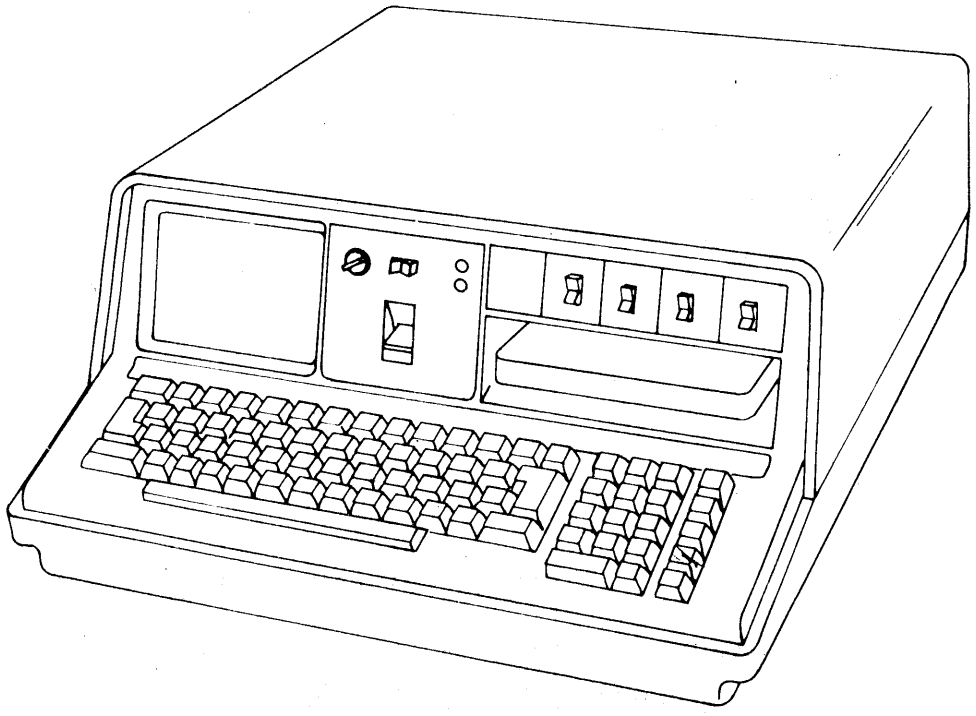
IBM

IBM 5110
BASIC Reference Manual

5110



IBM 5110
BASIC Reference Manual

SA21-9308-2

Preface

This publication is a reference manual that provides specific information about the use of the IBM 5110 Computer, the BASIC language, planning, and procedures. It also provides information about the insertion of forms and ribbon replacement for the 5103 printer. This publication is intended for users of the 5110 and the BASIC language.

Prerequisite Publication

IBM 5110 BASIC Introduction, SA21-9306

Related Publications

IBM 5110 BASIC User's Guide, SA21-9307

IBM 5110 BASIC Reference Handbook, GX21-9309

IBM 5110 Customer Support Functions Reference Manual, SA21-9311

Third Edition (April 1979)

This is a major revision of, and obsoletes, SA21-9308-1 and Technical Newsletters SN21-0299, SN21-0302, and SN21-0304. This publication should be reviewed in its entirety.

Changes are periodically made to the information herein; changes will be reported in technical newsletters or in new editions of this publication.

This publication contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Use this publication only for the purpose stated in the *Preface*.

Publications are not stocked at the address below. Requests for copies of IBM publications and for technical information about the system should be made to your IBM representative or to the branch office serving your locality.

This publication could contain technical inaccuracies or typographical errors. Use the Reader's Comment Form at the back of this publication to make comments about this publication. If the form has been removed, address your comments to IBM Corporation, Publications, Department 245, Rochester, Minnesota 55901. Comments become the property of IBM.

Contents

Chapter 1. Operation

IBM 5110 OVERVIEW

The IBM 5110 Model 1 (Figure 1) is a general-purpose desktop computer designed to meet the data processing requirements of a small business. The 5110 Model 1 has a display screen, a combined alphameric and numeric keyboard, a tape unit, switches, and indicator lights. The 5110 Model 2 is identical to Model 1 except that it has no built-in tape unit. The display screen and indicator lights communicate information to the user, and the keyboard and switches allow the user to control the operations the system will perform. Figure 1 shows a combined BASIC/APL 5110.

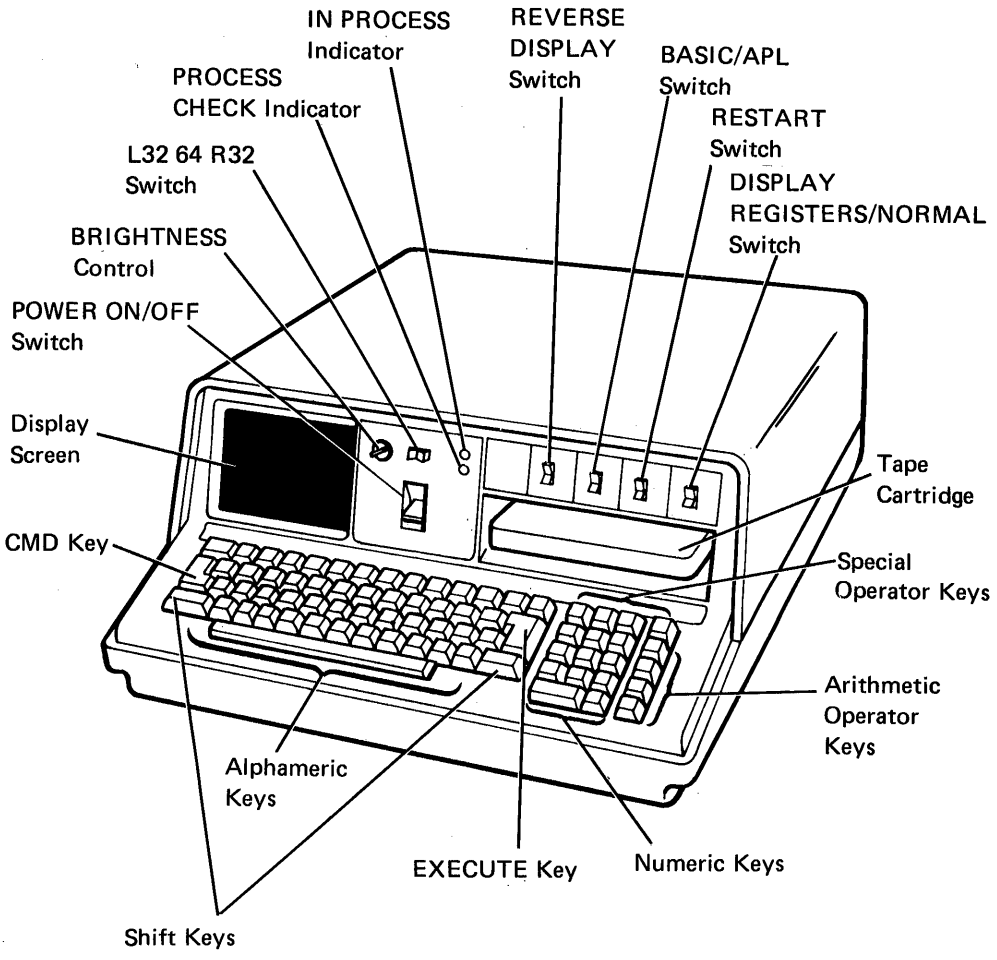


Figure 1. The IBM 5110 Computer

DISPLAY SCREEN

The display screen (Figure 2) can display 16 lines of data at a time, with up to 64 characters in each line. Input data (information supplied by the user) as well as output data (processed information) is displayed. The bottom line contains status information. The number in the lower right (NNNNN) indicates the number of character positions (bytes) in storage available to the user.

Line 1 (input line) contains information entered from the keyboard. The cursor (flashing horizontal line) indicates where the next input from the keyboard will be displayed. If the cursor is moved to a position that already contains a character, that character flashes. As BASIC processes the input, all lines of the on line 1 again. The number following NNNNN in the lower right identifies the current cursor position relative to the start of the input. This number can be up to 896 (14 lines of 64 each).

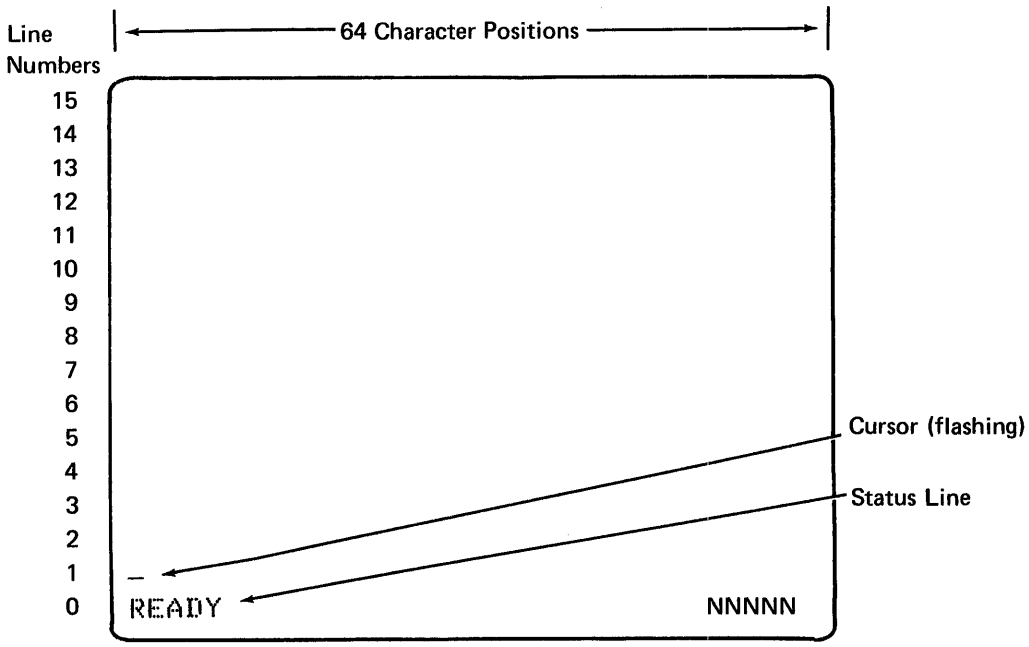


Figure 2. The 5110 Display Screen Format

KEYBOARD

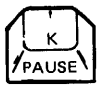
The APL/BASIC system keyboard (Figure 3) combines alphameric and numeric characters. The keyboard provides two modes of character keying selection: standard BASIC character mode and lowercase character mode. When you turn the system power on, you are in standard BASIC character mode. In standard BASIC character mode, you can enter uppercase alphameric characters (without using the shift key), the uppercase symbols (using the shift key), the BASIC statement keywords (on the front of the alphameric keys), and BASIC commands (above the top row of numeric keys) using the CMD key.

In lowercase character mode, you can enter lowercase (without using the shift key) and uppercase (using the shift key) alphameric characters, along with the uppercase symbols (selected using the CMD key). In lowercase character mode, you cannot select the BASIC statement keywords because the CMD key is used to select uppercase symbols.

To select lowercase character mode, press the HOLD key, then hold down the shift key while you press the ↓ Scroll Down key. To return to standard BASIC character mode, press the HOLD key, then hold down the shift key while you press the ↑ Scroll Up key.

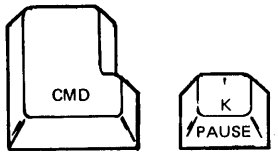
The following examples show the use of standard BASIC character mode and lowercase character mode.

In standard BASIC mode, press:



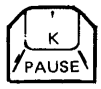
to enter the character K.

to enter the character ' (single quotation mark).



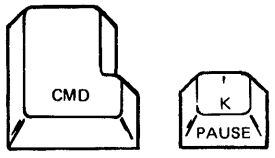
to enter the keyword PAUSE.

In lowercase character mode, press:



to enter the character k.

to enter the character K.



to enter the character ' (single quotation mark).

Note: All the examples in this manual are in standard BASIC character mode.

You can enter numeric data conveniently by using the calculator arrangement of numeric keys to the right of the alphameric keys. You can also enter this data by using the numeric keys above the alphameric keys. The arithmetic operator keys, located on the right of the keyboard, are also on the alphameric keys.

Note that on a BASIC-only keyboard (Figure 4) upper shift APL symbols above the alphameric characters can be entered as data, even though they do not appear on the BASIC keyboard.

When any of the keys are pressed, the characters entered appear in the input line on the display screen.

BASIC statement keywords are printed on the front of the alphameric keys. These words are entered starting at the current cursor position when the CMD key is held down and the corresponding key is pressed (in standard BASIC character mode).

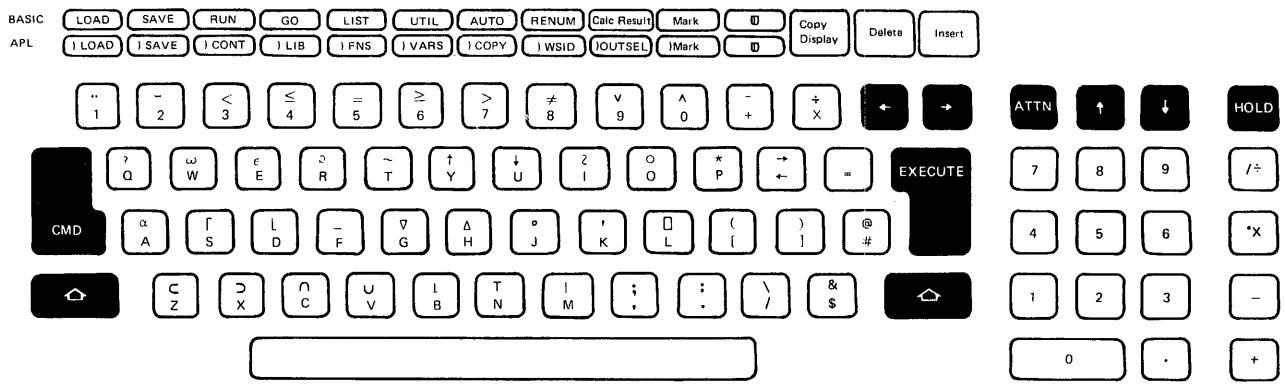


Figure 3. Combined BASIC/APL Keyboard

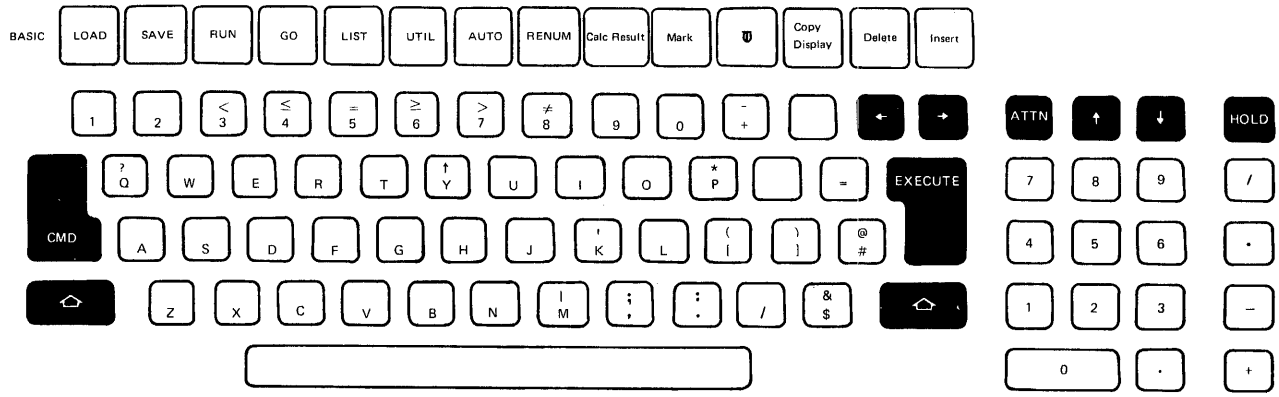


Figure 4. BASIC-Only Keyboard

Special Keys

The following keys have special functions relating to system operations:

(Attention)–You can press the ATTN Key to stop system operations. While you are entering data at the keyboard, you can press the ATTN key to blank everything from, and including, the cursor position to the right on the input line of the display screen. If the display is in use as an input device, the ATTN key will blank everything to the right and below the cursor position. You can then continue entering information. In addition, the ATTN key and the HOLD key are the only keys that are active when the display screen flashes to indicate detection of an error. After you press the ATTN key to stop the flashing display screen, all keys are active and you can proceed. The ATTN key must also be used (after the audible alarm has sounded) to resume procedure file operations.

(Command)–When you hold down the CMD key, you activate the top row of alphameric keys, which cause the command keyword to be inserted on the input line when the number below the word is also pressed. The command operation is executed when you press the EXECUTE key. The CMD key is also used with the numeric keys (on the right side of the keyboard) to initiate keys functions. Avoid holding down the CMD key and pressing the HOLD key; this activates a function restricted to use by service personnel. Holding down the CMD key (in standard BASIC character mode) and pressing a key with a BASIC statement keyword on the front will enter the keyword beginning at the current position of the cursor. In lowercase character mode, the CMD key allows you to type in the uppercase symbols. The CMD key is used with the [-/+] key to exit a program if data is requested. (The arithmetic operator [+] key should not be used for this operation.)

When you press the EXECUTE key, the system processes the line of data that you just entered on the input line. In addition, you can press the EXECUTE key to resume interrupted processing. Pressing the EXECUTE key when the input line is blank causes the same action as a GO command (see *GO Command* in Chapter 2).

When you press the HOLD key, all processing stops. Processing resumes when you press the HOLD key a second time. Thus, the HOLD key allows you to read displayed data during an output operation. While processing is stopped, the CMD key and the arithmetic operator keys on the right of the keyboard are restricted to use by service personnel. The HOLD key will not operate when the display screen is not activated. The HOLD key also allows you to change from standard BASIC character mode to lowercase character mode and vice versa. See *National Character Sets* in Chapter 3.

(Shift)–While you hold down a Shift key, you can select an uppercase symbol (in standard BASIC character mode) or an uppercase alphameric (in lowercase character mode) for input.

(Copy Display–when the CMD key is held down)–When you hold down the CMD key and press the key below Copy Display, all data displayed on the screen is printed by the attached printer. You can use the copy display function when the system is in a hold state, or anytime the system is waiting for input from the keyboard. On a combined BASIC/APL machine, the  key is used to activate the copy display function. The L32 64 R32 switch has no affect on printed data.

The following keys have a repeat capability, which means that they will continue to function for as long as they are pressed:

(Backspace)–When you press this key, the cursor moves one position to the left. When backspaced from position 1 of the input area, the cursor moves to the rightmost position of the input area. When you hold down the CMD key and press this key, you immediately delete the character at the current cursor position. All characters to the right of the cursor are shifted one position to the left each time you press this key. The cursor does not move.

(Forward Space)–When you press this key, the cursor moves one position to the right. When spaced beyond the end of the input area, the cursor returns to the first position of the input area. When you hold down the CMD key and press this key, you immediately insert a blank character at the current cursor position, and all data following the cursor position is shifted one position to the right. The cursor does not move; thus you can enter another character into this position. If the input area contains a character in the last position, the insert function is ignored.

(Scroll Up)–When you press this key, each displayed line moves up one line position (except the status line). This key also allows you to change from lowercase character mode to standard BASIC character mode (when used with the HOLD and shift keys). When the keyboard is open and the cursor is displayed on one of the top 14 lines, the Scroll Up key moves the cursor up one line.

(Scroll Down)–When you press this key, each displayed line moves down one line position (except the status line). This key also allows you to change from standard BASIC character mode to lowercase character mode (when used with HOLD and shift). When the keyboard is open and the cursor is displayed on one of the top 14 lines, the Scroll Down key moves the cursor down one line.

Note: The Spacebar also has repeat capability. Blanks are inserted if you hold down the Spacebar.

SWITCHES

The following switches are located above the keyboard on the console:

POWER ON/OFF – The Power switch turns the system on and off. When power is turned on, the system becomes operable in approximately 10-15 seconds.

RESTART – This is a spring-returned switch that reinitializes the system to its power on state. In the dual language system, the setting of the BASIC/APL switch determines which language is initialized.

BASIC/APL – This switch appears only on the dual language system and determines which language is initialized at power on and restart.

REVERSE DISPLAY – This two-position switch sets the display screen to white characters on a black background or to black characters on a white background. You may want to adjust the BRIGHTNESS switch after setting the REVERSE DISPLAY switch.

BRIGHTNESS – This control varies the intensity of the characters or the screen background.

DISPLAY REGISTERS/NORMAL – This switch is for use by service personnel. This switch should remain in the NORMAL position during system operation.

L32 64 R32 – The three positions of this switch are:

- L32 The leftmost 32 characters on the display screen are displayed with a blank between characters.
- 64 Up to 64 characters per line are displayed in adjacent positions.
- R32 The rightmost 32 characters on the display screen are displayed with a blank between characters.

INDICATORS

The console has two indicators:

PROCESS CHECK – When this indicator lights, a system malfunction has been detected, and further operations are not normally possible. Press the RESTART key. If the condition recurs, call for service.

IN PROCESS – This indicator lights only to inform you that the system is operating even though the display screen is turned off. Because some programs require several minutes of processing that turns off the display, the IN PROCESS indicator is your assurance that the system is operating. When this indicator light is on, the HOLD key does not stop processing.

EDITING INPUT LINES

If you detect an error in a line before you press the EXECUTE key to enter the line into the system, you can use the Forward Space or Backspace key to position the cursor at the error, then:

- You can use the insert or delete functions to correct the error.
- You can press the ATTN key to blank all data to the right of the cursor.
- You can enter the correct character. Note that the APL symbols (upper shift on several keys) in Figure 3 can be entered into an input line. On a BASIC-only keyboard (Figure 4), however, these symbols do not appear even though they can be entered into an input line. Because many of these symbols can be overstruck by other characters (such as ' and . to create !) you can take the following steps to replace an APL symbol:

  1. Position the cursor at the symbol.
  2. Press the Spacebar to erase the symbol.
  3. Backspace the cursor to the blank position.
  4. Enter the correct character.

If you detect an error in a line after you press the EXECUTE key to enter the line, you can use the Scroll Up or Scroll Down key to position the line to be corrected, then use the procedures above to correct the error.

If you want to change an entire line, you can simply enter the statement number of the line, then reenter the line and press the EXECUTE key. The new line will replace the old line.

If you want to delete one or more program lines, enter the statement number of the line you want to delete, then enter DEL and press the EXECUTE key. To delete several successive lines, enter the number of the first line, enter DEL, enter the number of the last line, then press the EXECUTE key. For example, to delete lines 20 through 90 in a program, enter:

```
20 DEL 90
```

then press the EXECUTE key. Note the DEL is invalid while you are entering a key group (see *Function Keys* in Chapter 2).

STORAGE CAPACITY

The base system (Model B11 or B21) has a storage capacity of 16K (K=1,024 bytes). Figure 5 shows how this storage is allocated to various requirements. Note that the work area available to the user is approximately 11,500 bytes, while 4,500 bytes are required for internal purposes. The storage capacity is increased in the following models of the system (the first digit represents model number 1 or 2, and the second digit represents storage size):

Model B12 and B22 – 32K
Model B13 and B23 – 48K
Model B14 and B24 – 64K

In these models, all additional storage is allocated to the user work area. For example, on the Model B14, the user work area is approximately 59,500 bytes, with the remaining 4,500 bytes used for internal purposes.

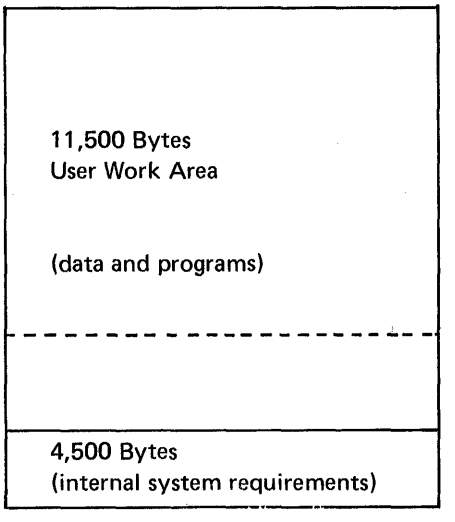


Figure 5. Storage Allocation for a 16K System

Chapter 2. System Commands

Some of the system commands are listed above the top row of numeric keys on the typewriter-like alphameric keyboard (Figure 4). These commands allow you to control diskette, tape, and printer operations, such as storing a program, loading a program into the system, and executing a loaded program. System commands can either be entered character-by-character from the keyboard, or the entire keyword can be entered by holding down the CMD key and pressing the appropriate number key below the command (except for those, such as LINK and MERGE, that are not listed). The latter operation inserts the keyword with a single keystroke, thus preventing possible keying errors and providing faster operation. The commands are used to direct the system to perform the following types of operations:

- Program Execution–Start or resume execution of a BASIC program or command.
- File Operation–Load or save programs or data on tape, or diskette, or mark tapes/diskettes.
- Program Operation–List and number program statements or merge several programs into one program.

Parameters required for a command can be entered after the command keyword is entered. The command operation then starts after you press the EXECUTE key. The command keywords and major functions of the BASIC system commands are:

ALERT	Alert the operator from a procedure file
AUTO	Automatic line numbering
CSKIP	Skip within a procedure file on a specified condition
GO	Resume interrupted processing
LINK	Load customer support functions, features, or IMFs
LIST	Display or print a BASIC program
LOAD	Load a BASIC program
MARK	Mark tape/diskette files
MERGE	Merge a BASIC program
PROC	Initiate input from a procedure file
RD=	Specify rounding position of printed numeric values
RENUM	Renumber statements
REWIND	Rewind tape
RUN	Run a BASIC program
SAVE	Save a BASIC program
SKIP	Unconditionally skip records within a procedure file
UTIL	Perform system support functions

DEVICE ADDRESS PARAMETER

Many BASIC commands and statements require entry of a device address parameter.

This address identifies the input/output device containing the file being processed. Valid device addresses for the 5110 are:

E80 – for the built-in tape unit (Model 1 only)
E40 – for the auxiliary tape unit (Model 1 only)
D80 – for diskette drive 1
D40 – for diskette drive 2
D20 – for diskette drive 3
D10 – for diskette drive 4
500 – for the printer
000 – for directing output to line 1 of the display screen (output only)
001 – for allowing GET statements to access data entered from the keyboard (in the same manner that INPUT statements receive data)
002 – for allowing up to 14 lines of the display screen to be used as keyboard input, output, or update as a record file

Note: The normal system default address on the 5110 Model 1 is device E80. The normal system default address on the 5110 Model 2 is device D80. These defaults can be changed with the UTIL SYS command, which allows you to select the default address you want to use (see UTIL Command). If you do not enter a device address with commands, the default address will be used. You can also enter 'SYS' for the device address parameter in the OPEN [FILE] and CHAIN statements to specify the system default device address.

FILE REFERENCE PARAMETER

The file reference parameter in many BASIC commands and statements can consist of either the file number or the file name, or both the file number and name. File names can be of the following types:

- For any files on tape, the file name can be any combination of up to 17 characters.

- For files on diskette, the file name can be either simple or complex. Simple file names can be from 1 to 8 characters in length. The first character must be an uppercase alphabetic character (A-Z). The remaining characters can be alphanumeric (A-Z or 0-9). Blanks are not permitted. A simple file name must be used when creating basic exchange diskette files (see *Customer Support Functions Reference Manual* for information on basic exchange files).

- Complex file names can be two or more simple names separated by a period. Total length of a complex file name is 17 characters including the period separator(s). As with simple file names, blanks are not permitted within complex file names.

When you enter a file name, be sure to enclose it in single quotation marks. If you enter both the nonzero file number and file name for a diskette file, the system accesses the file by name, but also compares the number you specify with the number of the file accessed. If the file number *and* name do not match, an error message is displayed.

Note: When a file is being accessed for output, an error will occur unless the file is unused (type 0), or unless the file name you specified matches the existing file name. See *UTIL Command* in Chapter 2 for information on making a file unused or changing a file name.

COMMAND SYNTAX

The syntax and description of each command is detailed in this section. Note that the syntax used in BASIC commands is also used for the BASIC statements (Chapter 4). In this syntax, parameters that must be specified as shown are in uppercase letters. Parameters you must supply are in lowercase letters. Optional parameters are enclosed in brackets ([ ]). Parameters enclosed in braces ({}) indicate that you must enter only one of the enclosed parameters. Ellipses (...) indicate that the preceding parameters can be repeated. Single quotation marks and parentheses must be entered where shown. Commas *must be* entered to separate parameters, except between the keyword and the first parameter.

Each command or statement entered from the keyboard is checked for syntax errors. If a syntax error is detected, an up arrow (↑) is displayed below the position in the line where the error was detected, the optional audible alarm sounds, the display screen flashes, and the keyboard is locked. The ATTN key and HOLD key are the only active keys when the display screen flashes. Press ATTN to stop the flashing screen, then correct the indicated error.

Some general rules that apply to system commands are:

- No preceding statement number is needed.
- Each command must begin a new line.
- Maximum command length is 64 characters.
- Blanks are ignored except in character strings enclosed in single quotation marks.
- Parameters must be separated by a comma.
- A comma is not required between a command keyword and the first parameter.

ALERT COMMAND

The ALERT command allows you to provide an indication to the operator that intervention is needed during the execution of a procedure file (see *Procedure File* in Chapter 3). When executed, the ALERT command halts system operation, sounds an audible alarm, and causes the display screen to flash the word ALERT and the optional comment. The operator must press the ATTN key to stop the flashing screen. The optional comment can be an instruction to change tapes or diskettes or perform other necessary operations. Comment length is limited only by line length. To resume normal operation, the operator can enter GO. To terminate operation, the operator can enter GO END. Also, the operator can enter another PROC command to initiate another procedure file and terminate the current procedure file.

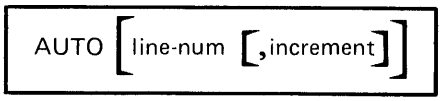
The ALERT command can only be entered in a procedure file.

Example

The following is a typical ALERT command.

```
ALERT REPLACE RECEIVABLES DISKETTE WITH PAYABLES DISKETTE
```

When this command is executed in a procedure file, the associated message will be flashed on the screen to instruct the operator to change diskettes.



AUTO COMMAND

The AUTO command allows you to initiate automatic line numbering for BASIC statements. Automatic numbering simplifies the task of entering statements in a BASIC program. You can specify both the beginning statement number and the increment between numbers. After you enter a statement and press the EXECUTE key, the next statement number is generated and displayed. The syntax of the AUTO command is as shown above, where:

line-num is a positive integer specifying the first statement number to be generated. The range of this number is 1 to 9999. If a beginning number is not specified, a beginning number of 0010 and an increment of 10 are generated.

increment is a positive integer used to increment succeeding statement numbers. If a beginning line number is not specified, the increment cannot be specified. The default is 10 for this optional entry.

Each statement number generated by the AUTO command is followed by a blank, then the cursor, as shown:

```
0010__
```

Note About AUTO

Automatic numbering continues until any other valid data, such as a command word, or a statement number other than the displayed number is entered on the input line. In this case, another AUTO command must be entered in order to resume automatic numbering. Other ways to terminate AUTO numbering are: press the Scroll Up key, which will display a new unnumbered line, or simply press the EXECUTE key when the input line contains only the line number.

Example

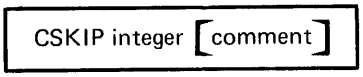
The following examples show AUTO commands:

AUTO (then press the EXECUTE key)

In this example, the display screen will show statement number 0010. After you press the EXECUTE key at the end of an entered statement, the statement number automatically increases by 10, producing statement numbers 0020, 0030, 0040, and so on.

In the following example, the beginning statement is 0320. After each succeeding statement is entered, the statement number is automatically increased by 20, producing statement numbers 0340, 0360, 0380, and so on.

AUTO 320,20 (then press the EXECUTE key)



CSKIP COMMAND

The CSKIP command allows you to conditionally skip records within a procedure file (see *Procedure File* in Chapter 3). This command is valid only when used with an active procedure file. The syntax of the CSKIP command is as shown above, where:

integer indicates the number of procedure file records to be skipped when the condition is satisfied.

comment is an optional comment.

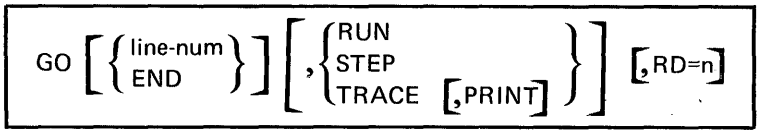
Upon execution of the CSKIP command, the value of the return code (the RC= parameter) set by the last END or STOP statement is checked. If the value is zero or less, no action is taken. If the value is greater than zero, the specified number of records in the procedure file are skipped. If the number of records specified exceeds the number of records remaining in the file, the procedure file is closed. The specified number must be a whole number. For example, an entry of 3.5 causes three records to be skipped, with the .5 assumed to be the beginning of the optional comment.

Example

The following example shows the execution of a SKIP command within a procedure file.

```
LOAD 4
RUN
CSKIP 3
LOAD 9
RUN
SKIP 3
ALERT INSERT TRANSACTION DISKETTE
LOAD 11
RUN
```

Upon execution of this procedure file, the program in file 4 runs. If the program in file 4 sets the return code to nonzero (see *STOP* in Chapter 4), the CSKIP 3 causes the next three records in the procedure file to be bypassed and ALERT to be processed. The program in file 11 is then run. If the return code was zero, the program in file 9 is loaded and run instead. The SKIP then causes the remainder of the procedure to be bypassed.



GO COMMAND

You can use the GO command to do the following:

1. Resume or end processing of a BASIC program that was halted by one of the following:
   a. A PAUSE statement
   b. Executing statements in step mode (see *RUN Command*)
   c. Some error conditions
   d. Your pressing the ATTN key

2. End a MARK command operation before it reaches its normal termination point or resume an interrupted MARK operation to re-mark a file.

3. Resume or terminate PROC file operations interrupted by an ALERT command.

You can resume processing by pressing the EXECUTE key on a blank line (implied GO). The GO command, however, allows you to resume processing in any of three modes of operation (normal, step, or trace). Thus, you can change the operating mode with the GO command. Program execution can be continued with the next sequential statement or at a statement number specified in the GO command. You can change the decimal position that activates rounding (see *RD = Command*).

To terminate the execution of a system command or a BASIC program, the GO command has this syntax:

```
GO END
```

END closes input and output files, thus maintaining the integrity of the files. After the files are closed, no program statements are executed.

To continue execution of an interrupted operation, the GO command has this syntax:

where:

line-num is the number of the statement at which processing is to be resumed. If this number is omitted, processing begins with the next sequential statement.

RUN specifies that processing is to continue in normal mode.

STEP specifies that processing is to continue in step mode (see *RUN Command*).

TRACE specifies that processing is to continue in trace mode (see *RUN Command*).

Note: If neither RUN, STEP, nor TRACE is specified, processing will continue in the mode that was in operation when processing was interrupted.

PRINT specifies that trace messages are to be displayed and printed. If PRINT is omitted, the messages are only displayed. This entry should only be specified with TRACE.

RD = n allows you to specify the number of digits to the right of the decimal point that will cause rounding. On printed or displayed output, n can be 1 to 15 and is initialized to 6. If n is not specified, it retains its last value. It also sets the comparison tolerance (See *Comparison Tolerance* in Chapter 3) when you do not use the image or FORM statements.

Notes About GO

- The statement number entry is valid only during execution of a BASIC program.
- When the input line is blank, pressing the EXECUTE key causes the same operation as a GO command.
- In response to an error indicating an attempt to mark a file already marked, GO must be entered *only* in positions 1 and 2 of the input line.

Example

The following examples show a variety of GO commands.

1. To change to or resume normal operation of a BASIC program:

   GO RUN (then press the EXECUTE key)

2. To change from step or normal mode to trace mode:

   GO TRACE (then press the EXECUTE key)

3. To change to step mode and begin execution at statement number 620:

   GO 620, STEP (then press the EXECUTE key)

LINK file-ref [,dev-address]

LINK COMMAND

The LINK command allows you to access the customer support function features, and IMFs. To access these programs, enter the LINK command with the syntax shown above, where:

file-ref can be either the file name (enclosed in single quotation marks), the file number, or both the file number and name (see *File Reference Parameter*).

dev-address is the address of the device containing the tape or diskette (see *Device Address Parameter*).

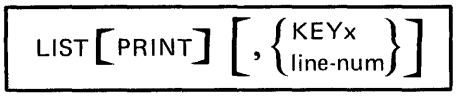
Upon successful execution of the LINK command, the system transfers control to the support function you specified. If an error occurs during the LINK operation, the system returns to load 0 status. IMFs may use enough workspace to prevent the successful execution of the LINK command. This condition will return the system to load 0 status. These IMFs may be cleared by a RESTART, and then you may retry the LINK command. If the LINK is successfully executed, and upon completion of the utility, you must reload any required IMFs when you return to load 0 status. If the error continues to occur, call your service representative. For a complete description of the customer support functions see the *IBM 5110 Customer Support Functions Reference Manual*.

Example

A sample LINK command is shown below:

```
LINK 2, 'ABC', D80
```

Upon execution of this command, the system will link to the program in file 2 (called ABC) on the diskette in drive 1.



LIST COMMAND

The LIST command allows you to display or optionally print the program or data lines from the work area. The contents of the work area are unchanged. The syntax of the LIST command is as shown above, where:

PRINT specifies that the list of lines in the work area is to be printed rather than displayed. If PRINT is not specified, the list will be displayed.

KEYx specifies that the indicated key group should be displayed or printed (x = 0 to 9).

line-num specifies that a group of 14 lines, ending with the indicated line number, is to be displayed. If PRINT is also specified, the entire work area, *starting* with the indicated line number, will be printed. If no line number is entered, the lowest line number in the work area is assumed to be the starting line number. This entry is not valid for a KEYx group.

When the listing is specified to the display, the line number specified in the LIST command appears on line 2 of the display screen. Up to 13 preceding lines (fewer, if less than 14 are defined) are displayed on the lines above line 2. You can use the ↑ and ↓ (Scroll Up and Scroll Down keys) to arrive at a particular line.

When the listing is specified to the printer, the entire work area, starting with the specified line number in the LIST command, is printed. Printing begins at the lowest line number if a line number is not specified in the LIST command.

Notes About LIST

- If the line number specified in the LIST command is not in the work area, the system will seek the next lower line number for the LIST operation. If a lower line number is not found, an error message is displayed.
- If the line length exceeds 64 characters, the succeeding line will contain the excess over 64 characters when the line is displayed. When printed, the full line will be printed up to 128 characters. The line length will not affect the execution of the statement.
- If the system is listing data lines longer than 118 characters on the display, the excess over 118 characters is not displayed.
- You can use the ATTN key to terminate a LIST operation to the printer.

Example

The following examples show several LIST commands:

- To display the first group of work area lines:

  LIST (then press the EXECUTE key)

- To display line number 250 and the preceding 13 lines:

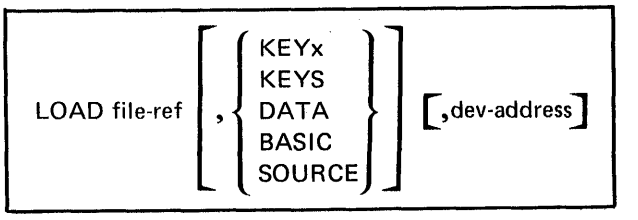
  LIST 250 (then press the EXECUTE key)

- To print line number 250 and all lines following in the work area:

  LIST PRINT, 250 (then press the EXECUTE key)

- To display the first 14 lines in the group associated with function key 5:

  LIST KEY5 (then press the EXECUTE key)



LOAD COMMAND

The LOAD command causes a previously saved file to be loaded into the work area (from tape or diskette) for modification and/or execution. LOAD can also be used to prepare the work area for the entry from the keyboard of a new BASIC program, data, or key group. The syntax of the LOAD command is as shown above, where:

file ref can be either the file name (enclosed in single quotation marks), the file number, or both file number and name, except for tape files which cannot be referenced only by the file name. (See *File Reference Parameter*.) An entry of 0 indicates that a new file will be entered from the keyboard, regardless of device address specified.

KEYx, KEYS, DATA, BASIC, or SOURCE specifies the type of data to be loaded into the work area. BASIC is the default when you are loading from the keyboard. BASIC or SOURCE is the default when you are loading from tape or diskette.

dev-address is the address code of the device containing a saved file to be loaded. (See *Device Address Parameter*.)

Entering a LOAD command of LOAD0 clears the entire work area, including all key groups. A LOAD0 command specifying BASIC or DATA as the second parameter will only clear the specified program from the work area, without altering existing key groups. If the new file type is a KEYx file, the new key file is added to the work area without altering the existing work area. If the KEYx file already exists in storage, the loaded KEYx file will replace the old one.

Line 0 will display READY when the system is in standard BASIC character mode, DATA when in lowercase character mode, and KEYx for key definition mode. You can begin entering the specified type of lines into the work area. The line type entered (BASIC, DATA, or KEYx) must be the same type as the work area, or the line will not be accepted and an error message will appear on line 0. KEYS and SOURCE are not valid options when you are loading from the keyboard (KEYx is valid).

A LOAD nonzero entry specifies that a saved file is to be loaded from tape or diskette into the work area. Any previous contents of the work area (except the keys files) are destroyed when the file type is specified as BASIC, DATA, or SOURCE. If the new file type is a KEYx file, the new key file is added to the work area without altering existing work area. If the KEYS/KEYx files already exist in storage, the loaded KEYS/KEYx files will replace the old ones. For example, if a new KEY6 function is loaded from tape or diskette, it will replace the existing KEY6 function. A copy of the saved file is loaded into the work area, and the following information is displayed:

- User-specified file identification (if any)
- Number of contiguous 1,024-byte areas of storage in the file
- Number of contiguous 1,024-byte areas of unused storage in the file
- First line number (for BASIC)
- Last line number (for BASIC)
- KEYx numbers (for KEYx file types)
- Amount of unassigned work area remaining (in bytes)—this does not include KEYx file types

Notes About LOAD

- File type KEYS (load all keys functions within the specified file) is invalid when the file number is zero (LOAD0).
- File type SOURCE (load a program in user source format from the specified saved file) is invalid when you are loading from the keyboard (LOAD0).
- When you are loading a record file (type 9) or a stream file (type 1, 2, or 3), SOURCE is assumed unless DATA is specified.
- If file type is omitted, BASIC is assumed if you are loading from the keyboard, and BASIC or SOURCE is assumed if you are loading from a saved file (depending on the type of file). In the latter instance, if the type of file is not BASIC or SOURCE, an error message is displayed.
- LOAD0, DATA provides automatic line numbering and the colon for each line. The display shows:

  0010:__

- If execution of a LOAD command is interrupted (when you press the ATTN key, or due to an I/O error) while you are loading a saved file, operation is terminated. If you are loading a program, the program may be incomplete and may not function as intended.
- Calculator statements are invalid if DATA is specified (LOADx, DATA).

Notes About LOAD SOURCE

- When a syntax error occurs during a LOAD SOURCE command, you must choose one of the following options:
  - Correct the line and press the EXECUTE key to continue.
  - Scroll up to ignore the line and continue loading.

- If you load a SOURCE file containing a line longer than 64 characters, you will get an error. The error must be corrected or the line must be scrolled up (which ignores the line) before loading can continue.

- The LOAD SOURCE command is terminated by an error (other than a syntax error) or by any use of a command key whether or not the key has a defined function (see *Function Keys*).

Example

The following examples show a variety of LOAD commands:

- To prepare for keyboard entry of a data file:

  LOAD0, DATA

- To prepare for entry of a function from the keyboard for function key 6:

  LOAD0, KEY6

- To load a saved program (file 3) from tape unit E40 (auxiliary tape unit):

  LOAD3, E40

- To load a program saved in user source format from file 6 on tape unit E80:

  LOAD6, SOURCE, E80

  or

  LOAD6, E80

Keyboard Generated Data Files

You can create a data file directly from the keyboard using the LOAD command. First, enter LOAD0, DATA, then press EXECUTE. The system responds with automatic line numbering (starting with 0010) followed by a colon. You can then enter numeric and character data. The end of a data file line is indicated when you press the EXECUTE key. Data file lines are not checked for proper syntax. A typical data file line is:

```
0010: APRIL, 312.41, 'JONES', 419.21, 'BALANCE'
```

After all data file lines are entered into the work area, they can be saved with the SAVE command. Data file lines are saved without line numbers or the colon. When data file lines are listed, the colon is displayed. Data in a keyboard-generated data file can be accessed by input/output statements during program execution or by a procedure file to control program execution. With line numbers and the colon removed, the data file is a continuous string of data items or records.

You can edit a saved data file by first loading it back into the work area with a LOAD command. When saved, the line numbers and colon are again removed. When loaded, data lines are preceded by line numbers, starting with 0001 and incrementing by 1, and the colon.

Function Keys

Ten function keys are available for your use in invoking programs or commands of your choosing. The function keys are the numeric keys 0-9 to the right of the typewriter-like keyboard. The functions invoked by these keys are defined in a LOAD command. First, enter a LOAD command specifying the key to which a function is to be assigned. For example, to assign a function to the 6 key, enter:

```
LOAD0, KEY6 (then press the EXECUTE key)
```

Now, you must define a key group, which is the statements associated with the function key. When you press the CMD key and the 6 key, these are the statements that will be executed. One of five valid key group header statements must be preceded by a line number of 999x, where x = 0 to 9 to represent the function key. In the preceding example, the line number is 9996. The header statements are:

- NULL, which specifies that you are not defining a function for a particular key or that you are deleting an existing definition.

- CMD, which specifies that the remainder of the line is a system command or a calculator statement. This key group can contain only one system command or calculator statement. The specified operation is performed immediately (without pressing the EXECUTE key).

- REM, which specifies that the key group is a series of BASIC statements. These statements will be executed when the CMD key and the specified key is pressed or when the specific key group is referenced in GOSUB. When referenced in GOSUB (GOSUB 999x), the program seeks the REM key group and branches to it. If the REM key group is not found, the program seeks statement 999x in the program and branches to it. The DATA, END, and STOP statements are not permitted in a function key. Thus READ, MAT READ should not be entered. A RETURN or CHAIN statement will end definition of a KEYx function. The RETURN statement must not contain an expression. A nested GOSUB within a key group will cause an error.

- TXT, which specifies that the character string (enclosed in single quotation marks) is to be inserted in the input line beginning at the current cursor position. For example, assuming a character string 'RATE' is assigned to function key 5, each time function key 5 is pressed while CMD is pressed, the constant RATE is inserted into the line being entered from the keyboard.

- KEYn, which indicates that the function assigned to the n key is to be reassigned to the x key specified in line number 999x. The n function key is set to NULL.

In the following examples, the function keys will be assigned a number of functions. First, the LOAD0, KEY6 command prepares the work area for assignment of the keys function.

- 9996 NULL indicates that the 6 key will have no defined function.

- 9996 CMD REWIND E80 indicates that the 6 key will cause the tape in the 5110 Model 1 built-in tape unit to be rewound each time the CMD key and the 6 key are pressed.

Another use for the function key is shown below:

```
9996 CMD A=&PI*R↑2
```

In this example, you can compute the area of a circle by assigning a value to R (the radius) and pressing the numeric 6 key while the CMD key is pressed.

- 9996 REM KEY6 FUNCTION
  0001 FOR X = 100 to 360 STEP 10
  0002 A = X/12
  0003 I = 2*A
  0004 PRINT FLP, I,X
  0005 NEXT X
  0006 RETURN

  These BASIC statements will be executed each time the numeric 6 key is pressed while the CMD key is pressed, or if a GOSUB 9996 is executed.

- 9996 TXT 'SPINDLES AND SPANGLES' indicates that the character string will be inserted into the current input line beginning at the current cursor position each time the 6 key is pressed while the CMD key is pressed.

- 9996 KEY4 indicates that the function assigned to the 4 key be reassigned to the 6 key. The 4 key is then set to NULL, meaning that it has no defined function.

Note that statements in an REM key group can be deleted by reference to the key group and use of the DEL editing function. For example:

```
KEY6, 10DEL
```

deletes line 10 from key group 6.

```
KEY6, 10DEL90
```

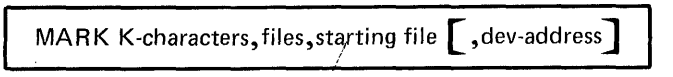
deletes lines 10 through 90 from key group 6.

Statements can be added or edited in the same manner:

```
KEY6, 10 PRINT A
```

adds line 10 or replaces the previous line 10 in REM key group 6.

Note that KEYx cannot be used to edit a key group header statement.



MARK COMMAND

You can use the MARK command to initialize one or more tape or diskette files to a specified number of contiguous 1,024-byte areas of storage. If end of tape is reached before the mark operation is complete, the last file number and the number of 1,024-byte areas successfully marked in the file are displayed. If end of diskette is reached, the number of files successfully marked is displayed. The ATTN key is not active during mark operations. If you try to re-mark a file on tape or re-mark a file that contains data on diskette, an error message is displayed (see Appendix C). To continue, enter GO. Note that you must use the Scroll Up key or blank the input line before entering GO *only* in the first two positions of the line. If you end the operation by entering anything other than GO in positions 1 and 2, the file already marked is unchanged.

Note: If an existing file on tape is re-marked, the original information in the re-marked file and the existing information in the files following the re-marked file cannot be used again. Files on diskette can be re-marked without losing information in files that follow.

The syntax of the MARK command is as shown above, where:

K-characters is the number of 1,024-byte areas of storage to be used on the tape or diskette.

files is the number of consecutive files to be marked.

starting file is the first (lowest numbered) file number to be marked.

dev-address is the address of the tape or diskette drive in which the file resides. Default is the system default device address (see *Device Address Parameter*).

Notes About MARK

- *If an existing tape file is re-marked, the original information on the re-marked file cannot be used again.* In addition, the information on files following a re-marked tape file cannot be used again. This does not apply to diskette files.

- If you are marking more than one file on the diskette, the system will check the starting file and each subsequent file within the range that you specify for a file that contains data. If any of these files contains data, the ALREADY MARKED error message is displayed.

- The MARK command can be issued any time the specified tape unit or diskette drive is not otherwise active.

- Tapes with CRC errors must be re-marked (reinitialized) starting with a file preceding the CRC error. A REWIND command is generally required before this MARK.

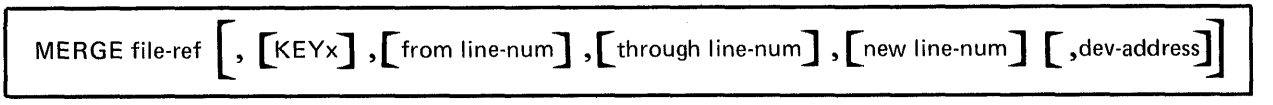
- You can determine the size required for a file by comparing the amount of work area available before and after you have entered data or programs into the work area. See *Storage Considerations* in Chapter 5 for information on storage requirements for data in the work area.

Example

A sample MARK command is as shown:

```
MARK 3,6,1
```

In this sample, six files will be marked, starting with file 1, with each file using 3,072 (3K) bytes on the tape or diskette residing in the tape or diskette drive that is the system default device.



MERGE COMMAND

The MERGE command allows you to merge all or part of a saved file with data or a program (saved in BASIC format) in the work area. In this way, you can add the same routine to several different programs, or add the same data items to several different data files. Only BASIC statements (in a BASIC file) and stream DATA files can be merged. The work area and saved file must be of the same type. If these files are different, the MERGE command is not executed, and an error message is displayed. Lines from the file are added to the work area lines in line number sequence. If a line from the file and a line in the work area have the same statement number, however, the line from the file replaces the work area line. The merged file could exceed the size of the work area, which causes an error message to be displayed (see Appendix C).

As the lines are merged, you can specify that lines merged from the file be renumbered, starting with a statement number of your choice and increasing by the original (SAVED) file increment. After the merge is completed, the display shows the READY message or DATA, along with the number of unused bytes in the work area. The syntax of the MERGE command is as shown above, where:

file-ref can be the file name (enclosed in single quotation marks), the file number, or both file number and name (see *File Reference Parameter*).

KEYx is an active function key group into which the file is to be merged.

from line-num is the first line to be merged in the saved file. If no number is entered, the first line in the file is the default.

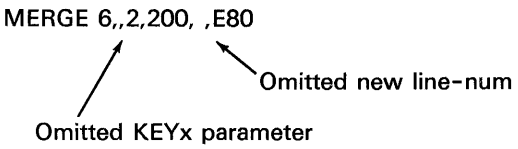
through line-num is the last line to be merged in the saved file. If no number is entered, the last line in the file is the default.

new line-num is the first line number to be used in renumbering the lines from the saved file. If no number is entered, the merged file will not be renumbered.

dev-address is the address of the device in which the saved file resides. The default is the system default device address (see *Device Address Parameter*).

Notes About MERGE

- This command must be entered character-by-character from the keyboard.
- Omitted parameters must be indicated by consecutive commas. For example:



Example

A sample MERGE command is as shown:

```
MERGE 6,,4,200,10,E40
```

In this example, data from file number 6 in the auxiliary tape unit will be merged with data in the work area. Lines 4-200 from the file will be merged. As the MERGE command is executed, lines from the saved file are renumbered, starting with line number 0010.

Another sample MERGE command is:

```
MERGE 5,KEY4,2,500,,D80
```

In this example, lines 2 through 500 from file 5 will be merged with the function key group currently assigned to the 4 key. File number 5 resides on diskette drive 1 (address D80).

With a file name, the command above could be:

```
MERGE 5, 'SYSIN',KEY4,2,500,,D80
```

PROC COMMAND

PROC file-ref [,REC=x][,dev-address]

The PROC command allows you to initiate the use of a procedure file (see *Procedure File* in Chapter 3). A procedure file is a record I/O file on tape or diskette that contains BASIC commands, statements, and/or input data. Data in a procedure file can be used to replace data that is normally entered from the keyboard (in response to an INPUT statement, for example), or to control loading and execution of BASIC programs. The syntax of the PROC command is as shown above, where:

file-ref can be the file name (enclosed in single quotation marks), the file number, or both file number and name (see *File Reference Parameter*).

REC = x specifies the record number to begin the procedure, where x is the record number. If this parameter is not specified, the procedure starts with the first record in the file.

dev-address is the address of the tape unit or diskette drive in which the procedure file resides. The default address is the system default device address (see *Device Address Parameter*).

Upon execution of a PROC command, the file with the specified number or ID (for diskette) on the specified device is accessed for procedure file data. You can create a procedure file just as you create record I/O data files with a program or by using the LOAD0,DATA command. The PROC command implicitly opens the procedure file.

Notes About PROC

- The procedure file will remain open while the procedure is active. Do not access the same file number and device while the procedure is active.
- A procedure file is closed by any error, by a GO END command (in response to an ALERT command) or by a PROC command embedded within the procedure that calls another procedure.
- A procedure file can supply data for an INPUT statement only if the RUN command at the beginning of the program contains the IN=P parameter (see *RUN Command*).
- A PAUSE statement within a program executed from a procedure file will execute the next record of the procedure file.

Example

The following is a sample PROC command.

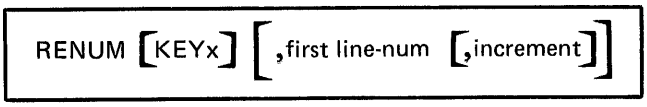
```
PROC 3,'DAILY',D80
```

Upon execution of this command, the procedure file (named DAILY) in file 3 of the diskette in drive 1 will be initiated.

RD=n

RD= COMMAND

Printed and displayed data can be rounded by using the RD= command to specify the number of digits to the right of a decimal point. If more digits are to be printed or displayed, they will be rounded to the number specified. At power on, rounding is initially set to six digits. Rounding can then be modified with the RD= command or in the RUN or GO commands. Variables and computational results are not rounded by this command. The range of rounding (n) is 1 to 15 digits. Comparison tolerance is changed by the use of the RD= command. (See *Comparison Tolerance* in Chapter 3 for more information.) Data printed with a PRINT USING statement is not affected by the RD= command.



RENUM COMMAND

You can use the RENUM (renumber) command to generate new statement numbers for all the BASIC statements or data in the work area. Like the AUTO command, renumbering begins with 0010 and the increment is 10, unless specified otherwise. In addition, all references to statement numbers such as in GOTO, IF, PRINT USING, GOSUB, and GET are changed to the new numbers. In function key groups (see *LOAD Command*), each key group is renumbered as if it is a separate work area, and is only renumbered if it is a BASIC program function. (RENUM does not alter the key group header record.)

The syntax of the RENUM command is as shown above, where:

KEYx specifies a key group (x = 0 to 9) to be renumbered.

first line-num is an integer (1-9989) identifying the number at which renumbering will begin. If this number is not specified, a beginning number of 0010 and an increment of 10 are the default values.

increment is an integer specifying the increment for succeeding statement numbers. Default is 10.

Notes About RENUM

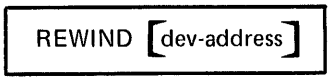
- Statement numbers 9990-9999 are not altered by a RENUM command.
- An error will occur if you try to renumber where a line number greater than 9989 will be generated.

Example

The following examples shows the execution of a RENUM command:

RENUM 20,10 (then press the EXECUTE key)

Before	**After**
0010 INPUT A, B	0020 INPUT A, B
0011 Q = INT (A/B)	0030 Q = INT (A/B)
0020 IFQ>0 THEN 30	0040 IFQ>0 GOTO 0060
0025 GOTO 10	0050 GOTO 0020
0030 PRINT Q	0060 PRINT Q
0035 GOTO 10	0070 GOTO 0020
0040 STOP	0080 STOP



REWIND COMMAND

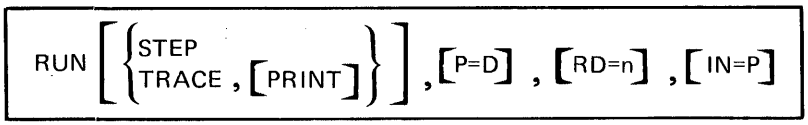
The REWIND command allows you to rewind the specified tape unit. The syntax of the REWIND command is as shown above, where:

dev-address is the address of the tape unit to be rewound (E80 for the built-in tape unit or E40 for the auxiliary tape unit). The default is the system default device address (see *Device Address Parameter*).

Example

A sample REWIND command is as shown:

```
REWIND E40
```



RUN COMMAND

The RUN command starts execution of a BASIC program at the lowest numbered executable statement. The program must already reside in the work area, and the work area must be defined as containing a BASIC program (see *LOAD Command* for loading programs and defining the work area type). BASIC programs can be run (executed) in three modes:

Normal Mode – All program steps are executed without interruption.

Step Mode – The system stops immediately before executing *each* program step (statement). The word STEP and the statement number of the next statement to be executed are displayed. To execute the next program statement, or to change execution mode, you must execute a GO command (see *GO Command*). Step mode allows you to display or alter variable values between steps for debug purposes.

Trace Mode – The statement number of each statement executed is displayed and/or printed while the statement is executed.

Both step and trace modes are useful in locating programming errors. Trace provides a more rapid view of the program steps as they are executed. Step, on the other hand, allows you to examine the contents of variables between program steps and to modify program steps or data.

The syntax of the RUN command is as shown above, where:

STEP specifies step-by-step execution.

TRACE specifies continuous statement execution, but the line number of the statement just executed is displayed and/or printed. Note that for *Normal* mode you do not specify STEP or TRACE.

PRINT specifies that trace messages are printed *and* displayed (only valid with TRACE). Also see *File FLS* in Chapter 3 for a discussion of dynamic trace to the printer.

P = D specifies that output from programs specifying the printer is directed instead to the display screen.

RD = n allows you to specify the number of digits (n) to the right of the decimal point that will cause rounding on printed output. The value n can be 1 to 15 and is initialized to 6 (see *Comparison Tolerance* in Chapter 3).

IN = P is valid only on a RUN command within a procedure file (see *Procedure File* in Chapter 3). This parameter allows you to specify that data for INPUT statements be supplied from the procedure file, rather than from the keyboard.

Notes About RUN

- The RUN command will be rejected by the system if the work area file type is DATA.
- The RUN command initializes all arithmetic variables and arrays to zeros, and character variables and arrays to blanks.
- When P=D is specified, all PRINT FLP, PRINT USING FLP, MAT PRINT FLP, and MAT PRINT USING FLP statements are interpreted as PRINT, PRINT USING, MAT PRINT, and MAT PRINT USING statements, respectively, during this run.
- The IN=P option is valid only when a procedure file is active.

Example

Some sample RUN commands are as shown:

1. To begin normal execution of a program:

   RUN (then press the EXECUTE key)

2. To begin a trace operation of a program and print the traced steps executed:

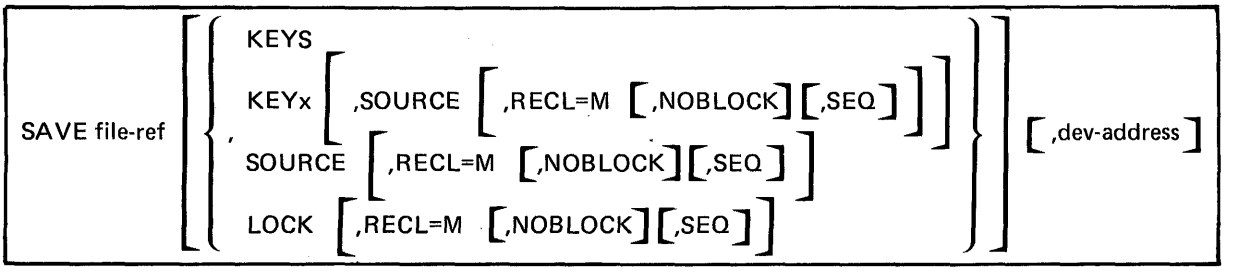
   RUN TRACE, PRINT (then press the EXECUTE key)

3. To begin normal execution of a program where the output as well as trace is displayed instead of printed:

   RUN TRACE, P=D

4. To begin execution of a program from a procedure with input data to be furnished by the procedure:

   RUN IN=P



SAVE COMMAND

The SAVE command allows you to save the contents of the work area in a specified file. After a SAVE command is completed, the display will show the READY or DATA message and the following information about the file:

- The number of contiguous 1,024-byte areas of storage allocated in the file.
- The number of contiguous 1,024-byte areas of storage unused in the file.

If the file comes to end of file before all of the work area is saved, an error message is displayed (see Appendix C) and only part of the work area is saved. To save the entire work area, enter another SAVE command specifying another file with the correct amount of available space.

The syntax of the SAVE command is as shown above, where:

file-ref can be the file name (enclosed in single quotation marks), the file number, or both the file number and file name (see *File Reference Parameter*). File name is required for saving diskette files.

KEYS specifies that the contents of *all* defined function keys are to be saved.

KEYx specifies that a function key (x = 0-9) is to be saved (see *Function Keys*).

KEYx,SOURCE specifies that the function key (x = 0-9) is to be saved as a BASIC program (without the function key header) in the same format in which the user entered it.

SOURCE specifies that the contents of the work area are to be saved in the same character format in which the user entered it.

LOCK specifies that the BASIC program being saved is to be locked. You cannot list, renumber, or edit a locked program. You can, however, perform the following with a locked program:

- Load for execution
- Save on tape or diskette (BASIC format only)
- Merge with lines in storage (the merged program is then considered to be locked)
- Run the program in normal mode (*not* in step or trace mode)

RECL = m specifies that the file to be saved is a record I/O file, and indicates the size (m) of the logical records in the file. Logical record size (m) must be a positive, nonzero integer.

NOBLOCK specifies that the file is to be saved on diskette in the unblocked format of one logical record per sector.

SEQ indicates that records can be sequentially relocated for I/O error recovery. If SEQ is specified, any attempt to access the file randomly with REC= or KEY= will cause an error.

dev-address is the address of the tape unit or diskette drive in which the specified file resides. The default is the system dafault device address (see *Device Address Parameter*).

Notes About SAVE

- If KEYx, KEYS, or SOURCE is not entered in the SAVE command, data in the work area will be stored for either a BASIC program or a DATA file, depending on the definition of the work area. If you are saving a data file, you can also specify the REC=m, NOBLOCK, and SEQ parameters.

- If you interrupt execution of a SAVE command by pressing the ATTN key, the operation is terminated. If the interruption occurs before the file has been changed, the file remains unchanged. The interrupt at any other time will result in only part of the program or data being saved.

- An attempt to SAVE a program that has been interrupted with ATTN may cause unpredictable results if the program is resumed from where it was interrupted.

- Only specified programs, data, and key functions are saved with the SAVE command. Data values, buffers, and IMFs are not saved. You must apply IMFs with the LINK command before you load the work area with any saved program requiring IMFs.

- A SAVE command to a used file must contain the same file name as the name of the used file.

Example

A sample SAVE command is as shown:

```
SAVE 4,'KEY 8',KEY 8,E80
```

In this example, the contents of function key 8 are saved in file 4, which resides on the cartridge in the tape unit built into the 5110 Model 1.

SKIP integer [comment]

SKIP COMMAND

The SKIP command allows you to unconditionally skip a specified number of records within a procedure file (see *Procedure File* in Chapter 3). The SKIP command is valid only when used within an active procedure file. The syntax of the SKIP command is as shown above, where:

integer indicates the number of procedure file records to be skipped.

comment is an optional comment.

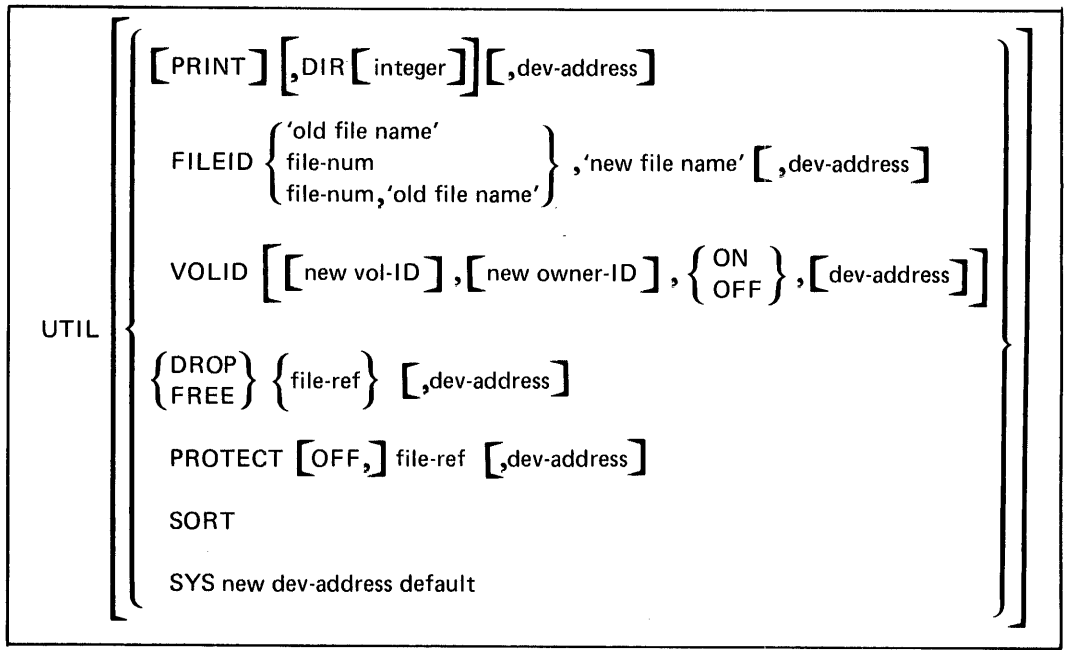
Upon execution of the SKIP command, the specified number of records is unconditionally skipped within the procedure file. If the number of records to be skipped exceeds the number of records remaining in the file, the procedure file is closed. The specified number must be a whole number. For example, an entry of 3.5 causes three records to be skipped, with the .5 assumed to be the beginning of the optional comment.

Example

The following example shows the execution of a SKIP command within a procedure file.

```
LOAD 4
RUN
CSKIP 3
LOAD 9
RUN
SKIP 3
ALERT INSERT TRANSACTION DISKETTE
LOAD 11
RUN
```

Upon execution of this procedure file, the program in file 4 runs. If the program in file 4 sets the return code to nonzero (see *STOP* in Chapter 4), the CSKIP 3 causes the next three records in the procedure file to be bypassed and ALERT to be processed. The program in file 11 is then run. If the return code was zero, the program in file 9 is loaded and run instead. The SKIP then causes the remainder of the procedure file to be bypassed.



UTIL COMMAND

You can use the UTIL command to do the following:

- Display or print a directory of file information currently on tape or diskette.
- Rename a file on diskette.
- Change or display a diskette volume ID and/or owner ID.
- Eliminate a file from diskette and make that file space available for allocation for other files, or discontinue the contents of a file on diskette or tape while leaving the file space allocated.
- Write protect a file or remove write protection from a file (diskette only).
- Transfer system control to the optional diskette sort feature.
- Change the system default device address.

Listing a File Directory

You need approximately 550 bytes of space in the work area to list a directory of file information. If sufficient work area is not available during execution of a UTIL command, an error message (see Appendix C) is displayed. You should save, then clear, the work area and continue with another UTIL command. Information about each file is printed or displayed on one line per file.

If you specify the PRINT parameter; all file information is printed. You can interrupt the listing to either the display or printer by pressing the ATTN key, which will terminate the command, or by pressing the HOLD key once, then pressing the HOLD key again to continue.

The following information is printed or displayed about each file:

- The file number.
- The identification you assigned to the file, if any (see *SAVE Command*).
- File type (see *File Types*).
- The number of contiguous 1,024-byte areas of storage allocated to the file.
- The number of contiguous 1,024-byte areas of storage unused in the file.
- A number (0-9) indicating the number of defective physical record areas in the file, or an asterisk indicating that the number is greater than 9. This value can indicate when you should relocate a file into another file (tape only) to avoid loss of data due to defective areas on the tape.
- File protection indicator (diskette only).
- Data set starting location (diskette only).
- Key numbers saved in the file for KEYS or KEY (0-9) files.

The UTIL command to print or display a directory of file information has the following syntax:

UTIL [PRINT] [,DIR [integer]] [,dev-address]

where:

PRINT specifies that the listing be printed. If the printer is not specified, output is displayed.

DIR integer specifies that a directory be listed. The integer, which identifies the starting file number for the listing, is optional. If the integer is not entered, the listing will begin at the current physical location of the tape or the beginning of the diskette.

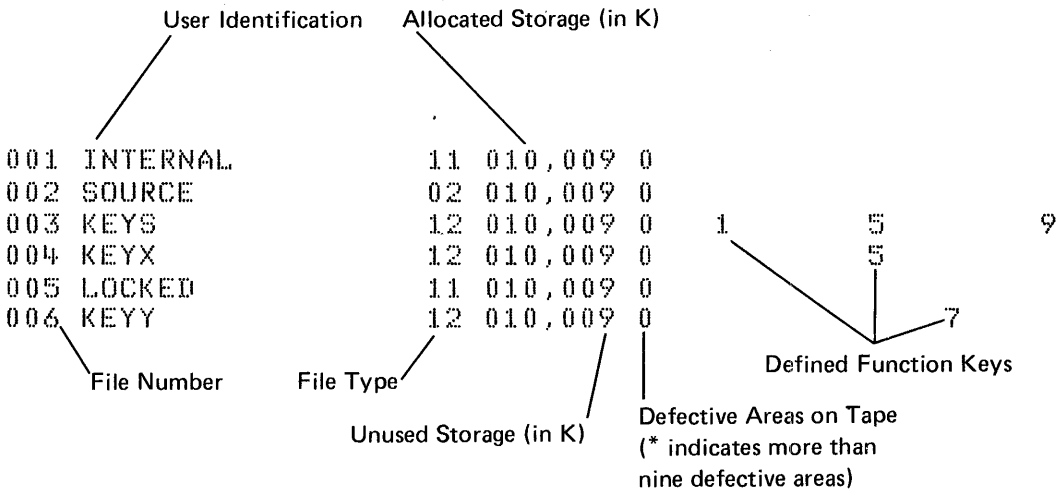
dev-address specifies the address of the tape or diskette drive from which the listing is to be made. The default is the system default device address (see *Device Address Parameter*).

Example

The following example shows a UTIL command, which specifies that a directory be printed, starting with file number 1 on the 5110 Model 1 built-in tape unit.

```
UTIL DIR1,E80
```

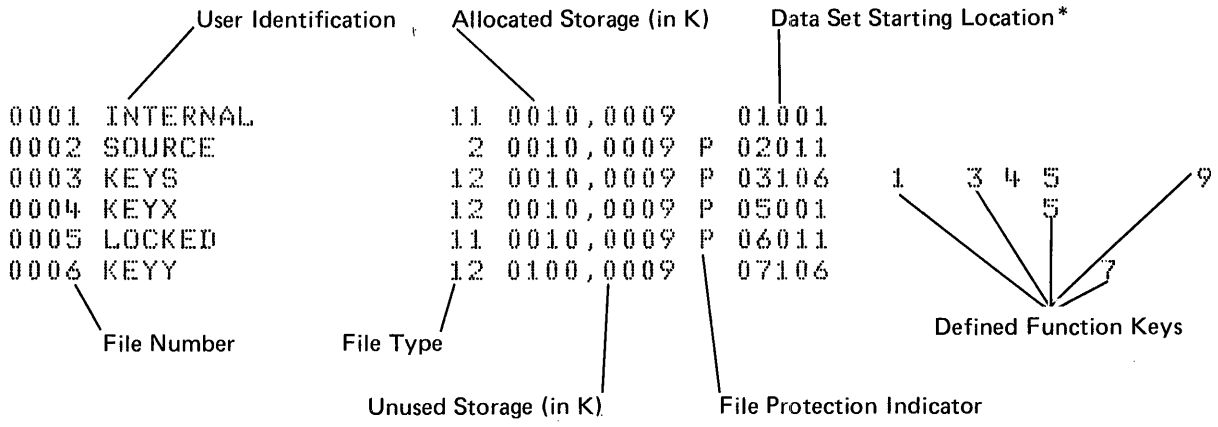
The resulting printed output for tape is shown in the following example:



For a diskette file directory, a UTIL statement is:

```
UTIL DIR1,D80
```

The resulting output for diskette is shown in the following example:



*Data set starting location is five digits (cchrr), where cc=cylinder, h=head, and rr=record number (see *IBM 5110 BASIC User's Guide*).

File Types

Valid file types listed with the UTIL command are:

Type	File
0	Marked, unused tape file or diskette file
1**	Data exchange file
2	General exchange file
3**	BASIC source file
4*	BASIC work area file
5*	BASIC KEYS file
6*	APL continue file
7	APL save file
8	APL internal data
9	Record I/O file
B9	Basic exchange, record I/O file
10	APL internal data (diskette only)
11	BASIC work area file
12	BASIC KEYS file
15	APL mixed record file (diskette only)
16*	Patch, tape recovery, and tape copy file
17*	Diagnostic file
18*	Communication file
19*	IMF file
21	Utility file
22	Feature file
23	IMF file
24	Diagnostic file
26	APL continue file (IBM 5110)

*These file types can be created on an IBM 5100, but cannot be created or loaded by an IBM 5110 (see *5110 BASIC Compatibility with IBM 5100 BASIC* in Chapter 5).

**These file types can be created on a IBM 5100 and can be read or loaded by a 5110, but they cannot be created by a 5110.

Renaming a File on Diskette

You can rename a file on diskette using the UTIL command with the following syntax:

UTIL FILEID { 'old file name' / file-num / file-num,'old file name' } ;new file name' [,dev address]

where:

FILEID specifies that a diskette file name be changed.

'old file name' indicates the current name assigned to the file.

file-num indicates the number of the file to be renamed. Note that either or both the old file name and file number may be specified. If both are specified, they must be separated by a comma with file number first. File name must always be enclosed in single quotation marks (up to 17 characters).

'new file name' specifies the new name to be assigned to the file.

dev address is the address of the diskette drive containing the file to be renamed. Default device address is the system default device address (see *Device Address Parameter*).

A sample UTIL command to rename a file is shown below.

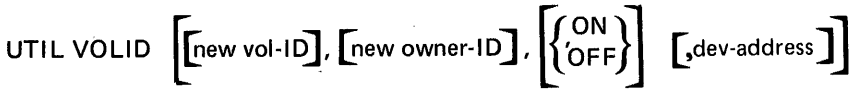
```
UTIL FILEID 7,'YTD.GROSS.SALES','YR76.GROSS.SALES',D80
```

File number 7 on diskette drive 1 will be renamed from YTD.GROSS.SALES to YR76.GROSS.SALES in this example.

Changing a Diskette Volume ID

You can change or display a diskette volume ID or owner ID with the UTIL command having the following syntax.

To change:



To display:

```
UTIL VOLID,,,D40
```

where:

VOLID specifies that a diskette volume ID be changed or displayed. If this is the only parameter entered for the UTIL command, the volume ID, owner ID, and sector size fields for the diskette are displayed only, and are unchanged.

new vol-ID specifies the new volume ID for the diskette. This entry can be from 1 to 6 alphameric characters. If you do not enter a new vol-ID, the volume ID remains unchanged.

new owner-ID specifies the new owner ID for the diskette. This entry can be up to 14 alphameric characters. If this parameter is not entered, the current owner ID is unchanged.

ON specifies that the volume be protected from unauthorized access. This parameter does not allow access to the diskette.

OFF specifies that the protection indicator for the volume be turned off, making the volume accessible. If this parameter is entered, both the current volume ID and owner ID must also be specified and must match those on the diskette.

dev-address specifies the address of the diskette drive on which the file resides. Default is the system device address (see *Device Address Parameter*). This address must not be enclosed in single quotation marks.

Note: Omitted parameters between other parameters must be indicated by consecutive commas. For example:

```
UTIL VOLID,,,D40
```

will display the VOLID for the diskette mounted on drive 2.

When the UTIL VOLID command is executed, the current volume ID, owner ID, and sector size fields are displayed, and the new volume ID and owner ID entries are then assigned.

A sample UTIL VOLID command is shown below:

```
UTIL VOLID DEBTS,ROWE,ON,D80
```

In this example, the diskette volume ID on the diskette in drive 1 (D80) will be changed to DEBTS, and the owner ID will be changed to ROWE. In addition, the volume protection indicator will be set on. With the indicator on, the diskette cannot be used (including the UTIL VOLID command) until the indicator is set off. Thus, before setting protection on, be sure you remember the volume and owner ID, which you must enter before protection can be set off.

Eliminating or Discontinuing a File

You can use the UTIL command to remove a file and make its physical file space available for reallocation, or simply reinitialize a file to unused status while leaving its file space allocated. The syntax of this UTIL command is:

```
UTIL {DROP} {file-num                } [,dev-address]
     {FREE} {'file name'             }
            {file-num,'file name'    }
```

where:

DROP specifies that the file be reinitialized to unused status, although the file space allocated for the file remains allocated. This parameter can be used to reinitialize both tape and diskette files.

FREE specifies that the file space allocated for the file is to be freed and can be used for allocation to another file. This parameter applies only to diskette files.

file-num specifies the number of the file to be deleted. This parameter is required for tape files; it is optional for diskette files.

'file name' specifies the name of the file to be deleted. This parameter can be up to 17 characters enclosed in single quotation marks. For diskette files, this parameter must be specified if a file number is not specified.

file-num,'file name' indicate that both the file number and file name can be specified, but must be separated by a comma.

dev-address specifies the address of the tape unit or diskette drive on which the file to be deleted currently resides. Default is the system device address (see *Device Address Parameter*).

A sample UTIL command to delete a diskette file is:

```
UTIL DROP 'YR76.TAXES',D80
```

In this example, the data in the YR76.TAXES file will be deleted from diskette drive 1. The file space will still be allocated, allowing another file (YR77.TAXES, for example) to be assigned to that space.

Assigning or Removing File Write Protection for Diskette Files

You can use the UTIL command to ensure the integrity of data in a diskette file by write-protecting the file. This type of UTIL command has the following syntax:

```
UTIL PROTECT [OFF,] file-ref [,dev-address]
```

where:

PROTECT specifies that a file be selected for assignment or removal of write protection.

OFF specifies that write protection be removed from a file that is currently write-protected. If this parameter is omitted, ON is assumed.

file-ref can be the file name (enclosed in single quotation marks), the file number, or both file number and name (see *File Reference Parameter*).

dev-address specifies the address of the diskette drive on which the file to be protected currently resides. Default is the system device address (see *Device Address Parameter*).

Note: Write protection prevents rewriting of a file. The file can, however, be updated (see *[MAT] REWRITE FILE* in Chapter 4).

A sample UTIL command to assign write protection is shown below:

```
UTIL PROTECT 'LOSSES', D80
```

In this example, the LOSSES file on diskette drive 1 will receive write protection, which ensures that other data cannot be written into the file.

Selecting the Diskette Sort Feature

You can gain access to the diskette sort feature using the UTIL command with the following syntax:

```
UTIL SORT
```

If the UTIL SORT command is specified in a procedure file, the next record is passed to the sort feature as if it were entered from the keyboard. After the sort is completed, the procedure file is still active, default device address remains the same, decimal rounding is reset to six positions, and all programs/data are cleared from the work area.

For details concerning the diskette sort feature, see the *IBM 5110 Customer Support Functions Reference Manual,* SA21-9311.

Changing the System Default Device Address

Normal system default device address for the 5110 Model 1 is the built-in tape unit (address E80). For the 5110 Model 2, normal default device address is diskette drive 1 (address D80). You can change the default device address using the UTIL command with the following syntax:

```
UTIL SYS new dev-address default
```

where:

SYS specifies that the system device address default be changed.

new dev-address default specifies the new default device address. Valid addresses are:

E80 – Primary tape unit (built into the 5110 Model 1)
E40 – Auxiliary tape unit (model 1 only)
D80 – Diskette drive 1
D40 – Diskette drive 2
D20 – Diskette drive 3
D10 – Diskette drive 4

The 5110 reverts to the normal default address (E80 or D80) after a LINK command.

Chapter 3. Data Constants, Variables, And Concepts

BASIC CHARACTER SET

The BASIC character set is used to represent arithmetic and character data entered from the keyboard as data constants and variables.

The characters that have syntactical meaning in the BASIC language fall into three categories: alphabetic, numeric, and special characters.

Alphabetic Characters

The alphabetic characters in BASIC are the upper/lowercase letters of the English alphabet (A-Z) and the following three characters called alphabet extenders:

@ (the commercial *at* sign)
# (the number or pound sign)
$ (the currency symbol)

Numeric Characters

The numeric characters in BASIC are the digits 0 through 9.

Special Characters

There are 22 special characters in BASIC:

Character	Name
	Blank
=	Equal sign or assignment symbol
+	Plus sign
-	Minus sign
*	Asterisk or multiplication symbol
/	Slash or division symbol
↑	Up arrow or exponentiation symbol
(	Left parenthesis
)	Right parenthesis
,	Comma
.	Period or decimal point
'	Single quotation mark
;	Semicolon
:	Colon
&	Ampersand
?	Question mark
>	Greater than symbol
<	Less than symbol
≠	Not equal symbol
≤	Less than or equal symbol
≥	Greater than or equal symbol
|	OR sign or vertical bar

Use of Blanks

Blanks may be used freely throughout a program to improve readability. They have no syntactical meaning except within character constants and in the image statement (see *[MAT] PRINT USING and Image/FORM* in Chapter 4), which specifies the format of printed or displayed data.

Underscore

The underscore character (uppercase F) is a valid overstrike character in the system.

Additional underscore capability is provided through a special keyboard operation. When you hold down the CMD key and press the spacebar, a symbol (■) will be displayed on the screen. This character can also be created if you use the hexadecimal constant FF.

Note: Do not use the ■ character in a DATA, OPEN, FORM, Image, PAUSE, RESTORE, STOP, END, FNED, or REM statement because these are not checked for syntax and unpredictable results may occur. It may be used only as a data constant in a variable within a PRINT statement that is to be printed. The ■ character can only be displayed; it cannot be printed. When entered in a line of data to be printed, this character causes all characters to its right to be underscored until another ■ character is encountered. When the data is printed, a blank will appear in place of the ■ characters. Spaces between characters are not underscored. An example of this function is shown below.

If you enter 'THE MANUAL IS THE ■BASIC INTRODUCTION■, SA21-9306'; this is printed:

```
THE MANUAL IS THE BASIC INTRODUCTION , SA21-9306
```

ARITHMETIC DATA

Arithmetic data is data with a numeric value. All numbers in BASIC are expressed to the base 10; that is, they are treated as decimal numbers.

Magnitude

The magnitude of a number is its absolute value. The range of numbers permitted in BASIC programs is greater than or equal to 1E-78 and less than 1E+74.

Precision

In BASIC, the precision of a number is the maximum number of digits it can contain. When you enter RUN, each numeric data item will have 15-digit precision, occupying 8 character positions of storage. All numbers in BASIC are converted to internal binary floating point format for processing. Because a decimal fraction may not have an exact binary equivalent, some truncation errors may occur after the conversion.

Arithmetic Data Formats

Arithmetic data can be entered, displayed, or printed in any of three formats: integer, fixed point, or floating point. The appropriate format for a given number depends on its magnitude and the level of arithmetic precision you require.

Numbers in any format can be positive or negative. Negative numbers must be preceded by a minus sign. When no sign is specified, a number is treated as a positive number, so plus signs are optional.

Integer Format

Numbers expressed in integer format (I-format) are written as a number of digits optionally preceded by a sign. Examples of numbers in integer format are:

```
 0
+2
-23
 2683
```

Fixed-Point Format

Numbers expressed in fixed-point format (F-format) are written as a number of optional digits preceded by an optional sign and followed by a decimal point. The decimal point can also be followed by a number of digits. These digits are required if a number does not precede the decimal point. Examples of numbers in fixed-point format are:

```
33.
33.00
 -.3
+3.56
```

Floating-Point Format

Numbers expressed in floating-point format (E-format) are written with an optional sign, followed by an integer or fixed-point number, followed by the letter E. An optionally signed one- or two-digit characteristic (exponent) must follow the E.

The value of a floating-point number is equal to the number to the left of the E, multiplied by 10 to the power represented by the number to the right of the E. This notation corresponds to standard scientific notation in which numbers are expressed as a power of 10. Note, however, that while the number 10^7 is permissible in scientific notation, the number E7 is not a valid floating-point number. The value 10^7 must be expressed as 1E7 in BASIC floating-point format. Thus, BASIC floating-point format requires a number to the left of the E.

Examples of numbers in floating-point format are:

Floating Point Number	**Equivalent Decimal Value**
.25E-4	.000025
+1.0E+5	100000
5E-7	.0000005
-15.33E6	-15330000

Selecting An Arithmetic Format

You can enter arithmetic values at the keyboard in the most convenient format for your application. The number one million, for example, can be entered in any of the following ways:

```
1000000
1000000.00
1E+6
```

The numeric size of arithmetic values is limited only by the magnitude (≥1E-78 and <1E+74). Note, however, that the physical length of values you enter is not limited, although entries exceeding 15 digits will be truncated on the right. Thus, very small and very large numbers can be entered in E-format. For example, you can enter:

```
1.4E12
```

or the equivalent I- or F-format value (14 followed by 11 zeros).

You can use the BASIC statements to control the format of arithmetic values displayed or printed (see *PRINT USING* and *Image/FORM* in Chapter 4). Thus, the form of the values originally entered does not affect the output format.

Arithmetic Constants

An arithmetic constant is either an integer, a fixed-point, or a floating-point number whose value is never altered during execution of the program. Thus, the integer 1 is a constant in the following statement:

```
X = X + 1
```

Internal Constants

An internal constant is an arithmetic constant with a predefined value. Unlike normal arithmetic constants, the internal constants are referred to by names, though like normal arithmetic constants, their values are never altered during program execution. The internal constants are:

Constant	Name	Value
pi π	&PI	3.141592653589793
Natural log	&E	2.718281828459045
Square root of 2	&SQR2	1.414213562373095
Centimeters per inch	&INCM	2.540000000000000
Kilograms per pound	&LBKG	0.453592370000000
Liters per gallon	&GALI	3.785411784000000

The internal constant names can only be used as parts of arithmetic expressions; for example:

```
2*&PI (then press the EXECUTE key)
```

The result is 6.283185.

Internal Variables

The 5110 provides three internal variables to aid in error recovery:

&LINE Contains the line number of the BASIC statement being executed when an error occurred

&ERR Contains the number of the error for which an ONERROR statement has caused a branch of program control

&REC Contains the relative record number of the last record referenced in a file

To access the contents of these internal variables, simply enter the variable as shown above and press the EXECUTE key or assign it to another variable.

Arithmetic Variables

A variable is a named data item whose value is subject to change during execution of the program. Arithmetic variables are named by a single letter of the extended alphabet (A-Z, @, #, and $), or by a single letter of the extended alphabet followed by a single digit (0-9). Examples of arithmetic variable names are A, A5, #1, and #7.

When a program is executed, the initial value of arithmetic variables is set to zero. The only exception is that variables assigned in a USE statement are not initialized (see *USE* in Chapter 4) for a program that is chained to (see *CHAIN* in Chapter 4).

CHARACTER DATA

Character data in BASIC is data without a numeric value. Like arithmetic data, character data can be in the form of constants or variables.

Character Constants

A character constant is a string of characters enclosed in a pair of single quotation marks. Any letter, digit, or special character can be included in a character constant. An apostrophe, however, must be indicated by two single quotation marks. For example, 'IT''S' represents IT'S. The following are all valid character constants:

```
'YES'
'THE SQUARE OF X IS'
'12345'
'AB'
```

The length of a character constant, when displayed or printed, is the number of characters it contains, including blanks, but excluding the *delimiting* quotation marks. Each pair of single quotation marks used to represent an apostrophe is counted as one character. The maximum number of characters in a character constant is limited only by the maximum number of characters on an input line, which is 64.

Character Variables

A character variable is a named item of character data whose value is subject to change during execution of the program. Character variables are named by a single letter of the extended alphabet (A-Z, @, #, and $), followed by the currency symbol ($). Examples of character variables are A$, #$, and $$. Character variables can be dimensioned (see *DIM* in Chapter 4) to a length of 1 to 255 characters.

When the program is executed, the initial value of character variables is set to blank characters. The only exception is that variables assigned to the common storage area are not initialized (see *USE* in Chapter 4) for a program that is chained to (see *CHAIN* in Chapter 4). These variables are initialized to blanks by a LOAD or RUN command (see Chapter 2).

Character constants assigned to character variables are adjusted to the length of the character variable. Longer constants are truncated; shorter constants are left-justified and padded with blanks on the right.

ARRAYS

An array is a collection of data items (elements) that is referred to by a single name. Only data items of the same type (numeric or character) can be grouped together to form an array.

Arrays can be either one- or two-dimensional. A one-dimensional array can be thought of as a row of successive data items. A two-dimensional array can be thought of as a matrix of rows and columns. Figure 6 shows a schematic representation of both types of arrays.

One-Dimensional Array Named A

A(1)	A(2)	A(3)	A(4)

Two-Dimensional Array Named B

B(1,1)	B(1,2)	B(1,3)
B(2,1)	B(2,2)	B(2,3)
B(3,1)	B(3,2)	B(3,3)
B(4,1)	B(4,2)	B(4,3)

Figure 6. Schematic Representation of One- and Two-Dimensional Arrays

Each element in an array is referred to by the name of the array followed by a subscript in parentheses, which indicates the position of the element within the array. The general form for referring to an array element is:

array name (rows, columns)

where *array name* is the name of the entire array, and *rows*, *columns* are any positive arithmetic expressions whose truncated integer values are greater than zero and less than or equal to the corresponding dimension of the array.

The expression in a subscript referring to an element of a one-dimensional array gives the position of the element in the row, counting from left to right. Thus, the third element of a one-dimensional array named A can be referred to by the symbol A(3), as in this example:

```
A(3) = 25
```

The first expression in a subscript referring to an element of a two-dimensional array gives the number of the row containing the referenced element. Rows are numbered from top to bottom. The second expression in the subscript gives the number of the column. Columns are numbered from left to right. Thus, the second element in the fourth row of a two-dimensional array named B can be referred to by the symbol B(4,2), as in this example:

```
B(4,2) = 1.53E6
```

The dimensions of an array and the number of elements in each dimension are established when the array is declared.

Declaring Arrays

Arrays can be declared either explicitly by use of the USE or DIM statement or implicitly by a reference to an element of an array that has not been explicitly declared.

When an array is declared explicitly, the dimensions and the maximum number of data items that can be contained in each dimension are specified in the USE or DIM statement.

When an array is declared implicitly, by a reference to one of its elements when its name has not appeared in a prior USE or DIM statement, it will have the number of dimensions specified in the reference, and each dimension will contain 10 elements. For example, when no prior USE or DIM statement exists for an array named A, the statement:

```
A(3) = 50
```

will establish a one-dimensional array containing 10 elements, the third of which will have the integer value 50. The remaining elements will be initialized to zero.

Likewise, when no prior USE or DIM statement exists for an array named B, the statement:

```
B(5,6) = 6.913
```

will establish a two-dimensional array containing 10 rows and 10 columns (100 elements), with the sixth element in the fifth row equal to 6.913.

Arrays with dimensions that contain more than 10 elements cannot be implicitly declared. Thus, without the appropriate prior USE or DIM statement, the following statements would both cause error conditions:

```
A(15) = 22.4
B(3,20) = 66.6
```

After an array has been declared, either explicitly or implicitly, it cannot be explicitly dimensioned by a DIM statement anywhere in the program.

Redimensioning Arrays

Numeric and character arrays can be redimensioned according to the following rules:

- Both one- and two-dimensional arrays can be redimensioned.
- The total number of elements in an array after redimensioning must not exceed the number originally specified when the array was declared.
- The number of dimensions can be changed.
- The maximum value for a dimension is 9999.
- An array can be redimensioned in a MAT assignment statement or MAT input statement.
- The new dimensions for the array can be specified with either a constant or by an expression.

Arithmetic Arrays

An arithmetic array contains only numeric data and can have one or two dimensions.

Arithmetic arrays are named by a single letter of the extended alphabet (A-Z, @, #, and $). Thus, the letter A can be used to name an arithmetic variable or an arithmetic array, or both, while the symbol A2 can only be used to name an arithmetic variable. For example:

```
A = 6
A(1) = 9
```

where the variable A = 6 (scalar) and A(1) = 9 (arithmetic array element). All elements of an arithmetic array are initially set to zero when the program is executed (except those assigned to the common storage area for use in a program that is chained to; see *USE* and *CHAIN* in Chapter 4).

Before being used in any of the matrix-handling statements, an arithmetic array must have been previously dimensioned, either explicitly or implicitly. Arithmetic arrays can be redimensioned as described previously.

Character Arrays

A character array contains only character data and can have one or two dimensions.

Character arrays, like simple character variables, are named by a single letter of the extended alphabet followed by the currency symbol ($). Thus, the name D$ can refer to either a simple character variable or a character array. The name D$(2,4) refers to the fourth element of the second row of a two-dimension character array. For example:

```
D$ = 'JONES'
D$(2,4) = 'SMITH'
```

Character arrays can be used in input, output, and simple matrix assignment statements (when no arithmetic operation is performed) and can be redimensioned as described previously.

Summary of Naming Conventions

Figure 7 shows a summary of the naming conventions previously described for variables and arrays. The symbol *ext* in Figure 7 denotes a letter of the BASIC extended alphabet (A-Z, @, #, and $). Information in brackets is optional.

Data Type	Name	Examples
Arithmetic variables	ext [digit]	A,$5
Arithmetic arrays	ext	A,$,#
Character variables	ext $	A$, $$, @$
Character arrays	ext $	A$, $$, @$

Figure 7. Naming Conventions for Variables and Arrays

SYSTEM FUNCTIONS

The IBM 5110 BASIC language includes system functions that perform a number of commonly used operations. In addition, you can define and name your own functions by using the DEF statement (see *DEF, RETURN, FNEND* in Chapter 4).

The system functions shown in Figure 8 can be used anywhere in a BASIC expression where constants, variables, or arithmetic array element references can be used, as shown in these examples:

```
A = SIN(&PI) + 1
B = SQR(X + 3)
R1 = RND
```

Most of the system functions have a single argument (optional with RND), which can be a valid expression (explained later in this chapter) and produce a single result. An invalid argument produces an error. The argument for the DET function must be a reference to a square arithmetic array. Minimum precision for these system functions is 10 digits. The system functions are listed in Figure 8.

Function Name	Description
ABS(x)	Absolute value of x
ACS(x)	Arc cosine (in radians) of x
ASN(x)	Arc sine (in radians) of x
ATN(x)	Arc tangent (in radians) of x
CEN(x)	Degrees Centigrade (celsius) corresponding to X degrees Fahrenheit
CHR(x)	Character string value of arithmetic expression X
COS(x)	Cosine of x radians
COT(x)	Cotangent of x radians
CSC(x)	Cosecant of x radians
DEG(x)	Number of degrees in x radians
DET(x)	Determinant of an arithmetic array[1]
EXP(x)	Natural exponent of x
HCS(x)	Hyperbolic cosine of x
HSN(x)	Hyperbolic sine of x
HTN(x)	Hyperbolic tangent of x
IDX(X$,Y$)	Position of the character string indicated by the second entry within the character string indicated by the first entry. These entries may be character constants, character variables, or substring entries. The number of characters searched for is defined by the length of the second entry. A result relative to position 1 is given, or a zero if the searched for entry is not present.

[1] Maximum matrix size is 50x50. Result of less than 1E-20 is the same as zero.

Figure 8 (Part 1 of 2). System Functions

Function Name	Description
INT(x)	Integral part of x[2] (positive values only)
KLN(X$)	Length in characters of the embedded key for file X$
KPS(X$)	Beginning character position of the embedded key for file X$
LEN(X$)	Length of character string X$ (less trailing blanks)
LGT(x)	Logarithm of x to the base 10
LOG(x)	Logarithm of x to the base e
LTW(x)	Logarithm of x to the base 2
MAX(X,Y...)	Maximum value of arithmetic scalars (X,Y...) or character arguments (X$,Y$...)[3]
MIN(X,Y...)	Minimum value of arithmetic scalars (X,Y...) or character arguments (X$,Y$...)[3]
NUM(X$)	Converted numeric value of character string X$
PRD(x)	Product of the elements of array x
RAD(x)	Number of radians in x degrees
RLN(X$)	Length of the last record referenced for file X$
RND[(x)]	Random number between 0 and 1[4]
SEC(x)	Secant of x radians
SGN(x)	Sign of x (-1, 0, or +1)
SIN(x)	Sine of x radians
SQR(x)	Square root of x
STR(X$,y[,z])	Substring of character variable X$, starting with the yth character and extending to the end of X$ or for z characters
SUM(x)	Sum of the elements of array X
TAN(x)	Tangent of x radians

[2]The comparison tolerance value is added to x before the integral part of x is taken.

[3]If the parameter list for MAX or MIN contains three or more character data items with different lengths, and the shorter data items appear to the left of longer data items, improper padding occurs on the shorter data items resulting in possibly incorrect values being returned. To prevent an error, use the MAX or MIN function with pairs of data items only, such as MIN(A$,MIN(B$,C$)); and arrange the data items so the shorter ones are located to the right ('AAA','AA','A').

[4]A nonzero argument to RND starts a series of random numbers determined by the argument value. A zero argument to RND starts a new series of random numbers with an undetermined value. If no argument is specified to RND, the next number in the current series is returned.

Figure 8 (Part 2 of 2). System Functions

EXPRESSIONS

An expression in BASIC is any representation of an arithmetic or character value. Constants, variables, arrays, array element references, and function references are all considered expressions. You can also form expressions by combining any of these value representations with symbols called operators.

An operator specifies either the relationship between data items, an arithmetic operation to be performed on them, or whether they are positive or negative. For example, the symbols >, *, and + are operators specifying greater than, multiplication, and addition (or positive value), respectively.

A special class of expressions, called relational expressions, is used with the IF statement to test the truth of specified relationships between two values.

Expressions referring to entire arrays, rather than individual array elements, are called array expressions. An expression that does not contain a reference to an entire array is called a scalar expression.

Arithmetic Expressions and Operators

An arithmetic expression can be an arithmetic variable, array element, constant, or function reference; or it can be a series of the preceding items separated by operators and parentheses. Some examples of arithmetic expressions are:

```
A1
X3/ (-6)
X+Y+Z
SIN(R)
-6.4
-(X-Y↑2/2+X)
```

You obtain the value of an arithmetic expression by performing the implied operations on the specified data items.

The five arithmetic operators are:

Symbol	Meaning
↑ or **	Exponentiation
*	Multiplication
/	Division
+	Addition
-	Subtraction

Note that the system stores ** as ↑.

The positive/negative operators are:

+ Positive (used only for clarity)
- Negative (changes the sign of the operand following it)

Special rules for the arithmetic operators and the resulting actions are as follows:

Exponentiation: The expression A↑B is defined as the variable A raised to the B power.

1. If A=B=0, an error will occur.

2. If A=0 and B<0, an error will occur.

3. If A<0 and B is not an integer, an error of a negative number to a fractional power will occur.

4. If A≠0 and B=0, A↑B is evaluated as 1.

5. If A=0 and B>0, A↑B is evaluated as 0.

Multiplication and Addition: A*B and A+B, multiplication and addition respectively, are both commutative; in other words, A*B=B*A and A+B=B+A. However, multiplication and addition are not always associative because of low-order rounding errors; for example, A*(B*C) does not necessarily give the same results as (A*B)*C.

Division: A/B is defined as A divided by B. If B=0, an error (overflow) will occur.

Subtraction: A-B is defined as A minus B. No special conditions exist.

Positive/Negative Operators: The + and - signs can also be used as positive/negative operators, which can be used in only two situations:

- Following a left parenthesis and preceding an arithmetic expression

- As the leftmost character in an entire arithmetic expression that is not preceded by an operator

For example:

-A+(-B) and B-(-2) are valid.
A+-B or B--2 are invalid.

Arithmetic Hierarchy

Arithmetic expressions are evaluated according to the hierarchy of the operators involved. Operations enclosed in parentheses are performed first. Operations with a higher priority level are performed before those with a lower priority level. Operations at the same priority level are performed from left to right. The hierarchy of the operators are:

	Operator	**Hierarchy**
1.	Enclosed in parentheses	Highest
2.	↑ or **	
3.	Positive + and Negative -	
4.	* and /	
5.	Addition + and Subtraction -	Lowest

You evaluate an expresion by reducing it to its component subexpressions. A subexpression is defined as a group that can be read *operand-operator-operand*, where an operand is one of the following:

- A simple reference to data (constant or variable)
- A subscripted array reference
- A function reference
- A parenthesized subexpression

Starting with the first operator to be executed according to the hierarchy, the operands of its subexpression are reduced to simple references to data in a left-to-right order. This process is repeated as many times as required in a left-to-right order or in a decreasing order of priority, or both, of the remaining operators until the entire expression is reduced to a simple reference to the evaluated result.

The following examples illustrate the successive steps in the evaluation of four arithmetic expressions according to the rules just described. In each expression, the variables A, B, and C have been assigned the integer values 4, 6, and 2, respectively.

Expression	Evaluation and Result
-A↑2+B/C*2.5	-4↑2+6/2*2.5 - 16 +6/2*2.5 - 16 +6/2*2.5 - 16 +3 *2.5 - 16 + 7.5 -8.5
(-A↑2)+B/C*2.5	(-4↑2)+6/2*2.5 - 16 +6/2*2.5 - 16 +3 *2.5 - 16 + 7.5 -8.5
-A↑(2+B/C)*2.5	-4↑(2+6/2)*2.5 -4↑(2+3) *2.5 -4↑5 *2.5 - 1024 *2.5 - 1024 *2.5 -2560
-A↑((2+B)/C)*2.5	-4↑((2+6)/2)*2.5 -4↑ (8/2) *2.5 -4↑4 *2.5 - 256 *2.5 - 256 *2.5 -640

Character Expressions

A character expression is a character constant, a character variable, a character valued function reference, a single element of a character array, or a substring (STR) function. The only operators ever associated with character expressions are the concatenations symbol and relational operators described next. The following are examples of valid character expressions:

```
D$(4)='DFR'
A$='SER'
STR(A$,1,3)=B$
A$=B$||C$
CHR(A-B)
MIN('A','C','E') or MIN (A$,B$,C$)
MAX('D','F','R') or MAX(D$,F$,R$)
```

Substring Function

A character string is a sequence of characters in a character expression. The string (STR) function allows you to extract, combine, or replace specific characters in the expression. The STR function is used with LET statements to do the following:

- Extract characters–For example, in the following statements:

```
10 A$ = 'PRODUCTION CONTROL'
20 STR(B$,1,10) = STR(A$,1,10)
```

statement 20 extracts the first 10 characters from A$ and assigns them to the first 10 characters of B$. In statement 20, A$ is the expression from which characters are to be extracted, 1 is the position of the first character to be extracted, and 10 is the number of characters to be extracted.

- Combine characters–For example, in these statements:

```
10 A$ = 'PRODUCTION'
20 B$ = 'CONTROL'
30 LET STR(A$,12,7) = B$
```

statement 30 places 7 characters from the content of B$ in the character string A$ beginning at the 12th position. The 7 is optional. If it is omitted, the remainder of the string (A$) is used starting at the specified position (12).

- Replace characters–For example, in these statements:

```
10 LET A$ = 'PART 789'
20 LET B$ = 'PART 1234'
30 LET STR(A$,6,4) = STR(B$,6,4)
```

statement 30 replaces the 789 in A$ with the 1234 from B$. Again, the numbers 6 and 4 in statement 30 specify the first character to be replaced and the number of characters to be replaced, respectively. These numbers do not have to be the same on both sides of the equal sign.

Concatenation

Concatenation is joining two character expressions using the concatenation symbol | |. The following is an example of concatenation:

```
0010 DIM A$5
0020 DIM B$4
0030 A$='MINNE'
0040 B$='SOTA'
0050 C$=A$||B$
```

In this example, the character string A$ is concatenated with the character string B$ to form string C$ (MINNESOTA). When two or more character strings are concatenated, the length of the resulting string is the sum of the individual strings.

If the data list of the PRINT statement has a character constant concatenated with a character variable, only the character constant is printed. In order to prevent this error, concatenate before executing the PRINT statement or replace the concatenation symbol with a semicolon. In the result, concatenation is limited to 127 bytes in length.

Relational Expressions

A relational expression compares the value of two arithmetic expressions or two character expressions. The expressions to be compared are evaluated and then compared according to the definition of the relational operator specified. According to the result, the relational expression is either satisfied (true) or not satisfied (false). *Relational expressions can appear in a BASIC program only as part of an IF statement.*

The relational operators and their definitions are:

Operator	Meaning
=	Equal
≠ or < >	Not equal
≥ or > =	Greater than or equal
≤ or < =	Less than or equal
>	Greater than
<	Less than

Note: The system stores < > as ≠, <= as ≤, and >= as ≥. Comparison uses all 15 digits. Thus, if you compare the relationship of two numbers, they must compare in all 15 digits for the relationship to be true. The general format of a relational expression is:

e_1 relational-operator e_2

where e_1 and e_2 are any expressions other than array or relational expressions, and *relational-operator* is any of the operators just described. Both e_1 and e_2 must be of the same data type (character or arithmetic), and only two expressions can be compared in a single relational expression.

When character data appears in a relational expression, it is evaluated according to EBCDIC value (see Appendix A) character by character, from left to right. Thus, the following relational expressions would all be satisfied:

```
'ABC'='ABC'
'ABLE'<'BALL'
'123'>'BALL'
'$12'<'7'
```

When character operands of different lengths are compared, the shorter operand is considered to be extended on the right with blanks to the length of the longer operand. Thus, in the preceding third example, values compared are 123␣ and BALL, where ␣ is a blank character.

Relational expressions can also contain the logical AND/OR expressions. In BASIC, the ampersand (&) is used to specify that two sets of arithmetic or character expressions be compared. For example:

e_1 relational-operator e_2 & e_3 relational-operator e_4

where e_1 through e_4 are expressions. In this example, the relational expression is true only if the relation between e_1 and e_2 and the relation between e_3 and e_4 is satisfied.

The vertical bar (|) is used to specify that either of two sets of expressions can compare in order for the relational expression to be true. For example:

e_1 relational-operator e_2 | e_3 relational-operator e_4

where e_1 through e_4 are expressions. In this example, the relational expression is true if either the relation between e_1 and e_2, or between e_3 and e_4 is satisfied.

For more information on these logical operators (& and |), see the IF statement in Chapter 4.

Array Expressions

In mathematics, a matrix is a group of arithmetic values arranged in a system of rows and columns, and a vector is a series of arithmetic values arranged in a single row. In the BASIC language, however, a matrix is a one- or two-dimensional array. Array expressions are used to perform operations on the entire collection of numeric or character array elements rather than on each element individually (scalar operations). (See *MAT Assignment Statements* in Chapter 4.)

DATA FILES AND ACCESS METHODS

The IBM 5110 is capable of processing two distinct types of data files: stream I/O and record I/O. Each of these files is described below.

Stream I/O Data Files

Stream I/O data files are useful for collecting streams of variable-length data items and storing them in sequential order as records in a tape or diskette file. Stream I/O files must be opened (see *OPEN/OPEN FILE* in Chapter 4) before they can be accessed, using the PUT statement to store data items in the file and using the GET statement to retrieve data items from the file. These files are organized for sequential access (one record after another in the order the records were entered). These files must also be closed after being used (see *CLOSE* in Chapter 4).

The data items within each logical record of a stream I/O file must be separated by a comma. When numeric data items are read from the file, the system retrieves them in succession and converts them into internal numeric format. When character data items are read from the file, the system locates the next nonblank character, ignoring the comma separators. If the retrieved character is a single quotation mark, the data item following must be a valid character constant. The end of the character constant is indicated by its closing single quotation mark. If the first character retrieved by the system is not a single quotation mark, the end of the character data item is indicated by the next comma. Thus, to have a comma in the character string, the entire character string must be enclosed in single quotation marks.

Record I/O Files

Record I/O files are useful for collecting related numeric and character data items and storing them as a unit in a fixed-length logical record. These files must be opened before you can access them using the WRITE FILE statement to store data items in the file and the READ FILE statement to retrieve data items from the file. Record I/O files can be accessed sequentially, directly, or key indexed.

Sequential Access

A record I/O file can be accessed in the order in which the records were placed in the file. This is called sequential access. Sequentially accessed files allow you to:

- Read records using READ FILE
- Read and update records using READ FILE followed by REWRITE FILE
- Add records to the end of a data file using WRITE FILE

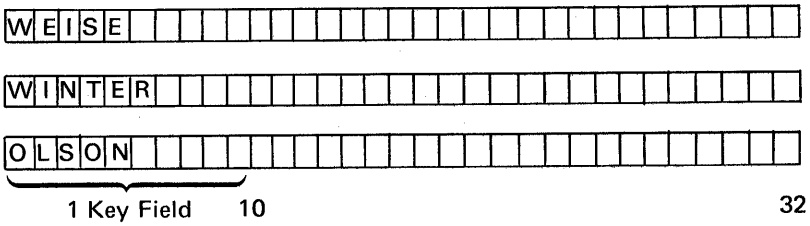
Direct Access

In a directly accessed file, each record is assigned a specific record number based on the position of that record in relation to the beginning of the file. This is called the relative record number (also called logical record number). In directly accessed files records can be accessed in any order if the REC=n parameter is specified on the READ FILE or REWRITE FILE statement. This access method is called direct access by relative record number. A directly accessed file can also be accessed sequentially if the REC=n parameter is not specified. Directly accessed files allow you to:

- Read records in any order using READ FILE REC=n
- Update records in any order using REWRITE FILE...REC=n
- Add records to the end of a file using WRITE FILE
- Read records using READ FILE
- Read and update records using READ FILE followed by REWRITE FILE

Key Indexed Access

An indexed file allows you to select a particular field (up to 28 bytes of character data) within each record to be used to identify that record. The field you select is called the *key* for the record. Each record must have a unique key, with the key field in the same position for every record. For example, assume a record length of 60 characters for each record in a file. Also assume that positions 1 through 10 in every record are selected as the key field.



Each record can then be accessed according to its unique key in the key field. When you access a record, you specify the record key (KEY=parameter), which is called a *search argument.*

For key indexed files, the system also maintains an index file containing the key and location (relative record number) on tape or diskette of each record in the file.

When you access a record by specifying its key, the system searches the index file for the key that is greater than or equal to the search argument and goes directly to the corresponding tape or diskette location to retrieve the record. This is called direct access by key index. Because the index file contains only the record key and location, it can be searched more quickly than all records in the file; therefore, record retrieval time is substantially reduced. Retrieval time can be reduced even further if you sort the keys in the index file and/or use the KW parameter.

When an indexed file is created, you must open both the data file that contains your records and the index file created by the system.

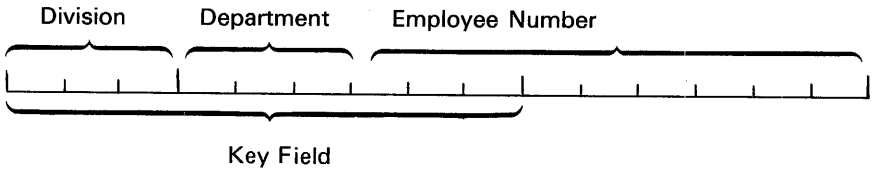
If a DUPKEY error occurs during the creation of an index file, and the record written to the file after the error contains a key with a value greater than all previous keys, then the index file being created is invalid and errors may occur during continued program execution. If the DUPKEY error occurs, close the index file; then reopen it as an update file by using the open keyword ALL.

If an EOF file error condition occurs while you are creating or adding to an indexed file and the files are closed, the master file or key file may have missing records. The files may have missing records because the 5110 writes all buffers at file close. The files should be examined to ensure that all records are present in both files.

If you omit the KEY= parameter, the system accesses the record with the next higher key than the last record accessed. This is called sequential access by key index.

You can also combine the direct access by key and sequential access by key by specifying a KEY= parameter on the first access only. This gives you the records in ascending sequence beginning with the key specified in the KEY= parameter.

If you specify a key in a KEY= or KEY≥ parameter that is longer than the key in the key index file, the specified key is truncated to the length of the key in the index file. If you specify a key that is shorter than the key in the index file, only the characters you specified will be used in the search. If the key field is made up of several parts, for example, then you could access the first record on part of the key as shown:




```
0010 DIM K$3,A$3
0080 K$= '003'
0090 READ FILE FL1, KEY=K$,A$,...
  .
  .
  .
  .
0200 READ FILE FL1,A$,...
0210 GOTO 100
  .
  .
  .
  .
  .
  .
```

The program would access the first record with division equal to 003 and then access all records with division 003 in sequence by key, assuming the master file is in ascending sequence by key.

Index File Format: The index file contains records whose only contents are the key value and the location of the record in the master file containing that key. Records in the index file may be 8, 16, or 32 characters long as shown in the following table:

Key Length in Master File	Index File Record Length
1 to 4 characters	8 characters
5 to 12 characters	16 characters
13 to 28 characters	32 characters

There are two types of records in the index file: key records and marker records. The key records contain the key (from the master record), which begins in position 1 of the index file record and the relative record number in binary in the last three bytes. The relative record number can be accessed with a B2 specification on the last two bytes (see *FORM,* B parameter, in Chapter 4) if there are less than 32,767 records.

There are two marker records in each index file. The first marker record contains the hex 00 character in all record positions except the last four. The last four positions contain two 2-character fields. The first 2-character field contains the length of the key field in binary fixed point format. The second 2-character field contains the starting position of the key field in the master record in binary fixed point format. The second marker record contains the hex FF character in all positions except the last four. The last four positions are not used.

The first marker record must be the first record in the index file. The second marker record is used to locate the end of the sorted portion of the index file. The second marker record is the second record of the file until the file has been sorted. In this position it indicates that there is no sorted portion in the index file. The records between marker records are assumed to be in ascending sequence by key.

When you specify a KEY=parameter, the index file is searched for a key that satisfies the search condition (=, ≥, or next sequential key). The keys between marker records can be searched rapidly because they are in sequence and because a work table can be used to shorten the portion of the file to be searched (see KW= parameter under *OPEN/OPEN FILE* in Chapter 4).

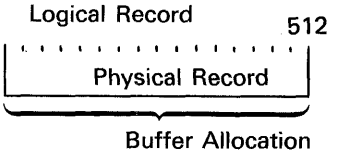
After the sorted portion of the index file has been searched, keys beyond the second marker record (if any) must be searched in order, which is much slower.

When you use the DELETE FILE statement, the record number field of the specified key record is set to zero and is not considered in the search. However, if a DELETE FILE specifies that the record with KEY=XYZ is to be deleted, and a later WRITE FILE statement creates a new record in the master file with KEY=XYZ; the entry in the index file for key XYZ is reused and will point to the new record. The old record, though still in the master file, is not accessible using the index file.

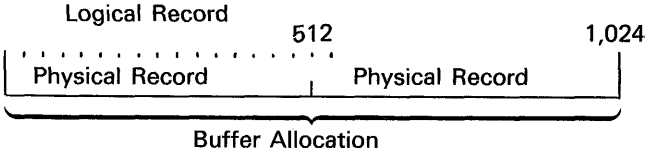
Record I/O File Buffer Requirements

The work area buffer required for a record I/O file depends on the physical record length of the media (tape or diskette) and the logical record length (data) in the file. Buffer space is allocated in multiples of physical record length. If logical record length is less than physical record length and can be divided evenly into physical record length, one buffer is allocated. If logical record length is less than physical record length and *cannot* be divided evenly into physical record length, two buffers are allocated. If logical record length is greater than physical record length in an even multiple (two, three, four times physical length, for example), an equal number of buffers are allocated. If logical record length is greater than physical record length, but is not an even multiple, buffers are allocated on the basis of the first multiple of physical record length that exceeds logical record length plus one physical record length. Following are allocation examples; assume that the physical record length is 512 characters for each example.

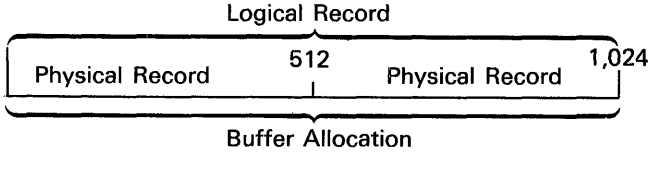
When the logical record length is 32 characters, the buffer allocation is 512 characters as shown:



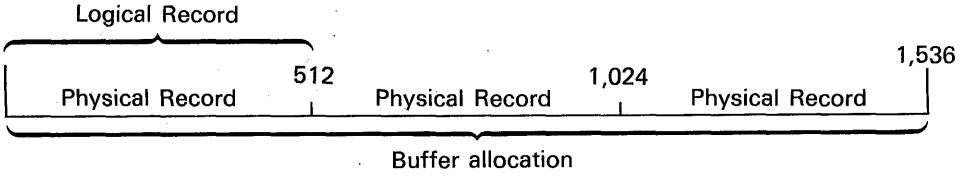
When the logical record length is 30 characters, the buffer allocation is 1,024 characters as shown:



When the logical record length is 1,024 characters, the buffer allocation is 1,024 characters as shown:



When the logical record length is 515, the buffer allocation is 1,536 characters as shown:



Note: If the buffer allocation is greater than 768 and you attempt to LOAD or SAVE record I/O files, an error message (604) will result indicating insufficient storage space.

FILE FLS

File FLS is a 35-byte, system-oriented file that allows you to indicate (with the WRITE FILE FLS statement) such information as console control, national character set selection, and rounding of numeric data. Also, you can use the READ FILE FLS statement to access such data as total user work area in the system, total work area available for variables and buffers, and total number of lines printed.

You can use the WRITE FILE statement referencing file FLS to indicate the following information.

Column 1 of file FLS indicates:

Character	Meaning
'N'	Turn on the display screen.
'F'	Turn off the display screen (this increases processing speed).
'S'	Sound audible alarm (if installed) and leave it on.
'A'	Sound the audible alarm for approximately one quarter second.
'Q'	Turn off the audible alarm.
'.'	Select EBCDIC character set.
'1'	Select Austria/Germany character set.
'2'	Select Belgium character set.
'3'	Select Brazil character set.
'4'	Select Denmark/Norway character set.
'5'	Select Finland/Sweden character set.
'6'	Select France character set.
'7'	Select Italy character set.
'8'	Select Japan character set.
'9'	Select Portugal character set.
'0'	Select International character set.
'/'	Select Spain character set.
'*'	Select Spanish Speaking character set.
'-'	Select United Kingdom character set.
'+'	Select French Canadian character set.

Also see *National Character Sets*.

Any other character entered in column 1 will be ignored.

Column 2 of file FLS indicates keyboard operation as follows:

Character	Meaning
'U'	Select standard BASIC character mode on the keyboard (this is the normal default).
'L'	Select lowercase character mode on the keyboard.

Any other character entered in column 2 of the buffer will be ignored.

Column 3 changes statement trace operations as follows:

Character	Meaning
'N'	Turn on statement trace (see *RUN Command* in Chapter 2).
'F'	Turn off statement trace.

Any other character entered in column 3 will be ignored.

Column 4 changes statement trace to the printer (see *RUN Command* in Chapter 2) as follows:

Character	Meaning
'N'	Turn on trace to the printer (trace must already be on).
'F'	Turn off trace to the printer.

Columns 5 and 6 allow you to specify a 2-character numeric value (a leading blank is valid) for rounding of numeric data (see *RD = Command* in Chapter 2).

If columns 7 through 9 contain FLx (where x is 0-9) of a stream I/O input file, subsequent input to character variables includes leading blanks and all commas and quotation marks. Thus, entire logical records can be read into character variables. If the file referenced (FLx) was open on device '001', both GET and INPUT statements will be effected.

Columns 10 and 11 allow you to set spacing control for printed lines. The printer spacing increment is 1/96 inch. Thus, 16 increments provide printing at the normal 6 lines per inch, 12 increments provide 8-line-per-inch spacing (96/12), 32 increments provide 3-line-per-inch spacing (96/32), and so on.

You can enter a value of 00, or a value of 08 through 99. An entry of 00 suppresses spacing, and an entry of 08 through 99 indicates the number of spacing increments for each line printed. If you enter 01 through 07, normal 6-line-per-inch printing will occur (the entry 01 through 07 is ignored). If an entry of 00 is used to suppress spacing, the lines printed counter is still incremented each time the print buffer contents are transferred to the printer. If you enter 08, 09, or 10, the printer prints over the last line printed. For example, if you enter 08, the printed output is as shown in the following example:

```
0010 PRINT FLP,'VΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔV'
0020 WRITEFILE FLS,'          08'
0030 PRINT FLP,'VΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔV'
0040 PRINT FLP,'VΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔV'
```

```
VΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔV
VΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔVΔV
```

Column 12 is the start of comparison tolerance See *Comparison Tolerance* in this chapter.

You can use columns 22 through 35 to hold any data using the WRITE FILE statement. You can access this data using the READ FILE statement.

You can access the following information using the READ FILE statement referencing file FLS:

Bytes	Meaning
1-5	Indicates the total area available in the system
6-10	Indicates the available user work area for variables and buffer usage
11-15	Indicates the number of data transfers from the print buffer to the printer. These are called *lines printed.*
16-18	Reserved
19-21	Indicates the return code (0-255) last set by the RC= parameter in the latest STOP or END statement

National Character Sets

As shown in the description of file FLS, you can select from several national character sets for processing on your 5110 within a BASIC program. These national character sets can also be selected from the keyboard whenever the keyboard is open for input. The following chart shows each national character set, the corresponding key that you must press to select the character set, and the resulting set, and the resulting characters that change according to the character set selected.

Press the HOLD key. Then hold down the SHIFT key and press one of the following keys on the calculator pad.	To select this national character set:	These characters change:	Hex Position										
			4A	5A	5B	5F	6A	79	7B	7C	A1	C0	D0	E0
1	Austria/Germany		Ä	Ü	$	^	ö	`	#	§	ß	ä	ü	ö
2	Belgium		[	]	$	^	ù	`	#	à	¨	è	é	ç
3	Brazil		Ë	$	Ç	^	ç	ã	Õ	Ã	~	õ	è	\
4	Denmark/Norway		#	¤	Å	^	ø	`	Æ	Ø	ü	æ	à	\
5	Finland/Sweden		§	¤	Å	^	ö	è	Ä	Ö	ü	ä	à	É
6	France		°	§	$	^	ù	`	£	à	¨	è	é	ç
7	Italy		°	è	$	^	ò	ù	£	§	ì	à	é	ç
8	Japan		£	!	¥	¬	¦	`	#	@	¯	{	}	$
9	Portugal		[	]	$	^	ö	`	Ã	Õ	ç	ã	´	Ç
0	International		[	]	$	^	¦	`	#	@	~	{	}	\
/	Spain		[	]	₧	¬	ñ	`	Ñ	@	¨	{	}	\
*	Spanish Speaking		[	]	$	¬	ñ	`	Ñ	@	¨	{	}	\
–	United Kingdom		$	!	£	¬	¦	`	#	@	¯	{	}	\
+	French Canadian		à	´	$	^	ù	`	#	@	¨	è	é	§
.	EBCDIC		¢	!	$	¬	¦	`	#	@	~	{	}	\

Note: Your service representative can change the character set that is in effect when the power is turned on or RESTART is pressed. In this case, you can still select any of the other character sets from the keyboard or use file FLS.

PROCEDURE FILE

The procedure file is a user-generated, record I/O tape or diskette file. You can create a procedure file just as you create a record I/O data file using the LOAD0,DATA command. A procedure file can contain commands, BASIC statements, and data to be used by the program for input. Record length in the procedure file is up to 64 characters.

Use of a procedure file is initiated by a PROC command, which causes the file to be opened implicitly. Records from the file are then loaded into a buffer area and sequentially executed as if they had been entered from the keyboard. When a record in the buffer has been executed, another record is loaded from the procedure file and used.

The SKIP and CSKIP commands allow selective use of records in a procedure file. The unconditional SKIP instructs the system to pass over a specified number of records. The conditional CSKIP instructs the system to test a condition (return code) from the STOP or END statement in the last program executed, then to pass over a specified number of records depending on the condition.

The ALERT command allows you to provide an indication to the operator that intervention is needed during execution of the procedure file.

Data in a procedure file can also be used as input supplied in response to an INPUT statement if the RUN command for the program included the IN=P parameter. Like all records in a procedure file, data records have a maximum length of 64 characters. Also, each value assigned from a procedure file must be of the same type (character or numeric) as the variable to which it is assigned in the INPUT statement.

If any errors occur while under PROC control, the procedure file is closed.

A PAUSE statement within a program under PROC control will cause the next record in the procedure file to be executed.

In the following example, the procedure file contains a series of records that will cause several BASIC programs to be loaded and executed.

UTIL SYS D80	Set the default device.
LOAD 'BUILD'	Load the BUILD program.
RUN IN=P	Start BUILD; data for input is taken from the procedure file.
D40,0,'NEWFILE'	This is data for an input statement.
CSKIP 5	If BUILD sets a nonzero return code, skip five records in the procedure file, otherwise get the next record.
LOAD 'READY.SORT'	If the CSKIP did not bypass records, load program READY.SORT.
RUN	Execute READY.SORT.
UTIL SORT	Transfer control to the sort feature.
FILE NEWSORT,1,SORT.CTL,1	This record is passed to the sort program (see *IBM 5110 Customer Support Functions Reference Manual*).
SKIP 2	Bypass the next two records.
LOAD 'NOSORT'	If the CSKIP bypassed records, load program NOSORT.
RUN	Execute NOSORT.
Alert change diskette in drive 2	Pause to let the operator change the diskette.
PROC 'PROC2'	Start another procedure.

BASIC EXCHANGE FILES

The purpose of the basic exchange file (type B9) is to allow the general exchange of data with other products. The 5110 supports this basic exchange when the OPEN FILE statement or SAVE command uses a simple name (8 or fewer characters) and includes the NOBLOCK and SEQ parameters. The diskette used must be initialized to type 1, type 4, or type 7 format. (See the *IBM 5110 Customer Support Functions Reference Manual*, SA21-9311, for a description of diskette formatting.)

Note: If you want to read diskettes written on the IBM 5110 on a 3741 you should use 128-byte records.

COMPARISON TOLERANCE

The comparison tolerance value determines the maximum tolerance (how different two numbers may be and yet be considered equal) when any relational function is used. For example, two numbers are considered equal if the relative difference between the two numbers is less than the comparison tolerance value. When power is turned on or the RESTART switch is pressed, the comparison tolerance is initialized to 5E-7 or .0000005. Even though the 5110 system initially sets the comparison tolerance value, you may change the value by using the following BASIC statement and commands:

- Before program execution by using:
  - RD=n
  - RUN RD=n

- Within a program by using:
  - GO RD=n
  - RD=n
  - WRITE FILE FLS

The command RD=n sets the comparison tolerance value to 5E-(n+1). The value of n may be any number from 1 to 15. When the RD=n command or the RUN RD=n is used to change the comparison tolerance value, the command must be entered before the program is executed.

When the GO RD=n command or the RD=n command is used to change the comparison tolerance value within a program, program execution must be interrupted by a PAUSE statement or the use of the ATTN key.

In the following example, the comparison tolerance value is .0000005:

```
0010 A=.99999951
0020 IF A=1 GOTO 50
0030 PRINT 'A ≠ 1'
0040 STOP
0050 PRINT 'A = 1'
RUN
A = 1
```

When the IF statement is used, the value of A is subtracted from 1. If the difference is less then .0000005, then A is considered equal to 1. If the RUN RD= command is used to reset the comparison tolerance value to .0000005, then A is not equal to 1 as shown in the example below:

```
RUNRD=5
A ≠ 1
```

For more information on comparison tolerance and the IF statement, see the *BASIC Statements* in Chapter 4.

When the WRITE FILE FLS statement is used to change the comparison tolerance value, certain conditions must be met. The comparison tolerance value must:

- Start in column 12
- Have long precision
- Be set to a value greater than or equal to 0 but less than 1

The WRITE FILE FLS statement can be used at any time within a program. The following example shows the WRITE FILE FLS statement used with a FORM statement that specifies a long precision value beginning in position 12:

```
0020 WRITEFILE USING 30,FLS,5E-4
0030 FORM POS12,L
```

When comparison tolerance is used with positive values of the INT function, INT takes the comparison tolerance value and adds it to the number before the integral part of the number is taken. If the comparison tolerance value is zero when arithmetic is performed, precision is lost because of binary conversion (see *BASIC Statements* in Chapter 4). In the following example, the comparison tolerance value is set to zero:

```
0010 WRITEFILE USING 20,FLS,0
0020 FORM POS12,L
0030 A=,01*100
0040 PRINT INT(A)
RUN
```

In statement 30, it appears that A equals 1 because .01*100 equals 1. However, because of the lost precision, A equals .9999999999999991. When you attempt to take the integer (INT) of A, a value of zero is returned. When the comparison tolerance value is reset to 5E-7 and the INT function is used, the value returned is 1. The actual value of A remains .9999999999999991.

The comparison tolerance value may be eliminated by the user by setting it to a value of zero with the WRITE FILE FLS statement as shown in the following example:

```
0015 WRITEFILE USING 16,FLS,0
0016 FORM POS12,L
```

Chapter 4. BASIC Statements

BASIC statements allow you to enter data and specify how that data is to be manipulated and what the outcome is to be. BASIC statements are either executable or nonexecutable. Executable statements cause a program action, such as value assignment or printing. Nonexecutable statements describe information needed by the program and the user, but cause no visible action.

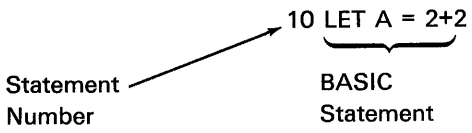
Executable and nonexeutable statements can be intermixed when a BASIC program is entered from the keyboard. The maximum number of statements permitted in a single BASIC program is limited only by the user work area size of the system and by the statement types.

Each statement in a BASIC program must begin with a statement number. The number determines the order of execution of the statements in the program. All statements are executed in numeric order, regardless of the order in which they were entered, unless the sequence of execution is altered by branches, loops, or subroutines.

The allowable range of statement numbers for a BASIC program is from 1 through 9999. You do not have to enter preceding zeros (0020, for example) because the system maintains statement numbers as four-position integers and inserts preceding zeros for statement numbers of less than four digits.

STATEMENT LINES

A BASIC statement preceded by a statement number is called a statement line. Statement lines are entered from the keyboard (one per display line) with a maximum of 64 characters. Statements cannot be split between two display lines, nor can there be more than one statement on each display line. A typical statement line is as shown.



In this chapter, all the BASIC statements are presented alphabetically in the syntax used for BASIC commands, except the statements dealing with matrix operations. These statements are discussed later in the chapter under *Matrix Operations*.

DESK CALCULATOR OPERATIONS

You can perform several desk calculator operations using BASIC statements without preceding statement numbers. These operations are executed immediately after you press the EXECUTE key. Desk calculator operations include:

- Assignment of values to variables (both character and numeric) *without* the keyword LET (Character variable assignment is a length of 18 unless dimensioned otherwise.)
- PRINT (to display) and PRINT FLP (to print) statements
- MAT PRINT and MAT PRINT FLP statements
- DIM statements to dimension arrays and variables
- Character and numeric expressions
- MAT assignment statements

BASIC STATEMENT LISTING

The statements used in the BASIC language for the system are listed below. A brief description of each statement is included.

CHAIN–Ends a program, then loads and begins executing another program.

CLOSE [FILE]–Deactivates open files.

DATA–Creates an internal data table of values you supply.

DEF–Defines a function to be used in the program.

DELETE FILE–Removes a specific record from a key-indexed file.

DIM–Specifies the size (dimensions) of an array or character variable length.

END–Ends a program.

EXIT–Specifies error exits for corresponding I/O error conditions.

FNEND–Ends a function defined in a DEF statement.

FOR–Begins a loop.

FORM–Specifies format for displayed/printed output and records in files.

[MAT] GET–Assigns values from a stream I/O file to variables or array elements.

GOSUB–Branches the program to the beginning of a subroutine.

GOTO–Branches the program to a specific statement.

IF–Branches the program depending on specific conditions.

:Image–Specifies formatting of data to be displayed or printed.

[MAT] INPUT–Assigns values from the keyboard to variables or array elements during program execution.

LET–Assigns values to variables.

MAT Assignment–Assigns values to all elements of an array.

NEXT–Ends a loop (see FOR and NEXT).

ONERROR–Specifies error recovery routine for debugging.

OPEN–Activates stream files for input or output.

OPEN FILE–Activates record files for input/output and access method.

PAUSE–Interrupts program execution.

[MAT] PRINT (FLP)–Displays or prints the values of specified variables, expressions, array elements, or constants.

[MAT] PRINT USING (FLP)–Displays or prints the values of specified variables, array elements, expressions, or constants in a format defined in an image or FORM statement.

[MAT] PUT–Writes the values of specified variables, expressions, or array elements into a stream I/O file.

[MAT] READ–Assigns values from the internal table (see *DATA*) to variables or array elements.

[MAT] READ FILE [USING]–Assigns values from record I/O files to variables.

REM–Inserts comments or remarks into a program.

[MAT] REREAD FILE [USING]–Allows reaccess to the last record read from a file.

RESET–Repositions a file to its beginning, to its end, or to a specific record.

RESTORE–Causes values in the internal data table (see *DATA*) to be assigned starting with the first table value.

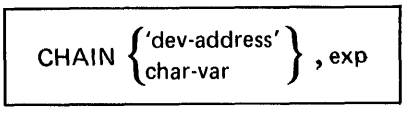
RETURN–Ends a current subroutine or user function.

[MAT] REWRITE FILE [USING]–Allows change/update to record I/O files.

STOP–Ends a program.

USE–Saves variables to be used by successive programs.

[MAT] WRITE FILE [USING]–Adds records at the end of a record I/O file.



CHAIN

The CHAIN statement performs the following sequence:

1. Ends the program currently being executed
2. Loads a new program
3. Begins executing the new program

The syntax of the CHAIN statement is as shown above, where:

'dev-address' is the address of the device containing the next program to be loaded and executed. You can enter the address directly, or enter a character variable containing the device address. See *Device Address Parameter* in Chapter 2.

char-var is a character variable to which you have assigned a device address.

expression can be an arithmetic or character expression. An arithmetic expression specifies the number of the file to be loaded. A character expression specifies the name of the file to be loaded. Upon execution, all open files in the current program are closed. The program in the file (determined by the expression or constant) on the device specified is then loaded and executed starting with the lowest statement number.

For example:

```
0110 CHAIN 'E80',14
```

In this statement, the program in file number 14 on device 'E80' is loaded and executed when this statement is executed.

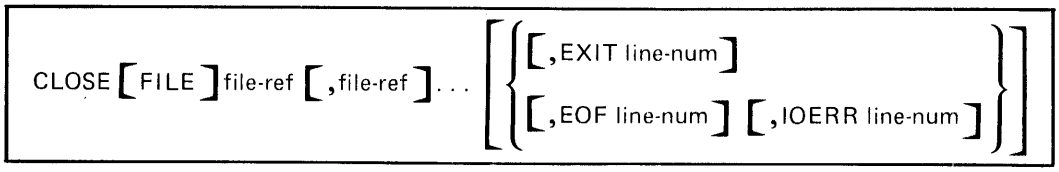
Notes About CHAIN

- When used in conjunction with the USE statement, the CHAIN statement allows variable values to be maintained from one program to the next (see *USE* in Chapter 4).
- The CHAIN statement can also be used to chain to a procedure file (see *Procedure File* in Chapter 3). When the expression value is character and begins with .PROC, the CHAIN statement closes all files and places the expression value (except for the leading .), a comma, and the device code enclosed in quotes on line 1 of the display screen. The system then processes the result as a PROC command.

Following is an example of a CHAIN statement:

```
60 A$='D80'
70 B$='.PROC4'
80 CHAIN A$, B$
```

In this example, the system will chain to the fourth file (the procedure file) on diskette drive 1. Note that the period preceding PROC4 in statement 70 is necessary to distinguish PROC4 from another valid file name when the CHAIN statement is executed.



CLOSE

The CLOSE statement specifies files to be deactivated. An implicit CLOSE statement is automatically executed for each active file at the end of program execution. The syntax of the CLOSE statement is as shown above, where:

FILE is specified when files to be closed are record I/O files only. Record I/O files are those with fixed-length records.

file-ref is from FL0 to FL9 and represents the same file specified in the OPEN statement. Only one file reference is required.

EOF, IOERR, and EXIT are error recovery exits, which direct the program to branch to the specified line number upon the occurrence of the indicated error (see *EXIT*).

Notes About CLOSE

- If a stream I/O file is used for both input and output operations during execution of a single program, the file must be closed and reopened between input and output references.
- If you do not close a file (CLOSE or end of program), the file may become unusable.
- If a file specified in a CLOSE statement is not active when the CLOSE statement is executed, the statement is ignored.
- The file references must be the same as those specified for the files in the OPEN statement.

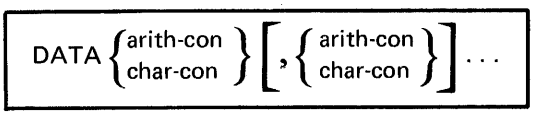
Examples

A sample CLOSE statement is as shown:

```
0020 CLOSE FL2, FL9
```

A sample CLOSE statement with error exits is as shown:

```
0030 CLOSE FL5, FL8, EOF 0090, IOERR 0200
```



DATA

The DATA statement is a nonexecutable statement that causes an internal data table to be created. The data table constants are supplied to variables and array elements specified in corresponding READ or MAT READ statements.

The syntax of the DATA statement is as shown above, where:

con is an arithmetic or character constant. Only one constant is required.

At the beginning of program execution (before any executable statements are executed), a table containing all the constants from all the DATA *statements* (in their order of appearance by statement number) is built. At the same time, a *pointer* is set to the first constant in the table. The pointer is advanced through the table, constant by constant, as the data is supplied to READ or MAT READ statement variables. (The pointer can be changed to point to the first constant again by the RESTORE statement.)

The length of character constants to be assigned to character variables is limited only by line length (64 characters).

Notes About DATA

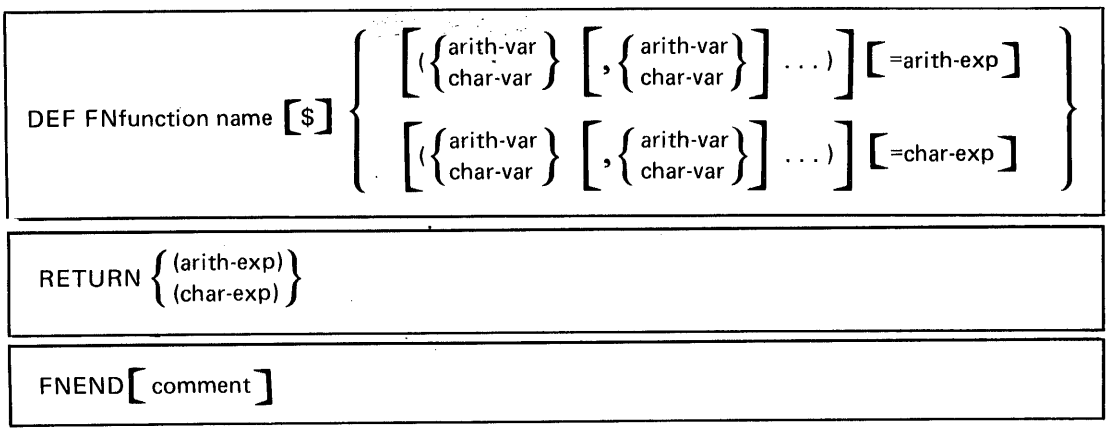
- Each constant in a DATA statement must be of the same type as that specified for the variable to which the constant is to be assigned in the corresponding READ statement. Thus, if the third constant in a DATA statement is a character constant, then the READ statement variable to which it is assigned must also be a character variable.
- DATA statements can be placed anywhere in a program, either before or after the READ statement to which they supply data.
- An error will occur if DATA statements do not contain enough constants for the READ statements issued.
- DATA statements cannot be used in a program assigned to one of the function keys.
- Character data need not be enclosed in single quotation marks unless leading blanks and/or embedded commas are significant.
- DATA statements are not checked for correct syntax.

Example

A sample DATA statement is as shown:

```
110 DATA BILL,21.60,CHARGE,15.40
```

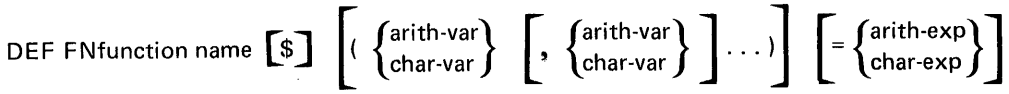
In this example, the character constants (BILL and CHARGE) and the arithmetic constants (21.60 and 15.40) are inserted into the internal data table.



DEF, RETURN, FNEND

The DEF statement is not executable, but informational. This statement allows the user to define an arithmetic or character valued function for reference elsewhere in the program. The FNEND statement indicates the end of the function, and the RETURN statement specifies the value of the function. The syntax of the DEF statement can be either a single line or multiline function. (The syntax of the FNEND and RETURN statements are shown after the DEF statement syntax.)

Single Line Function



where:

function-name is any character of the extended BASIC alphabet. This character with FN is the name of the defined function. For character valued functions, this character must be followed by a dollar sign ($).

(arith-var) is a simple arithmetic variable to which a value will be assigned when the function is called (these must be enclosed in parentheses).

(char-var) is a character variable to which a value will be assigned when the function is called. Assigned values cannot exceed 18 characters. Longer values will be truncated, and shorter values will be padded with blanks.

arith-exp, char-exp is an arithmetic or character scalar expression that specifies the value to be returned for the function.

A character expression must be specified if the function name is a character variable. Likewise, the expression must be arithmetic for an arithmetic function name (FNA).

Sample single-line DEF statements are shown below:

```
120 DEF FNA(R)=2*R+100

120 DEF FNA$ (R)=CHR(R+5)
```

Multiline Function

DEF FNfunction name [$] [({arith-var / char-var} [,{arith-var / char-var}] ...)]

where:

function-name is any character of the extended BASIC alphabet. This character with FN is the name of the function. For character valued functions, this character must be followed by a dollar sign ($).

(arith-var) is a simple arithmetic variable that receives a value when the function is referenced. These optional variables must be enclosed in parentheses.

(char-var) is a character variable to which a value will be assigned when the function is called. Assigned values cannot exceed 18 characters. Longer values will be truncated, and shorter values will be padded with blanks.

RETURN {(arith-exp) / (char-exp)}

where:

arith-exp, char-exp is an arithmetic or character scalar expression that specifies the value of the user-defined function to the referencing function. This expression must be of the same type (arithmetic or character) as the function defined.

FNEND[comment]

The FNEND statement is nonexecutable and simply indicates the end of a multiline function. The value of the function is specified in an expression in a RETURN statement. The comment is optional.

When a reference to a user-defined function is encountered during program execution, the value of each parameter in the expression is used to initialize the corresponding variable. The optional variables must match the number and type specified in the function reference. If the expression is present, the function is defined on the one line and its value is the value of that expression. This is a single line function. If the expression is not specified, the DEF statement is the start of a multiline function. In this case, the FNEND statement indicates the end of the function and the value of the function is specified in an expression in a RETURN statement.

Notes About DEF

- A function can be defined anywhere in a BASIC program, either before or after it is referenced.
- A function of a given name can be defined only once in a given program.
- A function cannot contain references to itself or to other functions that refer to it in their definitions.
- The expression in the RETURN statement is required for multiline functions (see *GOSUB and RETURN*).
- A function reference to a user-defined function can appear anywhere in a BASIC expression that a constant, variable, subscripted array element reference, or system function reference can appear (except desk calculator operations).
- The variables have a special meaning in the DEF statement. Consequently, it is possible to have a variable with the same name as a simple variable used elsewhere in the program. Each is recognized as being unique, and no conflict of names or values results from this duplicate usage.
- The maximum number of user-defined functions in a program is 29, and the maximum number of nested function references varies according to the complexity of the referencing statement.
- User-defined functions that are referred to during an input or output operation cannot themselves perform any input or output.
- After control is passed to a DEF statement without reference to the function, control goes to the first executable statement following the function definition–following the DEF statement for single-line functions, and following the FNEND statement for multiline functions.
- The last executable statement preceding the FNEND statement should be a RETURN, STOP, CHAIN, or unconditional GOTO to prevent control from passing to the FNEND statement.
- If a function definition alters the value of a variable that is referenced in the same statement that calls the function, unpredictable results may occur.
- A function may be defined in and referenced from a function key group. When a function is referenced from a key group, the system searches the current chain of statements, then the mainline program and definitions for keys 0-9 in order to find the function referenced.

Example

The following examples illustrate the execution of DEF statements:

```
10 DEF FNA (X) = X↑3/2
20 Y = 10
30 Z = FNA(Y)
```

After execution of statement 30, the variable Z will have the integer value 500.

In the next example, the variable R will have the integer value 72 after execution of statement 80. When statement 80 is executed, the current value of Y, which is 2, is substituted for each occurrence of the dummy variable X in the arithmetic expression of statement 100. Since the function FNC, defined in statement 100, uses the function FNB in its definition, the value 2 is substituted for each occurrence of X in the arithmetic expression of statement 90. The resulting value, 47, is then substituted for the function reference FNB(X) in statement 100. The current value of Y, which is 2, is then added to 47, and the resulting value of 49 is substituted for the function reference FNC(Y) in statement 80. This value is added to 23, and the resulting value of 72 is assigned to the variable R.

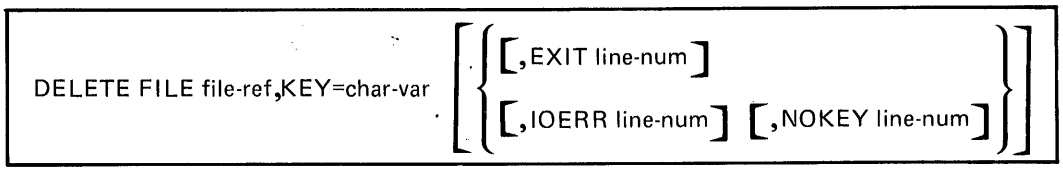
```
70 LET Y = 2
80 LET R = FNC(Y) + 23
90 DEF FNB(X) = 5*X**2+27
100 DEF FNC(X) = FNB(X) + X
```

The following example shows a multiline function definition. When these statements are executed, both C and D will have a value of 7.

```
10 A = 5
20 B = 2
30 DEF FNA (X,Y)
40 IF X>0 GOTO 60
50 RETURN X-Y
60 RETURN X+Y
70 FNEND
80 C = FNA (A,B)
90 D = FNA (A,B)
```

In the following example, the function returns the character string 'X SQUARED=4' when X=2:

```
10 DEF FNA$(X)='X SQUARED='||CHAR(X↑2)
```



DELETE FILE

You can use the DELETE FILE statement to logically delete a record from the index table according to the key field you specify. See *Key Indexed Access* under *Data Files and Access Methods* in Chapter 3. After the record is deleted, the file is positioned to a location immediately following the deleted record. The syntax of the DELETE FILE statement is as shown above, where:

file ref is FL0 to FL9 to identify the file containing the record to be deleted.

KEY=char-var specifies the key field in the record to be deleted. The file is searched for a matching key field, and the corresponding record in the index file is then deleted (see *Key Indexed Access under Data Files and Access Methods* in Chapter 3).

EXIT, NOKEY, and IOERR are error conditions and associated line numbers to which program control will transfer if the error occurs. EXIT is the line number of an EXIT statement, NOKEY indicates that a record with the specified key field cannot be found, and IOERR indicates that a hardware error prevents completion of this DELETE FILE statement.

Notes About DELETE FILE

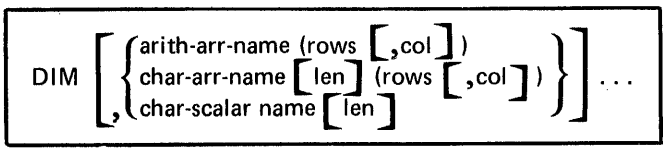
- The NOKEY and IOERR error conditions can be entered in any order.
- The file referenced in a DELETE FILE statement must have been opened with the ALL parameter in the OPEN FILE statement (see *OPEN/OPEN FILE* in Chapter 4). Otherwise, the DELETE FILE statement will cause an error.

Example

A sample DELETE FILE statement is shown below:

```
80 A$='ROWE'
90 DELETE FILE FL8,KEY=A$,NOKEY 999
```

In this example, the record with a key field equal to ROWE in file FL8 will be deleted. If the specified record cannot be found, program control will transfer to statement 999.



DIM

The DIM statement allows you to explicitly specify the size of arrays and character variables. The syntax of the DIM statement is as shown above, where:

arr-name is an arithmetic or character array to be dimensioned.

char-scalar is a character variable to which a length will be assigned.

rows,col are nonzero, unsigned integer constants specifying the dimensions of the arrays. One-dimensional arrays require only the rows entry. Two-dimensional arrays require both rows and columns entries separated by a comma.

len is the length of a character scalar, or the length of each element of a character array. This value can be from 1 to 255.

A one-dimensional array whose name is specified in a DIM statement is defined as having the number of elements represented by the rows entry. A two-dimensional array whose name is specified in a DIM statement is defined as having the number of rows and the number of columns entered in the statement.

The initial value of each arithmetic array element is zero; each character array element is initialized to blanks. If len is not specified, the default length of each character array element and character variable is 18 characters (see *Arrays* in Chapter 3).

Notes About DIM

- An array name cannot appear in a DIM statement if it has been previously defined, either implicitly or explicitly, in a USE statement or a prior DIM statement.
- An arithmetic or character array of one or two dimensions can be defined in a DIM statement.
- The maximum permissible size of each dimension in an array is 9999.
- Len can be specified as less than or equal to 255 characters.

Example

A sample DIM statement is as shown:

```
20 DIM Z$18(5), A(4,2), P$50
```

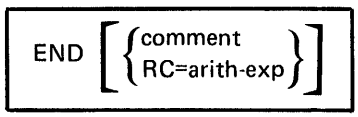
The result of the preceding statement is:

Z$ = five strings (array elements) of 18 blank characters each.

Array A has four rows of two columns each:

0	0
0	0
0	0
0	0

Character variable P$ has a length of 50 blank characters.



END

The END statement allows you to specify the logical end of a BASIC program and to terminate program execution. The syntax of the END statement is as shown above, where:

comment is optional.

RC = arith-exp is the return code used by the CSKIP command (see *Procedure File* in Chapter 3). This code can be accessed from file FLS (see *File FLS* in Chapter 3).

The END statement signifies that the program should be ended. It can be entered anywhere in a BASIC program. When an END statement is encountered during execution of a program, it causes all open files to be closed and it terminates processing. The actions of the END statement are identical to those of the STOP statement.

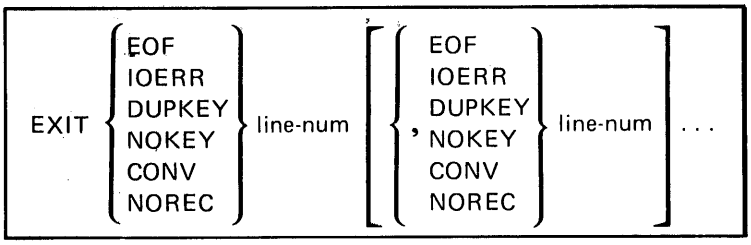
Notes About END

- The END statement is optional. If omitted, END is assumed by the system to follow the highest-numbered statement in the program.
- When you list the program (see *LIST Command* in Chapter 2), the system displays or prints a STOP statement to replace any END statement.
- If the value of the RC= parameter is less than zero or greater than 255, a value of zero is assumed.
- If RC= is not specified, the return code is set to zero.

Example

A sample END statement is as shown:

```
0910 END PG4
```



EXIT

The EXIT statement allows you to group in a single statement those error conditions (and their subsequent recovery line numbers) that can occur during the execution of input/output operations to files. The EXIT statement is nonexecutable and merely serves as a guide to the program, indicating the line number to which program control should transfer if an associated error should occur. If the EXIT parameter is entered for input/output statements, program control is transferred to the line number of the EXIT statement when an error occurs. The program then selects the appropriate error condition and transfers control to the corresponding line number as entered for the EXIT statement. The syntax of the EXIT statement is as shown above, where:

EOF, IOERR, DUPKEY, NOKEY, CONV, and *NOREC* are the error conditions that will transfer program control to their associated line number if the EXIT parameter in the input/output statement specifies a branch to this EXIT statement. For the various input/output statements, these error conditions have the following meanings:

EOF
For a *GET* statement, this error indicates that insufficient data remains in the file.

For a *READ FILE* statement, this error indicates that there are no more records in the file.

For a *PUT, WRITE FILE, PRINT,* or *CLOSE[FILE]* statement, this error indicates that there is insufficient file space to accommodate the data specified.

IOERR
For all input/output statements, this error indicates that a hardware error has prevented completion of the statement.

DUPKEY
For *WRITE FILE* statements specifying a key-accessed file, this error indicates that a record with the same key already exists in the referenced file.

NOKEY
For *READ FILE, REWRITE FILE, DELETE FILE,* and *RESET FILE* statements, this error indicates that no record matching the specified key can be found in the referenced file.

CONV For input/output statements referencing stream I/O files, this error indicates that a field cannot be converted to the type of variable specified.

For input/output statements referencing record I/O files, this error indicates one of the following:

- A value in a list of data cannot be converted to the format defined in the specified FORM statement.
- There are insufficient values in the record for the data items listed in the statement.
- There is insufficient space in the record to output the data items listed in the statement.
- A FORM specification specifies a position outside the defined record.
- A FORM statement referenced by a PRINT statement contains a format other than PIC or C.

NOREC This error indicates that the specified relative record number (in statements referencing record I/O files) is zero, negative, or greater than the relative record number of the last record in the file.

Notes About EXIT

- Each error condition can be entered only once in an EXIT statement.
- Error conditions can be entered in any order.
- Error clauses apply only to an error on that particular I/O statement. They do not apply to other statements accessing the same opened file. (See the various FILE statement formats.)

Example

A sample EXIT statement is shown below:

```
80 EXIT EOF 200,IOERR 220,NOKEY 240,NOREC 260
```

In this example, an input/output statement referencing line number 80 for the EXIT parameter will cause program control to transfer to line number 200 if an EOF condition caused the error, to line number 220 if an IOERR caused the error, to line number 240 if the specified key could not be found, or to line number 260 if the error was caused by an improper relative record number.

FNEND

For a complete description of the FNEND statement, see *DEF, RETURN, FNEND*.

```
FOR control-var=arith-exp TO arith-exp [ STEP arith-exp ]

NEXT control-var
```

FOR AND NEXT

Together, a FOR statement and its paired NEXT statement delimit a FOR loop—a set of BASIC statements that can be executed a number of times. The FOR statement marks the beginning of the loop and specifies the conditions of its execution and termination. The NEXT statement marks the end of the loop.

The syntax of the FOR and NEXT statements is as shown above, where:

control-var is a simple arithmetic variable.

arith-exp are expressions that specify an initial value for the control variable, the final value of the control value (at which execution of the loop will end), and the amount that the control variable will increase after each execution of the loop. If STEP and the last arithmetic expression are omitted, an increment of 1 is assumed.

Upon execution of these statements, all expressions are evaluated. The initial value of the control variable is tested against the final value of the control variable. If the initial value is greater than (less than for negative increments) the final value, the loop is not executed. Instead, the value of the control variable is left unchanged, and control goes to the statement following the NEXT statement.

If the loop is executed, the control variable is set equal to the initial value, and the statements in the loop are executed. When the NEXT statement is executed, the specified increment is added to the control variable, which is then compared with the specified final value. If the control variable is still less than (greater than, for negative increments) or equal to the final value, the loop is executed again and the cycle continues until an increment is made that renders the control variable greater than (less than for negative increments) the specified final value. At that time, the control variable is set back to its last value, and control falls through to the first executable statement following the NEXT statement.

Notes About FOR and NEXT

- If the optional STEP arithmetic expression is omitted in the FOR statement, the increment value is automatically set to +1.
- The value of the control variable can be modified by statements within the FOR loop, but its initial value, final value, and increment are established during the initial execution of the FOR statement and are not affected by any statement within the FOR loop.

- If the initial value to be assigned to the control variable is greater than (less than for negative increments) the final value when the FOR statement is evaluated, the loop is not executed, no value is assigned to the control variable, and execution proceeds with the first executable statement following the associated NEXT statement.

- If the value of the STEP arithmetic expression is zero, the FOR loop is executed an infinite number of times, or until the value of the control variable is purposely set beyond the specified final value.

- Transfer of control into or out of a FOR loop is permitted; however, a NEXT statement cannot be executed unless its corresponding FOR statement has been executed previously.

- FOR loops can be nested within one another as long as the internal FOR loop falls entirely within the external FOR loop (see the following example). Nested FOR loops should not use the same control variable.

- The maximum number of nested FOR loops is 15.

- If a program with an active FOR/NEXT loop is interrupted with ATTN, unpredictable results can occur if the program is saved.

Example

The following example shows a simple FOR loop that increases the control variable by 2 until the value of 25 is exceeded.

```
20 FOR I = 1 TO 25 STEP 2
 .
 .
 .
90 NEXT I
```

The next example shows the correct technique for nesting FOR loops. The inner loop is executed 100 times for each execution of the outer loop.

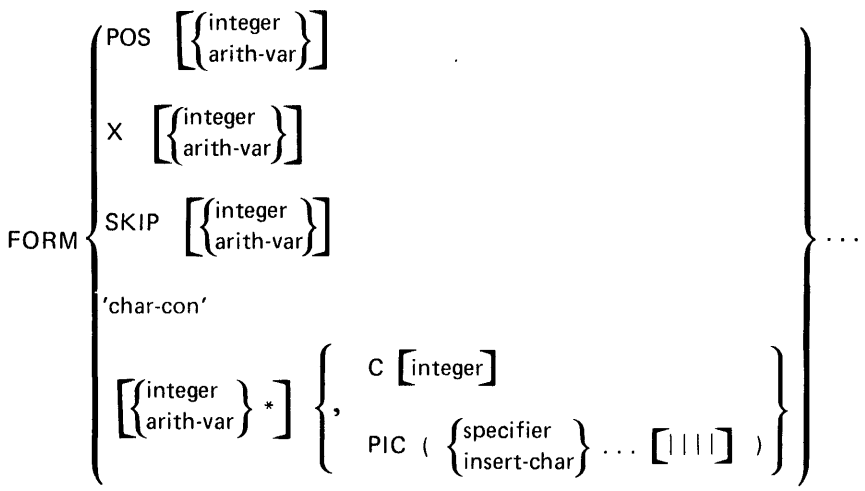
```
10 FOR J = A TO B STEP C
 .
 .
 .
150 FOR K = 1 TO 100
 .
 .
 .
250 NEXT K
 .
 .
 .
300 NEXT J
```

FORM

The FORM statement allows you to specify a format for both printed/displayed output and records in record I/O files. For the purpose of explanation, the FORM statement will be described as two separate statements: print formatting with the FORM statement, and record formatting with the FORM statement.

Print Formatting with the FORM Statement

When used to format printed output, the FORM statement has the following syntax:



where:

POS indicates the position in the line for the next value to be printed or displayed. This entry can be from 1 to the extent of line length. You can specify an integer constant or arithmetic variable for line position. If the truncated value of your entry is less than 1, this control specification is not used. If your entry exceeds total line length, the current line is displayed or printed, and line position is reset to the first position of the next line. Default value for this parameter is 1. POS can be used to position the cursor on the screen for a succeeding READ FILE statement.

X indicates the number of blanks to be displayed or printed. This parameter allows you to insert blank fields into displayed or printed data. Default value for this parameter is 1. It is not permitted as the first element in the FORM statement.

SKIP indicates the number of lines to be skipped when you are displaying or printing data. After the current line is printed or displayed, the next line begins at the SKIP value you entered, minus 1 (for the current line). Thus, if you enter 10 for SKIP, the system displays/prints the current line, then skips 9 lines, positioning the next line at line number 11. After the SKIP operation, the next output begins at the first position in the line. Default value for this parameter is 1. SKIP must be the last parameter in the FORM statement to cause printing.

'char-con' are character constants (enclosed in single quotation marks) that will be written exactly as entered when the FORM statement is used. If the character constant is used in a FORM statement for input, it will cause the corresponding positions in the record to be skipped over; for example, Xn, where n is the length of the character constant.

*integer/arith-var** indicates the replication factor for the data format that follows. This value indicates the number of times that the data format should be used. Thus you can use the same format repeatedly. This parameter must be greater than 0. Arith-var must be a single character variable.

C indicates the length of a displayed or printed field into which a corresponding character expression (in the PRINT statement) is to be output. The character expression is truncated or filled with blanks on the right to the length you enter for C. Default value for this parameter is the length of the character expression in the PRINT statement.

PIC indicates the length and conversion for a displayed or printed field into which a corresponding numeric expression (in the PRINT statement) is to be output. The PIC parameter can be up to 32 characters containing three specifications: the digit specifier, the insertion character, and the exponent specifier.

Digit Specifier

The digit specifiers are:

Specifier	Meaning
#	A digit must always appear in this position.
Z	Replace a leading zero with a blank.
*	Replace a leading zero with an asterisk.
$	Floating dollar sign. A dollar sign is to be printed immediately before the first significant digit.
+	Floating sign. A plus sign for a positive number, or a minus sign for a negative number, is to be printed immediately before the first significant digit.
-	Floating minus sign. A blank for a positive number, or a minus sign for a negative number, is to be printed immediately before the first significant digit.
CR	These positions can be used at the end of the PIC string to indicate a credit amount. If the value is negative, the characters CR will be printed after it. If the value is positive, the characters CR are replaced with two blanks.
DB	These positions can be used at the end of the PIC string to indicate a debit amount. If the value is negative, the characters DB will be printed after it. If the value is positive, the characters DB are replaced with two blanks.

The following are examples of digit specifiers. Assume a data item value of 112233 is to be printed.

PIC Specification	Printed Output
PIC(##########)	000112233
PIC(ZZZZZZZZZ)	112233
PIC(ZZZZZZ###)	112233
PIC(******###)	***112233
PIC($$$$$$###)	$112233
PIC(++++++###)	+112233
PIC(---######)	112233

If a floating dollar sign, plus sign, or minus sign is specified only once in a PIC specification, it does not float through the field but instead is printed in the indicated position. For example:

PIC Specification	Printed Output
PIC($ZZZZZ###)	$ 112233
PIC(+ZZZZ###)	+ 112233

Insertion characters

Insertion characters insert additional characters into a field, generally to improve readability. The following insertion characters can be specified:

Character	Meaning
B	Print a blank unconditionally.
,	Print a comma conditionally (only if a digit precedes the comma).
/	Print a slash conditionally (only if a digit precedes the slash).
.	Print a decimal point conditionally (if the value to be printed is nonzero and zero suppression is not in effect)
+	Trailing sign. When the + appears in the rightmost position of a PIC specification, it is treated as a trailing sign. A plus sign is printed for a positive number, a minus sign for a negative.
-	Trailing minus sign. When the - appears in the rightmost position, it is treated as a trailing sign. A minus sign is printed for a negative number, a blank for a positive number.

The following are examples of insertion characters. Assume a data item value of 112233 is to be printed:

PIC Specification	Printed Output
PIC(###B##B####)	000 11 2233
PIC(ZZZBZZBZ###)	11 2233
PIC(ZZZ,ZZZ,###)	112,233
PIC(ZZZZZ/Z#/##)	11/22/33
PIC(******#.##)	*112233.00
PIC($$$$$$###+)	$112233+
PIC($$$,$$$,$$$.##)	$112,233.00

Exponent Specifier

The exponent specifier appears in the four low-order positions of a data format as | | | |. The corresponding display-print positions are then: the letter E, the exponent sign (+ or -), and the two-digit exponent value. Use of an exponent specifier eliminates zero suppression. Thus, a previously defined decimal point will always appear in a field defined by a format specification containing the exponent specifier.

Notes About Printing/Displaying with FORM

- The line number of a FORM statement can be specified in a PRINT USING statement.
- A FORM statement can appear anywhere in a program.
- Array items are formatted in row order.
- If values in the PRINT statement exceed format specifications in the FORM statement, the format specifications are reused from the beginning of the FORM statement until PRINT values are exhausted.
- If a value in a PRINT statement exceeds the line width of the display or printer, the excess is displayed or printed beginning in the first position of the next line.
- Control specifications (POS,X, and SKIP) can be intermixed with data formats (C and PIC).
- Any data format can be preceded by a replication factor for repeated use of the format.
- SKIP is required as the last parameter in the FORM statement if the line is to be printed.
- Any control specifications (X,POS, or SKIP) after the last used conversion specification will be processed.

Example

A sample FORM statement is shown below:

```
30 FORM C18,PIC($$####.##)
```

When used with this PRINT statement (where B=999.09):

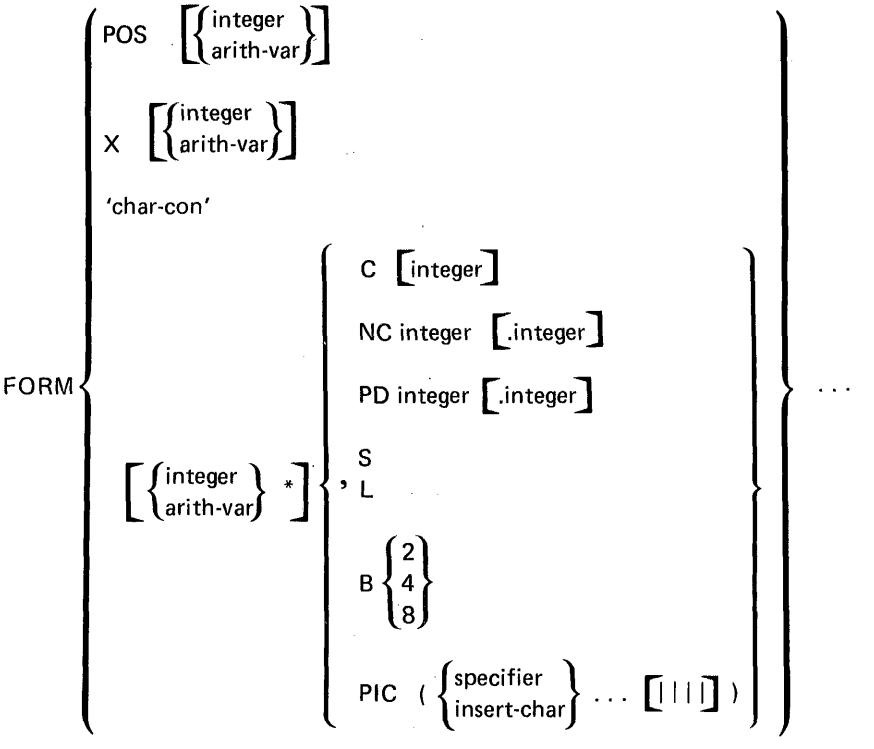
```
190 PRINT USING 30 FLP, 'BALANCE DUE=',B
```

the resulting output is:

```
BALANCE DUE= (7 blanks)$0999.09
```

Record Formatting with the FORM Statement

When used to format record data in record I/O files, the FORM statement has the following syntax:



where:

POS indicates the position in the record that you want to access (with READ FILE or REREAD FILE statements) or write to (with WRITE FILE or REWRITE FILE statements). This parameter allows you to select a specific area of the record for input or output. The value of this parameter can be from 1 to the length of the record. If the value is greater than the record length, it will cause an error. If the value is an arithmetic variable, the truncated integer portion of the value is used to determine the record position.

X indicates the number of positions to space forward in the record. This parameter allows you to space over unwanted portions of the record. If X causes a space beyond the end of the record, it will cause an error. If the value is an arithmetic variable, the truncated integer portion of the value is used to determine the number of forward spaces. The default value is 1.

char-con are character constants (enclosed in single quotation marks) that will be written exactly as entered when the FORM statement is used. If the character constant is used in a FORM statement for input, it will cause the corresponding positions in the record to be skipped. For example, FORM 'ABCD', B2 will read a 2-byte binary number beginning in position 5 of the record.

*integer/arith-var** indicates the replication factor for the format specification that follows. This value indicates the number of times that the format specification should be reused, which allows you to use the same specification repeatedly. This parameter must be greater than 0 and defaults to 1. If the replication factor is an arithmetic variable, the truncated integer portion is used to determine the number of times the specification should be reused.

C indicates the field length for character data. For input (with a READ FILE or REREAD FILE statement), the specified number of characters are assigned from the record to the corresponding character variables listed in the READ FILE or REREAD FILE statement. If the character variable length is less than the number of characters to be assigned, the number of characters specified is assigned and the excess characters are spaced over. If the variable length is more than the number of record characters to be assigned, the variable is padded on the right with blanks.

For output, the specified number of characters are assigned into the record from the data variables listed in the WRITE FILE or REWRITE FILE statement. If the number of characters to be assigned is less than the number you specify, the characters are padded on the right with blanks to the number you specified before being assigned. If the number of characters to be assigned exceeds the number you specify, only the specified number of leftmost characters are written into the record.

The C parameter is valid only for character data and will cause an error if used with numeric data. If a width is not entered, the number of characters equal to the length of the specified variable is written.

NC indicates the conversion of numeric data, NC integer[.integer]. For input, the next number of record positions you specify (integer) contain a numeric value that will be converted to BASIC internal numeric representation and assigned to a variable specified in the READ FILE or REREAD FILE statement. In the record, the numeric value is a string consisting of digits in combination with any of the characters $, +, -, *, /, b, comma, decimal point, or exponential notation (E ± numeric constant). Zoned decimal fields can also be read. The optional parameter (.integer) indicates the number of decimal positions in the field, and will override an explicit decimal point in the input field. If this causes the position of the decimal point to change, the value is also changed.

For output, an arithmetic value from an expression in the WRITE FILE or REWRITE FILE statement is converted to a signed, zoned decimal field of the length specified and placed in the record. The value is rounded if necessary. The optional integer (.integer) indicates the decimal positions that will be present in the record field; otherwise, all specified positions will represent the integer portion of the arithmetic value.

PD indicates the packed decimal format for numeric values, PD integer [.integer].

where:

integer is the width of the field in characters and *.integer* is the number of digits to the right of the decimal point. *.integer* must be ≤2**integer*-1.

For input, the number you enter (integer) identifies the positions in a record containing a numeric value in packed decimal form (two digits per position, with one digit and a sign in the low-order position). This value will be converted into BASIC internal representation and moved to a corresponding numeric variable in the READ FILE or REREAD FILE statement. The optional specification (.integer) identifies the number of decimal positions in the number. If the .integer specification is not entered, the fractional number is assumed to be zero.

For output, the number you enter (integer) indicates the record positons into which the corresponding numeric expression from the WRITE FILE or REWRITE FILE statement will be placed. The expression is first converted to packed decimal format (rounded if necessary) with the optional number (.integer) of fractional digits you specify.

B indicates the length (2, 4, or 8 bytes) of numeric data items in fixed-point signed binary integer format that are to be converted to BASIC internal data format. For record I/O file input, the next 2, 4, or 8 bytes in the record contain a signed binary value to be converted by the system into internal data format and assigned to the variable(s) specified in the READ FILE or REREAD FILE statement using a FORM statement.

For record I/O file output, the value of an expression in a WRITE FILE or REWRITE FILE statement using a FORM statement is converted by the system to fixed-point signed binary integer format, according to the length you specified (2, 4, or 8 bytes), and placed into the record. The maximum integer value that can be contained in 2, 4, and 8 bytes is listed as follows:

- 2 byte maximum is ± 32,767
- 4 byte maximum is ± 2,147,483,648
- 8 byte maximum is ± 999,999,999,999,999

S indicates short precision (4 characters) for numeric values. For input, this entry indicates that a four-position, short-precision value in the record is to be assigned to a corresponding numeric variable specified in the READ FILE or REREAD FILE statement. The value is extended on the right with zeros before being assigned to the variable.

For output, this entry indicates that a numeric expression in the WRITE FILE or REWRITE FILE statement will be written in the record without conversion and in short-precision format.

L indicates long-precision (8 characters) for numeric values.

For input, this entry indicates that an eight-position, long-precision value in the record is to be assigned without conversion to a corresponding numeric variable specified in the READ FILE or REREAD FILE statement.

For output, this entry indicates that a numeric expression in the WRITE FILE or REWRITE FILE statement will be written in the record without conversion and in long-precision format.

PIC indicates the format for numeric expressions in output statements only. The string of data (up to 32 characters) enclosed in parentheses consists of one of the following characters for each record position occupied by an edited numeric field. These characters can be digit specifiers, insertion characters, and the exponent specifier.

Digit Specifiers

Digit specifiers can be conditional or unconditional. They are:

Specifier	**Meaning**
#	This position must always contain a numeric digit.
Z	A leading zero in this position is replaced by a blank.
*	A leading zero in this position is replaced by an asterisk.
$	This character is placed in each position that can potentially contain a floating dollar sign; that is, a dollar sign to the immediate left of the first significant digit. Nonsignificant zeros are suppressed.
+	This character is placed in each position that can potentially contain a floating high-order sign. Nonsignificant zeros are suppressed.
-	This character is placed in each position that can potentially contain a floating high-order minus sign if the value in the record is negative. Nonsignificant zeros are suppressed.
CR	These positions can be used at the end of the PIC string to indicate a credit amount. If the value is negative, the characters CR will be printed after it. If the value is positive, the characters CR are replaced with two blanks.
DB	These positions can be used at the end of the PIC string to indicate a debit amount. If the value is negative, the characters DB will be printed after it. If the value is positive, the characters DB are replaced with two blanks.

Insertion Characters

Insertion characters are conditional or unconditional. They are:

Character	Meaning
B	This unconditional character always causes a blank to be inserted in the corresponding position of the record.
,	This character is inserted in the corresponding position of the record unless zero suppression is in effect and no significant digits appear to the left of the comma in the record. In this case, the comma will be replaced by a floating or zero suppression character.
/	This character is inserted in the corresponding position of the record unless zero suppression is in effect and no significant digits appear to the left of the slash in the record. In this case, the slash will be replaced by a floating or zero suppression character.
.	This character is inserted in the corresponding position of the record unless zero suppression has been specified for every digit position and the value is zero. In this case, the decimal point will be replaced by a floating or zero suppression character.
Trailing +	This character causes a plus sign or a minus sign to be inserted in the corresponding position of the record unless zero suppression is in effect and no significant digits appear to the left of the sign in the record. In this case, the sign will be replaced by an asterisk or a blank

Character	Meaning
Trailing -	This character is inserted in the corresponding position of the record if the value to be displayed is negative, unless zero suppression is in effect and no significant digits appear to the left of the minus sign in the record. In this case, the minus sign will be replaced by an asterisk or a blank.
Trailing $	This character is inserted in the corresponding position of the record unless zero suppression is in effect and no significant digits appear to the left of the dollar sign in the record. In this case, the dollar sign will be replaced by an asterisk or a blank.

Exponent Specifier

The exponent specifier | | | | causes the following sequence of characters to be placed in the corresponding positions of the record:

1. The letter E
2. The exponent sign (plus or minus)
3. Two digits representing the value of the exponent

These characters do not appear in the record when zero suppression is in effect and the value to be placed in the record equals zero.

Notes About Record Formatting with the FORM Statement

- The maximum number of image or FORM statements permitted in a single BASIC program is limited only by the amount of storage available.

- FORM statements are nonexecutable and may be placed anywhere in a BASIC program, either before or after the I/O statements that refer to them. However, they should not appear within a multiline function definition (between a DEF statement and its associated FNEND statement) to maintain compatibility with other BASIC systems.

- A PIC specification in a FORM statement may not contain both the Z and the * digit specifiers.

- A PIC string must be from 1 to 32 characters long.

- A single $, +, or - as the leftmost character in a PIC string is treated as a static character. Two or more $, +, or - signs at the leftmost end of a PIC string are treated as floating characters. The same character cannot appear as both a static character and part of a floating character string in a single PIC string.

- A string of floating characters must contain at least one more floating character than the maximum number of expected digits in the output field.

- A PIC string cannot end with a B, slash (/), or comma (,) insertion character.

- A PIC string cannot begin with a slash (/) or comma (,) insertion character.

- There cannot be more than one decimal point (.) insertion character in a PIC string.

- A PIC string must include at least one #, Z, *, or floating string.

- No # digit specifiers may appear to the left of a zero suppression character or a floating character.

- A # digit specifier may not appear in a PIC string that contains a decimal point followed by zero suppression or floating characters.

- The symbols + and - cannot appear in the same PIC string.

- A trailing character may not appear in a PIC string in which that trailing character is used as either a static character or as part of a floating character string.

Examples

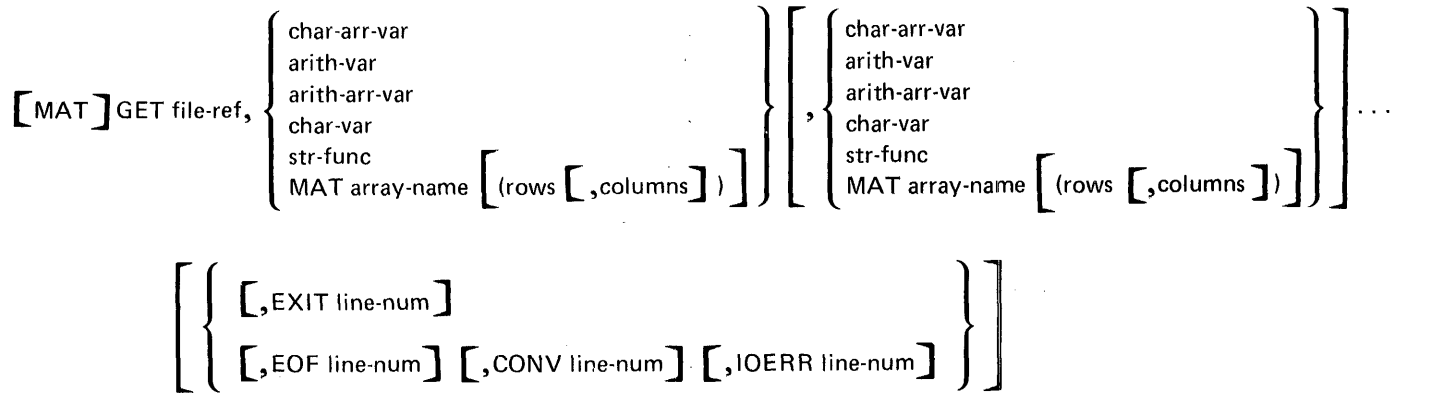
The following are examples of record formatting with the FORM statement:

```
25 READ FILE USING 30, FL1,KEY=N$, A$, G

30 FORM X25, C, X10, NC7
```

In the first example, the record that satisfies the key value in the variable N$ is read. Two values from the record are put into the variables described in the READ FILE statement. One is a character value placed into A$; the other is a numeric value placed into G. The FORM statement causes the first 25 positions of the record to be skipped and the number of characters equal in length to the variable A$ (18 by default in this example) to be read into A$. Ten more positions are skipped, and the next seven positions in the record are converted and read into the numeric variable G.

```
110 REWRITE FILE USING 100, FL1, A$, M

100 FORM X25, C, POS150, PIC(Z##.##)
```

In the second example, two values from the variables in the REWRITE FILE statement are entered into a record in the file reference FL1. The first 25 positions in the record are skipped, a character value equal in length to the variable A$ is inserted into the record, and, in position 150 of the record, the numeric value in M is inserted according to the PIC specification.



[MAT] GET

The GET statement allows you to assign values from a specified stream I/O file to referenced variables. The file containing the assigned values must already have been opened for input by an OPEN statement. The format of the GET statement is as shown above, where:

file-ref is from FL0 to FL9 to identify the file containing the values to be assigned. This entry must be the same as that specified in an OPEN statement.

var is a simple variable, a subscripted array reference, a whole array (preceded by MAT), or a substring reference. Only one variable is required. If more than one is entered, they must be separated by commas. If only arrays are referenced, the keyword MAT can precede GET.

EXIT is the line number of an EXIT statement for error recovery.

EOF, CONV, and IOERR are error recovery exits (see *EXIT*).

When the GET statement is executed, the values from the specified file (FL0-FL9) are assigned to the referenced variables. Subsequent GET statements for the same file cause the values in the file to be assigned beginning at the current file position.

Subscripts in variable references are evaluated as they occur, from left to right. Thus, an assigned variable in a GET statement may be used as the subscript of another variable in the same statement.

When a MAT GET statement is executed, or a MAT array name is used in a GET statement, the file values are assigned to the specified arrays row by row. The referenced file must have been previously opened for input by an OPEN statement. The file is positioned at its beginning for the first [MAT] GET statement, unless one or more GET operations were already executed. Subsequent [MAT] GET statements for the same file cause values to be assigned from the current file position.

If the optional *rows, columns* entries follow the array names in the [MAT] GET statement, the truncated integer portions of the expression values are used to redimension the arrays before data values are assigned from the file.

Notes About [MAT] GET

- A file currently activated as an output file cannot be specified in a [MAT] GET statement. It must first be closed, then reopened (in an OPEN statement as an input file).
- Each value assigned in a [MAT] GET statement must be of the same data type (arithmetic or character) as the corresponding variable.
- Referenced arrays can be redimensioned (see *Redimensioning Arrays* in Chapter 3).
- A [MAT] GET statement referencing a currently closed file causes program execution to be terminated.
- The keyword MAT must precede referenced arrays unless all references in the data list are arrays. In this case, use the keywords MAT GET.

Examples

A sample GET statement is as shown:

```
0090 GET FL4, X, Y, Z, MAT A(4), D$, EOF 620
```

In this example, the values in file FL4 are assigned, respectively, to variables X, Y, Z, the four elements of array A, and the character variable D$. On end of file, the program branches to statement number 620.

The following statements have the same meaning:

```
0090 GET FL1, MAT A, MAT B, MAT C
0090 MAT GET FL1, A, B, C
```

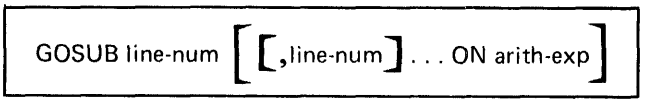
Note: Arrays cannot be intermixed with other variables unless they are preceded by MAT.

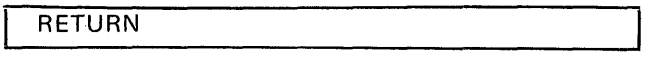
```
0090 GET FL1, A$, MAT B, D$
```

The following example shows a MAT GET statement:

```
0090 MAT GET FL2, A,B (5,10), Z(4,5), EOF 210
```

In this example, array A, array B (redimensioned to 5 rows, 10 columns), and array Z (redimensioned to 4 rows, 5 columns) will receive values from file FL2. When end of file is reached, control will transfer to statement 210.





GOSUB AND RETURN

The GOSUB and RETURN statements are used together to create subroutines. The GOSUB statement transfers control conditionally or unconditionally to a specified statement. The RETURN statement transfers control to the first executable statement following the last active GOSUB statement that was executed.

The syntax of the GOSUB statement is either simple or computed. The simple syntax is:

GOSUB line-num

where:

line-num is the number of the statement to which control is to be transferred.

The computed GOSUB syntax is:

GOSUB line [,line-num] ...ON arith-exp

where:

line-num is a statement number. At least one statement number is required.

arith-exp determines the statement to which control is passed.

The format of the RETURN statement is simply RETURN.

Execution of a simple GOSUB statement causes an unconditional transfer of control to the statement whose number is specified.

Execution of a computed GOSUB statement causes the arithmetic expression to be evaluated and control transferred to the statement whose numeric position in the list of statement numbers (reading left to right) is equal to the truncated integer value of the expression. Thus, an expression with a value of 2.75 would cause control to be transferred to the second statement in the list. If the expression has a truncated integer value less than 1 or greater than the total number of statements listed, control falls through to the first executable statement following the computed GOSUB statement.

When a GOSUB statement transfers control to a nonexecutable statement, control is transferred to the first executable statement following the specified nonexecutable statement.

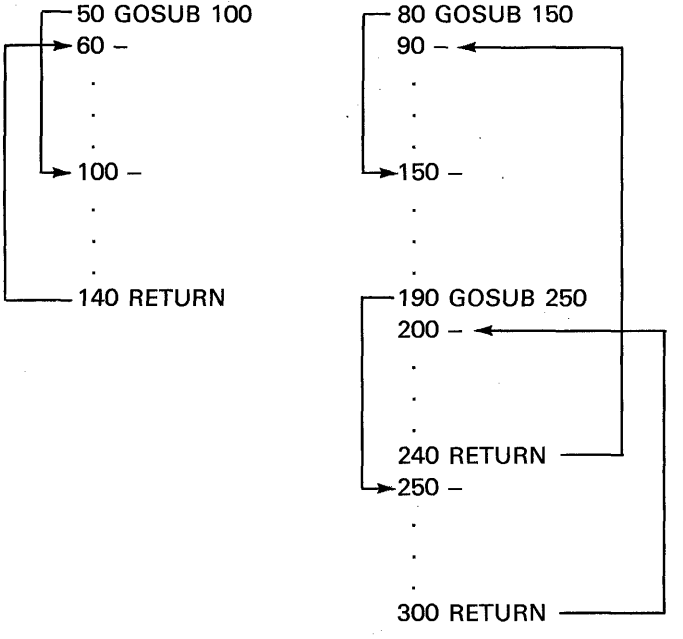
If the line number in a GOSUB statement is 999x, the system will call the corresponding defined function key group. If the function key group at 999x is undefined, the system transfers control to the indicated line number (999x). The called function key group must be defined as a REM key group.

Execution of a RETURN statement causes an unconditional transfer of control to the first executable statement following the last active GOSUB statement that was executed.

The maximum number of nested subroutines is 20, minus 1 for each keys function called (see *Function Keys* in Chapter 2).

Example

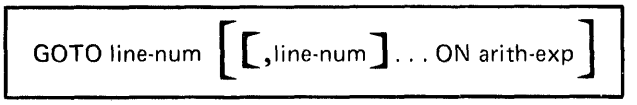
The following examples show the execution of GOSUB and RETURN statements:



The following examples show a GOSUB to a keys function:

```
100 GOSUB 9994          LOAD0,KEY4
.                       9994 REM
.                       0010 PRINT'IN KEYGROUP 9994'
.                       0020 RETURN
.
9994 PRINT 'IN MAIN PROGRAM'
9995 RETURN
```

In this example, statement 100 causes the program to branch to the REM function defined for key 4. If key 4 is not defined or is defined as something other than REM, the program seeks statement 9994 within the main program.



GOTO

The GOTO statement transfers control either conditionally or unconditionally to a specified statement.

The syntax of the GOTO statement can be simple or computed. The simple syntax is:

```
GOTO line-num
```

where:

line-num is the number of the statement to which control is to be transferred.

The computed GOTO syntax is:

```
GOTO line-num [,line-num] ...ON arith-exp
```

where:

line-num is a statement number. At least one statement number is required.

arith-exp determines the statement to which control is passed.

Execution of a simple GOTO statement causes an unconditional transfer of control to the statement number specified.

Execution of a computed GOTO statement causes the arithmetic expression to be evaluated and control transferred to the statement whose numeric position in the list of statement numbers (reading left to right) is equal to the truncated integer value of the expression. Thus, an expression with a value of 2.75 would cause control to be transferred to the second statement in the list. If the expression has a truncated integer value less than 1 or greater than the total number of statements listed, control falls through to the first executable statement following the computed GOTO statement.

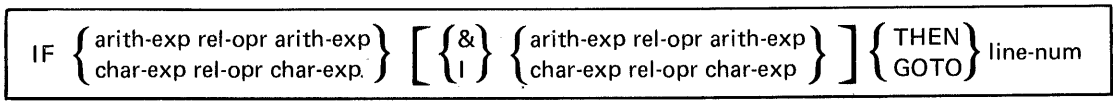
When a simple or computed GOTO statement transfers control to a nonexecutable statement, control is then passed to the first executable statement following the specified nonexecutable statement.

The following statement will unconditionally pass control to statement number 20:

```
100 GOTO 20
```

If X = 4, the following statement will pass control to statement number 60:

```
50 GOTO 40, 60, 15, 100 ON (X+4)/4
```



IF

The IF statement allows you to transfer program control according to the result of an evaluated expression. The syntax of the IF statement is as shown above, where:

arith-exp and *char-exp* are arithmetic or character expressions. Only one pair of expressions is required.

rel-opr is a relational operator. Only one operator is required.

& is the symbol entered for logical AND. | is the symbol entered for logical OR.

THEN and *GOTO* specify that control should be transferred. If you enter THEN, each time the program is listed on the display screen or printer, the system substitutes GOTO for THEN.

line-num is the number of the statement to which control is transferred if the relational expression(s) is true.

When an IF statement is executed, the expressions are compared as specified by the relational operator. If the relationship is true, control is transferred to the specified statement number. If the relationship is not true, control is passed to the first executable statement following the IF statement.

If & is specified, both relational expressions must be true before control passes to the statement number. If | is specified, control passes to the specified line number if either relational expression is true.

If the specified relationship is true and the specified statement is nonexecutable, control is passed to the first executable statement following the specified nonexecutable statement.

Notes About IF

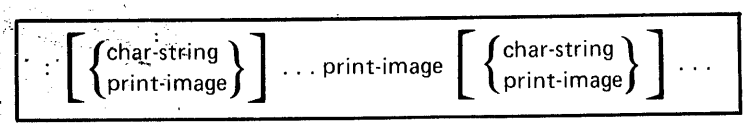
- The expressions being compared within the relational expressions must contain data of the same type (character or arithmetic).
- THEN and GOTO are interchangeable in the IF statement. Either can be used, but not both. GOTO is stored by the system.
- Comparison depends upon the comparison tolerance value (see *Comparison Tolerance* in Chapter 3).

Examples

The following examples show a variety of IF statements:

```
30 IF A(3)≠ X+2/Z GOTO 85

40 IF R$ = 'CAT' GOTO 70

50 IF S2 = 37.222 GOTO 120

60 IF X>Y GOTO 90

70 IF A>B | C>D GOTO 110

80 IF A$ = 'JOB' & B$ = 'DATE'GOTO 100
```

In statement 40, for example, if character variable R$ contains the word CAT, program control is passed to statement 70. In statement 70, if either A>B or C>D, control passes to statement 110.



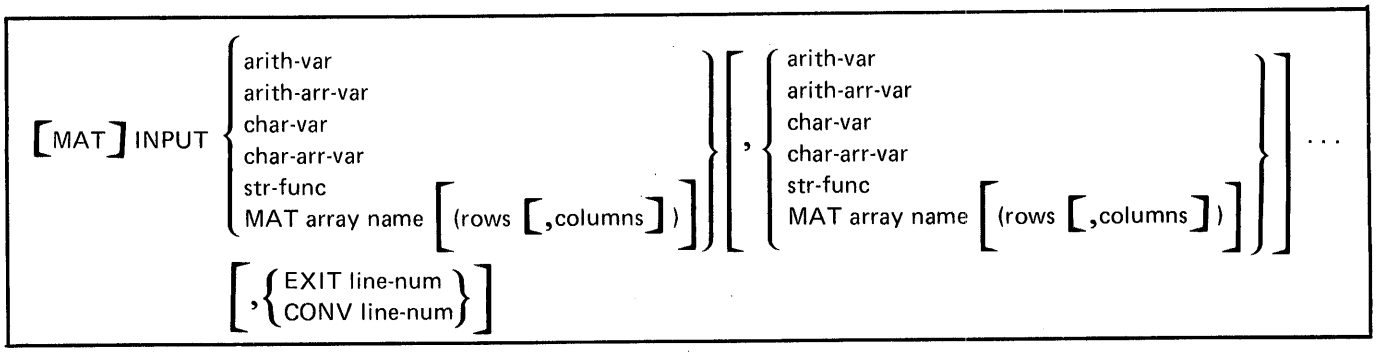
IMAGE

The image statement is used to control formatting of printed or displayed data. For a complete description of the image statement, see *[MAT] PRINT USING and Image/FORM.*

The image statement formatting specifications can also be assigned to a character variable, as shown below:

```
90 A$='TOTAL COST IS $###.##' or 90 : TOTAL COST IS $######
100 PRINT USING A$, FLP, T1  or 100 PRINT USING : 90, FLP, T1
```

In this example, variable A$ must have been previously dimensioned to the length of its assigned image specifications.



[MAT] INPUT

The INPUT statement allows you to assign values to variables from the keyboard or procedure file while your program is being executed. The syntax of the INPUT statement is as shown above, where:

var are simple arithmetic or character variables, subscripted references to an array element, whole arrays (preceeded by MAT), or substring references. The rows and columns references to an array element must be enclosed in parentheses. Only one variable is required in a [MAT] INPUT statement, although many variables can be specified. If only arrays are referenced, the keyword MAT can precede INPUT.

CONV line-num allows you to specify a line number to which program control will be transferred if you enter improper data for an INPUT statement (such as character data missing a quotation mark or an out-of-range numeric).

EXIT line-num is the line number of an EXIT statement for error recovery (see *EXIT*).

When an INPUT statement is executed, it displays a question mark on the display screen, and program execution halts. You must then enter a list of values that will be assigned, in the order they are entered, to the variables listed in the [MAT] INPUT statement or row-by-row to elements of specified arrays. When the complete list has been entered, press the EXECUTE key to resume program execution. If you entered the IN=P option in the RUN command, values are supplied from the active procedure file (see *Procedure File* in Chapter 3). Each [MAT] INPUT statement will get at least one record from the procedure file. If one record does not supply enough values for the data list, additional records will be read. If extra values are present in a record, they are ignored.

Subscripts of array variables in the INPUT statement are evaluated as they occur. Thus, an assigned variable in an INPUT statement can be used subsequently as the subscript of another variable in the same statement.

A character constant shorter than the defined length of the character variable to which it is assigned is padded on the right with blanks to the defined length before being assigned. Character constants longer than the defined length are truncated on the right before being assigned. The maximum length is 255 characters. Character constants containing no characters (null) are assigned as the defined length of blanks.

Notes About INPUT

- Each value entered must be of the same data type (character or arithmetic) as the corresponding variable reference in the [MAT] INPUT statement. Data types can be mixed in the same statement.
- Blanks within *numeric* data items are ignored.
- Each value entered must be separated from the next value by a comma. Two consecutive commas are ignored. To end the series, press the EXECUTE key.
- Character data must be enclosed in single quotation marks only if it contains commas, or if leading blanks are significant.
- If quotes, blanks, or commas are part of the input string, see *Write File FLS* in Chapter 3.
- The number of values entered at execution time must be equal to the number of variable references specified in the [MAT] INPUT statement. If you do not enter enough values, the program will request more input. If you enter too many values, the extra entries will be ignored by the program.
- Referenced arrays can be redimensioned (see *Redimensioning Arrays* in Chapter 3).
- The keyword MAT must precede referenced arrays unless all references in the data list are arrays. In this case, use the keywords MAT INPUT.
- The 'OUT' character (located above the [-/+] key) can be used to exit a program when input data is requested.

Examples

A sample INPUT statement is as shown:

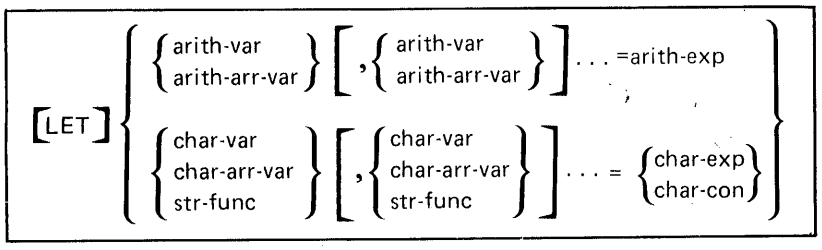
```
60 INPUT A$, B, X, Y(X), CONV300

'YES', 18.6, 8, 1.3597E-6
```

When line 60 is executed, you can enter the list of values shown. The program will branch to statement 300 if, for example, you entered character data for the second value.

The following example shows a MAT INPUT statement:

```
0100 MAT INPUT A, A$, B(3,6)
```

Upon execution of this statement, you must enter values for arrays A, A$, and B (redimensioned to 3 rows, 6 columns).



LET

The LET statement allows you to assign the value of an expression to one or more variables. The syntax of the LET statement is as shown above, where:

var are scalar variable names, subscripted to references to array elements, or substring references. Only one variable is required, although many can be specified. The row and column references to an array element must be enclosed in parentheses.

exp is an arithmetic or character expression, a string function, or a character constant.

When the LET statement is executed, the expression is evaluated, and the resulting value is assigned to the specified variables from left to right.

Character constants are limited only by line length (64 characters). Character expressions can be 1 to 255 characters in length. Character variables can be dimensioned to a specific length by using the DIM statement. The initial value of undimensioned character variables is set to 18 blank characters. When assigned to a character variable, a character constant is adjusted to the length of the variable. Longer constants are truncated, while shorter constants are left-justified and padded with blanks on the right.

Notes About LET

- Data values to the right of the equal sign must be of the same type (arithmetic or character) as the variables to which they are assigned.
- The keyword LET is optional.
- Hexadecimal constants can also be assigned to character variables in LET statements. The constant must begin with X' followed by an even number of characters 0 to 9 and A to F. The end of the constant must be indicated by a single quotation mark ('). Each pair of characters indicates a character to be assigned.
- Subscripted references to array elements are permitted in the assignment statement.
- The maximum number of variables to the left of the equal sign in a multiple assignment statement is limited only by the line size (64 characters).

Examples

Some sample LET statements are as shown:

```
10 LET Z$ = 'CAT'

20 LET X = 9

30 LET Y(X) = 2

40 Y(X),X = X/Y(X)
```

After execution of statement 10, the character variable Z$ will contain the word CAT followed by 15 blank characters (if Z$ is 18 characters in length). In statement 20, variable X receives a value of 9.

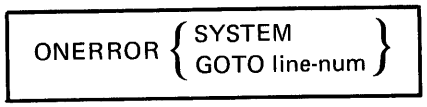
After execution of statement 30, the ninth member of the one-dimensional arithmetic array (Y) will have the integer value 2.

After execution of statement 40, the arithmetic variable X will have the decimal value 4.5. The ninth member of the one-dimensional arithmetic array Y will have the decimal value 4.5. The action of statement 40 is to first evaluate the expression on the right according to the current values of the variables Y(X) and X, 2, and 9, respectively. The resulting value, 4.5, is first assigned to Y(9), then to the variable X.

NEXT control-var

NEXT

For a complete description of the NEXT statement, see *FOR and NEXT*.



ONERROR

The ONERROR statement provides you with comprehensive error-trapping capability. When an error is detected during execution, two internal constants in the system (&LINE and &ERR) will contain the line number of the statement being interpreted and the number of the error. This enables you to recover from the error and continue from the point of interruption. The syntax of the ONERROR statement is as shown above, where:

SYSTEM specifies that normal error processing be used for all subsequent errors.

GOTO specifies the line number of the statement to receive program control when an error is detected.

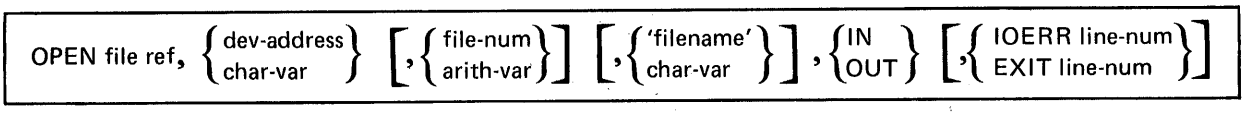
Notes About ONERROR

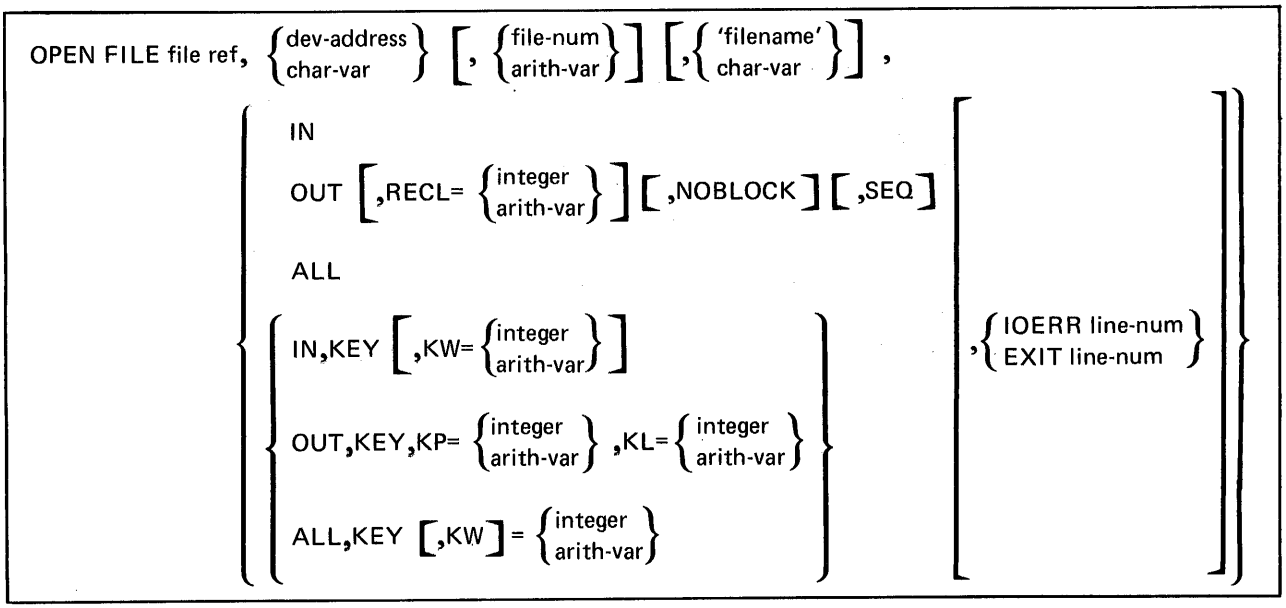
- I/O statements containing error exit parameters override the ONERROR statement.
- If an error occurs after execution of an ONERROR statement, &LINE and &ERR are set, but all other error indicators (internal pointers) are reset. Therefore, the data in &LINE and &ERR is the only information available about the error.
- To prevent repeated entry into your error routine, you should begin your error routine with an ONERROR SYSTEM statement.
- If the recovery for a detected error does not normally recommend a restart (&ERR>700), certain system information is lost (such as DEF and GOSUB statement return points). Continued execution after such an error is not recommended.
- If ONERROR is active when control is passed to a key group (GOSUB 999X), ONERROR is suspended until control returns from the key group. ONERROR may be specified in a key group but is active only within that key group and must refer to a statement in that key group.

A sample ONERROR statement is shown below:

```
010 ONERROR GOTO 50
020 A=100/0
030 PRINT 'A= 'A
040 STOP
050 ONERROR SYSTEM
060 PRINT 'ERROR' &ERR 'HAS OCCURRED, at LINE' &LINE
RUN
ERROR 681 HAS OCCURRED AT LINE 20
```

In this example, the ONERROR statement at statement number 010 causes the program to go to statement number 050 if any errors occur. The ONERROR statement causes the 5110 to return to normal error processing, then statement 060 causes the error message, ERROR 681 HAS OCCURRED AT LINE 020, to be printed.





OPEN/OPEN FILE

The OPEN statement allows you to:

- Identify an attached device with a file reference code.
- Allocate work space for the file.
- Specify the file number.
- Specify the identification to be used with the file.
- Specify the file usage (input, output, or both).

The syntax of the OPEN statement is as shown above, where:

file ref is FL0 to FL9 to specify the logical file to be associated with the physical file identified by the remaining entries in the OPEN statement. This file specification can also be referenced in input/output statements for stream I/O files.

dev-address is the address of the device referenced by the file reference code. You can enter the address directly, or enter a character variable containing the device address. Valid addresses are:

'SYS' for the current system default device address.
'E80' for the built-in tape unit (Model 1 only)
'E40' for the auxiliary tape unit (Model 1 only)
'D80' for diskette drive 1
'D40' for diskette drive 2
'D20' for diskette drive 3
'D10' for diskette drive 4
'500' for the printer
'000' for directing output to line 1 of the display screen (output only)
'001' for allowing GET statements to access data entered from the keyboard (in the same manner that INPUT statements receive data)

file-num is a constant or numeric variable ranging from 1 to 9999. This value is used to access the corresponding file on the device specified. Decimal values for variables are truncated to an integer.

'*filename*' is a character constant enclosed in single quotation marks or a character variable (char-var). The first 17 characters provide the identification field in the header record of a file being opened for output. For diskette files, those characters must be in a simple or complex name. See *File Reference Parameter* in Chapter 2.

IN or OUT indicates whether the file is to be used for input (IN) or output (OUT).

IOERR line-num allows you to specify the line number of a statement to which program control will be transferred if this OPEN statement cannot be completed because of a hardware error.

EXIT line-num allows you to specify the line number of an EXIT statement to which program control will be transferred if this OPEN statement cannot be completed because of an error condition defined in the EXIT statement.

A sample OPEN statement is as shown:

```
OPEN FL1, 'E80',3,IN,IOERR 999
```

In this example, file reference code FL1 will be assigned to reference file 3 on the tape unit built into the 5110 Model 1 for input, and program control will transfer to statement 999 in case of a hardware error.

The OPEN FILE statement provides the same capabilities for record I/O files as the OPEN statement does for stream I/O files. plus the following additional capabilities:

- Use of a file for both input and output at the same time
- Selection of where output is to begin
- Selection of logical record length
- Selection of file access method

The syntax of the OPEN FILE statement is as shown previously, where:

file ref, file-num, 'filename', IN, and OUT have the same meaning as described for the OPEN statement. The remaining parameters have the following meanings:

'dev address' has the same meaning as that for the OPEN statement with the additional option of addresses '000' and '001', which are invalid, and address '002', which opens the top 14 lines of the display as a record file with a record length of 896 characters.

RECL = is a nonzero, positive integer or a numeric variable with a value less than 9999, which indicates logical record length. If output is to start at the beginning of extent (BOE) of the file on diskette, this entry can only follow the OUT parameter. If this file is an empty file, the RECL= parameter is required. If the file is not empty and this parameter is entered, output will replace the contents of the file. If the file is not empty and this parameter is not entered, output will be added to the end of the file.

NOBLOCK indicates that the file is to be written with only one logical record to a physical record. This parameter can only be entered following OUT and RECL=.

SEQ indicates that the file can only be accessed consecutively and that records can be relocated sequentially for I/O error recovery. This parameter can only be entered following OUT and RECL=. If SEQ is specified, any attempt to access the file randomly with REC= or KEY will cause an error.

ALL indicates that the file can be used for both input and output, including extending and updating of the file. This parameter can only be entered for an OPEN FILE statement.

IN,KEY identifies the file as a key index file to be used for input only.

OUT,KEY identifies the file as a key index file to be used for output only.

KP = is a nonzero, positive integer or a letter-digit numeric variable with a value less than 9999, which identifies the first position of the key field in the data record. This parameter must be entered following OUT,KEY.

KL = is a nonzero, positive integer or a numeric variable with a value less than or equal to 28, which identifies the length of the key field in a data record in a key index file. This parameter must be entered following KP=.

KW = is a nonzero, positive integer or a numeric variable with a value less than 9999, which identifies the amount of system work area that can be used for key indexed file accessing (see *Data Files and Access Methods* in Chapter 3. This parameter is optional when IN,KEY or ALL,KEY is specified.

ALL,KEY indicates that the file is a key index file to be used for both input and output.

IOERR line-num and EXIT line-num have the same meaning as described for the OPEN statement.

Notes About OPEN

- An OPEN or OPEN FILE statement must be issued for a file before an input/output statement references the file.
- If a file is already open, the OPEN [FILE] statement causes an error message.
- Once a file is open, do not remove the tape cartridge or diskette until the file is closed.
- A quote cannot be embedded in a character constant used for a file name.
- The file name is required for diskette file output (see *File Reference Parameter* in Chapter 2).
- Tape file names are ignored by the system and are used for user documentation only.

Opening a Key-indexed File

When opening a key-indexed file, follow these general rules:

1. Open the master file using the OPEN FILE statement with the appropriate parameters.

2. Open the index file with the OPEN FILE statement. For the index file, enter the same file reference and access type (such as IN,OUT, or ALL) as entered for the master file. In addition, enter the KEY parameter and the KW parameter, which indicate key-indexed access and the amount of storage to be used for the index file. The key index file is built by the system. The index file contains only the key field for each record in the master file, along with the location of the record on tape or diskette. The KW parameter allows you to allocate space for a table that is used to improve access time when IN or ALL is specified and the index file is sorted. The size of this space is normally a multiple of the sum of the key length plus two.

3. When accessing records in the file with subsequent READ FILE, WRITE FILE, and other record I/O statements, be sure to specify the record key. The system locates the record key in the index file, then proceeds to the corresponding tape/diskette location for record access.

Examples

```
10 OPEN FILE FL1, 'D80', 'EMPL.MAST', ALL
20 OPEN FILE FL1, 'D80', 'EMPL.INDX', ALL, KEY, KW=80
```

The OPEN statements shown above enable the program to access the file 'EMPL.MAST' using the indexed access method. The files are both accessed by name only, and both files are opened for update operations (ALL). (The file 'EMPL.MAST' is opened first.) The keyword KEY in statement 20 indicates that this file is the index file for 'EMPL.MAST'. The KW=80 parameter assigns up to 80 characters in the work area for a table. The table improves access time when a search is made for a particular key value.

PAUSE

The PAUSE statement allows you to interrupt program execution to perform calculator operations. The syntax of the PAUSE statement is as shown above, where:

comment is optional. If the comment begins with a single quotation mark, it will be displayed on line zero when the PAUSE is executed and limited to 48 characters in length.

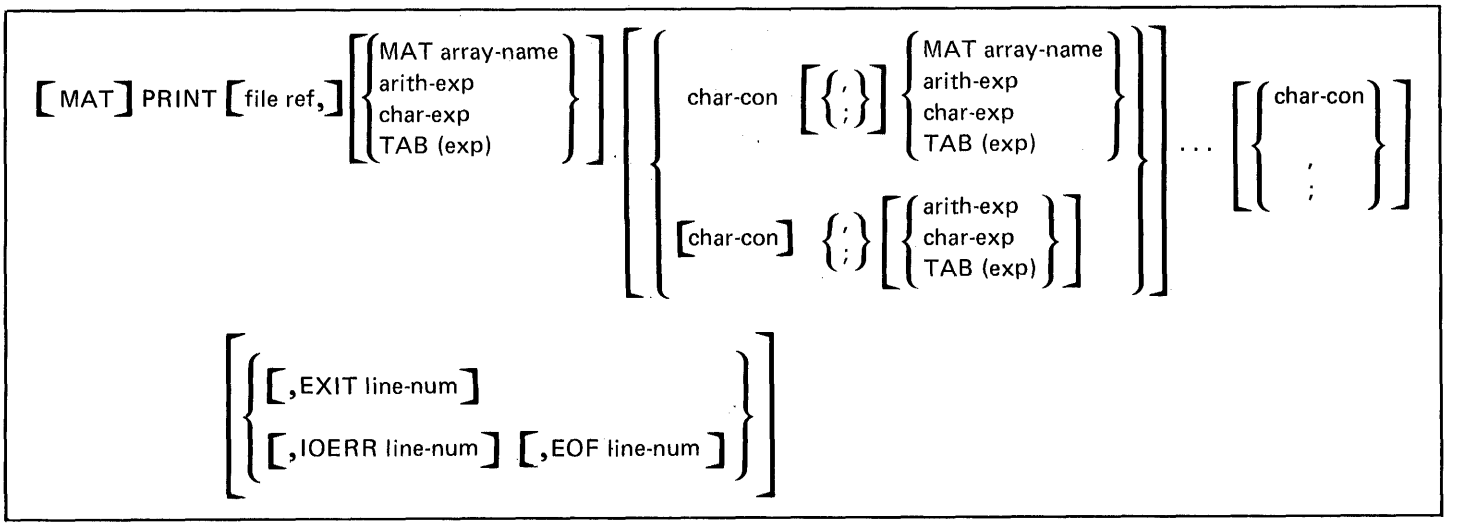
When a PAUSE statement is encountered during program execution, execution is interrupted and the message

PAUSE s

is shown on line 0 of the display screen, where s is the line number of the PAUSE statement. To resume program operation, you must issue a GO command. Note that you should not renumber statements (see *RENUM Command* in Chapter 2) or alter a calling statement (see *DEF, RETURN, FNEND*) while a program is interrupted. A comment beginning with a single quotation mark is displayed after the statement number. The following statement would cause the message PAUSE 0080 to be displayed and processing to be suspended until a GO command is issued:

80 PAUSE

A PAUSE within a program under PROC control will cause the next record in the procedure file to be executed.



[MAT] PRINT

The [MAT] PRINT statement causes the values of specified scalar expressions or arrays to be displayed on the display screen, printed by the printer, or written to a file. When using the [MAT] PRINT statement, the format of all displayed values is standardized, but the spacing between values on the same line can be controlled.

The syntax of the [MAT] PRINT statement is as shown above, where:

file ref is FLP (printer) or FL0 to FL9 (logical file). This entry is optional. FL0 to FL9 must have been previously opened in an OPEN statement.

exp are arithmetic or character expressions to be displayed or printed. Expressions can be separated by a comma or semicolon. You can control spacing of the displayed or printed expressions by inserting commas or semicolons between the expressions (see *TAB Function* in this chapter). Expressions can be arithmetic or character (character constants must be enclosed in single quotation marks). With no expressions specified, the PRINT statement can be used to complete the displaying or printing of a pending line or insert a blank line. Whole arrays (preceded by MAT) can also be specified. If only arrays are referenced, the keyword MAT can precede PRINT.

EXIT, IOERR, and EOF indicate that program control be transferred to the specified line number if the corresponding error should occur (see *EXIT*).

When a PRINT statement is executed, the value of each specified expression is converted to the appropriate output format as described under *Print Zones*, and displayed or printed in a left-to-right sequence in the order in which it appears in the PRINT statement.

When a MAT PRINT statement is executed, each array element is converted to a specified output format and displayed.

Each array is displayed by rows; the first row of each array begins at the start of a new line and is separated form the preceding line by two blank lines. The remaining rows of each array begin at the start of a new line and are separated from the preceding line by one blank line. After each array element is displayed or printed, the starting position of the next element is determined by the delimiting character, as described under *Spacing of Printed or Displayed Values*.

Print Zones

In general, each PRINT statement causes a new display or print line to be started, and each line is divided into *print zones*–either full print zones or packed print zones, or combinations of both.

Full Print Zones have 18 character positions and are specified by the comma delimiter. For example, the statement

```
200 PRINT A,B,C
```

would cause the value of variable A to be displayed beginning in the first position of the line, the value of B beginning in position 19, and the value of C beginning in position 37.

Packed Print Zones are variable in length and usually produce a more dense line of output than full print zones. The semicolon and null delimiters are used to specify packed print zones. The length of packed print zones for various types of data is explained under *Spacing of Printed or Displayed Values*.

Spacing of Printed or Displayed Values

The converted value of each expression or array element is printed or displayed in its own print zone. A print zone can be either full (18 positions) or packed (variable lengths) as specified by the delimiter following the array reference or expression.

Full Print Zones (Comma Delimiter):

For arithmetic expressions, the converted data items require one full print zone of 18 character positions. For character data, the print or display area is the smallest number of full print zones (smallest multiple of 18) large enough to accommodate the data.

Since most printed or displayed values are shorter than 18 characters, a line of full print zones usually produces columns of widely spaced output.

Packed Print Zones (Null [PRINT only] or Semicolon Delimiter):

For arithmetic expressions or numeric arrays, the length of the packed zone is determined by the length of the converted value, including the sign, digits, decimal point, and the exponent, as shown in Figure 9.

For character constants, the length of the packed print zone is equal to the length of the characters enclosed in single quotation marks, including blanks, but excluding the single quotation marks preceding an apostrophe.

For character variables, the length of the packed print zone is equal to the length of the character string, minus any trailing blanks.

Packed print zones usually produce a denser line of output than full print zones.

Length of Converted Data Item (characters)	Length of Packed Print Zone (characters)	Example (␣ represents a blank)
2-4	6	␣7.3␣␣
5-7	9	␣17.357␣␣
8-10	12	-45.63927␣␣␣
11-13	15	␣1.73579E-23␣␣␣
14-16	18	-8.92270493115␣␣␣␣
17-19	21	-1.234567890123E-22

The number of digits displayed or printed is controlled by the current value assigned to RD= (see *RD= Command* in Chapter 2).

Figure 9. Packed Print Zone Lengths for Arithmetic Expressions and Array Elements

Standard Output Formats for Printing or Displaying

Character Constants

The actual characters enclosed in single quotation marks (including trailing blanks but excluding the single quotation marks preceding an apostrophe) are printed or displayed.

However, if the data list of the PRINT statement has a character constant concatenated with a character variable, only the character constant is printed. In order to prevent this error, concatenate before executing the PRINT statement or replace the concatenation symbol (| |) with a semicolon.

Character Variables

The actual characters (excluding trailing blanks) are printed or displayed. However, if the data list of the PRINT statement has a character variable concatenated with a character constant, only the character constant is printed. In order to prevent this error, concatenate before executing the PRINT statement or replace the concatenation symbol (| |) with a semicolon.

Arithmetic Expressions

I-Format:

The integer, consisting of a sign (blank or minus) and up to 15 significant decimal digits for integers whose absolute value is less than 1E+14, is printed or displayed.

E-Format:

The floating-point number, consisting of a sign (blank or minus), up to 15 significant decimal digits, a decimal point following the first digit, the letter E, and a signed exponent consisting of one or two digits is printed or displayed. The E-format is used to print or display numbers whose absolute value is less than 1E-2 or greater than or equal to 1E+14. Printed or displayed values are rounded off, not truncated.

F-Format:

The fixed-point number, consisting of a sign (blank or minus) up to 15 significant digits, and a decimal point in the appropriate position is printed or displayed. The F-format is used to print or display values not included in the preceding I- or E-format descriptions.

Display Line Operation

As the values of expressions or array elements are transferred to the display screen, an internal line position pointer is maintained to keep track of the next available position where a character (or digit) can be placed. The movement of this line-position pointer before, during, and after displaying an expression depends on the type of expression and the delimiter following it in the [MAT] PRINT statement. Figure 10 shows the pointer actions that are possible.

The position of the display line-position pointer can be moved forward (to a higher-numbered character position) by the TAB function.

If an array delimiter is a comma, the line-position pointer is moved past any remaining positions in the full print zone after the system transfers the element value to the display screen. If the final delimiter is a semicolon, the line-position pointer is moved past any remaining positions in the packed print zone after the system transfers the element values to the display screen. See *Print Line Buffer Operation.*

Data Type	Delimiter	Pointer Position Before Displaying or Printing	Pointer Position After Displaying or Printing
Arithmetic Expression	Comma	If the record contains sufficient space for the expression, data will be written beginning at the current position of the pointer. If there is not sufficient space, the contents of the first record are written and the expression will start at the beginning of the next record.	The pointer is moved past any remaining positions of the full print zone. If the end of the record is reached, the record is printed or displayed and the pointer is set to the start of the next record.
	Semicolon		The pointer is moved past any remaining positions of the packed print zone. If the end of the record is reached, the record is printed or displayed and the pointer is set to the start of the next record.
	Null (not end of statement)		
	Null (end of statement)		The record is printed or displayed and the pointer is set to the start of the next record.

Figure 10 (Part 1 of 3). Positions of Display or Print Line-Position Pointer

Data Type	Delimiter	Pointer Position Before Displaying or Printing	Pointer Position After Displaying or Printing
Character Variable	Comma	If at least 18 character positions remain in the record, data will be written beginning at the current position of the pointer. If less than 18 positions remain, the record is printed or displayed and the constant will be written starting at the beginning of the next record. If the end of the record is encountered before the characters are written, the record is printed or displayed and the remaining characters will be written starting at the beginning of the next record.	The pointer is moved past any remaining characters of the full print zone. If the end of the record is reached, the record is printed or displayed and the pointer is set to the start of the next record.
	Semicolon	The variable is written starting at the current position of the pointer. If the end of the first record is reached before all data is written, the record is displayed or printed and the remaining data will start at the beginning of the next record.	The pointer is left at the position immediately following the last character.
	Null (not end of statement)		
	Null (end of statement)		The record is written and the pointer is set to the start of the next record.

Figure 10 (Part 2 of 3). Positions of Display or Print Line-Position Pointer

Data Type	Delimiter	Pointer Position Before Displaying or Printing	Pointer Position After Displaying or Printing
Null	Comma	No data is displayed.	The pointer is moved forward 18 character positions. If the end of the record is reached, the record is printed or displayed and the pointer is set to the start of the next record.
	Semicolon		The pointer is moved forward 3 character positions. If the end of the record is reached, the record is printed or displayed and the pointer is set to the start of the next record.
	Null (not end of statement)		
	Null (end of statement)		
Character Constant	Comma	No data is displayed.	The pointer is moved past any remaining spaces in the full print zone. If the end of the record is reached, the record is printed or displayed and the pointer is set to the start of the next record.
	Semicolon		
	Null (not end of statement)	The variable is written starting at the current position of the pointer. If the end of the first record is reached before all data is written, the record is displayed or printed and the remaining data will start at the beginning of the next record.	The pointer is left at the position immediately following the last character.
	Null (end of statement)		The record is written and the pointer is set to the start of the next record.

Figure 10 (Part 3 of 3). Positions of Display or Print Line-Position Pointer

TAB Function

The TAB (expression) function allows you to align columns of data. When (expression) is a noninteger, it is truncated. Thus, TAB(n) starts the next output in column n of the line. If the current position in the line is greater than n, data is put on the next line in position n. TAB(n) can be followed by a comma or semicolon with the same result. If n is greater than the line length, data is put at position X after computation of the following, where n = the TAB expression and L = line length:

```
X = n-L*INT((n-1)/L)
```

For example, if line length = 64 and the TAB(n) = 70, the integer value ((70-1)/64) = 1 is multiplied by 64 and is subtracted from n, which is 70-64=6. Thus, output begins in position 6 of the next line. This formula only applies where L = 1 to 9999.

Print Line Buffer Operation

The print line buffer is a temporary storage location where the characters for each print line are placed. After all data for a print line has been placed in the buffer, the data line is printed.

A buffer-position pointer is used to keep track of the next available position where a character (or digit) can be placed in the buffer. The movement of the buffer-position pointer before, during, and after moving an expression into the buffer depends on the type of expression to be printed and the delimiter following it in the PRINT statement.

The position of the buffer-position pointer can be moved forward by the TAB function. See the PRINT statement examples that follow.

The following examples show how various arithmetic values would be printed or displayed. The symbol ␣ represents a blank character, which always appears in the sign position of a positive number.

Value Given	**Value Printed**
0123	␣123
00123	␣123
-1234.5	-1234.5
135999999999	␣135999999999

Notes About [MAT] PRINT

- If there is no printer attached to the system, executing the PRINT FLP statement stops the program unless the P = D (printer = display) option was specified in the RUN command.
- Null delimiters are not permitted in a MAT PRINT statement except as the final delimiter.

Examples

The following examples show the output resulting from several PRINT statements:

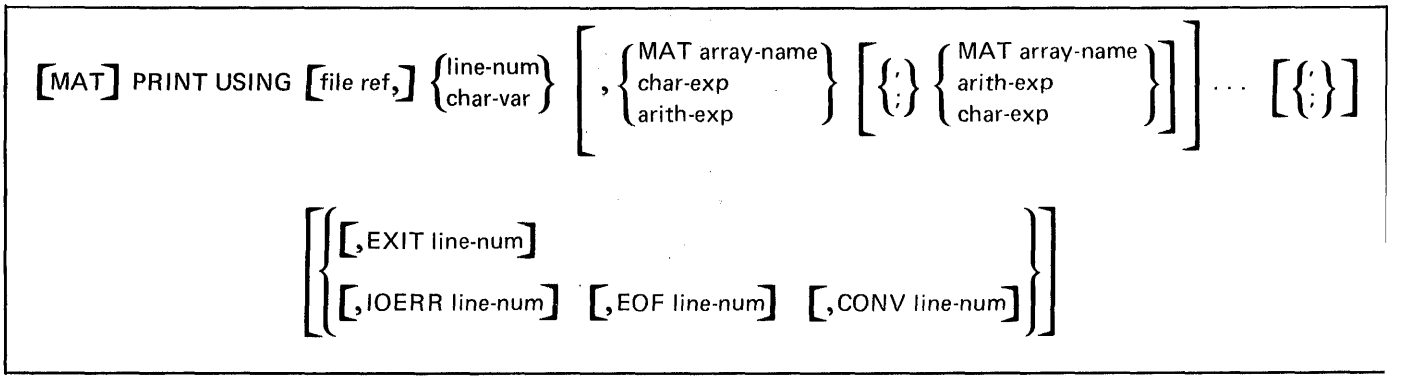
Statement	**Output**
10 PRINT FLP,'A','B'	A–17 blanks–B
20 PRINT 'A';'B'	AB
30 LET A$ = 'B'	
40 PRINT FLP,'A'A$	AB
50 PRINT FLP,A$'A' ,A$;A$	BA–17 blanks–BB
60 PRINT A$;'A'	BA
70 LET A$='	
80 PRINT FLP,'A';A$;'A'	AA
90 PRINT 'A';' ';'A'	A␣A
100 PRINT FLP	Blank line
110 PRINT 'NAME' TAB (30) 'ADDRESS'	NAME–25 blanks–ADDRESS
120 END	

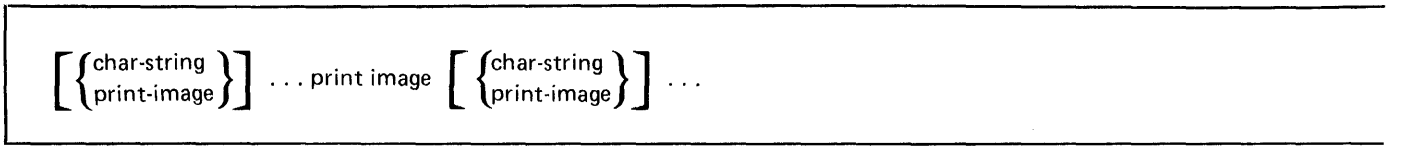
The following example shows the execution of a MAT PRINT statement:

```
10 DIM A(3,4), A$(12), C(2,2)
20 MAT READ A,A$
30 DATA 1,2,3,4,5,6,7,8,9,10,11,12,
40 DATA 'A','B','C','D','E','F','G','H','I','J','K','L'
50 MAT C = (1)
60 MAT PRINT A
70 MAT PRINT A$;
80 MAT PRINT C
```

Displayed output will be:

```
 1             2             3             4
 5             6             7             8
 9             10            11            12

ABCDEFGHIJKL

 1             1
 1             1
```





[MAT] PRINT USING AND IMAGE/FORM

The [MAT] PRINT USING statement and its associated image or FORM allow you to display specified scalar values or array elements in a format of your own choosing. The [MAT] PRINT USING statement specifies the values to be displayed and the image or FORM to be used. These statements specify the format of the line to be displayed. The syntax of the PRINT USING and image statements is as shown above, where:

file ref is an optional specification of FL0 to FL9 to identify the logical file into which the formatted data is to be placed or FLP to specify printing. If not specified otherwise, the formatted data is displayed. FL0 to FL9 must have been previously referenced in an OPEN statement.

line-num is the statement number of the image statement or FORM statement that defines how data is to be formatted when printed or displayed.

char-var is a character variable containing format information identical to that in an image or FORM statement. For an image statement, the contents of the character variable can be the same as the contents following the colon in a standard image statement. For a FORM statement, the first four positions in the character variable must be the characters FORM, followed by normal format specifications of a standard FORM statement.

exp is an arithmetic or character value separated by semicolons or commas. These values are incorporated (edited) into the format of the image or FORM statement specified. A comma or semicolon can also follow the last expression. Whole arrays (preceded by MAT) can also be specified. If only arrays are referenced, the keyword MAT can precede PRINT USING.

print-image is the format specification (see *Format Specifications*).

When a [MAT] PRINT USING statement is executed, the specified expressions or array references are evaluated, and their values are edited in order of appearance in the [MAT] PRINT USING statement into the corresponding format specifications in the specified image or form. A string of character constants will be printed exactly as entered in the image statement described in the following text. Array elements are displayed by row. For a description of FORM statement format, see *FORM*.

Conversion of Data Reference Values with Image

When the data referred to is a character value, the characters contained in it are edited into the line, replacing characters in the format specification including sign (+,-),pound sign, decimal point, and ||||.

If an edited character value is shorter than its format specification, blank padding occurs on the right. If an edited character value is longer than its format specification, it is truncated on the right. A character constant containing no characters (null) causes blank padding on the entire format specification.

An arithmetic expression is converted according to its format specification as follows:

- If the format specification contains a plus sign and the expression value is positive, a plus sign is edited into the line.
- If the format specification contains a plus sign and the expression value is negative, a minus sign is edited into the line.
- If the format specification contains a minus sign and the expression value is positive, a blank is edited into the line.
- If the format specification contains a minus sign and the expression value is negative, a minus sign is edited into the line.
- If the format specification does not contain a sign and the expression value is negtive, a minus sign and the negative number will be printed, provided that the format specification is long enough to contain both the number and the sign. If the format specification is not long enough, asterisks are edited into the line instead of the negative number. Asterisks are also edited into the line for positive values that are too large.
- The expression value is converted according to the type of its format specification as follows:

  I-Format: The value of the expression is converted to an integer, rounding any fraction.

  F-Format: The value of the expression is converted to a fixed-point number, either rounding the value or extending it with zeros according to the format specificaiton.

  E-Format: The value of the expression is converted to a floating-point number, rounding the value or extending it with zeros and adjusting the exponent according to the format specification.

For I or F format, the length of the integer portion of the arithmetic expression value is less than or equal to the length of the integer portion of the format specification, the expression value is edited, right-justified, and padded with blanks into the line. If the length of the integer portion of the format specification is less than the length of the integer portion of the expression value, asterisks are edited into the line instead of the expression value.

Format Specifications

For each occurrence of the pound sign (#) in an image statement or character variable, a single space is reserved in the display or print line for a character in the corresponding expression of the associated PRINT USING or MAT PRINT USING statement. The pound sign represents either character or arithmetic data. For arithmetic data, decimal points and the plus and minus signs, like the characters in the general format description, are printed as entered, provided the values are appropriate to the specified signs. (See *Conversion of Data Reference Values with Image* for a discussion of the displaying/printing of signs in the image statement.) For character data, the character string will override any format descriptors.

The various format specifications are:

- Character-Format–One or more # characters, as shown:

  ####

- I-Format (integer format)–An optional sign followed by one or more # characters, as shown:

  [+-] # [#] ...

- F-Format (fixed-decimal format)–An optional sign followed by either:

  1. No # characters, a decimal point, and one or more # characters, as shown:

     .#[###]

  2. One or more # characters, a decimal point followed by no # characters, as shown:

     #[###].

  3. One or more # characters, a decimal point, and one or more # characters, as shown:

     [+-] [#] ... .#[#] ... .[#] ...

- E-Format (exponential format)–Either the I- or F-format (described previously) followed by four | characters (||||).

The following rules define the start of a format specification:

1. A # character is encountered and the preceding character is not a # character, decimal point, plus sign, or minus sign.

2. A plus or minus sign is encountered, followed by:
   a. A # character, or
   b. A decimal point that precedes a # character.

3. A decimal point is encountered, followed by a # character and:
   a. The preceding character is not a # character, plus sign, or minus sign, or
   b. The preceding character string is an F-format specification.

The following rules define the end of a format specification that has been started:

1. A # character is encountered and:
   a. The following character is not a # character, or
   b. The following character is not a decimal point, or
   c. The following character is a decimal point and a decimal point has already been encountered, or
   d. The following four consecutive characters are not | characters (||||).

2. A decimal point is encountered and:
   a. The following character is not a # character, or
   b. The following character is another decimal point, or
   c. The following four characters are not | characters (||||).

Some examples of expressions and the way they are displayed or printed under various format specifications are as follows:

Format Specification	Expression Value	Displayed or Printed Format
#####	'APPLES'	APPLE
####	-123	-123
###	123	123
###	12	␣12
###	1.23	␣␣1
##.##	123	*****
##.##	1.23	␣1.23
##.##	1.23456	␣1.23
##.##	.123	␣␣.12
##.##	12.345	12.35
###||||	123	123E+00
###||||	12.3	123E-01
###||||	.1234	123E-03
##.##||||	123	12.30E+01
##.##||||	1.23	12.30E-01
##.##||||	.1234	12.34E-02
##.##||||	1234	12.34E+02

During output, specified expressions are displayed or printed beginning where the previous PRINT or PRINT USING statement ended in a line. If this is the first PRINT or PRINT USING statement, output begins on a new line. If the PRINT USING statement contains expressions or array elements that exceed format specifications in the specified image statement, output is controlled by the delimiter following the expressions. If the delimiter is a comma, the current line is displayed or printed, and the remaining expressions are formatted according to the beginning of the image statement, and output begins on a new line. If the delimiter is a semicolon, expressions remaining after the end of the image statement are formatted according to the beginning of the image statement, and output continues on the same line.

When the image statement is being used, the delimiter following the last expression value controls printing at the end of expressions. If the delimiter is a comma or blank, the current line is displayed or printed and the next output will start on a new line. If the delimiter is a semicolon, the current line is not displayed or printed and the next output will be added to the current line.

When a FORM statement is used (see *FORM*), the current line is not displayed or printed unless a SKIP specification follows the last data format specification used.

If the end of the image or FORM statement is reached *after* the last expression value or the first unused format specification is found, formatting is stopped.

Notes About PRINT USING

- The number of image or FORM statements permitted in a BASIC program is only limited by available storage.
- If the [MAT] PRINT USING statement output list contains at least one item, there must be at least one format specification in the corresponding image statement.
- If there is no printer attached to the system, executing the [MAT] PRINT USING FLP statement stops the program, unless the P = D (printer = display) option was specified in the RUN command.
- Image and FORM statements are nonexecutable and can be placed anywhere in a BASIC program, either before or after the PRINT USING statements that refer to them.
- If the number of elements in an array row exceeds the number of conversion specifications in the associated image statement, the image statement is reused for the remaining elements of that row, and if the delimiter following the array is a comma, a new line is started.
- The line for each row is terminated at the first unused conversion specification after the last element in the row is printed or displayed.
- During output, the trailing blanks in a variable image (see *Character Variables* in Chapter 3) are significant. For example: if a variable A$ is dimensioned to a length of 30 and a name C O JONES is entered into the variable, the PRINT USING statement will use all 30 positions of the variable and the next available position becomes position 31. See the following program.

```
10  DIM A$30
20 A$='################'
30 PRINTUSING A$,FLP,'C O JONES';
40 PRINT 'X'
RUN
C O JONES                      X
```

Examples

The following example shows execution of a PRINT USING statement where:

A = 342 and B = 42.02

```
100 :RATE OF LOSS####EQUALS####.##POUNDS
110 PRINT USING 100,A,B
```

The output is:

```
RATE OF LOSS 342 EQUALS 42.02 POUNDS
```

The same output would result from:

```
100 A$='RATE OF LOSS #### EQUALS ####.## POUNDS'
110 PRINT USING A$,A,B
```

Note that A$ must have been previously dimensioned to the appropriate length:

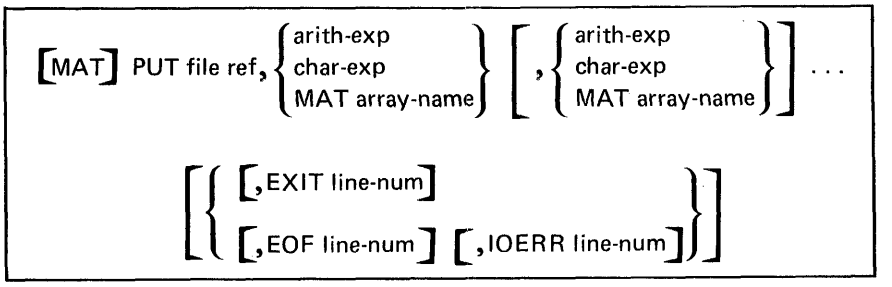
```
90 DIM A$39
```

The following example shows execution of a MAT PRINT USING statement:

```
0010 DIM A(4,3)

0020 :### ##.## ##.## ||||

0030 MAT A=(1)

0040 MAT PRINT USING FLP,0020,A
```

The output would appear as:

```
1      1.00    10.00E-01

1      1.00    10.00E-01

1      1.00    10.00E-01

1      1.00    10.00E-01
```



[MAT] PUT

The [MAT] PUT statement allows you to write the values of specified variables or array elements into a particular stream I/O file. The file must have already been opened with an OPEN statement. The syntax of the [MAT] PUT statement is as shown above, where:

file ref is FL0 to FL9 to identify the file into which variable values will be written.

exp are function references, subscripted array references or whole arrays (preceded by MAT) representing the values to be written. Only one value is required. If more than one is specified, the values must be separated by commas. If only arrays are referenced, the keyword MAT can precede PUT.

EOF, IOERR, and EXIT are error recovery exits (see *EXIT*).

When a [MAT] PUT statement is executed, the specified scalar reference or array element value is entered from left to right into a buffer for the specified file, beginning at the current file position. The file is written sequentially so that the first value entered by the [MAT] PUT statement will be the first value assigned from the file when the file is referenced in a [MAT] GET statement. When the buffer becomes full, the contents are written out. Tape buffers are 512 bytes in length. Diskette buffers are 128, 256, 512, or 1,024 bytes in length depending on diskette format. As many bytes as possible are used in each buffer.

Character data is written with enclosing quotation marks. Comma separators are written between data items. A new line character (hexadecimal 15) replaces the comma at the end of an array row or is written at the end of the data items listed in the [MAT] PUT statement.

A file can be activated only by an OPEN statement and is deactivated by a CLOSE statement or at the end of program execution.

Notes About [MAT] PUT

- A file currently activated as an input file cannot be specified in a [MAT] PUT statement. It must be closed, then reopened for output, or a RESET END statement must be issued.
- If space in the output file is exhausted before all values in the output list are placed in the file, program execution is terminated.
- A [MAT] PUT statement referring to a currently closed file causes a program error.
- A stream I/O file accessed with a [MAT] PUT statement must have been opened with an OPEN statement.

Numeric values are written as numeric constants with only insignificant zeros dropped. All other digits are kept regardless of the RD=value.

Examples

A sample PUT statement is as shown:

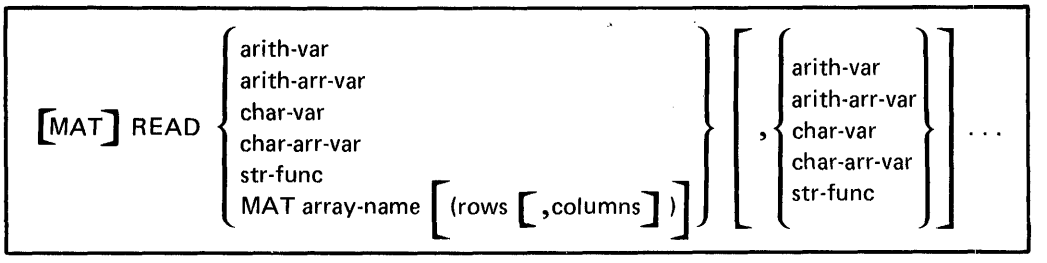
```
20 PUT FL1, Z3, 5*X-7,A,D$,9.005
```

In this example, the specified values will be written into file FL1.

The following example shows a MAT PUT statement:

```
0100 MAT PUT FL4, B, C$
```

In this example, the values of the elements in array B and array C$ are put into file FL4.



[MAT] READ

The [MAT] READ statement allows you to assign values to variables, array elements, or entire arrays from the internal data table created by DATA statements. The syntax of the [MAT] READ statement is as shown above, where:

var are simple arithmetic or character variables, substrings, subscripted references to a single array element or whole arrays (preceded by MAT). Only one variable is required. If only arrays are referenced, the keyword MAT can precede READ.

At the beginning of program execution, a pointer is set to the first value in the data table. When a READ or MAT READ statement is encountered, successive values from the data table are assigned to the variables or array elements in a READ statement, or to entire arrays in the MAT READ statement. The values are assigned to the array by rows, beginning at the current position of the data table pointer. The data table pointer can be reset by use of the RESTORE statement.

If a redimension specification follows the array name, the truncated integer portion of each value in rows and columns is used to redimension the array before values are assigned to it.

Subscripts of array variables in the [MAT] READ statement are evaluated as they occur; thus, an assigned variable in a [MAT] READ statement can be used subsequently as the subscript of another variable in the same statement.

Notes About [MAT] READ

- Before being used in a [MAT] READ statement, arrays must have been defined, either implicitly, or explicitly in a USE or DIM statement.
- If the data table is exhausted and unassigned variables, arrays or array elements remain in the [MAT] READ statement, an error occurs.
- If there are no DATA statements in the program, the [MAT] READ statement will cause an error.
- The [MAT] READ statement cannot be used in a program assigned to one of the function keys. This will cause an error when the function is executed.
- If redimension specifications are entered, the rules under *Redimensioning Arrays* in Chapter 3 must be followed.
- Each value read from the data table must be of the same type (character or arithmetic) as the variable to which it is assigned.
- If a conversion error occurs during processing of a MAT READ statement, and the preceding I/O statement executed included the CONV exit clause, the error on the READ statement will branch as indicated by the preceding CONV exit clause. To correct the error, correct the DATA list or READ statement to eliminate the error.

Examples

The following shows the execution of READ statements:

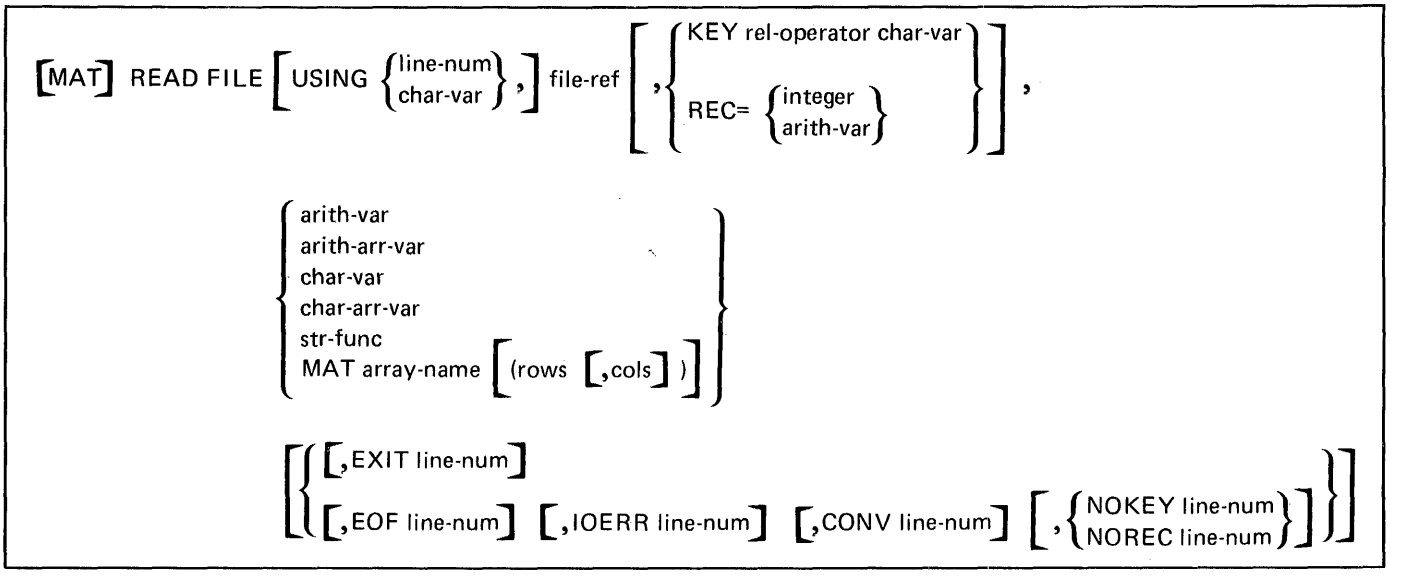
```
10 DATA'JONES', 15.00, 'SMITH',20.50

20 READ A$,A1,B$,B1

30 DATA 1,2,3,4,5,6

40 READ A,B,C,X(A),X(B),X(C)
```

After execution of these statements, the character variables A$ and B$ will contain the character strings JONES and SMITH, respectively, each padded on the right with blanks to a length of 18. The arithmetic variables A1 and B1 will contain the decimal values 15.00 and 20.50, respectively. The arithmetic variables A, B, and C will contain the integer values 1, 2, and 3, respectively, and the first three elements of the one-dimensional array X will contain the integer values 4, 5, and 6, respectively.

The following example shows a MAT READ statement:

```
0100 MAT READ A$ (2), A, B(12)
```

In this example, array A$ is redimensioned to two elements, array B is redimensioned to 12 rows, and then values from the data table are assigned to arrays A$, A, and B.



[MAT] READ FILE

The [MAT] READ FILE statement allows you to assign values from a record in a file (record I/O) to specified variables or arrays. By entering the optional USING parameter, you can also specify a FORM statement that permits conversion of data from the format defined in the FORM statement. Note that each value to be assigned must be of the same type as the variable to which it is assigned (numeric or character). The syntax of the READ FILE statement is as shown above, where:

USING line-num/char-var indicates that data to be assigned will be in the format defined in the FORM statement at the line number specified or assigned to the character variable specified. If this parameter is not entered, the system assumes that numeric data is in internal format, and that precision is identical to that currently in operation.

file-ref is FL0 to FL9 to identify the file from which variable values will be assigned, or FLS (see *File FLS* in Chapter 3).

KEY rel-operator char-var allows you to specify the key field used to access the record in the file. KEY indicates key indexed access of the file, rel-operator indicates equal (=) or greater than or equal (≥), and char-var contains the actual record key to be compared to those of records in the file until the matching record is found. The character variable can be less than or equal to the length of the actual key field. If the character variable is shorter than the key field, the search of the index will consider only that part of the key field equal to the length of the specified search argument. If this parameter is not entered, the next sequential record in the file is accessed. For a key indexed file, the record with the next sequential key is accessed.

REC = can be a positive, nonzero integer or numeric variable indicating the logical record number of the record to be accessed. If this parameter is not entered, the next (or first) logical record will be accessed.

var are simple arithmetic or character variables, substrings, subscripted references to a single array element or whole arrays (preceded by MAT) to which values are to be assigned. Only one variable is required. If only arrays are referenced, the keyword MAT can precede READ FILE.

EXIT, EOF, IOERR, NOKEY, CONV, and NOREC indicate that program control should be transferred to the appropriate statement number if the indicated error condition occurs. See *EXIT* in this chapter.

Notes About [MAT] READ FILE

- The REC parameter cannot be used to access a key indexed file.
- The KEY parameter can be specified only if a key indexed file has already been opened (using OPEN FILE).
- If redimension specifications are entered, the rules under *Redimensioning Arrays* in Chapter 3 must be followed.
- The keyword MAT must precede referenced arrays unless all references in the data list are arrays. In this case, use the keywords MAT READ FILE.
- EOF, IOERR, NOKEY, CONV, and NOREC error recovery exits can be entered in any sequence.
- Each value to be assigned from the record must be of the same type as the variable to which it is assigned (character or numeric).
- The length of the KEY parameter must be less than or equal to the actual record key field.
- If OUT was specified in the OPEN FILE statement for the referenced file, a program error will occur.
- A READ FILE issued to a type 2 file containing hex 1E or hex 15 causes unpredictable results. These characters are record delimiters.
- A READ FILE to device '002' will open the keyboard for input. Data entered on the keyboard is displayed on the top 14 lines of the screen.

A sample READ FILE statement is shown below:

```
20 I$='INVENTORY'

30 READ FILE FL8, KEY=I$,A$,D$,F$,R$,NOKEY 999
```

In this example, the record in file FL8 with a key field matching INVENTORY will be accessed. Next, values for A$,D$,F$,R$ will be assigned from the record. If no record in the file has a key of 'INVENTORY', program control will transfer to statement 999.

REM [comment]

REM

The REM (remark) statement allows you to insert remarks or comments in a BASIC program listing. The syntax of the REM statement is as shown above, where:

comment is one or more characters. This is an optional entry.

The REM statement is nonexecutable. It appears in program listing, but has no effect on program execution.

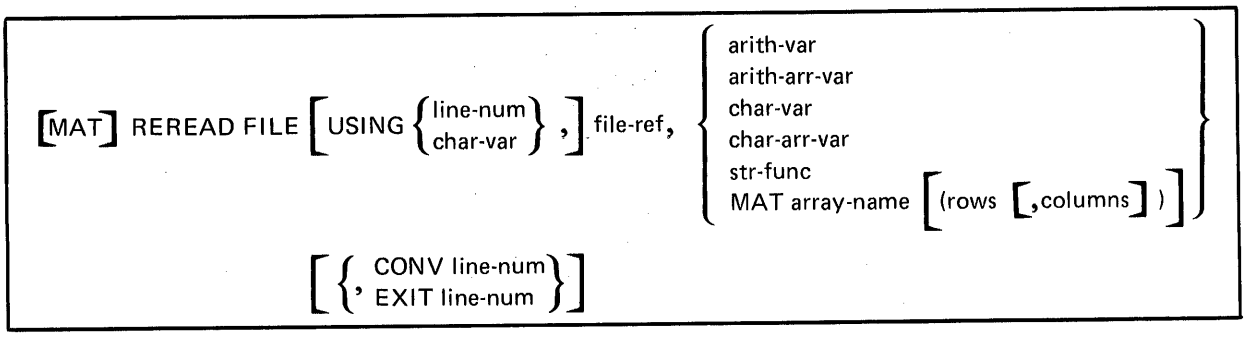
Notes About REM

- A REM statement can be used anywhere in a BASIC program.
- Also see *Function Keys* in Chapter 2.

Example

A sample REM statement is as shown:

```
10 REM THIS PROGRAM DETERMINES THE COST PER UNIT
```



[MAT] REREAD FILE

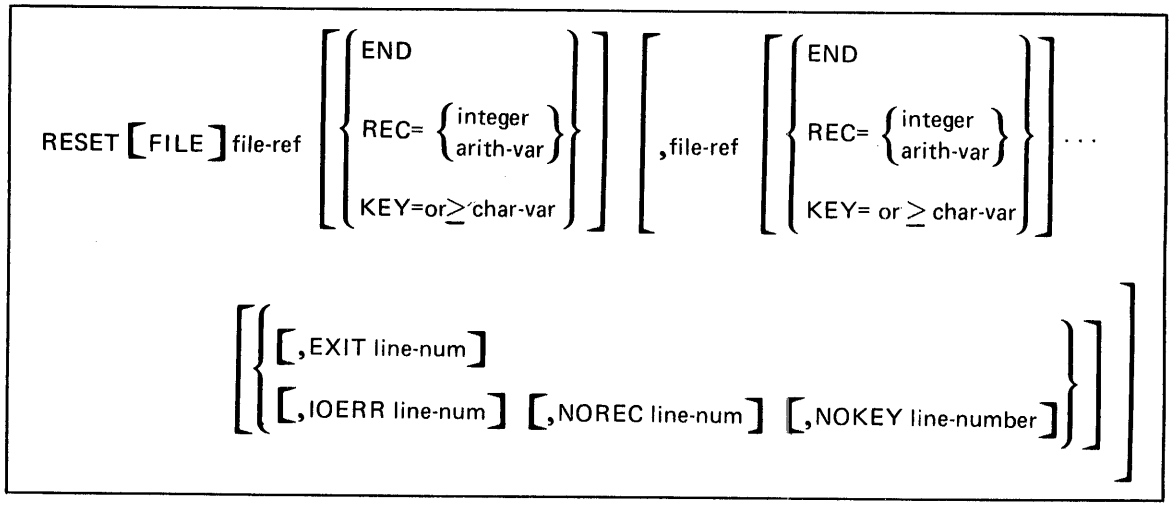
The[MAT] REREAD FILE statement allows you to access the last record that was read. After the record is accessed, the execution of a [MAT] REREAD FILE statement is identical to execution of a [MAT] READ FILE statement. The last access to the file being referenced must have been with a [MAT] READ FILE or another [MAT] REREAD FILE statement. The CONV and EXIT error recovery exits are the same as those described for the [MAT] READ FILE statement.

Note: A REREAD FILE statement referencing device '002' will reread whatever is on the top 14 lines of the screen without opening the keyboard for input.

A sample REREAD FILE statement is shown below:

```
010 MAT REREAD FILE USING 100, FL1,A,A$
```

In this example, the FORM statement at 100 is used to reread the file referenced by FL1, and the data in the record is assigned to array A and A$.



RESET [FILE]

The RESET [FILE] statement allows you to reposition an input file to its beginning or an output file to its beginning or end. The syntax of the RESET statement is as shown above, where:

file ref is FL0 to FL9 to identify the file to be repositioned. The file must have been previously opened with an OPEN statement. Only one file is required in each RESET statement. The END entry is optional and is only valid for stream I/O files.

REC = allows you to specify the number of the logical record to which you want the file reset. This record will then be the next record in the file to be accessed. This parameter is valid only for record I/O files.

KEY = or ≥ allows you to specify the key field in the record to which you want the file reset. This parameter is valid only for key-indexed record I/O files.

When a RESET statement is executed, the specified stream I/O file is repositioned so that subsequent GET or PUT file references to the file will refer to the first item in the file. When a file is opened by an OPEN statement, the first item in the file is automatically accessible.

When RESET END is specified to a stream I/O file opened for input, the file is closed and reopened for output. The file is reset so that writing of new data begins at the end of any existing data in the file.

If a file is to be used for input while open for output during execution of the same program, it must be closed and reopened with CLOSE [FILE] and OPEN statements.

When a RESET FILE statement is executed, the specified record I/O file is repositioned so that subsequent READ FILE and WRITE FILE references to the file will refer to the first item in the file. When a file is opened by an OPEN FILE statement, the first item in the file is automatically accessible. The REC= parameter allows you to specify the number of the record to which you want the file reset. The KEY= or KEY≥ parameter allows you to specify the key field within the record to which you want the key-indexed file reset. If you specify a particular key, and the system is unable to locate that key, the record with the next higher sequential key is accessed if KEY≥ is specified. Otherwise an error will occur.

Notes About RESET

- If a file specified in a RESET statement is not currently active, the file reference in the RESET statement is ignored.
- NOREC, IOERR, NOKEY, and EXIT are error recovery exits. See *EXIT*.

Example

In the following example, the RESET statement (number 100) repositions file FL6 to its beginning. The GET statement (number 110) then reads the first three values of file FL6 into A, B, and C, respectively.

```
90 GET FL6, X,Y,Z

100 RESET FL6

110 GET FL6,A,B,C
```

RESTORE [comment]

RESTORE

The RESTORE statement allows you to begin assigning values beginning with the first item in the first DATA statement (see *DATA* in this chapter) of the program according to the next READ statement executed. The syntax of the RESTORE statement is as shown above, where:

comment is one or more characters.

The RESTORE statement returns the internal data table pointer from its current position to the beginning of the table. The optional comment is a character string that does not affect the execution of the statement.

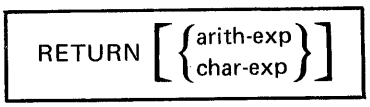
Notes About RESTORE

- A RESTORE statement in a program that contains no DATA statements is ignored.
- A RESTORE statement for an already restored data table pointer is ignored.

Example

After the following statements are executed, the variables A and C will each have a value of 1, and B and D will each have a value of 2:

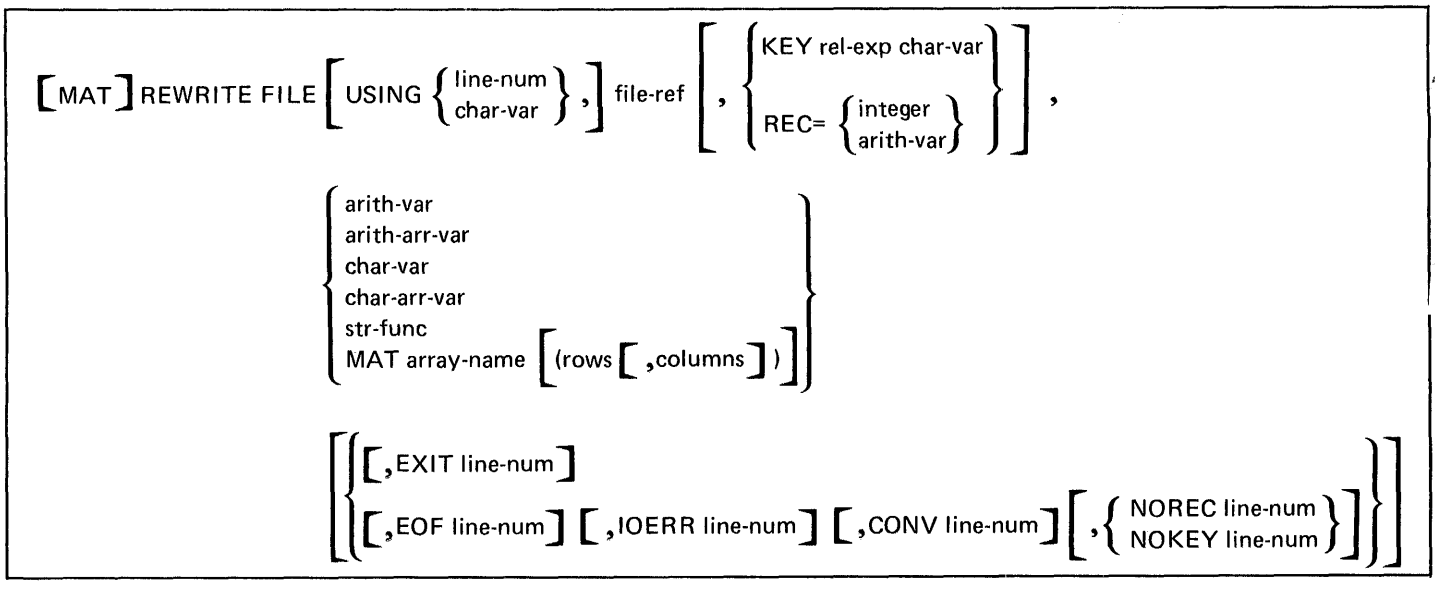
```
10 DATA 1,2

20 READ A,B

30 RESTORE

40 READ C,D
```



RETURN

For the use of the RETURN statement in the creation of subroutines, see the GOSUB and RETURN statements. Also see *Function Keys* in Chapter 2.

For the use of the RETURN statement with a multiline function definition, see the *DEF, RETURN, FNEND*.



[MAT] REWRITE FILE

The [MAT] REWRITE FILE statement allows you to change or update an existing record in a file (record I/O). The file, however, must satisfy the following requirements:

- The parameter ALL must have been specified in the OPEN FILE statement for the file.
- For tape and diskette data files, if the KEY or REC= parameter is not included in the [MAT] REWRITE FILE statement, the last access to the file must have been a READ FILE or REREAD FILE statement for the record to be changed. For record I/O using the display screen (device address '002'), the REWRITE FILE statement does not have to be preceded by a READ FILE statement.

Upon execution, the variables or arrays in the [MAT] REWRITE FILE statement are transferred to a buffer for the file reference specified. That buffer will contain the last record read from the file. If the record contains a key field (for key-indexed access), the field may not be altered. Variable values are converted for record output just as they are for the [MAT] WRITE statement, except that the X control specification in the FORM statement causes data to be bypassed rather than blanked.

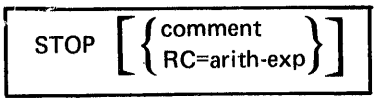
The syntax of the [MAT] REWRITE FILE statement is as shown above, where all of the parameters have the same meaning as those described for the [MAT] READ FILE statement.

Example

A sample REWRITE FILE statement is shown below:

```
20 REWRITE FILE USING 100, FL9, REC=8,D$,F$,R$,NOREC 999
```

In this example, record number 8 on the file referenced by FL9 will be rewritten with variables D$, F$ and R$ using the format specifications in the FORM statement at line number 100. If record number 8 cannot be found in the file, program control will be transferred to statement 999.



STOP

The STOP statement allows you to terminate program execution. The syntax of the STOP statement is as shown above, where:

comment is one or more characters.

RC = is the return code used by the CSKIP command (see *Procedure File* in Chapter 3).

When a STOP statement is encountered during execution of a program, it causes all open files to be closed and it terminates processing. The actions of the STOP statement are identical to those of the END statement. If the RC= parameter is not specified, it defaults to zero. If RC= is less than zero, or greater than 255, a value of zero is assumed.

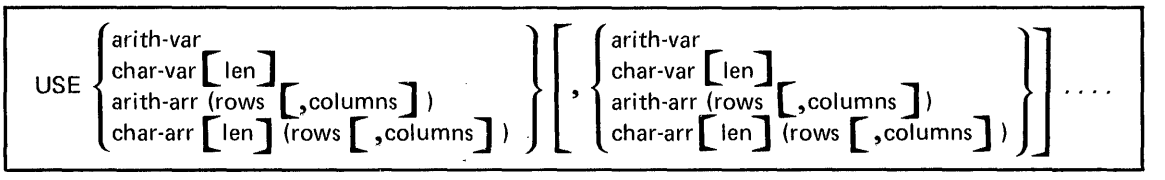
Note About STOP

A STOP statement can appear anywhere in a BASIC program.

Example

A sample STOP statement is as shown:

```
110 STOP
```



USE

The USE statement allows you to specify variables to be assigned to the common area of storage. This common area of storage holds these specified variables and passes them from one BASIC program to the next. These variables are initialized to binary zeros with the RUN command, and are not changed when one program CHAINs to another program (see *CHAIN* in this chapter). The syntax of the USE statement is as shown above, where:

var or *arr* is a numeric or character variable or array that is to be assigned to the common area of storage.

rows, *columns* are required only for arrays. These nonzero, unsigned integer constants specify the number of rows and columns to provide the dimensions of the array assigned to the common area of storage.

[len] allows you to specify the number of characters to be assigned to the char-var or each element of a character array.

Notes About USE

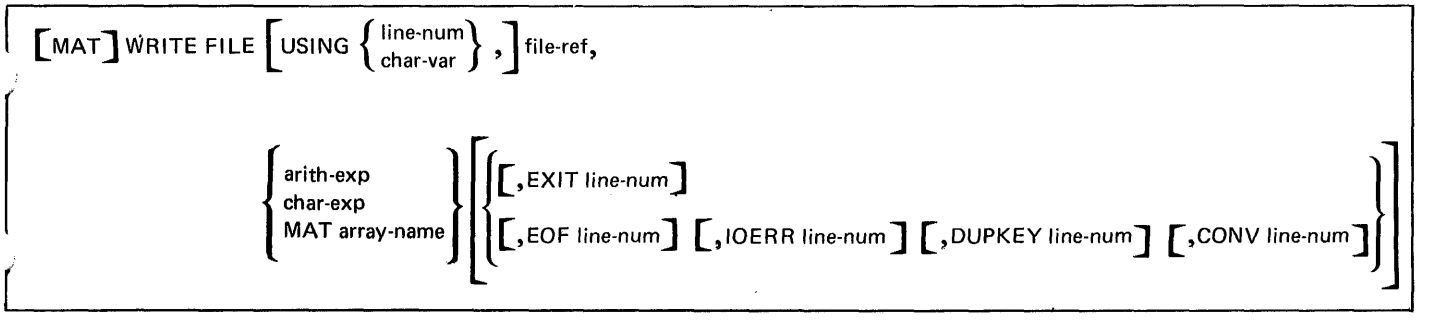
- If a variable is defined as an array in a USE statement, it must not be included in a DIM statement (see *DIM* in this chapter).
- USE must be specified in both the program chained from and chained to.
- If one or two dimensions are specified for a variable, the variable is assigned to the common storage area as an array with those dimensions.
- You can enter multiple USE statements in a program, but they must be the first statements containing variable references, and there must be no other statements between them. Thus, for example, the DEF, unconditional GOSUB, and GOTO statements can be executed before the USE statement.
- Variable data is assigned in the common area of storage in the same sequence in which it was entered in the USE statement. Thus, data values are unpredictable if data types differ from one chained program to the next.

Example

For instance, in this example:

```
0290 USE A(9,4), T, C
```

an array (A) with nine rows of four columns each is assigned to the common storage area. In addition, the scalar variables T and C are also assigned to the common storage area.



[MAT] WRITE FILE

The [MAT] WRITE FILE statement allows you to add a record at the end of a record I/O file. The record is written with a length equal to the length specified in the OPEN FILE statement. The USING parameter can be used to specify a character variable (which must contain valid FORM statement format specifications) or a FORM statement.

FORM statement specifications may cause truncation or padding of character variables and converison of numeric variables to various formats. Control specifications in the FORM statement also provide for changing the order of variables in the [MAT] WRITE FILE statement when written (through the POS specification). FORM statement control specifications also allow you to write null fields into the record. If the number of variables in the [MAT] WRITE FILE statement exceeds the control specifications in the FORM statement, the control specifications will be reused from the beginning of the FORM statement until all variables are formatted. Record fields not receiving a value are filled with blanks. Any control specifications beyond the last format specified will be honored. This is helpful in positioning the cursor in device '002'.

All of the parameters in the [MAT] WRITE FILE statement have the same meaning as those described for the [MAT] READ FILE statement.

Notes About WRITE FILE

- The KEY and REC parameters are not valid in the [MAT] WRITE FILE statement.
- If IN was specified in the OPEN FILE statement for the file referenced in a WRITE FILE statement, an error will occur.
- The EOF, IOERR, DUPKEY, and CONV error exits can be entered in any order.
- If RECL was specified in the OPEN for this file, the file will be written from the start. Otherwise, records will be added at the end of existing records with record lengths equal to those in the file.

Example

A sample WRITE FILE statement is shown below:

```
020 WRITE FILE USING 090, FL6,A$,B$,C$,EOF 888
```

In this example, values for A$, B$, and C$ are written at the end of file FL6, according to the control specifications in the FORM statement (line number 090).

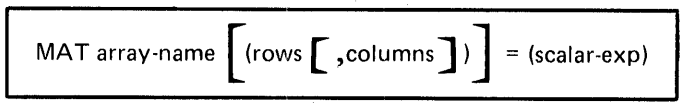
MATRIX OPERATIONS

In your system, an array can contain either numeric or character data, while a matrix can contain only numeric data. Before being used in MAT statements, arrays and matrices must have been defined, either implicitly or explicitly in a DIM or USE statement.

MAT ASSIGNMENT STATEMENTS

MAT assignment statements allow you to assign values to elements of an array. The value assigned can be derived from any of the following array expressions. Each array expression is discussed in detail later.

Array Expression	Meaning
(e)	Scalar value
A	Simple array
A + B	Matrix addition
A - B	Matrix subtraction
A * B	Matrix multiplication
(e) * A	Scalar multiplication
IDN	Identity function
INV (A)	Inverse function
TRN (A)	Transpose function
AIDX (A)	Index function (ascending order)
DIDX (A)	Index function (descending order)



MAT ASSIGNMENT (SCALAR VALUE)

This statement allows you to assign a specified scalar value to each element of an array. The syntax of this statement is as shown above, where:

array-name is the name of the array that receives the values.

rows,columns are the redimension specifications of the array (see *Redimensioning Arrays* in Chapter 3).

scalar-exp is the value to be assigned.

When this statement is executed, the parenthesized scalar expression is evaluated and each element in the named array is set to that value.

If redimension specifications are included, the truncated integer portion of each expression in *rows, columns* is used to redimension the array before the scalar value is evaluated and assigned to each of the array elements.

For character arrays, if the expression value is shorter than the array element, it is padded on the right with blanks to the specified length before being assigned. If there are more characters in the expression value than the array element length, only the leftmost characters are assigned to the array elements, and the remainder is truncated.

Notes About MAT (Scalar)

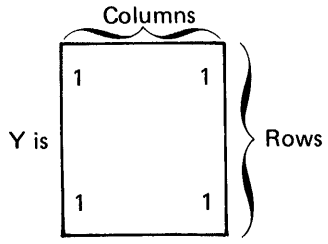
- The scalar expression to the right of the equal sign must be of the same type (arithmetic or character) as the array to which it is assigned.
- If redimension specifications follow the array name, the rules as stated under *Redimensioning Arrays* in Chapter 3 must be followed.

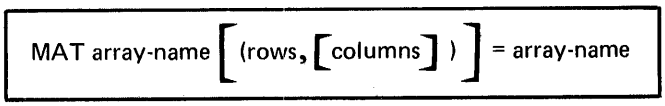
Example

The following example shows the execution of a MAT assignment (scalar) statement:

```
40 DIM Y (3,3)

50 MAT Y(2,2) = (1)
```

The resulting values are:





MAT ASSIGNMENT (SIMPLE)

The simple MAT assignment statement allows you to assign the elements of one array to another array. The syntax of this statement is as shown above, where:

array-name is the name of the array.

rows, columns are the redimension specifications for the first array (see *Redimensioning Arrays* in Chapter 3).

Each element of the array specified to the right of the equal sign is assigned to the corresponding element of the array specified to the left of the equal sign.

If redimension specifications follow the name to the left of the equal sign, the truncated integer portion of each expression value in *rows, columns* is used to redimension the array before values are assigned to it.

Notes About MAT (Simple)

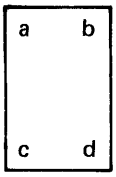
- Both arrays specified must be the same type (arithmetic or character).
- Both arrays specified in the array assignment statement must have identical dimensions (after redimensioning, if any).
- If redimension specifications are included, the rules described under *Redimensioning Arrays* must be followed.

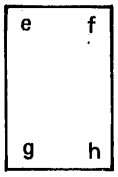
Example

The following example shows the execution of a MAT assignment (simple) statement:

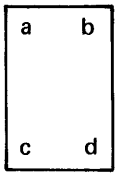
```
20 DIM A(2,2),B(2,2)
.
.
.
100 MAT A = B
```

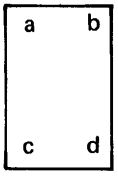
The resulting values are represented below:

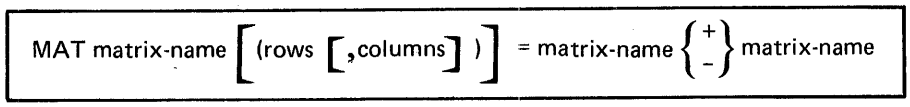
If B = 

and A = 

then, after statement 100:

B = 

A = 



MAT ASSIGNMENT (ADDITION AND SUBTRACTION)

The MAT assignment statement allows you to add or subtract the contents of two matrices and assign the result to a third matrix. The syntax of the statement is as shown above, where:

All parameters of the statement are the same as those for other MAT assignment statements.

The corresponding elements of the matrices specified to the right of the equal sign are added or subtracted, as indicated, and the result of the operation is assigned to the corresponding elements in the matrix specified to the left of the equal sign.

If redimension specifications follow the matrix name to the left of the equal sign, the truncated integer portion of each expression value in *rows*, *columns* is used to redimension the matrix before values are assigned to it.

Notes About MAT (Addition and Subtraction)

- All three matrices must be numeric.
- All three matrices specified in the statement must have identical dimensions (after redimensioning, if any).
- If redimension specifications are included, the rules described under *Redimensioning Arrays* in Chapter 3 must be followed.

Example

The following example shows execution of this statement:

```
10 DIM X(3,3), Y(2,2), Z(2,2)
.
.
.
100 MAT X (2,2) = Y + Z
```

The resulting values are:

If Y = $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ and Z = $\begin{bmatrix} e & f \\ g & h \end{bmatrix}$, then X = $\begin{bmatrix} a+e & b+f \\ c+g & d+h \end{bmatrix}$

MAT matrix-name [(rows,columns)] = matrix-name * matrix-name

MAT ASSIGNMENT (MATRIX MULTIPLICATION)

This statement allows you to perform the mathematical matrix multiplication of two numeric matrices and assign the product to a third matrix. The syntax of this statement is as shown above, where:

all parameters in the statement are the same as those for other MAT assignment statements.

In matrix multiplication, a matrix (A) of dimensions (p,m) and a matrix (B) of dimensions (m,n) yield a product matrix (C) of dimensions (p,n) such as that for i = 1,2...,p and for j = 1,2...,n:

$$C(i,j) = \sum_{k=1}^{m} A(i,k)* B(k,j)$$

If redimension specifications follow the matrix name to the left of the equal sign, the truncated integer portion of each expression value in *rows, columns* is used to redimension the matrix before values are assigned to it.

Notes About MAT (Matrix Multiplication)

- All three matrices specified must be numeric.
- If the matrix specified to the left of the equal sign is the same as either matrix to the right of the equal sign, you will get incorrect results.
- All of the following relationships must be true (after redimensioning, if any) where:

  A = B*C

  1. All three matrices must be two-dimensional
  2. The number of columns in the second matrix (B) must be equal to the number of rows in the third matrix (C).
  3. The number of rows in the first matrix (product matrix A) must equal the number of rows in the second matrix (B).
  4. The number of columns in the first (product matrix A) must equal the number of columns in the third matrix (C).

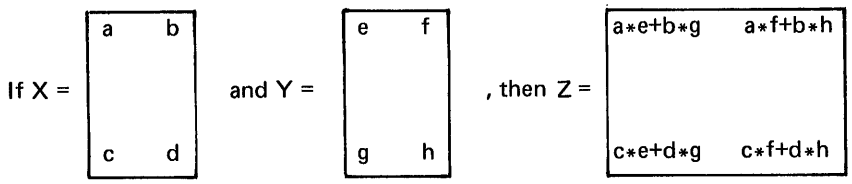
- If redimension specifications are included, the rules described under *Redimensioning Arrays* in Chapter 3 must be followed.

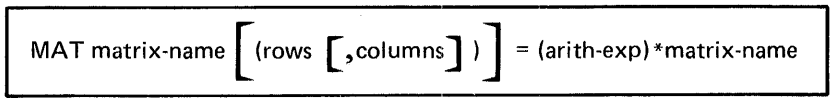
Example

The following example shows the execution of this MAT assignment statement:

```
10 DIM X(2,2), Y(2,2), Z(2,2)
 .
 .
 .
100 MAT Z = X * Y
```

The resulting values are:





MAT ASSIGNMENT (SCALAR MULTIPLICATION)

This statement allows you to multiply the elements of a numeric matrix by the value of an arithmetic expression, and assign the resulting products to the elements of another numeric matrix. The syntax of this statement is as shown above, where:

matrix-name is the name of a numeric matrix.

rows,columns are redimension specifications (optional).

(arith-exp) is a scalar arithmetic expression, which must be enclosed in parentheses.

The scalar expression is evaluated, and each element in the matrix to the right of the equal sign is multiplied by the value of the expression. The result is assigned to the corresponding elements of the matrix to the left of the equal sign.

If redimension specifications follow the matrix name to the left of the equal sign, the truncated integer portion of each expression value in *rows, columns* is used to redimension the matrix before the multiplication.

Notes About MAT (Scalar Multiplication)

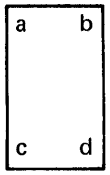
- Both matrices specified must be numeric.
- Both matrices specified must have identical dimensions (after redimensioning, if any).
- If redimension specifications are included, the rules under *Redimensioning Arrays* in Chapter 3 must be followed.

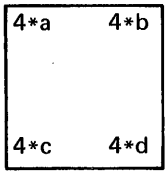
Example

The following example shows execution of a MAT assignment statement:

```
20 DIM X(2,2), Y(2,2)
 .
 .
 .
100 MAT Y = (4) * X
```

The resulting values are:

If X = 

, then Y = 

```
MAT matrix-name [(rows,columns)] = IDN
```

MAT ASSIGNMENT (IDENTITY FUNCTION)

This statement allows you to make a numeric matrix assume the form of an identity matrix. The syntax of the statement is as shown above, where:

all the parameters are the same as those for other MAT assignment statements, and IDN specifies identity matrix.

Each element of the specified matrix for which the values of both subscripts are equal, for example, A(2,2) or A(3,3), is assigned the integer value 1. All other elements, for example A(2,3) or A(3,1), are assigned the value 0.

If redimension specifications follow the matrix name, the truncated integer portion of each expression value in *rows*, *columns* is used to redimension the matrix before the assignment of 1 or 0 to each of its elements.

Notes About MAT (Identity Function)

- The matrix specified must be numeric.
- The specified numeric matrix must be a square matrix; that is, the number of rows must equal the number of columns (after redimensioning, if any).
- If redimension specifications are included, the rules under *Redimensioning Arrays* in Chapter 3 must be followed.

Example

The following example shows the execution of a MAT (identity function) statement:

```
50 DIM X(16)
 .
 .
 .
60 MAT X(4,4) = IDN
```

The resulting values are:

X is

1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

MAT matrix-name [(rows, columns)] = INV (matrix-name)

MAT ASSIGNMENT (INVERSE FUNCTION)

This statement allows you to assign the mathematical matrix inverse of one matrix to another matrix. The syntax of the statement is as shown above, where:

all parameters are the same as those for other MAT assignment statements, and *INV* specifies the inverse function.

The matrix inverse of the matrix specified to the right of the equal sign is assigned to the matrix specified to the left of the equal sign. For the square matrix A of dimensions (m,m), the inverse matrix B, if it exists, is a matrix of identical dimensions such that:

A*B = B*A = I

where I is an identity matrix.

Not every matrix has an inverse. The system function DET (see *System Functions* in Chapter 3) can be used to determine whether a given matrix has an inverse. The inverse of matrix A exists if DET(A)≠ 0.

If redimension specifications follow the matrix name to the left of the equal sign, the truncated integer portion of each expression value in *rows*, *columns* is used to redimension the matrix before values are assigned to it.

Notes About MAT (Inverse Function)

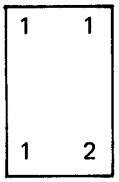
- Both matrices specified must be numeric.
- Both matrices specified must be square, and both must have identical dimensions (after redimensioning, if any).
- The determinant is considered zero and the matrix singular (inverse undefined) if the result is 1E-20 or less.
- If redimension specifications are included, the rules under *Redimensioning Arrays* in Chapter 3 must be followed.

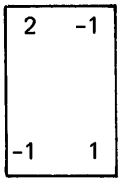
Example

The following example shows execution of a MAT (inverse) statement:

```
20 DIM X(2,2), Y(2,2)
 .
 .
 .
80 IF DET (Y) = 0 GOTO 300
90 MAT X = INV (Y)
 .
 .
 .
295 GOTO 310
300 PRINT 'SINGULAR MATRIX'
310 STOP
```

The resulting values are:

If Y is 

, then X is 

```
MAT array-name [(rows,columns)] = TRN (array-name)
```

MAT ASSIGNMENT (TRANSPOSE FUNCTION)

This statement allows you to replace the elements of one array with the matrix transpose of another array. The syntax of the statement is as shown above, where:

all parameters in the statement are the same as those for other MAT assignment statements, and *TRN* specifies the transpose function.

The transpose matrix of the array specified to the right of the equal sign is assigned to the array specified to the left of the equal sign. The values in column y of one array become the values in row y of the other array.

If redimension specifications follow the array name to the left of the equal sign, the truncated integer portion of each expression value in *rows, columns* is used to redimension the array before values are assigned to it.

Notes About MAT (Transpose Function)

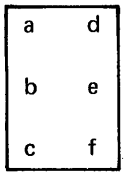
- Both arrays specified must be two-dimensional, and the number of rows in each array must be equal to the number of columns in the other (after redimensioning, if any), and they must be the same type (character or numeric).
- The same array cannot be used on both sides of the equal sign. This will cause incorrect results.
- If the redimension specifications are included, the rules under *Redimensioning Arrays* in Chapter 3 must be followed.

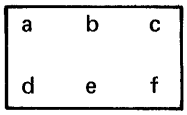
Example

The following example shows the execution of a MAT (transpose function) statement:

```
40 DIM A(3,2), B(2,3)
 .
 .
 .
80 MAT B = TRN(A)
```

The resulting values are:

If A is 

, then B is 

MAT matrix-name [(rows)] = AIDX (array-name)

MAT ASSIGNMENT (ASCENDING INDEX)

This statement allows you to index the elements of an array and assign the index values to another array. Character arrays are indexed alphabetically, and numeric arrays are indexed numerically. The syntax of the statement is as shown above, where:

matrix-name, array-name are one-dimensional character or numeric arrays.

rows is the dimension of the array named.

AIDX indicates the ascending index function.

When a MAT AIDX statement is executed, index values are assigned to the matrix on the left of the equal sign, according to the order of the values entered into the array on the right of the equal sign. In other words, the system determines the sequence of the values, then indicates the positions in the array of the values in ascending order.

Notes About Ascending Index

- Both arrays specified must be one-dimensional with the same number of elements.
- The array on the left of the equal sign must be numeric.

Example

The following example shows the execution of a MAT AIDX statement:

```
30 DIM A(10),B(10)
40 MAT B = AIDX(A)
```

If array A =	then array B will contain
9	4
5	9
6	5
0	10
2	8
7	2
8	3
4	6
1	7
3	1

Note that the numbers in array B show the position of the numbers in ascending order as they appear in array A (0 is in the fourth position, 1 is in the ninth position, and so on).

MAT matrix-name [(rows)] = DIDX (array-name)

MAT ASSIGNMENT (DESCENDING INDEX)

This statement allows you to index the elements of an array and assign the index values to another array. Character arrays are indexed alphabetically and numeric arrays are indexed numerically. The syntax of the statement is as shown above; where, all of the parameters have the same meaning as those for the ascending index statement (AIDX).

The execution of the descending index statement is the same as execution of the ascending index statement (AIDX) except that array elements are indexed in descending order.

Notes About Descending Index

- Both arrays specified must be one-dimensional with the same number of elements.
- The array on the left of the equal sign must be numeric.

Example

The following example shows the execution of a MAT DIDX statement:

```
30 DIM A(10) ,B(10)
40 MAT B= DIDX(A)
```

If array A =	9	then array B will contain	1
	5		7
	6		6
	0		3
	2		2
	7		8
	8		10
	4		5
	1		9
	3		4

Note that the numbers in array B show the position of the numbers in descending order as they appear in array A (9 is in the first position, 8 is in the seventh position, and so on).

Chapter 5. More Information About Your System

5110 BASIC COMPATIBILITY WITH IBM 370/VS BASIC

The BASIC language used in the 5110 differs from IBM 370/VS BASIC in the following areas:

- System file reference codes are limited to FLS,FLP, and FL0-FL9, which appear in I/O statements as unquoted strings.
- Double quotation marks are not available.
- System OPEN statements contain more information than VS BASIC.
- The system provides hexadecimal constants, which are enclosed in single quotation marks and preceded by X.
- The system provides ascending/descending index on MAT assignment statements (AIDX/DIDX).
- The system allows you to direct formatted output to either the printer, or the display screen, or a tape/diskette file. This capability is provided through an additional parameter (FLP) in PRINT, MAT PRINT, PRINT USING, and MAT PRINT USING statements. An entry of FL0-FL9 in this position directs output to the file referred to in a corresponding OPEN statement. VS BASIC uses PRINT TO statements for this purpose.
- The system allows you to specify file description information that is provided by the operating system (in 370/VS BASIC) via the OPEN [FILE] statement.
- The system allows you to CHAIN to a specified program without initializing the data area reserved by variables specified in a USE statement.
- The system allows you to list data in the USE statement, including dimensions for arrays. In conjunction with a CHAIN statement, the USE statement allows you to pass data from one program to another.
- The system does not provide the following VS BASIC functions: CLK,CNT,CPU,DAT(x), DOT(X,Y), JDY(x), and TIM.

- The system provides error exit clauses on INPUT and PRINT statements.
- The system requires statement numbers with values up to 9999, and permits special functions to be assigned by the user to numbers 9990-9999.
- The system does not support ascending/descending sort (ASORT/DSORT) on MAT assignment statements.
- The system does not support the ELSE clause on the IF statement.
- The system does not support the overflow/underflow and INERR clauses on the ONERROR statement.
- The system does not recognize the exclamation (!) symbol in an image statement, although the symbol can be entered by overstriking the single quotation mark and decimal point.

5110 BASIC COMPATIBILITY WITH 5100 BASIC

Differences Between 5110 BASIC and 5100 BASIC

The 5110 BASIC differs from the 5100 as follows:

- Programs saved in internal code are not compatible between the 5110 and the 5100. For example, if the command SAVE 3 is used to save a program on the 5100, that program cannot be loaded into a 5110.

  Note: Programs saved in source code are compatible between the 5110 and the 5100. For example, if the command SAVE 3 SOURCE is used to save a program on the 5100, that program can be loaded into a 5110. For more information on loading source code, see *Converting 5100 Programs to 5110 Programs* in this chapter.

- The 5110 uses the EBCDIC character set; the 5100 does not. For example, the hex constants for the new line, end of record, and % characters are X'E3', X'FF', and X'73' for the 5100 and X'15', X'1E', and X'6C' for the 5110.

- The internal numeric format for the 5110 is binary (base 2) floating point. The internal numeric format for the 5100 is packed decimal (base 10) floating point. In some situations this difference can cause different results (for more information, see *Binary Floating-Point Arithmetic Considerations* in this chapter).

- The 5110 user-defined functions use the value of local and global variables as follows:

  First, assume the following conditions exist:
  - The variable X is globally defined.
  - FNA uses X as a parameter and local variable.
  - FNA invokes FNB.
  - FNB does not use X as a parameter, but FNB does refer to variable X.

When FNA invokes FNB, the value of X used by FNB is the same value as parameter X for FNA. When FNB is invoked independently from FNA, the value of X used by FNB is the global value of X.

For the 5100, the value of variable X in FNB is always the global value of X.

- For the 5110, stream I/O data files are type 2 files. For the 5100, you use negative file numbers to create type 2 files. If you use negative file numbers on the 5110, an error occurs when the file is opened.
- The 5110 requires 241 bytes more overhead from the user work area than did the 5100. Therefore, some programs that executed on the 5100 might generate a work area full condition in the 5110.
- When 5110 keyboard-generated data files are used, the word DATA is replaced with a colon (:).
- For the 5110, the CHAIN statement does not cause a single line feed to the printer.
- For the 5110, character variables used with the USE statement are initialized to binary zeros. For the 5100, the character variables are initialized to blanks.
- For the 5110, if a stream I/O file is opened for output, issuing a RESET END statement causes an error.
- For the 5110, when you are chaining to another program, pressing ATTN while the tape drive is operating does not cause the tape to stop. Instead, you must remove the tape from the tape drive to stop the operation.
- For the 5110, when you open a file for input and specify a character variable for identification, the user ID from the file is *not* placed in the character variable when the OPEN statement is executed.
- The range of valid numbers for the 5110 differs from the 5100. Constants greater than 7.237E75 and less than 9.9999999999999E99 are not accepted by the 5110. In addition, constants with magnitude less than 5.3976E-79 and greater than 1E-99 are not accepted by the 5110.

Converting 5100 Programs to 5110 Programs

Any syntactically valid 5100 BASIC SOURCE program will run on the 5110, except programs that open a file with negative file numbers or use the RESET END statement with stream I/O files. Following are some considerations that will help you use 5100 programs on the 5110:

- You should check the 5100 program to determine how the differences between the 5100 BASIC and 5110 BASIC might affect the results generated by the program (see *Differences Between 5110 BASIC and 5100 BASIC* earlier in this chapter). Also a sample program, which might aid you when using 5100 programs on the 5110, is discussed later in this chapter.

- Because the BASIC internal code is not compatible between the 5100 and the 5110, you must load any 5100 programs that are stored in internal code in the 5100 work area and then save the program in source code. The source code can then be loaded into a 5110. When a BASIC SOURCE program is loaded into the 5110, each statement is displayed and the statement is syntax checked.

  In some cases, when BASIC statements are entered on the 5100, you might delete leading zeros in the statement number and blank spaces in the statement. Then, when the program source code is loaded in the 5110, the 5110 inserts the leading zeros and blanks back into the statement. This might cause the statement to exceed 64 characters; therefore, an error occurs when the statement is syntax checked. You can then press ATTN (to stop the display screen from flashing), modify the statement, and press EXECUTE to continue loading the source code.

- You can convert existing tape data files to diskette data files by using the tape-to-diskette copy function (see the *Customer Support Functions Reference Manual*). Remember that diskette files must have a simple or complex name. If the tape file name is longer than 8 characters, the tape-to-diskette copy function allows you to enter a new file name.

- User-defined keys cannot be saved in source code on the 5100; therefore, they must be rekeyed on the 5110.

The following sample 5100 BASIC to 5110 BASIC conversion assistance program can be used to aid you in converting 5100 programs to 5110 programs. The program does the following:

1. Prints a listing of the program and indicates the statements that may require your attention.

Type of Condition	**Possible Changes**
Open	– Change device. – Add file name for output files to diskette. – Change file name.
CHAIN	– Change device.
Hex assignment	– Change hex code, if necessary.
IF	– Examine to determine whether implications of binary floating-point must be considered.
INT	– Examine to determine whether implications of binary floating-point must be considered.

2. Edits lines over 64 characters by eliminating unneeded zeros and blanks. If this cannot be done, the line is displayed for you to edit.

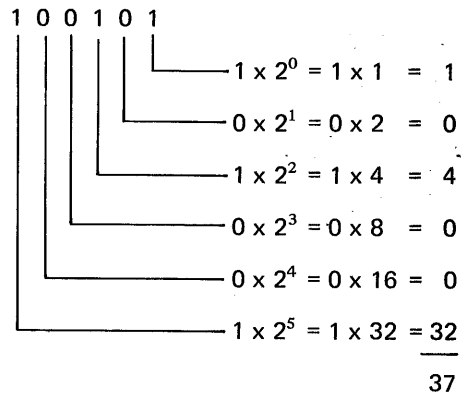
3. Writes a new copy of the program in SOURCE code to tape or diskette.

5100 Conversion Assistance Program

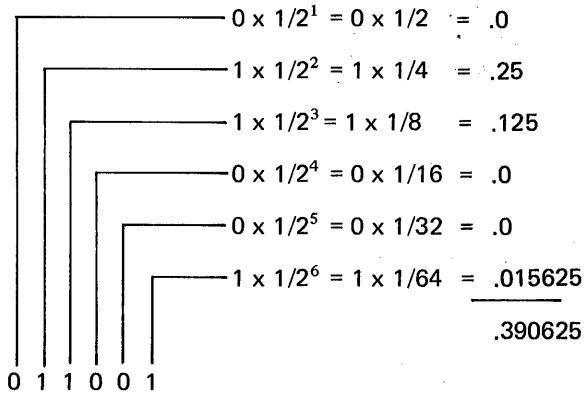
```
0005 REM PROGRAM TO CONVERT BASIC PROGRAMS FROM 5100
0010 DIM R$128,X$1
0020 X$=X'15'
0030 PRINT 'ENTER DEV.ADDRESS, FILE # OF INPUT FILE'
0040 Y=I
0050 INPUT D$,F1
0060 PRINT 'ENTER DEV.ADDRESS,FILE# & FILE NAME OF OUTPUT FILE'
0070 INPUT E$,F2,F$
0080 OPEN FL1,D$,F1,IN
0090 OPEN FL2,E$,F2,F$,OUT
0100 WRITEFILE FLS,'       FL1'
0110 GET FL1,R$,EOF 480
0120 IF STR(R$,6,5)='CHAIN' GOTO 210
0130 IF STR(R$,6,4)='OPEN' GOTO 210
0140 IF STR(R$,6,2)='IF' GOTO 210
0150 X=IDX(R$,'=X''')
0160 IF X≠0 GOTO 210
0170 X=IDX(R$,'INT(')
0180 IF X≠0 GOTO 210
0190 N$=' '
0200 GOTO 220
0210 N$='NOTE--->'
0220 IF LEN(R$)≤64 GOTO 450
0230 FOR I=1 TO 3
0240 IF STR(R$,I,1)≠'0' GOTO 260
0250 NEXT I
0260 R$=STR(R$,I,5-I)‡‡STR(R$,6)
0270 IF LEN(R$)≤64 GOTO 450
0280 FOR I=1 TO LEN(R$)
0290 IF STR(R$,I,1)≠'''' GOTO 320
0300 I=IDX(STR(R$,I+1),'''')+1
0310 GOTO 350
0320 IF STR(R$,I,1)≠' ' GOTO 350
0330 R$=STR(R$,1,I-1)‡‡STR(R$,I+1)
0340 IF LEN(R$)≤64 GOTO 450
0350 NEXT I
0360 PRINT '*** UNABLE TO SHORTEN TO 64 ***'
0370 WRITEFILE FLS,'A'
0380 OPEN FL3,'001',IN
0390 WRITEFILE FLS,'       FL3'
0400 PRINT 'PLACE CORRECTED ENTRY ON INPUT LINE'
0410 PRINT R$
0420 GET FL3,R$
0430 CLOSE FL3
0440 GOTO 220
0450 PRINT FLP,N$;TAB(10);R$
0460 PRINT FL2,R$;X$;
0470 GOTO 110
0480 STOP
```

Binary Floating-Point Arithmetic Considerations

Arithmetic operations in the 5110 are performed using the binary representation of the numbers. For example, the number 37 is represented in binary as:



When fractions are used, the number 37.4 appears to be a simple number. However, the 5110 can only approximate .4. For example:



Notice that as the powers of 2 are increased, the accumulated total becomes closer to .4. The 5110 uses 24 bits (short precision) or 56 bits (long precision) for decimal numbers. For output, the 5110 converts the number back to decimal and then rounds the number so that the output you see is .4.

However, it is important for you to remember that even though the number stored in the 5110 is only slightly different from .4, the number is in fact different. For example, assume that the 5110 is used for accounting purposes to charge projects according to how an employee spends his time during the day. Management expect 8 hours to be accounted for, and the 5110 receives the following numbers:

2.3

2.5

3.2

When the 5110 adds these numbers together, a number slightly less than 8 is stored internally. If a program compares the calculated number to 8, the 5110 would indicate that 8 hours were *not* accounted for when in fact they were.

Because whole numbers can be exactly represented in the binary system, one way to obtain exact comparisons is to multiply both numbers by 100 after rounding, then making the compare on the integer portion. For example, instead of IF A = B GOTO 200 use:

```
IF FNC(A) = FNC(B) GOTO 200
```

where DEF FNC(X) = INT(100*(X+(SGN(X)*.0001))).

Another way to obtain exact comparisons is to check for a certain number of digits to be the same. For example:

```
IF ABS(A-B) <= .000001 THEN 0070
```

Another situation occurs when the INT function is used:

```
INT(8) = 8

INT(7.999999) = 7
```

If the calculated result is slightly less than the expected result, the INT function can return an integer value that is less than expected. In this case, you should use a user-defined function instead of the INT function. For example, instead of the INT(X) use:

```
FNI(X)
```

where DEF FNI(X) = (INT(X+SGN(X)*.0001)) so that FNI(7.999999) = 8.

TAPE CARTRIDGE HANDLING AND CARE

- Protect the tape data cartridge from dust and dirt. Cartridges that are not needed for immediate use should be stored in their protective plastic envelopes.

- Keep data cartridges away from magnetic fields and from ferromagnetic materials which might be magnetized. Any cartridge exposed to a magnetic field may lose information.

- Do not expose data cartridges to excessive heat (more than 130° F or 54° C) or sunlight.

- Do not touch or clean the tape surface.

- If a data cartridge has been exposed to a temperature drop exceeding 30° F or -1° C since the last usage, move the tape to its limits before using the tape. The procedure for moving the tape to its limits is:

  1. Use the UTIL command to move the tape to the last marked file.

  2. Use the MARK command to mark from the last marked file to the end of the tape. For example:

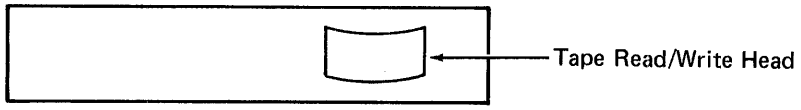
     MARK 200,1,n

     where n is the number of the last marked file, plus one. This will cause an error (end of tape). Press the ATTN key to continue.

  3. Use the REWIND command to rewind the tape.

Tape Head Cleaning Procedure

Occasional cleaning of the tape read/write head provides more reliable operation of the tape drive. Use a soft lint-free cloth or paper towel dampened with isopropyl to clean tape oxide from the tape head. Then wipe the tape head dry.



Tape Drive Opening

STORAGE CONSIDERATIONS

The following list shows how many bytes of storage are required for each data type that can be stored in the work area:

Data Type	Number of Bytes Required
Character variable 18-byte format	22
Character variable not 18-byte format	Length of the variable plus 5
Character array 18-byte format	18 times the number of elements plus 10
Character array not 18-byte format	Element length times the number of elements plus 11
Numeric variable	12
Numeric array	8 times the number of elements plus 10

Because the 5110 work area contains a fixed amount of storage, it is a good practice to conserve as much storage as possible.

DISKETTE HANDLING AND CARE

Operation

Diskette Insertion

CAUTION
If a diskette has been exposed to temperatures outside the recommended range (50° F to 125° F or 10° C to 51° C), keep it at room temperature for about five minutes before inserting it in the drive.

1. Open the diskette drive cover.
2. Remove the diskette from its envelope by grasping its upper edge and lifting.
3. Insert the diskette into the drive by grasping the diskette by its upper edge and carefully placing it in the drive.
4. Close the cover only after the diskette has been fully inserted.

Diskette Removal

1. Open the diskette drive cover.
2. Remove the diskette by grasping its upper edge and pulling it straight out.
3. Slide the diskette into its envelope and return it to a clean storage area.

Handling Defective Cylinders

With use, areas can develop on the disk surface on which readable records cannot be written. A diskette with a defective area should normally be removed from service. The 5110 is capable of reinitializing diskettes and assigning up to two alternate cylinders. In this case, do not use a diskette with a defective area before reinitializing it to bypass the cylinder containing the defective area. Then record the number of the defective cylinder on the permanent label.

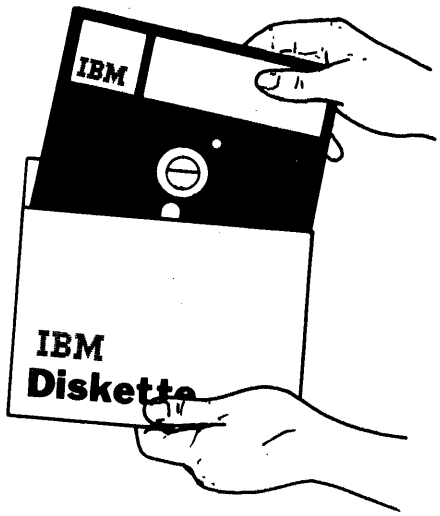
If diskette errors occur, you must make a decision regarding replacement of individual diskettes. The following procedures can help with this decision:

- When using a new diskette, assign a serial number to it and record that number on the diskette permanent label and in the space provided in the diskette internal label (volume ID field).
- Keep a log of diskette serial numbers and the initial date used so you can estimate wear by the diskette age.
- Whenever a diskette error occurs on the same cylinder repeatedly, reinitialize the diskette as soon as possible. To reinitialize, first copy any useful data from the diskette. As part of the initialization routine, the device assigns cylinder and sector numbers to the diskette, bypasses the defective cylinder, and assigns the cylinder number of the defective cylinder to the next good cylinder. For two-sided diskettes, both tracks of a cylinder must be relocated if either is defective. Two defective cylinders per diskette can be replaced in this manner.
- Periodically examine the log of diskette serial numbers and the permanent labels on the diskettes. If a diskette is too old for further use, or if there are more than two defective cylinders, replace the diskette.

Handling Precautions

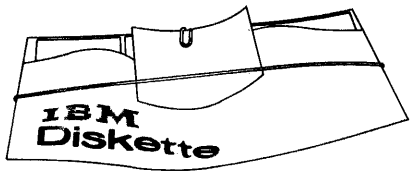
Replace the diskette if it is physically damaged (torn, folded, creased), or if the recording surface becomes contaminated. It is particularly important that you do not use diskettes which are contaminated with sticky fluids (soft drinks, coffee) or abrasive substances (metal filings) on the recording surface. Placing a contaminated diskette in a device can contaminate the read/write head, causing operation errors. In addition, contaminants can be passed to clean diskettes. A substance spilled on the diskette jacket can be removed and the data recovered only if the contaminant does not reach the recording surface. After recovering the data, discard the diskette.

To remove a diskette from its envelope, grasp the diskette by its upper edge and pull.

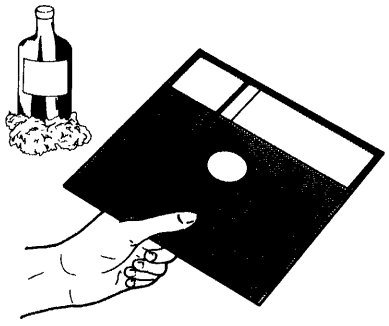


Return the diskette to its protective envelope whenever it is removed from the diskette drive and whenever you are writing on a label on the diskette.

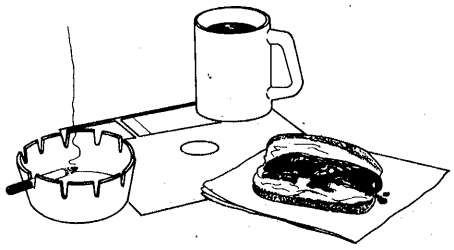
Do not bend or fold the diskette. Do not use rubber bands or paper clips on the diskette.



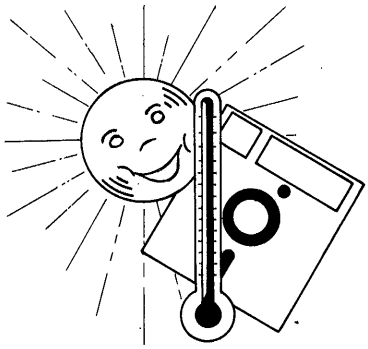
Do not touch or clean the exposed diskette surface.



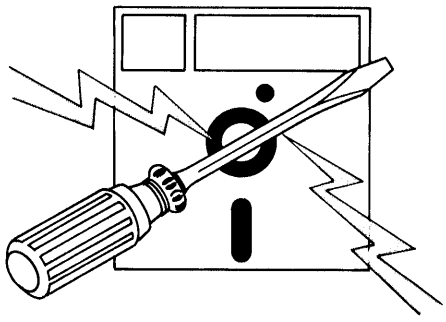
Do not smoke, eat, or drink while handling the diskette.



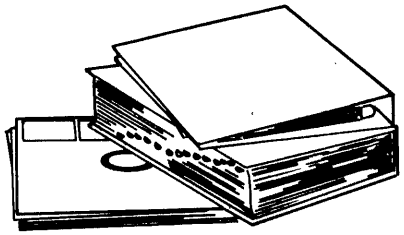
Do not expose the diskette to excessive heat or sunlight.



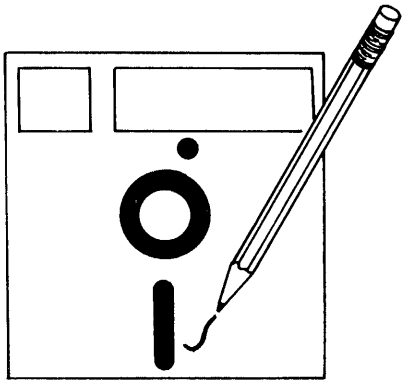
Do not use magnets or magnetized objects near the diskette. Data can be lost from a diskette that is exposed to a magnetic field.



Do not place heavy objects on the diskette.



Do not erase labels attached to the diskette, or make any erasures on or near the diskette. Erasure residue could get in the diskette, and this should be avoided. To discourage erasures, IBM recommends that you use a fiber-tip or ball-point pen when marking on the diskette labels. Mark temporary labels before attaching them to the diskette. Alter temporary labels with the diskette in the envelope.



Storage

Environment

Temperature: 50° F to 125° F (10° C to 51° C)
Relative humidity: 8% to 80%
Maximum wet bulb temperature: 85° F (29° C)

Short-Term Storage

Store diskettes needed for immediate use flat in their envelopes, in stacks of ten or less. If storing vertically, support the diskettes so they do not lean or sag.

Long-Term Storage

Store diskettes not needed for immediate use in their original shipping cartons, with each diskette in its protective envelope. Shipping cartons can be stored either vertically or horizontally.

CAUTION
Do not apply pressure to diskette envelopes or cartons, because pressure can warp the diskettes.

Shipping and Receiving

When shipping a diskette, always label the package DO NOT EXPOSE TO HEAT OR SUNLIGHT. When receiving a diskette, check the carton and the diskette for damage.

To pack one diskette:

- Place the diskette in its protective envelope.
- Put the envelope in a single-diskette carton.

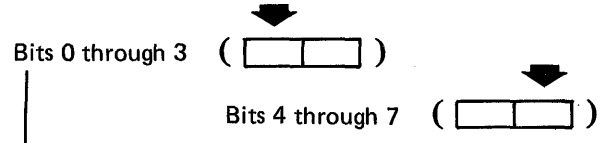
To pack multiples of 10 diskettes:

- Place each diskette in its protective envelope.
- Put 10 diskettes in a 10 diskette box.
- Put each 10-diskette box between spacers to prevent damage during shipping.
- Insert top and bottom pads in the carton.
- Place the 10-diskette boxes and their spacers in the appropriate size carton.
- Fill the open space in partially filled cartons and 10-diskette boxes with a filler that cannot contaminate the diskette or enter the diskette jacket.

CAUTION
Do not use so much filler that diskettes are tightly compressed; compression can warp the diskettes.

Appendix A. 5110 BASIC Characters and Hexadecimal Representation

The following chart lists all the EBCDIC characters and their hexadecimal representation.



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
2																
3																
4		A	B	C	D	E	F	G	H	I	¢	.	<	(	+	¦
5	&	J	K	L	M	N	O	P	Q	R	!	$	*	)	;	¬
6	-	/	S	T	U	V	W	X	Y	Z	¦	,	%	_	>	?
7	&	^	¨	A		W			v	`	:	#	@	'	=	"
8	~	a	b	c	d	e	f	g	h	i	↑	↓	≤	Γ	L	→
9	[]	j	k	l	m	n	o	p	q	r	⊃	⊂		o		←
A	¨	~	s	t	u	v	w	x	y	z	∩	U	⊥	[	≥	°
B	α	€	ι	ρ	ω		×	\	÷		∇	Δ	т	]	≠	ı
C	{	A	B	C	D	E	F	G	H	I	∝	∾	∫	Φ	Ψ	Ω
D	}	J	K	L	M	N	O	P	Q	R	∝	!	Ψ	Φ	[]	A
E	\		S	T	U	V	W	X	Y	Z	∕	\	∩	⊖	⊟	₮
F	0	1	2	3	4	5	6	7	8	9	¦	∾		⊕	±	

For example, the character 0 has value of X'F0'.

Note: Graphics are assigned to all blank positions, except hex 40, for maintenance use only. They are incompatible with other systems and cannot be used for exchange purposes. These graphics may be removed or changed as a result of maintenance or new versions of this product.

In the following illustrations (Figures 11 and 12), the characters on the keys correspond to the following diagram.

KEYBOARD CHARACTERS FOR XXXX

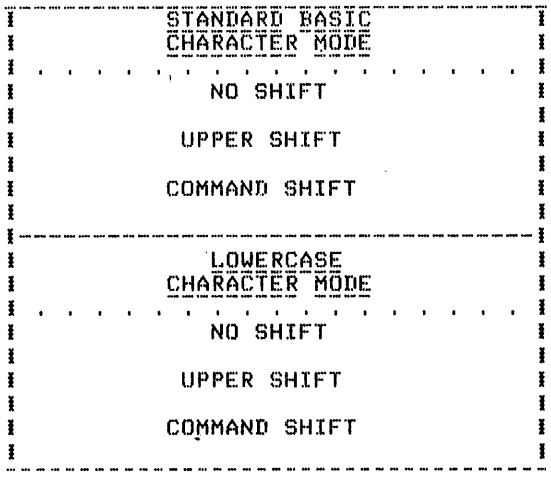


Figure 11. 5110 Keyboard Characters

Figure 12. 5110 Keyboard Characters

Appendix B. 5103 Printer

The 5103 Printer has the following characteristics:

- Bidirectional printing (left to right and right to left). The print head moves from the left margin and prints a line. Succeeding lines will be printed in either direction depending on which end of the new line is closest to the current position of the print head. The print head will be returned to the left margin periodically when printing is not imminent.

- A maximum print line of 132 characters.

  Note: If 132 characters are formatted for forms less than 132 characters wide, loss of data will occur as the print head leaves the form.

- Capability of using individual or continuous forms. Maximum number of copies is six, but for optimum feeding and stacking, IBM recommends a maximum of four parts per form.

- Adjustable forms tractor that allows the use of various width forms. The forms can be from 3 to 14.5 inches (76.2 to 368.3 mm) wide for individual forms, and from 3 to 15 inches (76.2 to 381 mm) wide for continuous forms.

- Print position spacing of 10 characters per inch (2.54 cm) and line spacing of six lines per inch (2.54 cm).

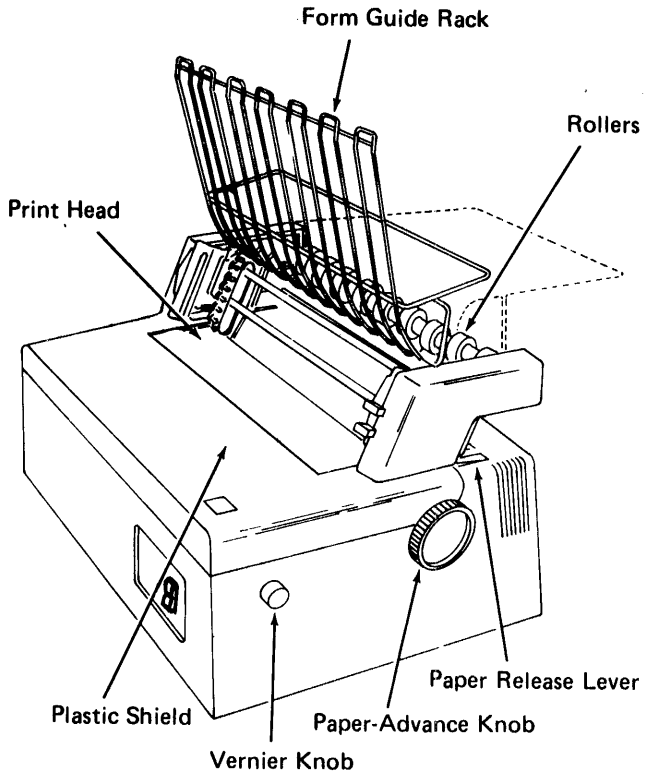
- Stapled forms or continuous card stock cannot be used.

- The character printing rate is 80 or 120 characters per second. The throughput in lines per minute is program-dependent.

- A vernier knob (located on the right side of the printer) that allows for fine adjustment of the printing position. This knob should only be used when the print head is in its leftmost position.

HOW TO INSERT FORMS

Continuous Forms



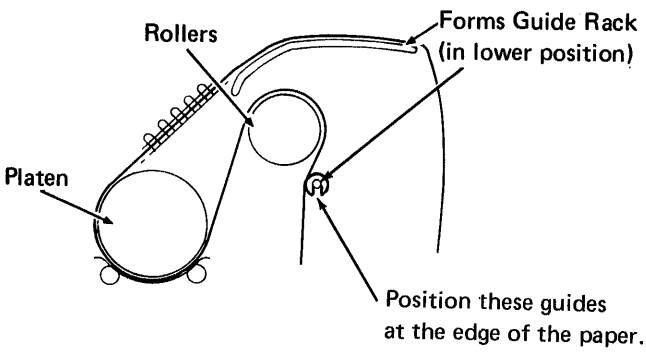
1. Slide the printer cover forward.
2. Push the print head to the extreme left position.
3. For singlepart forms pivot the form guide rack up and forward to a vertical position. For multipart forms, leave the forms guide rack in the horizontal position.

   The diagrams below show the proper forms path for singlepart and multipart forms.

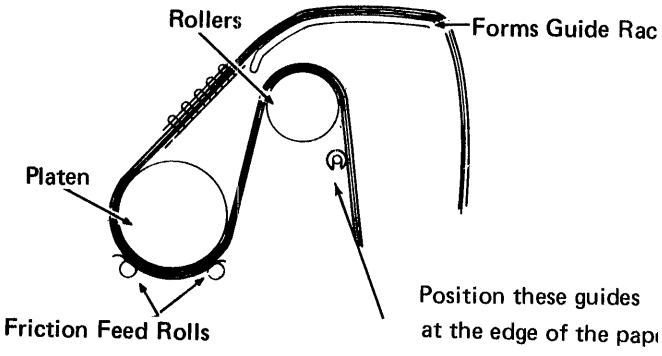
4. Push the paper release lever to the rear to activate the friction feed rolls.
5. Place the forms on the table behind the printer.

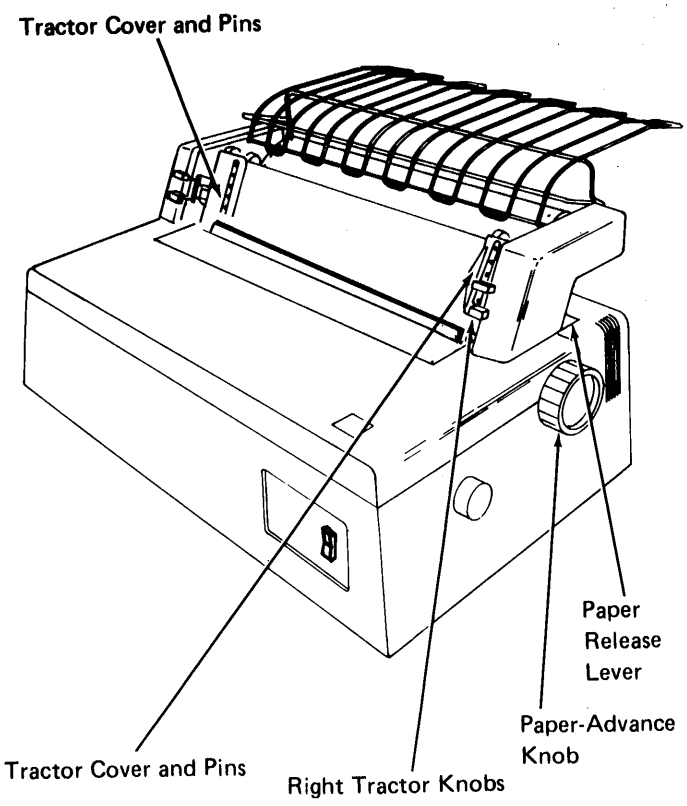
   Note: The forms must be positioned behind the printer so that the forms feed squarely into the printer.
6. Thread the paper down, over the rollers, behind the tractors, and behind the platen.
7. Turn the paper-advance knob to move the paper around the platen until you can grasp it with your fingers.

Forms Path for Singlepart Forms



Forms Path for Multipart Forms





8. Open both tractor covers.

9. Pull the paper release lever forward to disengage the friction feed rolls.

10. Pull the paper up and place the left margin holes over the tractor pins. Be sure the left tractor is in its leftmost position.

11. Close the left tractor cover.

12. Squeeze the two knobs on the right tractor and slide the tractor to align the pins with the right margin holes.

13. Place the right margin holes over the tractor pins.

14. Close the right tractor cover.

15. For singlepart forms, pivot the form guide rack to a horizontal position.

16. Turn the paper-advance knob to position the form for the first line to be printed. The paper should exit over the forms guide rack.

    Note: To move the form backward, turn either paper-advance knob backward and pull the form from behind the printer to keep the form from buckling at the print head.

17. Slide the printer cover closed.

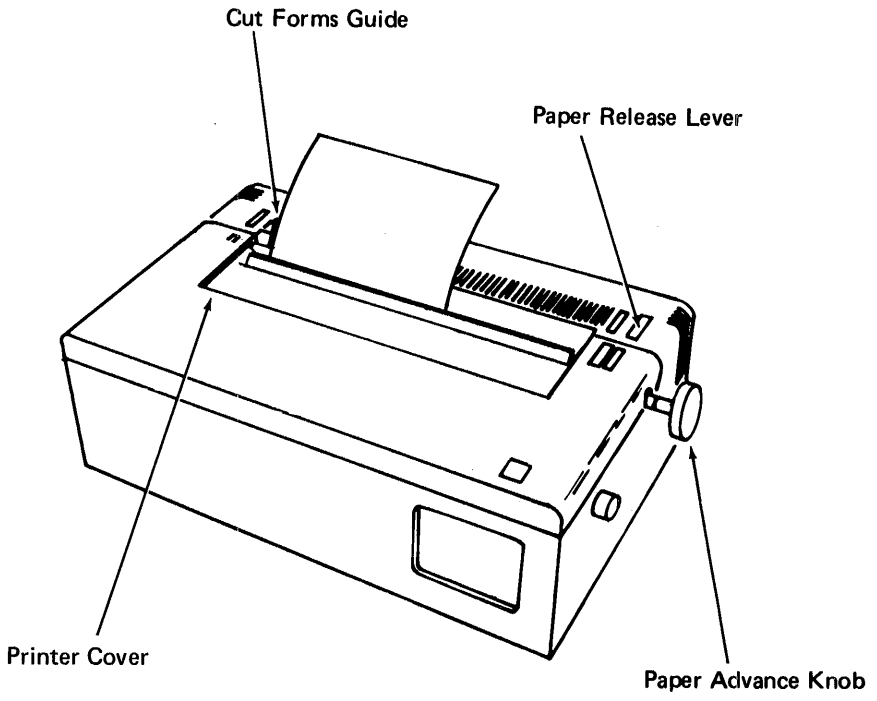
18. The plastic guides on the rear of the wire rack should be positioned (one on each side of the forms) so as to aid in guiding the forms for proper feeding. These guides are positioned by sliding them back and forth.

CAUTION
The switch that senses end of forms is deactived when the friction feed rolls are engaged. Thus, the print wires could hit the base platen if no forms are in the printer.

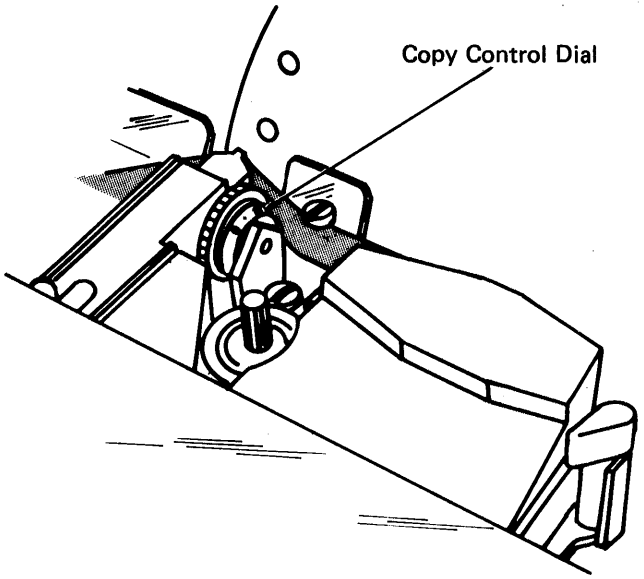
Cut Forms

1. Remove the forms tractor by tilting it back and lifting it off.
2. Move the cut forms guide forward.
3. Slide the printer cover forward.
4. Push the print head to the extreme left position.
5. Push the paper release lever to the rear to activate the friction feed rolls.
6. Place the form in position behind the platen and against the cut forms guide.
7. Turn the paper-advance knob to position the form for the first line to be printed. Improve the paper alignment if necessary by using the paper release lever.
8. Slide the printer cover closed.

CAUTION
The switch that senses end of forms is deactivated when the friction feed rolls are engaged. Thus, the print wires could hit the base platen if no forms are in the printer.

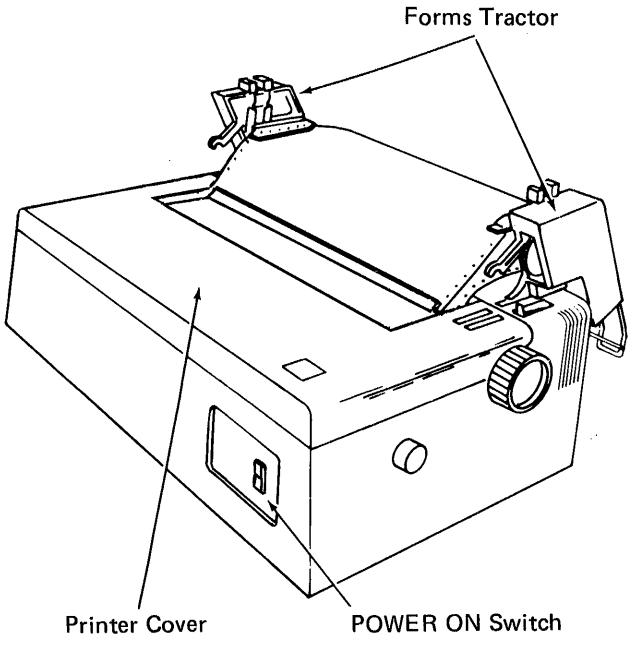


HOW TO ADJUST THE COPY CONTROL DIAL FOR FORMS THICKNESS

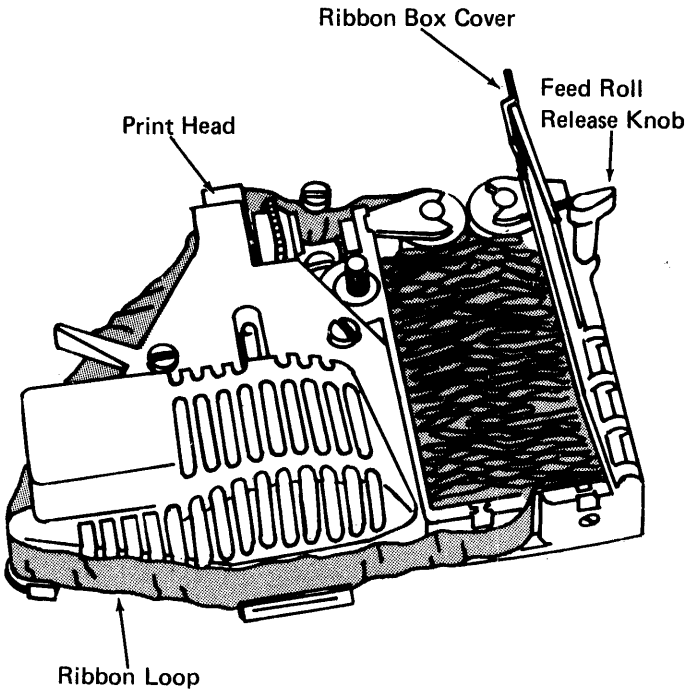


1. If you are using singlepart forms, set the copy control dial on 0.
2. If you are using multipart forms and the last sheet is not legible, rotate the copy control dial toward 0 one click at a time to obtain the legibility you desire.
3. If you are using multipart forms and the ribbon is smudging the first sheet, rotate the copy control dial toward 8 one click at a time until smudging stops.

HOW TO REPLACE A RIBBON (PART NUMBER 1136653)



1. Turn off power to the printer.
2. Tilt the forms tractor back by lifting both sides at the front.
3. Slide the top cover forward, then lift the front edge of the top cover and remove it.



4. Be sure that the print head is to the extreme left.

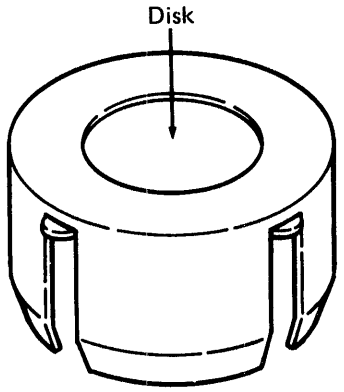
5. Turn the feed roll release knob counterclockwise until it points to the right.

6. Open the ribbon box cover.

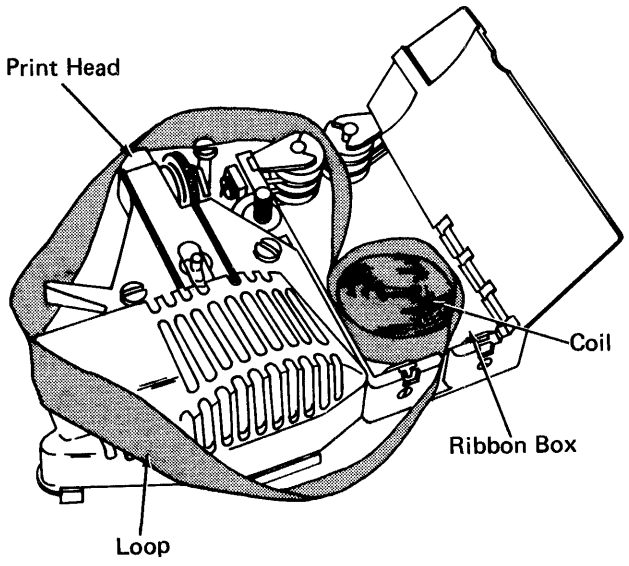
7. Put on the gloves supplied with the new ribbon.

8. Remove the old ribbon from the guides being careful to disengage it from the clip on the print head.

9. Lay the ribbon loop on the top of the ribbon in the ribbon box. Pick up the entire ribbon and discard it.



Ribbon Holder

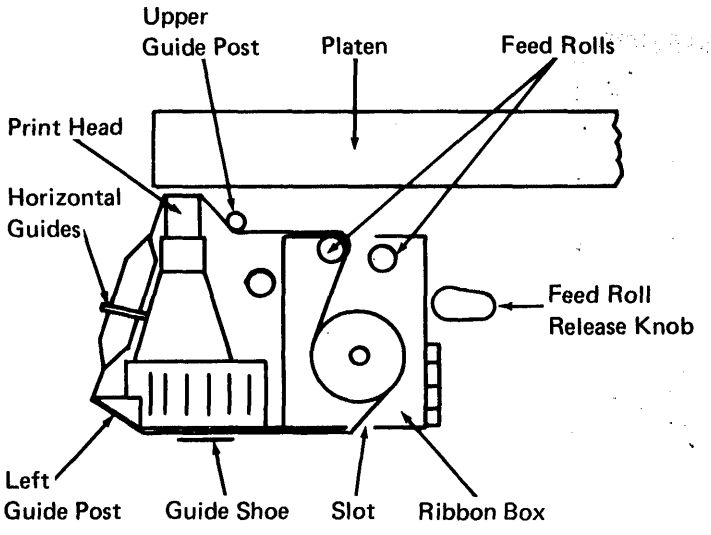


10. Eject the new ribbon from its holder into the ribbon box by pressing on the disk.

11. Remove the disk from the ribbon and discard the disk and the holder.

12. Hold the coil lightly with one hand and pull about 10 inches (254 mm) of ribbon from the coil.

13. Form a loop from the ribbon across the print head.



14. Thread the part of the loop nearest the platen between the feed rolls and on the inside of the upper guide post.

15. Turn the feed roll release knob clockwise to close the feed rolls.

16. Thread the ribbon between the print head and the platen. Be sure the ribbon is under the clip on the print head.

17. Thread the other part of the loop through the slot in the bottom of the ribbon box.

18. Thread the ribbon through the guide shoe and around the left guide post.

19. Insert the horizontal part of the ribbon twist (bottom edge first) between the two horizontal guides.

20. Move the print head back and forth across the platen to remove the slack from the ribbon. Continue moving the print head until you are sure that the ribbon feeds properly. Leave the print head at the extreme left.

21. Close the ribbon box cover.

22. Close the printer cover and turn the power on.

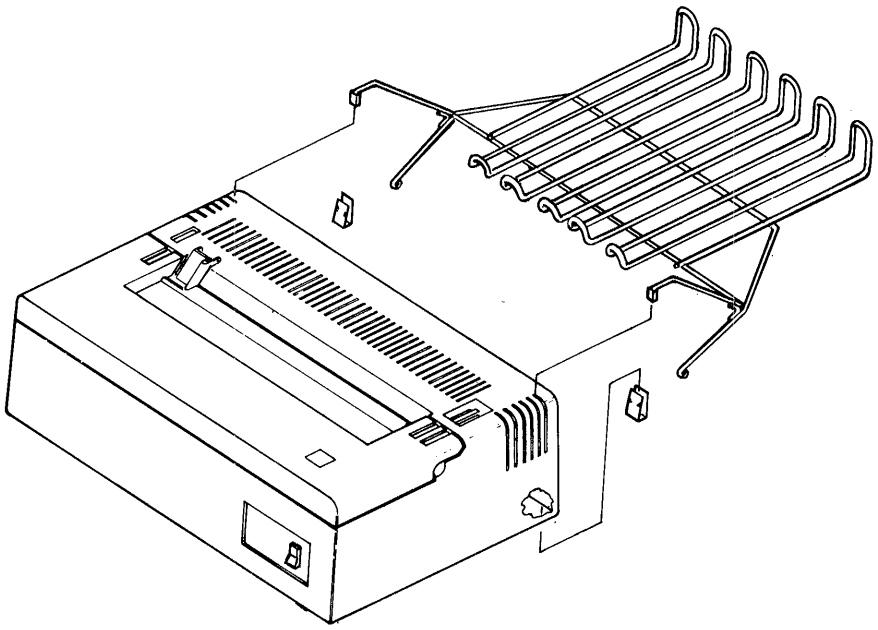
23. Reposition the forms tractor.

INSTALLING THE 5103 PRINTER STACKER

A folded-form paper stacker is supplied with 5103 printers. The wire stacker hooks onto the back of the printer cover as shown in the drawing. The lower wires on the stacker should contact the metal clips on the cover.

The stacker can be bent if too much weight is applied. Under normal conditions, printed forms should not be allowed to accumulate higher than 1 inch in the stacker.

Note that, because of the relatively small free-fall distance of the paper as it leaves the printer, you may have to manually fold the first two or three sheets to get the folding operation started.



Appendix C. BASIC Error Messages And Operator Recovery

The following list contains all the system error codes. Any detected error will deactivate keys on the keyboard and cause the display screen to flash. To stop the flashing display, press the ATTN key. Error codes below 100 are I/O errors and are displayed as a three-digit message, followed by a device identifying character (E for tape, D for diskette, or 5 for the printer).

The device identifying character is followed by 80 (built-in tape or diskette drive 1), 40 (auxiliary tape or diskette drive 2), 20 (diskette drive 3), or 10 (diskette drive 4).

You can recover from each execution error with one of the following procedures labeled (A), (B), (C) or (D):

(A) Enter GO or GO END to end the program or enter GO x to continue, where x is the statement number of any statement in your program. Use this recovery for errors that occur during program execution.

(B) Enter GO END to end the program or GO to continue.

(C) Enter GO or GO END to end the program.

(D) Correct the statement or command in the input line, then press the EXECUTE key. Use this recovery for errors that occur during command or source input.

Note that if errors are detected during execution of a program involving tape or diskette operations, you should enter GO END to ensure that files are properly closed.

If I/O errors (01-99) occur on a tape file, the file is automatically marked *not open* and cannot be accessed by GET or PUT statements. An output file has not been properly closed and may not be accessible. GO END closes all other files.

I/O ERRORS

Error codes 004 through 008, or 015 through 045 can be caused by problems within the tape cartridge or diskette respectively. Time may be saved, and a service call may be eliminated, if you will make a complete check of the media referenced by the error code for any of the following problems:

- Worn or damaged oxide coating
- Binding tape cartridge spools
- Loose tape cartridge drive band
- Binding diskette
- Worn or off-center diskette drive hole
- Damaged diskette index hole

I/O ERRORS

Error	Meaning	Recovery
001	Diskette drive timing error occurred when the rotational speed of the diskette was not within specification.	Retry the operation. If the error recurs, call your service personnel.
002	An attempt was made to access tape file 0, or mark more files than existed on a diskette.	Verify that the command is correct. Retry the operation. If the error recurs, call your service personnel.
003	Tape hardware error.	Retry the operation. If the error recurs, call your service personnel.
004	An attempt was made to access an unmarked tape cartridge.	Verify that the tape is marked. Retry the operation. If the error recurs, call your service personnel.
005	Tape cartridge was not inserted.	Insert a cartridge and retry the operation.
006	Tape was file-protected with the file protect switch in the SAFE position.	If you want to write on the tape, turn the SAFE switch on the tape cartridge to off, the SAFE position.
007	Tape read error occurred when an incorrect CRC was detected during a tape read operation.	Use the LINK command to load the Tape Recovery program and recover as much data as possible. Re-mark the tape.
008	The next expected physical record cannot be found. A system malfunction occurred, or the tape cartridge was removed from the tape unit when data or a workspace was being written. The data in the file cannot be used.	Retry the operation again. If the error recurs, copy the files following the file that caused the errors onto another tape. Then use the MARK command and re-mark the tape from the file that caused the error.

Error	Meaning	Recovery
009	An attempt was made to read a record after the end of data address.	
010	End of file has occurred during a read or a write beyond the last record in the file.	Use the MARK command to mark a larger file and retry the operation.
011	End of marked tape occurred when a nonexistent tape file was specified.	Specify the correct file or use the MARK command to mark the tape.
012	End of physical tape was encountered.	Use another cartridge.
013	Device not attached. Error occurred when the device addressed was not attached or the power switch on the 5114 was off.	Change the address to the correct device or turn the 5114 power switch on.
014	Device error.	Retry the operation. If the error recurs, call your service personnel.
015	System error occurred and the volume or header label could not be read or written.	Use the Diskette Recovery program to correct the problem. If the error recurs, copy all available data onto another diskette; then reinitialize the diskette.
016-019	Diskette read error indicating a faulty diskette. Error occurred during one of the following: • When reading an ID field on a data field • When an error was detected on all the ID fields of a track • When data miscompare occurred on a write data operation	Use the Diskette Recovery program to correct the problem. If the error recurs, copy all available data onto another diskette. Reinitialize the diskette.

Error	Meaning	Recovery
020	ID search failure occurred when the correct ID was not found on the diskette.	Copy all available data onto another diskette; then reinitialize the diskette. Retry the operation. If the error recurs, call your service personnel.
021	Hardware error occurred when no ID fields could be found on the track.	Same as error 020.
022	Hardware error occurred when no data was found following an ID field.	Use the Diskette Recovery program to correct the problem. If the error recurs, copy all available data onto another diskette; then reinitialize the diskette.
023	Data write timeout (hardware error)	Retry the operation. If the error recurs, call your service personnel.
024	No VOL1 label on the diskette occurred because the first four characters of the volume label were not VOL1 or the volume label was not a valid record.	Use the Diskette Recovery program to correct the problem. If the error recurs, copy all avaliable data onto another diskette. Reinitialize the diskette.
025	Invalid volume label occurred because of one of the following: • An invalid surface indicator • An invalid physical record length • An error in the extended label area indicator	Same as error 024.
026-028	Diskette hardware error.	Retry the operation. If the error recurs, call your service personnel.
030	Diskette drive cover was opened and a different diskette was inserted.	Insert the correct diskette, close the diskette drive cover, and retry the operation.
031	Diskette write error occurred due to a defective diskette.	Copy all available data onto another diskette; then reinitialize the diskette.

Error	Meaning	Recovery
032	Diskette read/write error occurred due to a diskette timing problem.	Use the Diskette Recovery program to correct the problem. If the error recurs, call your service personnel.
033-035	Diskette hardware error.	Retry the operation. If the error recurs, call your service personnel.
036	An I/O operation was specified with a file number that was zero or greater than the maximum number of files allowed on that type of diskette.	Correct the file number specified and retry the operation.
	On a MARK command, the number of files to be marked were more than were permitted on the diskette.	Correct the number of files in the MARK command and retry the operation.
037	Invalid control flag occurred when a control record other than sequential relocate or delete was found in the data area of the diskette.	Use the Diskette Recovery program to correct the problem. If the error recurs, copy all available data onto another diskette; then reintialize the diskette.
038	Invalid header error occurred because of one of the following: • The first 4 characters were not HDR1 • The header record length indicator did not match the volume length indicator • A diskette Type 2D had an exchange type indicator set to blank • The EOE (end of extent) was past cylinder 74 • The BOE (beginning of extent) was greater than the EOE • The EOD (end of data) was greater than EOE + 1	Use the Diskette Recovery program to correct the problem.

Error	Meaning	Recovery
039	An attempt was made to write to a write protected file.	To write to the file, turn off the write protect indicator and then write to the file.
040	An I/O operation was specified with only the file name, and more than one file on the diskette had the same file name.	Issue the command using the appropriate file number or change the name of the file.
041	An I/O operation was specified with only the file ID, and no matching file ID was found.	Insert the correct diskette and reissue the command using the correct file ID.
	An I/O operation was specified with only the file number and that file was not marked.	Issue the command or statement using a new file ID.
042	A MARK command was issued, but there was not enough unallocated continuous storage available on the diskette to format the file(s).	Use the diskette support function to combine all unused space on the diskette. Then retry the MARK command. If the error recurs, use another diskette.
043	An attempt was made to access a volume protected diskette.	Use the UTIL VOLID command to turn off the volume protection indicator.
044	A sequential relocate or delete control record was encountered when sequential relocate was prohibited.	To read the data, turn the sequential relocate indicator on.
045	The diskette was inserted incorrectly, the drive cover was not closed, or there was no diskette in the drive.	Make sure the diskette is inserted correctly and retry the operation.
050	End of forms occurred when the printer ran out of paper.	Insert new forms. If the error occurred on a command operation, reenter the command. If the error occurred during program execution, enter GO x to continue, where x is the statement number displayed. This may cause the last line to be printed twice.

Error	Meaning	Recovery
051	Printer not ready occurred because the power was turned off.	Turn on the printer. If the error occurred on a command operation, reenter the command. If the error occurred during program execution, enter GO x, where x is the statement number displayed.
052-053	Printer errors.	Check that the forms tractor and the platen are not engaged simultaneously. If the error recurs, call your service personnel.
054	The printer error occurred because the printer was turned off within 2 seconds after a system command was entered. The error continues to occur when the printer is turned on again.	Save any data in the workspace; then enter the LOAD0 command to synchronize the printer with the 5110.
055-059	Printer errors.	See error 052-053.
070	System timing error.	To recover, press the CMD and HOLD keys; then press the HOLD key to release the machine.
071	Printer errors occurred when the printer was running in overlap mode and an error occurred in the previous print line.	Check for end of forms, paper jams, and so on. Retry the operation.
072	Printer paper or ribbon was changed while the program was running.	Press the HOLD key prior to changing printer paper or ribbon.

EXECUTION ERRORS

Normal execution errors are simply displayed as three-digit messages, as shown in the following list.

Error	Meaning	Recovery
500	Syntax error in statement, in key group header, or in FORM specification from a character variable during program execution.	Ⓐ or Ⓓ
502	Unbalanced parentheses in expression.	Ⓓ

Error	Meaning	Recovery
503	Missing delimiter between character constants.	D
505	First statement following a LOAD0,KEYX was not a header statement.	D
506	Invalid line number for KEY specified.	D
507	Statement type not allowed in BASIC KEY group.	D
508	Size of dimensions is not in the range of 1 to 9999, or character variable is not in the range of 1 to 255.	D
509	Invalid error exit parameter for statement as specified. Error exit displayed after error number.	D
510	Data conversion specification missing from FORM statement.	A or D
511	Invalid character in PIC string for FORM statement.	A or D
512	FORM statement PIC string starts with a slash or a comma.	A or D
513	FORM statement PIC string ends with a blank (B), slash, or comma.	A or D
514	FORM statement PIC string contains more than one decimal point.	A or D
515	FORM statement PIC string contains an incorrect exponent specifier.	A or D
516	DB or CR in a FORM statement PIC string is not the last item.	A or D

Error	Meaning	Recovery
517	FORM statement PIC string contains numeric (#) digit specifier after the decimal point when a zero suppression charcter or floating sign also appears after the decimal point.	A or D
518	FORM statement PIC string contains a zero suppression character following a numeric digit specifier.	A or D
519	FORM statement PIC string contains intermixed zero suppression characters.	A or D
520	FORM statement PIC string contains a zero suppression character after floating sign specified.	A or D
521	FORM statement PIC string contains the same leading and trailing signs.	A or D
522	FORM statement PIC string contains a trailing sign that is not the last item in the string.	A or D
523	FORM statement PIC string contains invalid use of sign character.	A or D
524	FORM statement PIC string not in range of 1 to 32 characters between parentheses.	A or D
525	FORM statement PIC string contains no digit specifiers or floating sign specification.	A or D
526	FORM statement contains character other than B or trailing sign after the exponent specifier.	A or D
527	Insufficient storage to syntax-check the statement.	D

Error	Meaning	Recovery
528	Available storage exceeded.	(D)
550	Invalid command syntax or syntax error in a DEL (delete) statement.	(D)
551	Invalid use of a command: • No place to go to on GO. • No statements to renumber. • No program to RUN. • No statements to LIST. • Program is locked.	(D)
552	Unable to locate line number or next lower line number for a LIST command.	(D)
553	Function key not loaded or not defined for request.	(D)
554	Next line generated by AUTO command would exceed 9999 or attempt to load a data file with more than 9999 data lines.	Auto numbering is terminated, enter line numbers via the keyboard.
555	RENUM will cause a line number greater than 9999.	(D)
556	Statement number referenced in a BASIC statement not found during RENUM. The statement number of the referring statement is displayed after the error number.	(D)

Error	Meaning	Recovery
557	One of the following has occurred: • UTIL was specified to tape with an open file. • Device code 0, 1, 2, 3, or 4 was specified on a LOAD, SAVE, or OPEN [FILE] command. • A UTIL sort was attempted when the diskette sort feature was not installed. • Input has been requested for an output only device, or output has been requested for an input only device.	(D)
558	End of file occurred before completion of a SAVE command, SAVE is terminated.	Specify a larger file and reenter the SAVE command.
559	Input line on LOAD SOURCE command exceeds 64 bytes.	(D) or scroll up the line and ignore it.
560	One of the following has occurred: • The file type is incorrect for a LOAD. • The file type for a MERGE is not 11 when the work area is BASIC, or the file type is not 2 when the work area is DATA. • The file type is incorrect for a LINK.	(D)
561	Data file with line longer than 330 bytes cannot be loaded.	System limit, no recovery.
562	PROC command issued for a file that is not record I/O.	(A)

Error	Meaning	Recovery
563	Procedure file record length exceeds 64 bytes.	(A)
564	Invalid command because procedure file is not active.	(A)
565	Invalid value to skip on SKIP or CSKIP, so the procedure is terminated.	Correct the value in the procedure file.
566	MARK command issued for a file already marked.	Enter GO in positions 1 and 2 to continue or enter any other character to end the MARK operation.
567	MARK command error: • For tape, reached the end of the reel before MARK completed. • For diskette, reached end of the diskette before MARK completed. For tape, the last marked file and K-bytes allocated to the file are displayed. For diskette, the number of files allocated to the specified size are displayed.	Information only, no recovery.
568	File number already exits on another file.	(D)
569	On a SAVE command, the source line is greater than the RECL specified.	(D)
600	Attempt to OPEN an open file.	(D)
601	OPEN [FILE] specifies a file reference (FL0 through FL9) for a file that is already open.	(A)
602	OPEN FILE with a KEY specified when a master file is not currently open.	(A)

Error	Meaning	Recovery
603	Attempting to open more than one file at a time on tape or other single file device.	(A)
604	Insufficient storage for file OPEN request. See *Record I/O File Buffer Requirements* in Chapter 3.	(A) or (D)
605	Diskette label that is invalid on this system has been found.	(A) or (D)
606	Unsupported diskette label entries found.	(A) or (D)
607	Attempting to open a password protected tape file for output.	(A) or (D)
608	File type on the file is not valid for input or the RECL= not specified for a new output file.	(A) or (D)
609	One of the following has occurred: • OPEN FILE specified to device 000 or 001. • OPEN specified to device 002. • OPEN [FILE] specified to device '003' through '00F'.	(A)
610	Invalid file ID or a complex name was specified for a BASIC exchange file.	(A) or (D)
611	OPEN for output to write-protected file or a file with a nonblank expiration date.	(A) or (D)
613	Invalid device code specified (other than 0-9 or A-F).	(A) or (D)

Error	Meaning	Recovery
614	File name found but not at file number specified, or file found by number, but the header ID and user ID do not match.	(A) or (D)
615	Maximum logical records on a file is greater than 2**24-1.	(A) or (D)
616	NOBLOCK specified and logical record size exceeds physical record size.	(A) or (D)
617	RECL= specifies a zero value.	(A) or (D)
618	Record length on record I/O file is 0.	(A) or (D)
619	OPEN specified output to a file that is not empty and the file ID does not match the ID in the header specified.	(A) or (D)
620	New media mounted while a file is open on the device.	Insert the correct diskette, close the diskette drive cover, and retry the operation.
621	Insufficient storage to verify that correct media is inserted.	(D)
622	Device not valid for KEY file.	(D)
623	Invalid KEY parameter: KP exceeds 64K, KL exceeds 28, or KW exceeds 64K.	(D)
624	File specified for KEY is not direct.	(D)
625	Key length plus key position exceeds master record length.	(D)
626	Master file not same access method as index file.	(D)
627	Storage exceeded allocating key information block.	(D)

Error	Meaning	Recovery
628	Key length exceeds 28 positions.	(D)
629	Invalid index file specified.	(D)
630	System error.	If error continues to occur, call your service representative.
631	Invalid key specified (first byte is X'00' or X'FF').	(D)
632	Record in master file (indicated by index file) does not have matching key.	(D)
633	REWRITE to indexed file has modified the key field (record not written).	(D)
634	Conversion error detected.	(D)
635	DUPKEY error detected.	(D)
636	Key not found.	(D)
637	Record not found.	(D)
638	KEY specified on unindexed file, or REC specified on indexed file.	(D)
639	DELETE FILE specifies file that is not indexed, record, or ALL.	(D)
640	RESET specifies KEY to nonindexed file.	(D)
641	RESET with KEY specifies output-only file.	(D)
642	RESET FILE references stream I/O file, or RESET references record file.	(D)
643	ALL or REC= specified on a file created for sequential access.	(D) or enter GO line number to continue. REC= will be accepted on further references to the file; however, incorrect results can occur if the file contains a bad sector.
645	I/O access to unopened file, or DELETE FILE specifies unopened file.	(D)

Error	Meaning	Recovery
646	One of the following has occurred: • The file referenced is not open. • A GET or PUT referred to a record file. • A READ or a WRITE FILE referred to a stream file. • A GET or READ FILE referred to an output file. • A PUT or WRITE FILE referred to an input file. • PRINT referred to an input, record, or stream input file.	(D)
647	An I/O operation was attempted during evaluation of a function referenced in an input or output statement.	(D)
648	REWRITE does not follow a READ or REREAD to a specified file.	(D)
649	REREAD does not follow a READ or REREAD statement.	(D)
650	Statement specified in USING parameter is not a FORM or image statement.	(D)
651	PRINT USING specifies a data item and there is no conversion specification in the image statement.	(D)

Error	Meaning	Recovery
652	Statement specified in EXIT parameter is not an exit statement.	D
653	Too few DATA statements for the READ statements issued.	D
654	Insufficient storage available for workspace needed for variable image/FORM specification.	D
660	USE statement located after the first data reference.	D
661	DIM statement contains previously defined data.	D
662	Invalid redimension specified (zero or exceeds 9999).	D
663	New array dimensions exceed original allocation.	D
664	MAT statement specifies an array not previously defined.	D
665	INV argument is singular.	D
666	IDN function specified for a nonsquare matrix.	D
667	Operand dimensions are invalid for the function used and the specified result array.	D
668	A referenced statement number cannot be found.	D
669	No FNEND statement found for a DEF statement.	D
670	No matching NEXT statement found for a FOR, or the control variable did not match.	D
671	NEXT statement executed before the matching FOR statement.	D
672	A RETURN without a value has been encountered while no GOSUB is active.	D

Error	Meaning	Recovery
673	A RETURN with a value has been encountered while no user function is active.	Ⓓ
674	CHAIN to procedure file: command is too long.	Ⓓ
680	Data underflow has occurred (zero is assumed).	Enter GO or GO statement number to continue or GO END to terminate
681	Data overflow has occurred (plus or minus 7.237E75 is assumed).	Enter GO or GO statement number to continue or GO END to terminate
700	Too many subscripts used for an array reference.	Ⓒ
701	Subscript not in range of specified dimensions (too large or negative).	Ⓒ
702	Reference found to a user-defined function that is undefined for this program.	Ⓒ
703	System error.	Retry the operation. If the error recurs, call your service representative.
704	Incorrect number of parameters specified in a user-defined function reference.	Ⓒ
705	Invalid data type returned for a user-defined function. For example, FNA$ returned a numeric value or FNA returned a character value.	Ⓒ
706	Nonsquare matrix for DET function.	Ⓒ
707	Argument for LOG is negative.	Ⓒ
708	Invalid argument to EXP function (too large).	Ⓒ
709	Argument for square root is negative.	Ⓒ

Error	Meaning	Recovery
710	Argument for arcsine or arccosine is greater than 1.	C
711	Power function attempt to raise 0 to 0 power, or attempt to raise 0 to a negative power.	C
712	Power function attempt to raise a negative number to a noninteger power.	C
713	Invalid character string found in NUM intrinsic function or error in LEN function.	C
714	File reference in KLN, KPS, or RLN is not FLP, FLS, or FL0-FL9.	C
715	Parameters for STR are zero or exceed the first argument length.	C
716	Storage exceeded trying to allocate space for a variable.	C
717	Insufficient work area to calculate determinant.	C
718	Error during concatenation (too long).	C
719	Operator stack, operand stack, or temp stack has overflowed. Statement is too long, or there are too many nested function calls.	C

Appendix D. Attaching a TV Monitor

The 5110 is designed to drive a monitor or combination television/monitor to display the same information shown on the 5110 display screen. The number of monitors that can be driven by the 5110 depends on the monitor input requirements and the distance between the monitor and the 5110. There is 40 mA of current available to drive monitors (the last monitor in a parallel string must be terminated with 75 ohms). Monitors are connected through the BNC connector located on the back of the 5110. The recommended cable is RG59/U or an equivalent.

MODIFIED TV SETS

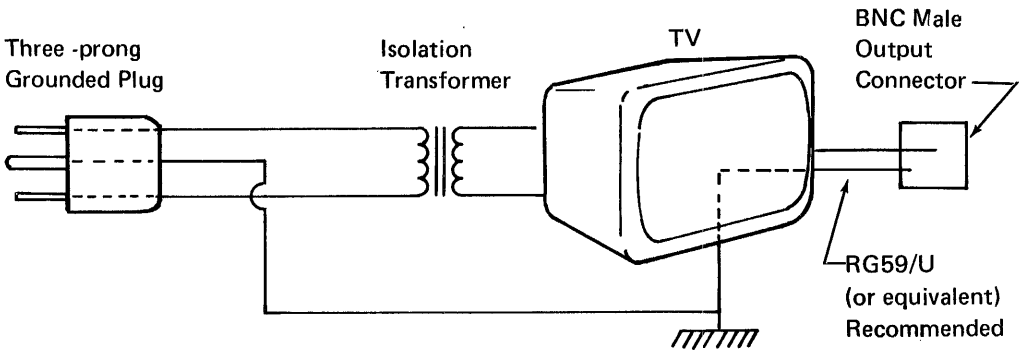
Modifying a standard TV set for use as a video monitor is not recommended because it yields less satisfactory results than a regular video monitor. The TV set contrast and resolution circuitry are not usually as good as they are on a monitor class unit. Therefore, the screen image is frequently not as sharp and usually more difficult to read.

However, if you choose to modify a TV receiver and use it as a video monitor you must observe the following or you may damage the 5110 and expose yourself to a severe electric shock when you attempt to hook up the TV set to the IBM 5110.

A modified TV set must have isolation between the primary line voltage and the set's chassis and circuitry. You can usually accomplish this by using an isolation transformer between your outlet line voltage and the input voltage to the TV set. This transformer should be permanently wired into the circuit. The new input power plug must be a three-prong grounded plug with the ground connected to the chassis of the TV set. This grounding circuit must be electrically connected to the 5110 grounding circuit.

Before the video input is connected to anything, it should be tested to verify that the connector's external shell is at ground potential and that no line voltage is present on either the external shell or the center conductor.

It is the responsibility of the TV modifier to ensure that the input circuit meets the requirements of the 5110 output and will not damage the 5110.



Note: If you do elect to modify a TV receiver for use as a video monitor, IBM accepts no responsibility for safety precautions during conversion and hookup, for damages incurred to the TV receiver or 5110, or for the quality of the TV receiver as a video monitor.

Index

Please use this form only to identify publication errors or request changes to publications. Technical questions about IBM systems, changes in IBM programming support, requests for additional publications, etc, should be directed to your IBM representative or to the IBM branch office nearest your location.

Error in publication (typographical, illustration, and so on). **No reply.**

Page Number *Error*

Inaccurate or misleading information in this publication. Please tell us about it by using this postage-paid form. We will correct or clarify the publication, or tell you why a change is not being made, provided you include your name and address.

Page Number *Comment*

IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Name _____

Address _____

_____

● No postage necessary if mailed in the U.S.A.

Cut Along Line

Fold and tape | Please do not staple | Fold and tape

NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N. Y.

POSTAGE WILL BE PAID BY . . .

IBM CORPORATION
General Systems Division
Development Laboratory
Publications, Dept. 245
Rochester, Minnesota 55901

Fold and tape | Please do not staple | Fold and tape

IBM

International Business Machines Corporation

General Systems Division
4111 Northside Parkway N.W.
P.O. Box 2150
Atlanta, Georgia 30301
(U.S.A. only)

General Business Group/International
44 South Broadway
White Plains, New York 10601
U.S.A.
(International)

IBM

International Business Machines Corporation

General Systems Division
4111 Northside Parkway N.W.
P.O. Box 2150
Atlanta, Georgia 30301
(U.S.A. only)

General Business Group/International
44 South Broadway
White Plains, New York 10601
U.S.A.
(International)

IBM 5110 BASIC Reference Manual Printed in U.S.A. SA21-9308-2

SA21-9308-2