required if the file is to be deleted. The CREATE-FILE verb constructs a D-pointer. If a data file is being added to an existing dictionary, the pointer to the dictionary will be a D-pointer.

1.4 CHANGES IN THE EDITOR.

The primary improvements in the EDITOR are the addition of prestored commands, a considerable improvement in the speed of the string locate, replace and update commands, and extension of the merge and file instructions.

The increase in speed in the locate, replace, and delete commands is due to the use of more powerful instructions which are part of the R80 firmware and are otherwise transparent. The same instructions have proliferated across other processors in the system, but will not generally be noted. There are simply various places where the system will run locally faster than it did under R77, but this is one of the few that is at all visible to the user.

This discussion will begin with extensions to the EDITOR introduced during R77, because such documentation as they received were a few crabbed lines inserted in an already cramped text.

1.4.1 Printing unprintable characters.

Characters which are unprintable include the control characters, between X'00' and X'1F', inclusive. Under R80 the EDITor marks control characters by inserting a period, '.', where the control character stands in the text line. It does not indicate what the character is, however. It may then be removed by replacing a unique string which includes the control character with the string of your choice. The control character should be marked with an ^ in the first string in the replace.

1.4.2 Changes to the Replace command.

Between R77-1C and R77-1D the replace command was extended to allow several executions of the replace command on a single line, and an option was made available to allow replacement of all copies of a string within a line with the specified replacement string.

1.4.2.1 Multiple replacements within a line.

The intent of the capacity to execute multiple replacements within a line is to minimize typing and buffer switching (the F command). If there are several elements of a line which you wish to change, you may change them one at a time, using the R command for each, without using the F command in between. On each use of the R command in this case, the command operates on the result of the last command. Only the first use of the R command operates on the original line. This means that if the X command is used, you move back to the original line, rather than the line as it was before the last use of the R command, because the last copy is not saved. This means that it may be more efficient to back out of an errant R command which is the last of several modifications to a particular line by executing an R command which reverses the last R command. In general, you can modify a line indefinitely.

The column limit parameters and the replacement of all cases of a string option may be used with multiple R commands.

If the replacement was a full-line replacement of the form R, carriage-return, followed by the prompt, followed by the text and a terminal carriage-return, the line may not be modified by a string replace until the buffers have been exchanged using the F command. The premise is that the X command can be used, followed by another replace. If this is not satisfactory, then the sequence, 'DE<I text<' will have the same result, and will allow replacements within the inserted line.

1.4.2.2 Replacement after multiple-line replacement.

It has been possible to replace a given string in several lines by using the form Rn for a long time. It is now possible to replace text in the last line in the Rn group using another R command without first flipping the buffers (the F command) in the same way it can be modified after a single line replacement command. It is not possible to access lines prior to the last without using either the F command, which exchanges the buffers, or the X command, which cancels the Rn replace command.

1.4.2.3 Replacement after input.

When inputing a body of text, a typographical error might occur or one might change one's mind as to the text being entered. One can either use the X command to cancel all of the input string, or the F command to flip the buffers to make the text available for correction. Either action is time-consuming and destructive of the natural flow of the text. It is now possible to execute two carriage-returns, one to terminate the current line, and one to return to command level, and then commence executing R commands to modify the last line of text inserted. Lines preceding the last line inserted can not be modified in this manner for the same reasons that the lines before the last after a multiple-line replacement can not be modified.

After the line is modified to the point that it is satisfactory, one may reenter the input process by executing the usual I, carriage-return.

This may be done after a single-line insertion generated by the form

'I text<'.

This may not be done at the end of the first body of text in an item being defined by this execution of the EDIT verb, because when the item is new, the EDITOR automatically exchanges the buffers on return to control level after the execution of an input command. It does not automatically exchange the buffers after text has been merged in to the new item from another item, allowing modification of the last line of the text which has been merged in from another item. The automatic buffer exchange can be subverted by executing a F command as soon as the new item is entered. Note that you can file the item and the file will contain an item whose only data is its item-id. There are uses for key-only items. Once the buffers have been exchanged, the item will behave in the usual manner.

1.4.2.4 Multiple replacements and the Merge command.

It is now possible to merge one or more lines of text into the current location in the text, and then modify the only or last line merged in using the multiple replace facility. Lines prior to the last can not be so modified for the reasons noted above. It is possible to do a lot of

text manipulation very quickly using the merge, delete and replace commands.

1.4.3 A note on column specification.

Under R80, the first column is referenced by 1, and as usual by using a colon, ':', as the delimiter between fields. Columns are referenced according to the numbers returned by the C command. If a single-column reference is executed using the form

R/X/Y/20

Then the first 'X' in line after column 19 will be replaced with a 'Y'. The search will continue to the end of the line.

If a two-column reference is executed using the form

R/X/Y/20-22

Then an 'X' in column 20, 21 or 22 will be replaced with a 'Y'. If the form

R/CAT/DOG/20-22

is used, then the whole source string 'CAT' must be in the field starting at column 20 and ending at column 22. This is equivalent to looking for the string 'CAT' which starts in column 20. In other words, if the field being located by the locate, replace or delete commands is longer than one character and if it is being searched for in more than one column, then the ending column value must be the beginning column plus the length of the string plus the number of columns in which the string may start minus 1. The example which follows will hopefully clarify this proceedure.

If the line being considered is the following:

```
084 the difference between the beginning and ending
```

and we wish to replace the second 'the' with 'any', then

```
C<
          1         2         3         4         5
12345678901234567890123456789012345678901234567890123456789012345
```

allows us to observe that the second 'the' commences at column 24. We may then use the form

```
R/the/any/24<
```

which will yield

```
084 the difference between any beginning and ending
```

In order to use the column range specification, then we must use the form

```
R/the/any/24-26<
```

```
084 the difference between any beginning and ending
```

If the form

```
R/the/any/24-25<
```

is used, then the process will return to the prompt character without doing anything. The same will occur with

```
R/the/any/22-25<
```

or

```
R/the/any/25-27<
```

In general, success will occur if the beginning column is prior to or on the first character to be identified in the line, and the ending column is greater than or on the address of the last character to be identified in the line.

Use of column specification with the replace.

The protocols above are identical for the string locate and the form of the delete which deletes lines which contain a given string.

1.4.4 The Replace-Universal command.

This is the form of the replace command, referenced above, which allows the replacement of all cases of the first string in the replace command with the second string in the line or lines specified. This option is indicated by simply using the form RU for the command where R would be used in the normal case. The option allows multiple-line replacements using the form RUn for the form Rn, and it allows column specification, as above. An example of the replace-universal command demonstrating an alternative approach to the example above is below.

```
084 The difference between the beginning and ending

RU/the/any

084 any difference between any beginning and ending

r/any/the

084 the difference between any beginning and ending
```

Using the Replace-Universal command.

This approach may be faster in real time, since it requires less thought, and avoids typing the trailing delimiter in the replace statement and calculating or estimating the location of the target string in the line.

1.4.5 EXTENSIONS TO THE MERGE COMMANDS.

4.5.1 The historical MErge command.

The EDITor now allows items to be merged in from other files.

Historically, the merge command, ME, allows text to be merged from anywhere in the current item if the item-id field is null, or from any item in the file being edited, including the file copy of the current item, by the inclusion of the desired item-id in the merge command. The Editor has been extended so that text may be merged in from other files. Note that the syntax of the merge command has been as follows:

MEnumber-of-lines/ITEM-ID/starting-line-number

Thus:

ME10/DOG/5

will merge in ten lines of the item DOG in the file which contains the item currently being edited (the FILENAME referenced in the EDIT command), starting at line 5. In other words, line 5 through line 14 will be inserted after the line currently displayed on the screen.

1.4.5.2 MErging in items from other files.

The extended syntax requires the use of the delimiters (and) in place of the / delimiter used above. They thus become reserved when using the merge command in the sense that the colon, :, is reserved when using the locate, replace, and delete commands. In this case there is the further peculiarity that (and) are not the same character, whereas any character may normally be used as a delimiter, so long as all the delimiters in a particular string are identical.

The new feature allows the following forms:

MEnumber-of-lines(DICT FILENAME ~~NEW~~ MERGE FROM ITEMNAME)starting-line-number

MEnumber-of-lines(FILENAME NEWITEMNAME)starting-line-number

MEnumber-of-lines(DICT FILENAME)starting-line-number

MEnumber-of-lines(FILENAME)starting-line-number

The use of DICT is conventional. It means the same thing here as is does at TCL in the reference to files, and in the COPY processor. If there is no item-id specified, then the processor defaults TO the item-id of the item being edited at the moment. This is useful if one wishes to get a copy of an item into a test file and edit it quickly, or if one wishes to assure that the item will not be filed inadvertently over the old copy. Combined with the prestore command structure and the increase in speed of the replace command, some very powerful things can be done very quickly and easily.

There are certain other defaults which apply to the merge command, and which are carried over into this extended form which will be noted below

as a reminder.

MEnumber-of-lines(DICT FILENAME NEWITEMNAME

MEnumber-of-lines(FILENAME NEWITEMNAME

MEnumber-of-lines(DICT FILENAME

MEnumber-of-lines(FILENAME

These forms do the same thing as above, except that the starting line number defaults to line 1 in the merge source item.

ME(DICT FILENAME NEWITEMNAME)starting-line-number

ME(FILENAME NEWITEMNAME)starting-line-number

ME(DICT FILENAME)starting-line-number

ME(FILENAME)starting-line-number

This does the same thing as above, except that starting-line-number is the only line which is mergeu into the destination item. As such, the line may then be mouified using the replace command, as noted above.

ME(DICT FILENAME NEWITEMNAME

ME(FILENAME NEWITEMNAME

ME(DICT FILENAME

ME(FILENAME

This simply returns the first line of the merge source item. Note that the trailing right parenthesis is optional if the starting line number defaults to the first line of the source.

Obviously, these defaults all apply to the normal merge statement, leading to the minimal form 'ME/', which simply inserts the first line of the item currently being edited into the current location in the item, which is useful if you wish to put a given line in several different places in an item.

1.4.6 EXTENSIONS TO THE FILE COMMANDS.

Along with the extension to the merge command come a similar extension to the FI and FS commands. You may now file the item currently being edited to either a different item in the current file, or to the same item name or to a different item name in a different file. The syntax for the FI and FS commands are identical. The difference is that the FS returns you to the top of the item currently being edited. You can not FD any item other than the one which you are currently editing, by the way. Massive use of the FD can be accomplished with the DELETE verb (PROC) or by the use of a prestore command. The delete verb will be faster.

1.4.6.1 Filing items in other files.

The R77 and earlier releases allowed the forms FI< and FS<. The extension

has the folowing form:

FI(DICT NEWFILENAME NEWITEMNAME

FI(DICT NEWFILENAME

FI(NEWFILENAME NEWITEMNAME

FI(NEWFILENAME

FI NEWITEMNAME

In the list of FI commands above, the first will file the item which you are editing into item NEWITEMNAME in the dictionary of NEWFILENAME; the second, into the same item name in the dictionary of NEWFILENAME; the third, into item NEWITEMNAME in the data portion of NEWFILENAME; the forth, into the same item name in the data portion of NEWFILENMAME; and the fifth, into item NEWITEMNAME in the file referenced in the EDIT command. In each case, an existing item will not be overwritten, and the EDITor will proceed to the next item in the list, if any, to the next line in the initiating PROC, if any, or to TCL.

Forms symmetrical with those noted above include

FS(NEWFILENAME NEWITEMNAME

FIO(NEWFILENAME NEWITEMNAME

FSO(NEWFILENAME NEWITEMNAME

The left parenthesis must immediately follow the I or S, because the delimiter which specifies that this is not a file reference is a blank. It is therefore possible to generate items commencing with a left parenthesis in the file which you are currently referencing. If something gets lost, that is where to look. Note that item-ids may have embedded blanks; in order to retrieve them in the EDITor, surround the item-id including the embedded blanks with quotes, and the item will appear.

The FI and FS commands with a new item or file reference will not overwrite items with the specified name which are already stored in the reference file. Instead the EDITor will respond with a rather cryptic

CMND?

and return to the prompt character. In general, anything which the EDITor either does not understand or of which the EDITor disapproves will result in CMND? error message when the FI, FS, or EX processes are involved.

Should you wish to overwrite an existing item, then the forms FIO and FSO are available, in the tradition of the O option used by the copy processor. Again, the left parenthesis must immediately follow the O for the reasons noted above. The processor will not recognize a lower-case 'o'.

1.4.7 EXTENSION TO THE EX COMMAND.

The purpose of the EX command is to exit from an item. A problem occurred if one was editing a long list or the whole file and wished to exit from the EDITor entirely. The only facility available was to BREAK and END.

R80 uses the extended form EXK (K for kill), which will cause the editing process to proceed to TCL or the PROC which called the EDITor. The exit process will not recognize a lowercase 'k'.

Upon exit from the EDITor, the processor now specifies the name of the item exited, rather than simply printing 'EXIT'.

1.4.8 An extension to the X command.

The X command is used to reverse the effect of the last update instruction. The XF command is now available to reverse the effect of all updates executed since the last buffer exchange (F command).

The ? command now displays the item-id of the item currently being edited, as well as the current line number.

1.4.9 Assorted EDITor supervision facilities.

There is now a check which does not allow items which exceed the maximum allowable item size to be filed. A message is printed which notes the condition, and the number of bytes by which the item in the EDITor buffer exceeds the maximum item size. Note that the maximum item size is less than 32767 bytes, and that the buffers available to the EDITor are about 53000 bytes, subject to a COLD-START option.

Associated with this feature is a new command, S?, which causes the the size of the item which you are currently editing to be displayed.

1.4.10 An aside on the construction of null attributes.

There are three ways to create a null attribute or line. The first is to generate a line with some text, and then to replace that text with a null. The second is to use the Insert command. This is done by typing an 'I' at the prompt, followed by a blank, followed by a carriage-return. This will insert a null attribute at the current location and return to the prompt. The third allows the creation of several null attributes with a single input line. For each null attribute desired, simply include a CONTROL-up-arrow in the line. This has the effect of including an attribute mark in the text, and is useful for the creation of data definition items.

1.4.11 An aside on clearing and trimming the right end of the line.

If a replace statement specifies an attribute mark as the second string, then all of the line to the right of the last character before the first character in the string in the line which matches the first string specified by the replace statement will disappear.

A field of blanks has a special place as the first string in a replace statement. The processor will search for a string of blanks of the same or greater length in the line. The peculiarity of blanks is that the processor takes the line to be implicitly padded to the right with an indefinite number of blanks. It will therefore take any string of blanks longer than the longest string of blanks within the line to mean the end of the line. It is therefore possible to trim all trailing blanks off an item using the form

R1000/ //

or

R1000/ /^

presuming that the blank string is suffitiently long.

1.4.12 Clearing the left end of a line.

It is possible to clear the left end of a line by replacing a string of ^'s of necessary length with a null, as in the example below.

	Given the line,
.032 This is a dog. This is a cat.	to remove the first sentence
.R/^^^^^^^^^^^^^^^^//	will yeild
.032 This is a cat.	to remove the first sentence

Clearing the left end of a line.

1.4.13 THE PRESTORE FACILITY.

The prestore facility allows the storage of strings of EDITor commands, and the execution of the string by using the name of the string as the command. The allowable string names are P0, P1, ... , P9. . There are therefore a maximum of ten prestored commands available at any one time. Further, each prestored command is allocated 100 bytes, so that, if one wishes to generate a prestored command which exceeds 100 bytes, simply do not initialize the command whose name is ordinally next. In other words, if P1 is 150 bytes long, do not use P2. This may be inconvenient on rare occasions, but it makes things run faster.

In order to create a prestored command, type in the name of the prestored command, P0, P1, ..., P9, followed by a space, followed by the first command to be executed, followed by the prestore command delimiter, which is a start buffer mark (X'FB'), and which may be input by typing CONTROL-[(control-left-square-bracket) or ESCAPE (esc), followed by the next command, and so on. Any valid command is usable, including prestore command names.

P0 L22	This is loaded when you enter the EDITor. It has the following synonym:
P< L22	This allows the traditional L22< to be done by P<, which is generally convenient, being next to the carriage-return key.
P1 R100/DOG/CAT[FIR100/dog/cat[FI	This has the effect of changing dogs to cats in the first hundred lines of text.

Creating simple prestores.

In the above examples, note first that 'P' is a synonym for 'P0'. It is automatically loaded with 'L22' at entry to the EDITor. P1 through P9 are null at entry. Executing them will cause a CMND? response. All prestores created since the entry to EDITor by use of the EDIT verb are retained until the EDIT verb is exited. Any of them may be changed by creating another prestore command string with the same name. The prestores persist from item to item, whether the EDITor is using an explicit item list, a selected list, or the whole file.

If one is going to use a prestore command for a repetitive task, it may either be activated each time it is to be used, or it may call itself, at which time its termination conditions must be considered. A prestore command which calls itself will terminate only when it runs out of items to process. This means that a prestore which calls itself must have an EX, FI, or FO in the command string. If it does not have such an item iteration command in the string, it will loop indefinitely on the current item. The only exit from this condition is a BREAK and END. The primary use of the prestore calling itself is to manipulate many items with a single instruction string initiated once. It is particularly useful for searching for specified strings in text files and replacing them as

necessary. The following example searches a BP (BASIC program) file for the name GENERAL.LEDGER.

```
ED BP *<                          Edit the file.
ITEMNAME                          The first item.
TOP                               Standard mark.
.P1 _500/GENERAL.LEDGER[EX[P1     Define the search.
.P1                               Initiate the run.

                                  At this point the EDITor will
                                  exhibit all lines in the current
                                  item with the desired string,
                                  then display
EOI nnn                           the number of lines in the item
'ITEMNAME' EXITED
NEWITEMNAME                       The name of the next item.
TOP                               The top mark.
                                  All the lines with the string, if
                                  any, and so on, until the list is
                                  exhausted, at which time the
                                  process will return to TCL.
```

A prestore command calling itself.

The same maneuver may be executed to the printer by appending the (P option to the EDIT verb. In this case, all information which would have been displayed on the terminal will be sent to the printer.

1.4.13.1 Displaying the current prestore commands.

It is possible to display all currently initialized prestore commands by using the PD (Prestore Display) command.

1.4.13.2 Using prestores in PROCS.

It is possible to create the desired prestored command strings in PROCS in the same manner that instructions are sent to the various processors from a PROC.

```
PQ                                The PROC definition.
MED BP *                          The verb.
STON                              Turn the stack on.
HP1 _500/GENERAL.LEDGER<          Specify the prestore.
HP1<                              Execute the prestore.
P                                 Execute the verb.
```

Defining a prestore in a PROC.

The example above assumes that a list is in existence. The verb activation may include an explicit item list or specify the whole file using the conventional asterisk. On entry to the first item from the EDIT verb, the prestore is automatically set up, and is available for use. All

ten prestores may be initialized this way, allowing the development of powerful customized EDITor commands. Further, it is implicit that the existence of this facility should de-emphasize the use of BATCH-strings inside PROCs for batch upuates, since the ability to prestore strings of commands, and to create multiple prestores provides the user with many of the facilities in bATCH.

1.5. ENGLISH

This section will discuss such changes to ENGLISH as may be of use to the user, and it will attempt to extend certain sections of the existing current ENGLISH documentation which are excessively opaque or thin.

The primary effort which has gone into ENGLISH has been the clarification of the code. This is in general transparent to the user, but results in speed improvements in some areas, predictability improvements in other areas, and symmetry improvements in still other areas.

1.5.1 EXTENSIONS TO ENGLISH.

The following syntactic extensions to ENGLISH have been made.

There is now a USING connective which allows the specification of an arbitrary file as the dictionary from which data definition attributes are to be retrieved.

In all cases the master dictionary is checked for for any data definition items not found in the specified dictionary. If found, they are then included in the ENGLISH compilation.

Selection items may now take NOT and EACH connectives for a single selection criterion. Further, a NOT connective allows a value string.

SSELECT will now consistently allow data arguments as output.

Multicolumnar labels will now consistently stay within their assigned columns.

Data elements with a length of zero may be totalled. Null multiple sub-values may be displayed by the non-columnar processor.

Detail suppression and control breaks are now available for MODEL3 processors.

An LPV, Load Previous Value, command is available with F- and A-correlatives.

F-correlatives may now return strings up to 500 bytes in length.

F-correlatives may contain conversions which contain embedded parentheses.

The compiler now scans I-, F-, and A- correlatives more intensely, allowing some optimization of the runtime code at compile time, and allowing an error trap for some erroneous cases.

The A-correlative compiler now allows calls to A- and F-correlatives.

The LPV command is usable in A-correlatives.

The L conversion has been modified so that there is an option which returns the actual length of each data element.

The Translate conversion has been modified so that you may specify the particular multi-value in the target attribute to be returned, if desired.

The translate processor has been modified so that a broader range of file names will be accepted.

If the combination of 'ONLY', ID-SUPP, and DET-SUPP is used, then the page header and column header will be printed, followed by any error messages generated by the process. The process will not print a blank line for each item processed.

The headings and footings now know whether they are going to the printer or the terminal for centering purposes.

Page numbers may be left adjusted in the page number field if desired.

1.5.2 DATA DEFINITION ITEM RETRIEVAL.

Historically, ENGLISH has allowed data definition items to be taken from the file level immediately above the file specified in the ENGLISH sentence. This meant that a data file could only be LISTed using data definition items in the dictionary of that file, that file dictionaries could only be listed using data definition items in the account master dictionary, and that the master dictionary could only be listed using data definition items which it found in itself. Dictionaries of dictionary-only files could be listed using internal data definition items if they had a data level D-pointer which pointed back into the dictionary itself, and if the key word DICT was not used in the ENGLISH sentence, or using the master dictionary data definition items if the key word DICT was included in the sentence.

There are two modifications to this protocol in R80. If explicitly specified data definition items are not found in the referenced dictionary, then the master dictionary is searched for them. Second, a new connective has been added which allows the specification of a file other than the normal dictionary level as the source of the data definition items.

1.5.2.1 Master dictionary data definition item default

If a data definition is explicitly specified in the ENGLISH sentence, and the key word DICT is not in the sentence, and the data definition item is not in the dictionary of the file referenced by the sentence, and the data definition item is in the master dictionary of the account, then the data definition item found in the master dictionary will be included in the ENGLISH compilation. This is useful for data definition items which are of global use, for example, SIZE, which is defined as

A^9999^ITEM LENGTH^^^^^^^^^^.

The substitution of master dictionary data definitions does not occur if the default data definition items are requested (1, 2, and so on).

1.5.2.2 The USING connective.

If more flexibility is desired, there is a USING connective which has been added to the set of connectives available to ENGLISH. The legal forms are:

USING DICT FILENAME

USING FILENAME.

The first references the dictionary of the referenced file; the second references the data-level file FILENAME. Note that the data-level file may be of the form FILENAME,SUBFILENAME. In these cases all data definition items will be taken from the file and level referenced by the USING connective, except those data definition items which default to the master dictionary, as above.

Only one USING connective is allowed in an ENGLISH sentence. If the USING connective is used, it must be immediately followed by either DICT FILENAME or FILENAME. The source of the data processed remains specified by the conventional file name element in the sentence. Q-pointers are usable for either or both references, of course.

Default data definition items will come from the file referenced by the USING connective.

1.5.2.3 Use of the USING connective.

The intent of the USING connective is, first, to allow the use of the contents of a dictionary on a structurally unrelated file. It has the effect of giving different users access to different pieces of data within a common file by means of different retrieval locks on different dictionaries, or, more accurately, dictionary equivalent files. It allows separation of data definition items by function and meaning. It allows the use of a test dictionary during development in order to avoid the collection of incorrect, obsolete, and intermediate data definition items in the permanent file dictionary. It allows the use of existing data definition items on new, temporary or test files without going through the COPY routine. A few syntactic examples follow. -

LIST WORKFILE USING DICT TESTDICT	The data source will be WORKFILE. The data definition items will be retrieved from DICT TESTDICT. The default data definition items will be used.
LIST DICT WORKFILE USING DICT WORKFILE	The data and data definition items have the same source.
LIST WORKFILE USING WORKFILE,WFDICT1	The data comes from WORKFILE. The data definition items come from a data-level subfile of WORKFILE named WFDICT1. There may be an indefinite number of these.

Examples of the USING connective.

This facility will probably be of more use during system development or modification than in a stable application. It could be used in a stable

application in order to control who has access to what data within each record within a file. The spreading of data definition items could be equally useful for the update case if a dictionary-driven update program is used. This is probably best accomplished if each class of user has an actual account rather than separate Q-pointer synonyms which log onto the actual account. In this case each account has Q-pointers to the relevant data files, and associated D-pointer dictionaries which contain the data definitions items to which that class of user is privy. This necessarily leads to a heavily PROC-driven application, which is also useful for data security.

1.5.2.4 The ITEM-ID definition with Q-pointers to the data file.

The file definition item in an ENGLISH sentence allows the use of attributes 7, 9, and 10 as meaningful elements in the data definition. The label comes from the D- or Q-pointer name because attribute 3 of the D- or Q- pointer is in general otherwise occupied. Attribute 2 is an obvious force to 0. Note that the selection, sort and output processors all ignore attribute 8. The selection and sort processors take the item-id as it is; the output processor allows the use of an attribute 7 conversion.

The justification is of importance to both the sort and output processors; the field length is of importance to the output processor, especially if the item-ids are significantly longer than the file name or its nominal field length, and especially in the columnar processor. Note that item-ids do not fold, unlike other data processed by the columnar processor. All of the characteristics of item-id handling may be gotten around by using a data definition item which references data attribute 0 with ID-SUPP.

In the case that a Q-pointer is used to reference a file, the definitions in attributes 7, 9, and 10 in the Q-pointer definition take precedence over the same attributes in the D-pointer if they exist in the Q-pointer. Those that do not exist in the Q-pointer will be retrieved from the D-pointer. In the case that they do not exist in either, attribute 7 will be defined as null, attribute 9 will become L and attribute 10 will become 9. These defaults will be taken for all data definition items processed by the ENGLISH compiler.

This allows the creation of multiple Q-pointers which treat the item-id field in different ways, as may be convenient. It remains that Q-pointers require at most that the first three attributes be defined.

1.5.3 THE SELECTION PROCESSOR AND ITS EXTENSIONS.

This section is intended to clarify the action of the selection processor as well as note the extensions to it.

1.5.3.1 Item-id selection.

ENGLISH allows the following item-id selection protocols. The default is the whole file, or the item-ids specified in a list if a list is active. Restrictions may be put on the item-ids returned in two ways which are syntactically similar, but which operate in different ways.

1.5.3.2 Explicit item-ids.

If explicit item-ids are specified, then only those item-ids will be returned. If there is a list in effect, it will be ignored.

Explicit item-ids are specified as follows:

```
LIST FILENAME 'ITEM1''ITEM2''ITEM3'
```

or

```
LIST FILENAME "ITEM1""ITEM2""ITEM3"
```

or even

```
LIST FILENAME \ITEM1\\ITEM2\\ITEM3\
```

Each of these will yield a listing of the three items, ITEM1, ITEM2, and ITEM3. The processor does this by retrieving each of these items directly from the file referenced. The collection of explicit item-ids becomes a list, which the processor uses to obtain the next item-id until it is exhausted, at which time the process terminates.

The delimiter ' is reserved for item-id specification. The delimiters " and \ are normally used for value specification, but will be taken as item-ids under certain circumstances.

1.5.3.3 Item-id tests.

ENGLISH also allows tests on item-ids. Be aware that all tests are on the item-id as it stands in the file. No conversions or correlatives will be applied to the item-id before the test is made.

The specification of an item-id test rather than the retrieval of a specified set of items is done by including a relational operator, hereinafter referred to as a relational connective, in the item-id specification string. For example:

```
LIST FILENAME 'ITEM1' = 'ITEM2''ITEM3'
```

or

```
LIST FILENAME "ITEM1""ITEM2" = "ITEM3"
```

or even

```
LIST FILENAME = \ITEM1\\ITEM2\\ITEM3\
```

will all have the same effect. ENGLISH will search the whole file, or all the items in a list if there is one in effect, looking for items which have the item-ids ITEM1, ITEM2, and ITEM3. This will take longer than using the explicit item-id reference given above, and is not recommended when you know which item-ids you want.

The intent of relational connectives is to allow specifications of the form

```
LIST FILENAME < 'CAT'
```

which will have the effect of selecting all items which are alphanumerically less than CAT, presuming that the file is left-justified. The full effects of justification will be considered below.

Note that in the examples above only one relational connective was included. In that case, all the elements not preceded by a connective are automatically assigned the connective '='. This is true throughout ENGLISH in those cases when values are usable.

There need not be spaces between the item or value strings, and the file name, data definition items or connectives may be concatenated to a value in the input sentence, as in the form

```
="ITEM1"
```

In all other cases all elements in the sentence which are to be retrieved from a dictionary or dictionary-equivalent file must be surrounded by blanks.

It is possible to specify either a list of item-ids for retrieval or to specify a test on item-ids using this mechanism. It is not possible to retrieve certain items directly and to test all the other items for admissability using only item-id tests. In other words, the item-id list is either a list of explicit item-ids or it is a sequence of values against which to test each item-id in the file.

1.5.3.4 The use of delimiters in the ENGLISH sentence.

In the above examples the delimiters ', ", and \ are all used equivalently to delimit item-ids or values, as the case may be. In general, only the delimiter ' is reserved for item-ids or item-id-related values. The " and \ may be used for values related to data definition item selection criteria and print-limiters as well, although they prefer to be associated with the data selection criteria rather than with item-id selection criteria. In general, values delimited by either " or \ will be treated as item-id selection criteria only if they follow the file reference and precede data definition items.

The delimiter ' will cause the value which it surrounds to be treated as an item-id selection criterion wherever it may be in the sentence.

1.5.3.5 Item-id selection criteria.

Presume that we have a set of values with an associated relational connective, so that ENGLISH is scanning the whole file in order to test the item-ids for acceptability. Be aware that the item-id test is logically ANDed with all other selection criteria. If the item-id fails the item-id test, the item will be discarded. If one wishes to 'or' an item-id test with other selection criteria, then a data definition item must be included which references the item-id. For instance:

Consider the following data definition items.

```
    0          1                          DDI item-id
001 A      001 A                          DDI typifier.
002 J      002 1                          AMC specifier.
003        003                            null label
004        004
005        005
006        006
007        007
008        008
009 _      009 L                          Justification.
010 10     010 10                         Length
```

and the following ENGLISH sentences:

`LIST MD < "CAT" WITH 1 = "D"`

This will select all items whose item-ids are alphanumerically less than 'CAT' AND which have a 'D' in attribute 1.

`LIST MD WITH 0 < "CAT" WITH 1 = "D"`

This will select all items whose item-ids are alphanumerically less than 'CAT' OR which have a 'D' in attribute 1.

`LIST MD WITH 0 < "CAT" AND WITH 1 = "D"`

This will select all items whose item-ics are alphanumerically less than 'CAT' AND which have a 'D' in attribute 1, as in the first case.

`LIST MD WITH 0 < "CAT" AND 1 = "D"`

This is erroneous. It will have the effect of selecting all items whose item-ids satisfy the CAT criterion, as above. The rest of the sentence has to do with print-limiters, which are taken up below.

`LIST MD < "CAT" AND WITH 1 = "D"`

This is erroneous. It will terminate in the ENGLISH compiler with an error, because the AND connective must be followed by another value which may be ANDed with CAT, and because the AND connective may not immediately precede the first WITH connective in the ENGLISH sentence.

The relationship of Item-id selection criteria to data selection criteria.

1.5.3.6 An aside on item-id structure.

In TCL the universal delimiter is a blank. Verbs, file names, connectives and data definition names are normally delimited by blanks. Non-ENGLISH verbs which reference files and items will take a string of characters which is delimited by blanks to be the file name or one of the item names, depending on its location in the command sequence.

If one constructs an ENGLISH sentence which references a data definition item which it cannot find in either the specified file dictionary or the master dictionary, it will then generate another item-id by taking the item-id for which a record did not exist and concatenate the next string delimited by blanks in the sentence to it, with a blank between the two character strings. This will now be used as an item-id. This is why the error message which is trying to tell you that the data definition item is not on file may includes more elements of the ENGLISH sentence. For example, if in the example below, the data definition item DOG is not on file,

```
LIST MD CAT DOG RAT
```

the error message

```
[24]  THE WORD "DOG RAT" CANNOT BE IDENTIFIED
```

will be returned.

The sequence of concatenated strings will terminate at the end of the sentence, or at the first connective which succeeds the unidentified word, or at the first value or item-id which succeeds the unidentified word.

What this means is that in general a blank is a character which is allowable for item-ids in the system. If you wish to EDIT a item-id which includes one or more blanks, enclose the string in one of the value delimiters above.

You may also use the delimiters within a string enclosed in delimiters. Simply use a delimiter which is not part of the item-id as the value surrounding delimiter. For example:

If DOG RAT is an attribute definition item in MD, then

LIST MD DOG RAT	Will return the one attribute definition item whose name is DOG RAT.

In order to modify the item DOG RAT, use the form

EDIT MD "DOG RAT"	Which will obtain the item.

If you have an item named O'HARA, use the form

LIST MD "O'HARA"	This will return the item O'HARA.

Similarly, the form

SELECT CUSTOMERFILE WITH LASTNAME "O'HARA"	Will find all the O'HARAs in the file CUSTOMERFILE.

Examples of infrequent but legal item-ids.

There are characters which are not acceptable for inclusion in item-ids. None of the system delimiters is allowable. An attempt to retrieve an item-id which contains a system delimiter will have the effect of terminating the key at the delimiter. In other words, if you have an item-id with a system delimiter in it, it can be accessed only by processors which use a sequential file search rather than an item name retrieval. This means that you can list it or select it but you cannot EDIT it, COPY it, DELETE it, or otherwise update it, or retrieve it using a list. It can be removed by using a file-to-file copy, since it will not copy to the new file. The fragment of the errant item-id up to the delimiter will be reported as not being on file at the end of the copy process if the copy is driven from a selected list.

The control characters (X'00' through X'1A') are generally inadvisable, although they may work occasionally.

1.5.3.7 Selection by data values.

ENGLISH has always supported the capability of selecting items based on the values contained in data fields. The discussion below includes a recapitulation of the historical characteristics of the selection processor with a focus on subtities, and a consideration of extensions which appear in R80.

Data definition items are marked as selection criteria in the ENGLISH sentence by preceding the data definition item name with the connective WITH. The standard form of a WITH phrase is

WITH {NOT} {EACH} DDINAME {value string},

which is called a selection criterion. For convenience, the connectives NOT and EACH following the WITH shall be referred to as selection modifiers.

The basic form of selection,

WITH DDINAME

will check for the existence of at least one non-null value returned from the item as a result of the directives contained in DDINAME. The item will succeed if there is at least one non-null value. The modified form

WITH NOT DDINAME

will succeed if there all values returned are null. The addition of the connective WITHOUT in R77 allows the more natural form of the above:

WITHOUT DDINAME

which has the same effect. The modified form

WITH EACH DDINAME

will succeed if each value returned is non-null. The form

WITH NOT EACH DDINAME

or

WITHOUT EACH DDINAME

is an extension for R80. It has the effect of accepting all items which have at least one null value returned.

The meaning of the term 'value' in this context is considered below.

1.5.3.7.1 Evaluation of data by the selection processor.

The selection processor processes the data according to attribute 6 of the data definition item. That is, it executes any conversions or correlatives which are in attribute 6. The result of this calculation is returned to the selection processor. The conversions or correlatives which may be in attribute 7 of the data definition are not applied to data

within an item. The contents of attribute 7, however, may have a significant effect on the success of the selection process as we shall see below.

.5.3.7.2 Obtaining a value to test.

The selection processor will ignore leading null sub-values within each value. That is, if an attribute of a data item contains multiple values, which themselves contain sub-values, as below,

^\\3\4]6\\7]1\2\3]]\\^

then the processor will retrieve one data value from each value. In this case the values returned will be:

3; 6; 1; null; null,

whether the search is based on a direct data call from the AMC in attribute 2 of the data definition item or on a virtual data call by an F- or A-correlative. In either case, if a request for the next value results in a null followed by a sub-value mark, the processor will proceed to the next value in the string. It will return from the data search, if a non-null data value has been retrieved, if it encounters a null value followed by a value mark, or if it encounters a null value followed by an attribute mark.

If the search results in a null data value, the process returns to the value test routine. If a non-null data value is returned, the process will then execute any conversions remaining in attribute 8 of the data definition item. The data resulting from the conversion, if any, will then be returned to the value test.

This is the 'value' referred to above. Note particularly that only the first non-null sub-value is returned. If another value is requested, then the next value is taken from the next value, that is, from the right side of the next value mark if there is one. All sub-values which follow the first non-null sub-value are never inspected by the selection processor.

1.5.3.7.3 The test for existence.

What occurs at the value test level when testing for existence depends upon the selection modifiers. If there are no modifiers, then if any non-null sub-value is returned from the item, the selection phrase will succeed. If a null value is returned, then the tester will request the next value in the item, unless the last null value was terminated by an attribute mark. In this case, the values within this attribute of this item have been exhausted, and the item does not have the required value. Therefore, this selection phrase fails.

If the selection modifier is NOT, then all values defined by the data definition must be inspected in order to succeed. If any value is returned which is non-null, then the clause will fail.

If the selection modifier is EACH, then all values defined by the data definition must be inspected in order to succeed. If any value is returned which is null, then the clause will fail.

If both modifiers are used, WITH NOT EACH or WITHOUT EACH, then the clause

will succeed if any value is null. It will fail only if all values are non-null. Note that this is an R80 extension in the interest of symmetry.

WITH DDINAME	succeeds if
(VALUE1 # NULL) OR (VALUE2 # NULL) OR (VALUE3 # NULL) OR ...	
WITHOUT DDINAME	succeeds if
(VALUE1 = NULL) AND (VALUE2 = NULL) AND (VALUE3 = NULL) AND ...	
WITH EACH DDINAME	succeeds if
(VALUE1 # NULL) AND (VALUE2 # NULL) AND (VALUE3 # NULL) AND ...	
WITHOUT EACH DDINAME	succeeds if
(VALUE1 = NULL) OR (VALUE2 = NULL) OR (VALUE3 = NULL) OR ...	

Success conditions for with and its modifiers
under the test for existence.

WITH DDINAME	fails if
(VALUE1 = NULL) AND (VALUE2 = NULL) AND (VALUE3 = NULL) AND ...	
WITHOUT DDINAME	fails if
(VALUE1 # NULL) OR (VALUE2 # NULL) OR (VALUE3 # NULL) OR ...	
WITH EACH DDINAME	fails if
(VALUE1 = NULL) OR (VALUE2 = NULL) OR (VALUE3 = NULL) OR ...	
WITHOUT EACH DDINAME	fails if
(VALUE1 # NULL) AND (VALUE2 # NULL) AND (VALUE3 # NULL) AND ...	

Failure conditions for with and its modifiers
under the test for existence.

1.5.3.8 The value string.

In the syntax of the WITH phrase above, there is an optional value string which has not been mentioned since, although all of the tests for existence assume a null string as the value. A value string is made up of value phrases of the following form

{relational connectives} VALUE.

The relational connectives are optional in the sense that the relation will default to '=' if there is no relational connective preceding the value.

A value is of the form

"text string"

or

\text string\

Remember that the delimiter ' will always specify an item-id reference.

The contents of the text string may be any characters with the exception of the system delimiters. Avoid the control characters if possible. There are three special symbols, ^, [, and] which have a special meaning to the selection processor, and will be considered below.

5.3.8.1 The relational connectives.

The master dictionary contains definitions of the usual relational connectives: =, #, <, >, =>, >=, <=, =<, EQ, NE, GT, GE, LT, and LE. These may be used in any combination except with the condition #, which must be used by itself. Note that all normal combinations are already defined. The form = < may be used as well as =<, for example. Note that the space between the connectives requires that two look-ups must be done, while the =< form is retrieved in a single master dictionary reference. If you have a syntactic preference for the form <>, you may copy the item '#' in the master dictionary to the item '<>'. The operators are logically equivalent.

1.5.3.9 The specified value and attribute 7.

It was noted above that the contents of attribute 7 might have an unexpected effect on the results of the selection processor. This is because attribute 7 is generally thought of as an output conversion, because that is what it is designed to effect. For this reason, the ENGLISH compiler will execute an inverse conversion on the data values defined in the value string, so that the output conversion does not need to be done for each value in the file referenced by the data definition item. The compiler then throws away the contents of attribute 7.

The ENGLISH compiler will not attempt to execute an F- or an A-correlative in attribute 7. These will be ignored.

There are conversions which will yield unexpected results, however, and

should be avoided.

1.5.3.9.1 Allowable conversions -- the date.

There are conversions which are of use, primarily the date and time conversions. The date conversion will take a date in display format and return the internal form, which is a decimal number representing the number of days since December 31, 1967. In this case you would not wish to execute a date output conversion in attribute 6, since it is unlikely that you would ever get a match. Note that an input date conversion will only transform the external form of a date into the internal form, and that an output conversion will only transform an internal date into the external form of the date. The only time that an input conversion is done in ENGLISH is for the evaluation of values associated with selection phrases according to attribute 7.

It is preferable to do the data conversion associated with selection in attribute 7 because it only needs to be done once, at compile time, and because, if it is done in attribute 8 on each value, it will be necessary to remember the precise form which will result, and because the form which derives from attibute 8 will be evaluated according to the alphanumeric form of an external date, rather than in the normally-desired numeric form of the internal date. The internal date is represented as an increasing integer, so that less than and greater than relational connectives have the expected meaning of before and after. The external date does not have this characteristic.

1.5.3.9.2 The time.

Time conversions are allowable. Again, it is preferable to use the attribute 7 form. Time is represented in the machine as an integer which is the time since midnight in seconds.

1.5.3.9.3 The mask conversions.

In general the forms MK, ML, and MD will treat only the scaling and decimal location characteristics available with these masks. Nothing else will be touched. This means that they will have an effect only if they are immediately followed by one or two numeric digits. They will have an effect only on a value string which represents a number. If the value is a number, it is scaled and the decimal place is attended to. Remember that the internal form is an integer. That is, there is no decimal point in the number. If there are some numeric digits at the front of the value, then these will be taken as the number, and the rest of the value will be thrown away. If the first character of the specified value is not numeric, then the value string will be taken without modification.

Thus, if attribute 7 has a MK, ML, or MD conversion in it, numeric values are recommended.

1.5.3.9.4 Other masking functions.

Since the functions of the form MCA, MCN and so on have the effect of stripping non-admissable characters from the string which they process, there is no point in using them. Clearly the inverse function is meaningless.

The MCXD (convert from hex to decimal) and MCDX (convert from decimal to hex) conversions have inverse functions, so that they are useable in attribute 7 of a data definition item being used for selection. The MCXD will convert decimal to hex as an input conversion, and the MCXD will convert hex to decimal as an input conversion.

1.5.3.9.5 Translate conversions.

If there is a translate conversion in attribute 7, then it will be executed as an input conversion. This means that the first of the translate attribute mark count numbers in the translate syntax will be used. If the field is null, then the translate will return the item-id if it found an item-id.

If the value specified does not yield an item-id, and if the translate option byte is an 'X', then the value for which the processor will search will be a null. If the option byte is a 'C', then the value for which the processor will search will be the specified value.

What the an input translate will not do is search the file specified by the translate for an item which has the specified value in the correct attribute, and return the item-id as the value.

If there is a direct one-to-one correspondence between the source and destination items, then it is possible to have a set of translate elements within the file which are an inverse transformation. That is, if you supply the value generated by the output translation, and if that value is an item in the file which has as contents the item-id the value which translates to the value supplied, then a translation in attribute 7 is valid. For instance,

```
In a file we may call CUSTOMER,

    232                     Pacific Printing  The item names.
001 Pacific Printing   001 232                The translate references.

Then if attribute 7 of the data definition item CUSTTRANS is

TCUSTOMER;C;1;1

and the data file reference to Pacific Printing is '232', then

LIST FILENAME WITH CUSTTRANS = "Pacific Printing" ... CUSTRANS ...

will yield the desired result, because Pacific Printing will be
translated into 232 for the selection, and 232 will be translated
into Pacific printing for output.
```

A translate which will work in a selection.

If the output translate function which takes many different data strings and translates all of them into a single output result, requiring an inverse function which is multi-valued, then a translate in attribute 7 is inappropriate, because only the first value found by the attribute 7 manipulation will be included in the resultant value string. In this

case, the translate must be put in attribute 8, so that the processor is comparing the translated value to the value originally specified in the value string.

5.3.10 Summary of conversions for selection.

It is generally a good idea to use the date and time conversion in attribute 7. The MCXD and MCDX conversions will work. The MR, ML, and MD conversions will work sometimes, and will do strange things other times.

The translate may possibly work if the data structure is just right.

The various other masks and conversions which have no natural inverse functions will tend to fail in a data-sensitive way, and are not recommended.

The processor will not even try to deal with A- and F-correlatives. In all cases the contents of attribute 7 are discarded during the compilation process.

1.5.3.11 Forming conditional values.

As noted above, there are three special characters associated with value specification: '[', '^', and ']'. These are not optional and they are not modifiable. They have the following meanings:

[stipulates that any leading string is acceptable.

^ stipulates that any character is acceptable in this position.

] stipulates that any trailing character is acceptable.

The test will terminate at this point with success. For purposes of evaluation, the inclusion of a special character forces evaluation from left-to-right, on a character-by-character basis. For example,

= "[6"	will accept any data value which terminates in a '6', such as '6' or 'ABC6' or '123456'.
= "3^5"	will accept any three-character string which begins with a '3' and ends with a '5', such as '305' or '3A5' or '3*5'.
= "6]"	will accept any string which starts with a '6', such as '6' or '6ABC' or '654321'.
= "[6]"	will accept any string which contains a '6', such as '6' or 'ABC6' or '6ABC' or 'ABC6XYZ'
= "[3^5]"	will accept any string which contains any three-character string which starts with a '3' and ends with a '5', such as '335' or '305XYZ' or 'ABC3X5' or 'ABC3X5XYZ'

Use of special characters in selection values.

There are certain forms which will not work. If the '[' is used in the value specification, it must be the first character in the string, and it must be the only '[' in the string. If the ']' character is used in the string, it will terminate the specified string at that point. Any characters which may occur after a ']' will never be inspected. The ^ may be used anywhere, and any number of them may be included in the value specification. The form '^^^' may be used to retrieve all three-character strings, for instance, although there is a conversion, 'L', which performs this function.

1.5.3.11.1 The special characters and the relational connectives.

The following examples use the relational connectives < (less than), > (greater than) and =, since the other permutations can be derived from these.

The case of equality is shown above. If the form

WITH 2 < "5]"

is used, the test is on the first character, and is straightforward.

If the form

WITH 2 < "[5"

is used, the test is on the last character, and is straightforward.

If the form

WITH 2 < "[5]"

then, if there is a '5' anywhere in the string, equality will be true, and inequality will fail. If there no '5' in the data string, then the condition 'less than' will hold if the last character is less than the 5, and the condition 'greater than' will hold if the last character in the data string is greater than '5'.

Similarly, if the string of actual data specified is several digits long, the test which generates the type of equality will be as follows: if the process reaches the end of the data string when it is on the first real character of the test string, it will compare those two characters and yield a result as above. If the it is on a character other than the first real character in the specified string, it will generate the result expected if the compare were on the first K characters in the specified string against the last K characters in the data string.

"[567]" < "12486"	becuase 5 < 6.
"[567]" > "12484"	because 5 > 4.
"[567]" < "12457"	because 56 < 57.
"[567]" > "12455"	because 56 > 55.
"[567]" < "12568"	because 567 < 568.
"[567]" > "12566"	because 567 > 566.

Examples of results under inequality in the [] case.

Equality will not occur unless all of the real character string is found in the data string.

1.5.3.11.2 Justification and evaluation.

If the data definition item is left-justified or if there is a special character in the value specification string, then comparison will be alphanumeric. That is, it will proceed from left to right, and inequality will be declared as soon as characters in the same location in the two strings are different. The collating sequence is that of the ASCII character set, with the particular characteristic that numbers precede letters, and capital letters precede lower-case letters. An absolute null is less than any character, including an ASCII null. An absolute null occurs when the end of a string is reached, with the result that 'ABC' comes before 'ABC0'. Also note that blanks are the non-control character with the lowest value.

If the data definition item is right-justified, and there are no special characters in the string, and the data string and value string are numeric, then the test will be on the magnitude of the two numbers such that 12 > 2. If these were left-justified, 12 < 2 because 1 collates before 2. If the data are not numeric, then they will be compared in the usual left-to-right manner until either inequality is discovered, the strings terminate, or numeric fields are found. If both the data and the specified value are equal up to the start of a numeric field, then the numeric fields will be evaluated as binary integers and compared. If inequality is found, then the string with the smaller imbedded integer is accepted. If they are equal and both strings terminate at this point, then the strings are equal. If the strings continue with non-numeric data, the left-to-right process continues until inequality occurs or the strings terminate.

1.5.3.12 Value strings containing many value phrases.

It is possible to select based on more than one value. The relation associated with each value is the relational connective which immediately precedes the value. If there is no relational connective which precedes the value, then a default '=' will be inserted into the value string. The implicit relation between the value phrases is 'OR'. If the data value must pass both of two criteria, then there must be an 'AND' between the two value phrases.

The relational connective default:

... WITH X "A""C""E""G" . is equivalent to

... WITH X = "A" = "C" = "E" = "G"

which will succeed if

(DATA = "A") OR (DATA = "C") OR (DATA = "E") ...

where DATA in each case represents only one value which may be returned to the value comparison processor. Therefore we may say,

IF ((DATA = "A") OR (DATA = "C") OR (DATA = "E") ...)

then DATA IS TRUE else DATA IS FALSE.

A data value is said to succeed if the test returns TRUE.

The cases of inequality are similar:

... WITH X < "A" > "C" # "E" <= "G"

. is equivalent to

IF ((DATA < "A") OR (DATA > "C") OR (DATA # "E")

OR (DATA <= "G") ...)

Then DATA IS TRUE else DATA IS FALSE.

(This particular case will succeed in all cases).

ORed values with relational connectives.

1.5.3.12.1 The AND connective within a value string.

If you desire to specify a range of values which will be acceptable, or a collection of conditions on a given data value such that they must all be true in order for the condition to succeed, then the specified values may be ANDed together. The required form is:

... value AND relational connective value.

For instance,

```
... WITH X <= "A]" AND >= "C]"      will accept all values which
                                    start with A, B, or C, as in

IF ((DATA <= "A]") AND (DATA >= "C]"))

then DATA IS TRUE else DATA IS FALSE.

... WITH X = "1]" AND < "[5"        will have the effect of
                                    accepting all values with start
                                    with 1 and end with a character
                                    less than 5, as in

IF ((DATA = "1]") AND (DATA < "[5"))

then DATA IS TRUE else DATA IS TRUE.
```

Examples of AND value specification phrases.

1.5.3.13 The evaluation of one selection item.

An indefinite collection of value phrases may be ANDed together into what we may call an AND value specification phrase. Further, several AND value specification phrases may be ORed togetner into what we have been calling a value string.

Essentially, and AND phrase, which may consist of sub-conditions, acts as a single entity which can either pass or not pass. For an AND value specification phrase to pass, all elements must pass. Within the string in general, that is, as amongst the ORed value specification phrases, if any element succeeds, then the selection criterion succeeds.

1.5.4 The relationship of selection criteria within a sentence.

It was just noted that if one ORed value specification allows a data element to pass, the selecion criterion will pass. It is possible to have several selection criteria within a single sentence. The default connective between the selection criteria will be an OR, so that if any of the criteria pass, tne item will pass. This is of course modified by the selection modifiers NOT and EACH. Prior to R80 the modifier NOT could not be used in the same criterion with a value string, on the grounds that the criteria in the value string could always be reversed to obtain the complement of the criteria. It should be clear that value strings can be constructed such that the construction of their complement is at least thought-provoking, if not time-consuming and unpleasant. Therefore, the NOT modifier has been enabled so that it will cause the criterion to fail in the case that the value string succeeds and vice versa. We may therefore replicate the table of success and failure under the conditions of WITH, WITHOUT, WITH EACH and WITHOUT EACH which was displayed for the case of existence only. Note that the case of existence is equivalent to the value string # "", although using the explicit string is inefficient. In the table below the form '# NULL' is replaced by the form 'IS TRUE', and the form '= NULL' is replaced by the form 'IS FALSE', as values returned from the value test processor.

WITH DDINAME <VALUE STRING>	succeeds if
(VALUE1 IS TRUE) OR (VALUE2 IS TRUE) OR (VALUE3 IS TRUE) OR ...	
WITHOUT DDINAME <VALUE STRING>	succeeds if
(VALUE1 IS FALSE) AND (VALUE2 IS FALSE) AND (VALUE3 IS FALSE) ...	
WITH EACH DDINAME <VALUE STRING>	succeeds if
(VALUE1 IS TRUE) AND (VALUE2 IS TRUE) AND (VALUE3 IS TRUE) ...	
WITHOUT EACH DDINAME <VALUE STRING>	succeeds if
(VALUE1 IS FALSE) OR (VALUE2 IS FALSE) OR (VALUE3 IS FALSE) ...	

Success conditions for WITH and its modifiers
under test against a value string.

WITH DDINAME <VALUE STRING>	fails if
(VALUE1 IS FALSE) AND (VALUE2 IS FALSE) AND (VALUE3 IS FALSE) AND ...	
WITHOUT DDINAME <VALUE STRING>	fails if
(VALUE1 IS TRUE) OR (VALUE2 IS TRUE) OR (VALUE3 IS TRUE) ...	
WITH EACH DDINAME <VALUE STRING>	fails if
(VALUE1 IS FALSE) OR (VALUE2 IS FALSE) OR (VALUE3 IS FALSE) ...	
WITHOUT EACH DDINAME <VALUE STRING>	fails if
(VALUE1 IS TRUE) AND (VALUE2 IS TRUE) AND (VALUE3 IS TRUE) ...	

Failure conditions for WITH and its modifiers
under test against a value string.

1.5.4.1 Selection criteria AND clauses.

We may now consider the behavior of the traditional AND clause. Note that there may be a maximum of 9 AND clauses. The sentence will be very difficult to comprehend long before it has acquired 9 AND clauses.

We define the term SELECTION-CRITERION to be of the form

WITH {NOT} {EACH} DDINAME {<VALUE-STRING>},

such that each tests one data definition item against any value string, and modifies it as specified, across such data values as are available and are required, within one item.

Then an AND clause is of the form

SELECTION-CRITERION AND SELECTION-CRITERION AND SELECTION-CRITERION ....

The criterion for success of an AND clause is that each SELECTION-CRITERION succeed, as per the table above.

1.5.4.2 Data selection criteria.

The data selection criterion is made up of an indefinite number of selection-criteria which are ORed together, which may include at most 9 AND-clauses and any number of ORed selection criteria which are not members of AND clauses. The condition for success of the data selection criteria is that at least one of the selection criteria which are ORed together succeed.

5.4.3 Item selection criteria.

The condition for item selection is that the item-id tests succeed, and that the data selection criteria succeed. In other words, the item-id

test is implicitly ANDed with the data selection test.

1.5.4.4 Some things which will not work.

The form

```
LIST MD < "CAT" AND WITH 1 = "D"
```

will not work because the item-id test is implicitly ANDed with the data selection criteria, and because in this context the AND must either attach an item-id test value to "CAT" or generate an AND clause based on a prior selection criterion. This will generate error message 71.

The form

```
LIST MD < "CAT" OR WITH 1 = "D"
```

will not work because of the implicit ANDing, and has been discussed above.

If you desire the case ((1 = "A" OR 2 = "B") AND 3 = "C"), then it must be written in the following manner:

```
LIST MD WITH 1 = "A" AND WITH 3 = "C" WITH 2 = "B" AND WITH 3 = "C".
```

Two data values cannot be compared by the form WITH 1 = 2, because the system has only one temporary data area. If this is desirable, an F-correlative can be generated of the form F;1;2;=, which will return the value 1 when the statement is true, or the value 0 when the statement is false. The above form would be written 'WITH 1=2? = "1"'. This form is probably more efficient than the alternative would be.

Returning to the relationship between the character surrounding a value and the treatment of the value by the ENGLISH compiler, we consider the following example:

```
LIST "20""30""40" FILENAME "50""60" WITH 1.
```

This will compile as though the following had been entered:

```
LIST FILENAME '50''60' WITH 1 = "20" = "30" = "40" = "50 = "60".
```

The values which fall between the file name and the first succeeding data definition item will be construed as constituting an item-id test. Since there are no relational connectives associated with these item-id test elements, the process will explicitly retrieve items 50 and 60. It will then test them to see if the data definition item whose name is '1' will return the value 20, 30, 40, 50, or 60. In this case the item will succeed. Otherwise it will fail. This result may be unexpected.

On the other hand, the form

```
LIST '20''30''40' FILENAME '50''60' WITH 1.
```

will behave like the following sentence:

```
LIST FILENAME '20''30''40''50''60 WITH 1
```

Further, the sentence

LIST "20""30""40" FILENAME "50""60".

will yield the following error message:

[19] A VALUE WITHOUT AN ATTRIBUTE NAME IS ILLEGAL.

The gist of this is that values delimited by ' will be taken as item-ids or item-id test values, that values delimited by " or \ which fall between the file reference and the first data definition item will be taken as item-ids or item-id test values, and that all other values in the string delimited by " or \ will be associated with either the immediately preceding data definition item, if there is one, or with the next data definition item, or if there are no data definition items in the string, then the sentence will fail in the compiler.

1.5.5 On generating lists whose elements are data.

The historical intent of the SELECT and SSELECT verbs was to select a list of item-ids for retrieval of a specified collection of items in a specified order. There was an extension under R77 which allowed selected parts of the contents of items to be returned instead. The intent was to be able to present lists of data to BASIC or RUNOFF. There was a problem in the case which included a sort pass, or in which a list was selected immediately prior to the data selection, which involved the list of item-ids being overwritten by the results of the data retrieval in some cases. This will no longer occur. It was always possible to execute the SELECT form of this operation from a saved list.

1.5.6 Multi-line column-header labels in the columnar processor.

It has been the case for some time that under certain circumstances the second and subsequent lines of the column-header labels would shift to the left or to the right. They will no longer do so. If you have used column-headers which slid around, and if you fixed the problem by padding with blanks here and there, then the following comments may be of interest.

The processor will take as the field width of a column the maximum of the longest line in the column-header for that field or the length of the field specified in attribute 10 of the data definition item. This may make certain reports wider than they used to be. The processor used to take the maximum of the width of the label element specified for this line or the field length specified in attribute 10. The problem occurred when a label element for a particular line was longer than the field specified in attribute 10. It occurred particularly in the case of file definition elements where the file name was longer than the field length definition in attribute 10, because on the second and subsequent lines the offset would shrink to the field length specification, causing all the rest of the label elements on the second and subsequent lines to shift to the left. If you have compensated for this by padding an element on the second line out to a length sufficient to justify the column-headers, your report may become non-columnar.

1.5.7 Totalling data elements of length zero.

Data elements with a length of zero will now be totalled in the columnar processor, although it will still take up one column in the output report.

The use of zero length elements is at break time. They will not print at detail time, but they will print at break time if desired. In that case, they are available to overlay data elements which printed at detail time, but which are not intended to print at break time. Arrangements of this nature are used in conjunction with function correlatives in both attributes 7 and 8 of the data definition item to get numbers which are the result of operating on totals generated at detail time. There are facilities for doing some fairly extensive modifications to the output structure at break time.

1.5.8 Null sub-multi-values and the non-columnar processor.

Historically, the non-columnar processor has terminated the display of sub-values within a value when it encountered the first null sub-value, when it was processing a sentence which did not include the controlling and dependent specification. This had the effect of displaying a value as null if it was made up of a series of sub-values, the first of which was null. The routine has been modified to scan past the sub-values which are null, and display the non-null sub-values, rather than immediately terminating the value on the first null. This may affect your non-columnar reports. The intent of this was to get the same numbers in the columnar and non-columnar reports.

1.5.9 BREAK-ON and DET-SUPP have been enabled for MODEID3 verbs.

A MODEID3 verb is a verb which uses the non-columnar processor to generate a string which represents the results of executing the data definition items on an item, which then exits to special code to process the string into the required form. The LIST-LABEL and SORT-LABEL verbs use this path, for instance. Note that this is one of the ways to extend the power of the system. The user writes the code which is executed as MODEID3, and defines a verb which will exit to MODEID3 with the results of the non-columnar processor, rather than sending those results to a terminal or to a printer.

1.5.10 The Load Previous Value (LPV) operator.

In prior releases, if a data definition item included F-correlatives in both attributes 7 and 8 and the results of the calculations done in attribute 8 were desired in attribute 7, they had to be replicated in attribute 7. In general, 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 attriute 2 of the data definition item. It is now possible to load this data into the function correlative stack using the LPV instruction. Noting that a conversion may call a function correlative, we 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 totalled, 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 3*1]MX%8]F;LPV;"52";K;"*C";:]TFILE;C;;3

This has rather less motivation, since it is equivalent to

008 F;5(3*1]MX%8);"52";K;"*C";:;(TFILE;C;;3).
```

Use of the LPV operator in F-correlatives.

In general, the LPV should be used as the first operator in an 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 may happen. In some cases they might be useful, but be aware that they are subject to change between releases.

The LPV operator is available for use in A-correlatives.

5.11 Length of the string returned by an F-correlative.

It has been possible to get a string back from an F-correlative which was up to 160 bytes long. This limit has been increased to 500 bytes due to

the increasing use of the F-correlative structure. If the string exceeds 500 bytes, it will be truncated on the right. In general, if any single data string retrieved or generated exceeds 500 bytes in ENGLISH, it will be truncated. A single data string is construed as the operation of one data definition item on one sub-value.

1.5.12 T-, A- and F-correlative compilation.

The ENGLISH compiler has been modified to allow more complete checking of correlatives and conversions. All translates will be "opened" during the compile phase, and marked in a more distinct way.

Sequences of conversion written in the form (CONV1);(CONV2);... will be compiled to the form (CONV1]CONV2]...), which will run faster. Certain other minimal improvements of this nature have been made.

A-correlatives may now call A-correlatives and F-correlatives using the N(ATTRNAME) form. The tree of A-correlative calls may be 7 levels deep.

1.5.13 Inclusion of '(text)' in conversion called from an F-correlative.

The F-correlative will now execute a conversion which includes parentheses. This is useful for pattern-matching and for including parentheses in output masks.

1.5.14 An extension to the translate.

The translate syntax now allows the specification of value mark count in the item which is the target of the translate, presuming that it is multi-valued. The form is as follows:

TFILENAME;CN;IAMC;DAMC;BAMC.

THE 'N' following the C is the VMC of the value to be returned, if desired. If no VMC is specified, and the target is multi-valued, then the string returned will include all of the values in the attribute specified by DAMC or BAMC delimited by blanks. Sub-values are always delimited by blanks.

1.5.15 Blank line suppression.

It is now possible to LIST a file without any output by using the form

LIST FILENAME ATTRIBUTENAME ID-SUPP DET-SUPP

This has the effect of printing the page header and the column header without printing a line for each item in the file. If ATTRIBUTENAME was left out of the statement, and there were no default attribute definition items picked up from the file dictionary, or the 'ONLY' connective was used, then the processor used to print a blank line for each item in the file.

The columnar LIST processor has been modified so that the form

LIST ONLY FILENAME ID-SUPP DET-SUPP

will not print a blank line for each item in the file.

If a blank line for each item in the file is desired, then the form

LIST ONLY FILENAME ID-SUPP

will suffice.

The intent of this modification is to find those items which are in a list which are not in a given file. This is useful if you have two files which should have approximately the same item-ids in them, or if you have a saved list which has the desired record keys as its contents. In the first case one of the files must be selected, and in the second case the list must be retrieved. You should then execute the following command:

LIST ONLY FILENAME (CDI

which has the effect of printing only

[780] ITEM "ITEMNAME" NOT ON FILE.

for each item not on file, and

NNN ITEMS LISTED.

where 'NNN' is the number of items found in the file.

If the option 'P' is included, the list will be sent to the printer. If the assignment is to a hold file, then the resulting print file can be moved from a hold file to a data file and massaged with the EDITor to generate a %SELECTable item which may be turned into a list of what is not in the file.

1.5.16 Left-adjustment of page numbers in headings.

Use of the form 'PN' in place of the form 'P' will cause the page number to be left-adjusted in the required field, rather than right-adjusted in a field of four blanks.

1.5.17 Measuring the length of a data string.

It is possible to obtain the length of a data string by using the 'L' conversion with an argument of '0'. Simply execute the phrase L0 in attribute 7 or 8.

1.6 CHANGES TO RUNOFF.

RUNOFF has been improved in ways which are mostly transparent. Some odds and ends which improved symmetry or utility are noted below. There has been a slight change to the handling of upper and lower case commands, and the single-character overstrike and underline indicators.

1.6.1 Upper- and lower-case controls.

The modification of the upper-case, lower-case control structure causes the text to go to the case specified. The prior form caused the text to go to lower-case of LC was employed. The text then reverted to its natural state upon UC. This was useful when terminals tended to be upper-case only, and the printer would execute lower-case. Terminals which handle lower-case are now prevalent, though there are still printers which do not handle lower-case. UC was modified to switch the output to upper-case if UC was used. The new instruction, ENDCASE or EC has been implemented to turn off both the upper-case condition and the lower-case condition to allow the text to go to its natural condition.

The forms ^^ and \\ cause the text to switch to upper-case or to lower-case in the same way that UC and LC cause the switch, except that ^^ and \\ may be imbedded in a line. Turning off the condition ^^ or \\ requires the use of EC.

The forms ^, \, &, and @ will produce one character of upper-case, lower-case, underline, or overstrike, as they were previously documented, but as they did not previously do. Each will be treated as the character itself if it is followed by a blank. The backarrow or underline character, _, continues to cause the succeeding character to be taken as a text character rather than a control character. This means that if you have existing RUNOFF text with forms such as '40@ 851.28/4', the dollar sign will be overstruck and the @ will disappear unless the backarrow, _, is inserted in front of the @. The same is true of the '&' character if it occurs in a character string. This was implemented because it was felt that the use of ^, \, &, and @ should be symmetrical, and that the use of these characters in character strings is less frequent than their use as control characters.

The single-character forms affect only the succeeding character. They override a UC or LC command for that character, and they have no effect on the UC and LC command.

The example below is an attempt to display the interactions of the several commands above. The first part is the text which was sent to RUNOFF and the second part is the output from RUNOFF. First, note that the 'I' in 'is' is always capitalized by the single-character ^, and that the 'a' is always in lower-case due to the single-character \ command.

The first line is in its natural form. The second line is uniformly capitalized by the UC command, excepting the 'a'. The third line is uniformly sent to lower-case, except for the 'Is'. The fourth and fifth lines contain a '^^ text \\' string, which is uniformly capitalized, excepting the 'a'. After the \\ the string reverts to lower-case. The only way to retrieve the capitalization of the string 'UC AND 'LC' is by the use of EC command. Thus, the sixth line is in its natural form.

```
.LINE LENGTH 66.PARAGRAPH 5.J
 This ^is \a test of UC AND LC.
.uc
 This ^is \a test of UC AND LC.
.lc
 This ^is \a test of UC AND LC.
 This ^is \a ^^test of UC AND LC.
 This ^is \a test\\ of UC AND LC.
.ec
 This ^is \a test of UC AND LC.


 This Is a test of UC AND LC.

 THIS IS a TEST OF UC AND LC.

 This Is a test of uc and lc.

 This Is a TEST OF UC AND LC.

 THIS IS a TEST of uc and lc.

 This Is a test of UC AND LC.
```

. Example of .UC, .LC, .EC and the associated ^ and \ characters.

6.2 The comment instruction.

The .* command has been added. This will inform the RUNOFF processor that all of the rest of the text in the line in which it occurs is a comment. It must either be at the beginning of the line, or after another command in a command line. It is always the last command in a line. This allows text to be commented out, and the intent of READs and CHAINs to be noted.

6.3 Treatment of hyphens.

Hyphens which are surrounded by alphabetic characters will allow a word-break on the hyphen in fill and justify modes. That is, if a term is a concatenation of two words separated by a hyphen, and the line overflows within the second part of the term, then the first part and the hyphen are left in the line, and the next line is commenced with the second part of the word.

Similarly, if a line in the source text terminates with a hyphen preceeded by an alphabetic character, and the first character in the next line is an alphabetic character, then the last word in the line and the hyphen will be concatenated with the first word in the next line and output together in a line with the hyphen between the two parts. If there is a line overflow which occurs during this process, the hyphenated word will be handled as above. What the processor will <u>not</u> do is remove the hyphen.

If the hyphen does not have this meaning, then the back-arrow character may be placed in front of it to suppress this action.

1.6.4 A change to HILITE.

The highlight command no longer causes a break in the text. This allows parts of paragraphs to be highlighted in justify or fill mode.

The execution of the hilite command has also been modified so that if the term .HILITE is the last character string in command line, then it is equivalent to .HILITE OFF.

1.6.5 A note on SECTION spacing.

Conventionally the .SECTION command is followed by a blank line before the next paragraph starts. Since the SECTION command causes a break which terminates the preceding paragraph, and since the text following the SECTION command is placed immediately into an ouput line and output prior to a consideration of the next line, the blank line after the SECTION command can be avoided by not indenting the first line of the next paragraph. That is, if the processor does not know that the next line starts a paragraph, it will not skip a line. It may be necessary to use an INDENT MARGIN if paragraph indentation is desired, however.

1.6.6 New options.

The U option has been added to force the whole RUNOFF output to upper-case, if that is desired. The J option will suppress highlighting. The C option will suppress the .CHAIN and .READ commands if it is desired to RUNOFF one element of a chained or treed structure. The I option will cause the name of the next item to be output by RUNOFF to be placed in the last line of the last item RUNOFF. This of use with relative large documents. The S option, which suppresses underlining and overstriking remains.

1.6.7 Other emmendations.

You will note that the sequence <<< will tab over to the third tab if tabs are set, and that right tabs are more co-operative. Note the discussion in the spooler documentation under SP-EDIT on retrieving hold file entries for execution by RUNOFF.

The .READ or .CHAIN {FILENAME} ITEMNAME commands allow any form of the file name and item name. Comments may succeed the .READ or .CHAIN command in the line if the item name is terminated with a trailing blank. Executable instructions in the line after a .READ will not be executed.

1.7 CHANGES TO PROC.

1.7.1 The GO Command

In R80, the PROC branch command, G or GO, has been extended to allow 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. An example follows:

```
001 PQ                          Define PROC
002 RI                          Clear input buffers
003 J 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
 .
 .
 .
```

Example of "GO An" Command

1.7.2 The Secondary Input Buffer

The secondary input buffer is now 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 spooler documentation in the PERIPHERALS manual. The user should note that the secondary input buffer is a very temporary entity and that if its contents are to be used, this should be done immediately subsequent to the execution of the processor which loaded the buffer.

1.8 R80 BASIC IMPROVEMENTS

There are many areas of improvement to Basic. They can be generally broken down into External differences, Speed improvements (both in the compiler and in the run time package), and new or modified Basic syntaxes. Of these only the external differences can cause upgrade problems.

1.8.1 BASIC program file structure.

On the old systems, Basic source files had no structure. Source, object and symbol records were intermixed on the files. This cluttered up the files and made it difficult to manipulate only the source programs. Object code was frequently cataloged into the pointer file for the sake of run time efficiency. This caused problems because anyone could access programs in the unprotected pointer file. Also the object did not get saved/restored with the account on an account save/restore.

Now there is a fixed structure for Basic source files. The file MUST have a dictionary and a separate data level. The Basic source programs are stored in the data level of the file. The compiler writes the object and the symbol file as one record into the dictionary. This makes it much simpler to manipulate the program source. It can be LISTed, T-DUMPed, T-LOADed, and so on, without having to select the source items. The object record has the same format as a pointer-file record and so the dictionary "D" pointer must have a "DC" in attribute one. The primary advantages of this new format are:

1. The object can now be protected with access/update locks.
2. The object saves/restores with the account on account-saves.
3. The CATALOG function is not necessary for run time efficiency.
4. There is less disk space utilized and fewer steps to perform.
5. The Basic Debugger can tell the name of the item and verify the object code integrity.

1.8.2 CATALOGed BASIC programs.

Since the output of the compiler is the same as the result of the verb CATALOG under R77 and earlier releases, the CATALOG and DECATALOG verbs have changed meaning to some extent. Both are now TCL-II verbs. They require the BASIC program file name and one or more explicit item-ids, or a '*', meaning all, as usual, or that a list be in effect. It also means that you can catalog all of the basic programs in one file by using the CATALOG verb only once, and similarly with the DECATALOG verb.

The effect of the CATALOG verb is to point to the file which contains the pointer to the object code. The CATALOG verb no longer needs to go through the pseudo-loading process required under R77 and before.

The meaning of the DECATALOG verb has changed as well. It has the primary purpose of removing the object code string from the system. This string is pointed to by the pointer record in the dictionary of the BASIC program file where the program resides. The program does not need to be cataloged

in order to use the DECATALOG verb. It will also remove the pointer left in the master dictionary by the CATALOG verb if there is one. The DECATALOG verb requires the name of the file in which the BASIC programs are stored.

Note that with both of these verb the BASIC program file name and the program name are the only parameters used by the system. The basic program name is transferred to the object code pointer as its name, and to the master dictionary pointer to the dictionary of the BASIC file. Similarly, the object pointer will reside in the dictionary of the file which contains the source program in the data section, and the pointer which results from the CATALOG verb will point to that file, and the verb requires reference to the file.

1.8.3 SPEED IMPROVEMENTS

Much work has gone into speeding up basic. Obviously most of this kind of work is transparent to the user. The Basic Compiler is now two to three times faster than on R77. This will improve programmer productivity. It may also help to reduce system development time. The compiler is now doing more work than it used to. Optimizing logic has been added to improve run time performance especially in the areas of IF statements, literal array subscripts, and concatenate functions. Other areas of the run time package that have notably improved are all dynamic array functions, locate function, case statement and format masks.

1.8.4 BASIC DEBUGGER

On R77 the use of the BASIC debugger required the generation of a symbol table item. This had the name of the program preceded by an asterisk, eg. *PGM. On R80, the symbol table is embedded in the object code which is placed in the catalog space. The debugger, therefore, has instant access to the symbol table, and no longer requires the use of the 'Z' command except when access to the source code is required. In addition, a number of changes have been made to the overall debugger structure in order to correct a number of peculiar problems which arose with its use. Note that the user may still suppress generation of the symbol table by using the (S) option when compiling programs.

It should also be noted that a user now requires SYS2 privileges to use the Basic debugger. This prevents users from making unauthorized changes to data during reporting and data entry. Also, a new command "?" has been added which will display the current program name and line number.

1.8.5 SYNTAX IMPROVEMENTS

There are a number of minor syntax changes designed to reduce programmer effort and to increase the readability of the Basic program.

1.8.5.1 Change to the OPEN syntax.

Note that the ONLY change to require a possible source code change is the OPEN change.

OLD	NEW
OPEN 'D','filename'	OPEN 'DICT','filema

The word DICT must be explicitly supplied to open a dictionary level file.

OPEN '','filename'	OPEN 'filename'

The leading null expression is optional. Also the file name may be specified as 'DICT filename'. simplifying the inputing of filenames from the terminal.

1.8.5.2 The DATA statement.

DATA x	DATA x,x,x, ...

Multiple expressions are now allowed on the DATA statement. Each expression becomes the response to one input request from the CHAINed process.

1.8.5.3 Changes to EXTRACT et alia.

EXTRACT(da,am,vm,svm)	EXTRACT(da,am)
DELETE(da,am,vm,svm)	DELETE(da,am)
INSERT(da,am,vm,svm,new)	INSERT(da,am;new)
REPLACE(da,am,vm,svm,new)	REPLACE(da,am;new)

Trailing zero subvalue or value mark counts are no longer required. Note that in the INSERT and REPLACE commands if they are omitted the delimiter before the new value must be a semicolon.

1.8.5.4 A new EXTRACT syntax.

EXTRACT(da,am,vm,svm)	da<am,vm,svm>

This is a direct replacement for the EXTRACT statement that should make the source code easier to read and understand. Note that trailing zero

subvalue or value mark counts can be dropped.

1.8.5.5 A new REPLACE syntax.

```
da=REPLACE(da,am,vm,svm,new)      da<am,vm,svm>=new
```

This is a direct replacement for the REPLACE statement.

1.8.5.6 The IF ... ELSE form.

```
IF x THEN NULL ELSE                IF x ELSE
IF NOT(x) THEN                     IF x ELSE
```

The THEN clause of an IF statement is optional if the ELSE clause is present. One or the other MUST be present.

1.8.5.7 The LOCATE() THEN form.

```
LOCATE() ELSE                      LOCATE() (THEN/ELSE)
```

A THEN clause has been added to the LOCATE statement. The LOCATE statement now operates exactly like the new IF statement.

1.8.6 NEW BASIC SYNTAXES

1.8.6.1 BREAK inhibition.

```
BREAK ON
BREAK OFF
```

These commands increment/decrement the break inhibit counter. Note that they are cumulative. If two BREAK OFFs are executed, two BREAK ONs must be executed to restore a breakable status.

1.8.6.2 ECHO control.

```
ECHO ON
ECHO OFF
```

These commands turn the system echo-back on or off. They may be used to suppress the echo back of terminal input.

1.8.6.3 Format Masks

The format mask has been extended so that the user may specify a date format which is the same as any of the valid formats for the standard system 'D' (date) conversion.

1.8.6.4 Masked input.

```
INPUT @(x,y):variable 'mask'
```

This is a VERY complex input function! It is capable of replacing as many as twenty lines of basic code used in screen input. Its functions include cursor addressing, output masking, editing, error messages, input masking, and exception trapping.

This command itself is used for the actual entry of the data. Ancillary functions can be performed by the commands described below. In the above example, "variable" represents the name of the variable being input, and "mask" represents a standard EVOLUTION format mask. If the variable being used already has a value it will be displayed at the specified cursor address using "mask" as the output mask. Regardless, the cursor is positioned one character back of "x" in the "@(x,y)" specification, the prompt character is printed and input is requested. If the user presses the return key, then whatever default value was there before will be accepted. Otherwise, the input will be verified against the mask, and, if acceptable, will be assigned to "variable". If the mask contains a decimal digit specification and/or a scaling factor, then numeric checking will be performed. If the mask contains a length specification (eg. R#10), then length checking will be performed. If the mask is 'D' (or any other valid date mask) then a date verification will be performed.

Note that data is converted on output and input. Thus, if you wish to input a date, the default should be stored in internal format, will be displayed and input in output format and will be placed back in the variable in internal format. Note also that the '%' is a numeric character verification symbol. Thus, for example, if the statement executed is INPUT @(20,10):SOC.SEC '%%%-%%-%%%%' and the data entered is 423-15-6897 then SOC.SEC will contain the value 423156897. If an error condition is encountered, then a message is printed at the bottom of the screen. Some simple examples follow:

```
INPUT @(25,2):INV.DATE 'D'        Inputs a date.

INPUT @(35,7):AMOUNT 'R2,'        Inputs a dollar value.

INPUT @(20,14):NAME 'L#40'        Inputs a text field with
                                  a length specification.

INPUT @(0,10):DESC                Inputs data with no mask.
```

Sample "INPUT @" Statements

1.8.6.5 Other INPUT forms.

```
INPUTERR expr

INPUTTRAP 'xx' GOTO n,n,n,n ...
          or
INPUTTRAP 'xx' GOSUB n,n,n,n ...

INPUTNULL x
```

These are all support functions for the new form of input statement. They allow the user to tailor the INPUT function to conform to local standards.

INPUTERR causes a message, specified by "expr", to be printed on the last line of the screen. This differs from an explicit PRINT statement in that it sets a flag indicating that a message has been printed. Thus, when the next valid entry is made, the system will check the flag and clear the bottom line.

INPUTTRAP allows the user to set a trap for a particular character or characters. Each character in the string specification corresponds to a label in the GOTO or GOSUB clause. Thus, for example, if the statement INPUTTRAP '_X' GOTO 10,20 is executed, the subsequent entry of a '_' character will cause a branch to "10" and the entry of 'X' will cause a branch to "20". The GOSUB form of this expression will cause a subroutine call to be issued instead. Caution - the subroutine RETURN statement will cause a return to the statement following the INPUTTRAP statement - not the one following the INPUT statement.

The INPUTNULL statement allows the user to define a character which is to signify that whatever default value was present is to be replaced by the null string. Thus, if the statement INPUTNULL '/' is executed, the subsequent entry of a '/' character will cause a defaulted value to go to null. Note that the default character is '_'. Some examples follow:

INPUTERR 'INVALID DATA!'	Displays error message
INPUTTRAP '*/' GOTO 150,170	Causes branching if either '*' or '/' is entered.
INPUTNULL '@'	Causes the '@' character to null defaults in INPUT statements.

Examples of INPUTERR, INPUTTRAP and INPUTNULL

1.9 SYSTEM MANAGEMENT CHANGES

The process of management of the system has not changed a great deal. This section will note various changes which may affect the management of the system. To a great extent this will be a cross-reference to things already noted.

1.9.1 The 'O' response during file restores.

When a tape reaches the end-of-tape mark without having finished the routine which it is executing, it will send the "mount next tape" message. When the next tape is mounted, the process will wait for the character 'C', at which time it will check the tape label on the new tape for admissability. If it does not like the tape label, it will say so, and wait for the tape to be changed to the correct tape, and the character 'C' to be entered.

It may occur that the tape that was mounted, which had a label that the processor did not like, was the correct tape, however. It is now possible to execute the response 'O', for override. This will cause the tape accept the new reel without continuing to complain about the label. It may also cause the processor to complain about the label on the next tape, at which time the use of the 'O' response is recommended.

1.9.2. Group format errors.

Group format errors have been discussed elsewhere in this document. The various processes which may encounter group format errors by either reading or writing a file will enter a group format error handler when a group format error condition is encountered.

1.9.2.1 The definition of a group.

The term group is used to specify one 'bucket' of storage. A file is made up of a collection of groups, such that there are the same number of groups as the number specified for the modulo of the file. Put another way, the modulo of the file specifies the number of groups which make up the file.

The hashing algorithm takes the specified item-id and decides in which group it is or should be stored. The file retrieval or storage routine then searches that group for the specified item. The hashing algorithm may be thought of as dividing the item-id by the modulo in order to obtain the remainder. This remainder is then the 'group number', and specifies the group which is to be searched.

Within each group the items are stored physically end to end. Each item is made up of a count field, a key, and the data. The documentation for this system has conventionally used the term 'item-id' in place of the term 'key'. It remains that the item-id is the key which is used to look up the location of the item.

The count field exists only in a file representation of the item. It is a sixteen-bit binary number, such that the high-order bit is zero, represented in the file in ASCII hexadecimal notation, and as such takes

up 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 termination mark of the last 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.

1.9.3 Transient group format errors.

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 ENGLISH, 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 makes up the group may not all be updated synchronously. There is, in other words, a stochastically-determined 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.

1.9.4 Real group format errors.

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

The count field is definitely wrong if any or all of the four digits which make up the count field are not ASCII hexadecimal digits, which are X'30' - X'39' or X'41' - X'46', which are 0-9 and A-F.

The end of item data may be wrong if the count field contains the wrong ASCII hexadecimal digits, or if the end of item data is actually wrong.

The end of item data may be wrong in several ways. If the item is contained in a frame, then the end of item data may be wrong in the ways that the the count field may be wrong. If the item spans a frame boundary, certain other mechanisms come into play. If a process was in the process of updating an item, to the extent that the first frame containing the item was written to disc, but that the last frame was not written when the process was interrupted by something like a cold start, then a group format error will occur. If the overflow handler becomes confused, the frames attached to a group may be acsuired by another data file or by a print file. The difference should be obvious on inspection, using the DUMP verb. Print files do not normally contain attribute or value marks and data files do not normally contain carraige-return, line-feed sequences.

If the damaged frame is the result of an incomplete update, then the difficulty is localized. Repair of this group will usually attend to the matter. If the damage appears to be due to co-ownership of the frame, the problem may be greater. In this case it is best to leave the frame with the frame to which it has a back-link, presuming that the data is

consistent in that chain. Then cut the forward link in the spurious chain and terminate the group.

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.

1.9.5 Recovery from group format errors.

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 context, note that the organization of file-save tapes written by R77 and later releases 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 occurs in the middle of the desired account, the an A option is to be used.

1.10 CHANGES TO THE ASSEMBLER

1.10.1 Assembler Directives

The Assembler has been changed to suppress the execution of the EJECT and CHAIN directives. These can be re-enabled by using the (J) option when assembling programs.

1.10.2 Assembler Files

In R77 and prior releases, the OSYM, PSYM and TSYM files were one level (dictionary only) files.

On R80, these files must have a dictionary. Note that the R80 sysgen procedure will take care of this for the standard system files, but users with their own versions should be certain to make the same change.

1.10.3 Comments Option

The Assembler on R80 no longer asks for comments when assembly is performed.

1.11 R80 ASSEMBLY DIFFERENCES

1. Changes where made in the encoding of some of the instructions. This means that ALL code must be reassembled before it can be run.

2. Branch conditions have been extended to cover all six possible conditions. All branch instructions have Low, Low or Equal, Equal, Not Equal, High or Equal, and High conditions (L,LE,E,NE,HE,H). For the single operand instructions these are Less than Zero, Less than or Equal to Zero, Zero, Not Zero, Higher than or Equal to Zero, and Higher than Zero (LZ,LEZ,Z,NZ,HEZ,HZ).

3. A branch decrementing by 1 instruction has been added. It has the same format as the single operand test & branch but is a BDxx instead of Bxx. For example, BDNZ T0,Label

4. Monitor calls have been given unique names. This will avoid possible nasty results from mis-coding a monitor call and will help minimize source changes in the future.

5. Register detach functions should be done with the new commands DET, DETZ and DETO. These commands detach a register or detach it and zero (DETZ) or one (DETO) the displacement field. Manipulating the WA or DSP fields of a register directly is not valid and may cause may problems.

6. To improve system security, setting the tally USER to X'F512' no longer releases system access/update locks. There is no way for a user account to bypass access/update locks. Please be careful -- you can disable your system!! To prevent total disability, the account SYSPROG is considered privileged, and can access any system file regardless of access/update locks.

7. The firmware will activate the user on every character keyed. This helps to keep the user near the top of the priority list and prevents a considerable amount of disc thrashing.

8. User code cannot access PIB locations other than the status bytes. Access to these two bytes must be through the monitor calls PIB.AND/PIB.OR

9. The only approved way to open a file is by using a new system function FILEOPEN. This replaces separate calls to the old functions GBMS and GOLID. The OPEN processor has a simplified interface and will generally mean a reduction in the code required to open a file.

10. All string decrementing instructions were deleted because of a lack of firmware space. If you used them you must recode them in software.

11. As mentioned in the R77 manual, the arithmetic condition flags no longer exist. If you used ZROSIT or NEGSIT as arithmetic flags after an accumulator operation, you must change the code to use explicit tests.

1.12 SYSTEM DEBUGGER IMPROVEMENTS

A number of new features have been added to the System Debugger. These are most likely of interest only to the Assembly Language programmers. There is only one change - the way the debugger is enabled/disabled. Everything else is in the form of extensions or new commands.

1.12.1 DEBUGGER ENABLE/DISABLE

The procedure to enable/disable the system debugger has changed. On R77 it involved physical front panel switches and an IKI sequence. This was inconvenient, especially if a problem occurred when inexperienced personnel were operating the machine. The function of enable/disable is now done by typing DB when in the system debugger and on SYSPROG. Note - you must be on SYSPROG to enable/disable the debugger. The debugger will respond DB ON if now enabled or DB OFF if now disabled.

1.12.2 KILL ALL BREAK POINTS

It is now possible to kill all existing break points by typing K, carriage-return. The debugger will respond with K-.

1.12.3 KILL ALL TRACE VALUES

It is now possible to kill all existing trace values by typing U, carriage-return. The debugger will respond with U-.

1.12.4 BREAK AT EVERY STATEMENT IN A FRAME

By typing bx.0 as a break point, the debugger will break at every instruction in the frame x. This is equivalent to using an E1 function execept that it only traces instructions within a given frame. This is a standard break and is killed explicitly by typing Kx.0 or K carriage return.

1.12.5 VARIABLE VALUE TRACE

A new function - trace by value - has been added to the system debugger. When you type Yvariable the debugger will examine the contents of the variable. Whenever the contents change, the debugger will break and print the address and the new contents of the variable. This is extremely valuable when a variable is being changed for unknown reasons or by an unknown processor. The process will run quite slowly if a value trace is in effect because the system is breaking after each instruction to test the contents of the variable. Variable may be any direct, indirect, or symbolic variable that would be valid in a trace statement. You may value trace two variables at a time.

To kill value traces you either type Y carriage return or END to the debugger.

1.12.6 FRAME SUBSTITUTION

A new function - frame substitution - has been added to the system debugger. When you type Fx,y to the system debugger, you initiate FRAME SUBSTITUTION. X and y are frame numbers in decimal. When frame substitution is in effect, any external BSL or ENT to frame x will be made to the corresponding entry point in frame y. This allows one to test a new or modified frame without disrupting the operations of the rest of the system. Presently you may only substitute one frame.

To kill frame substitution you type either F carriage return or END to the debugger.

1.12.7 ARITHMETIC FUNCTIONS

As an add to debugging, all the system arithmetic functions have been made available when in the system debugger. Typing ADDD, ADDX, SUBD, SUBX, MULD, MULX, DIVD, DIVX, XTD, or DTX followed by the standard parameters will return the result to the CRT.

These are the steps to upgrade from R77 to R80:

1. The C.E. Installs the new firmware board and performs any hardware tests necessary.

2. Boot the system with the R80 SYSGEN tape.

3. Select the F option and restore the ABS section from the SYSGEN tape.

4. Change tapes to the first reel of your FILE-SAVE and restore the DATA section from your tape(s).

5. When the restore is finished DO NOT do the COLDSTART proceedure! Instead, BREAK and type 'OFF'.

6. Logon to SYSPROG.

7. Type 'ED MD T-EOD' and change line two (2) to '709A'. File the item.

8. Remount the R80 SYSGEN tape and type 'T-EOD'.

9. When the tape stops type 'T-FWD'.

10. Type 'T-LOAD MD (CI)'.

11. Type 'SYSGEN-R80'. The SYSGEN Proc will begin executing. It will load system files R80-UPDATE, SYS-ERRS, SYSPROG-PL, BLOCK-CONVERT, NEWAC, ERRMSG, DSYM, PSYM, STAT-FILE, PROCLIB, and ACC. It will then update the verb definitions in the master dictionaries of all the accounts on the system. It then performs the COLDSTART proceedure which will allow you to set the date and time.

12. At this point, the primary updating is done. All that remains is to re-organize the POINTER-FILE lists, to convert BASIC source files to DICT/DATA files, and re-compile/catalog all your BASIC programs.

LOGON HAS BEEN IMPROVED TO REQUIRE ATTRIBUTE 8 OF THE SYSTEM TO CONTAIN THE LETTERS 'SYSx' BEFORE THE ENTRY WILL BE CONSIDERED VALID FOR LOGON. 'X' MAY BE ANY ASCII CHARACTER. IF 'x' IS A 1, SYS1 PRIVILEGES WILL BE SET. IF 'x' IS 2, SYS2 PRIVILEGES WILL BE SET. ALL OTHER VALUES OF 'x' WILL RESULT IN SYS0 PRIVILEGES.

ATTRIBUTE 11 MAY CONTAIN A LIST OF LINE NUMBERS FOR WHICH LOGON IS VALID. IF ATTRIBUTE 11 CONTAINS ANYTHING, IT IS ASSUMED THAT THE ENTRIES ARE VALID LINE NUMBERS FOR THE LOGON PROCESS TO VERIFY. WHEN THE LINE WHICH IS ATTEMPTING IS NOT SPECIFIED IN ATTRIBUTE 11, THE SYSTEM RESPONDS AS IF THE ACCOUNT DID NOT EXIST.

THE VALID LINE NUMBERS MUST BE STORED IN ASCENDING ORDER. NUMBER RANGES ARE ALLOWED. EXAMPLE: 0,2-4,7,9,13-15
THE ACCOUNT COULD ONLY LOGON TO THE FOLLOWING LINES:
0 2 3 4 7 9 13 14 15