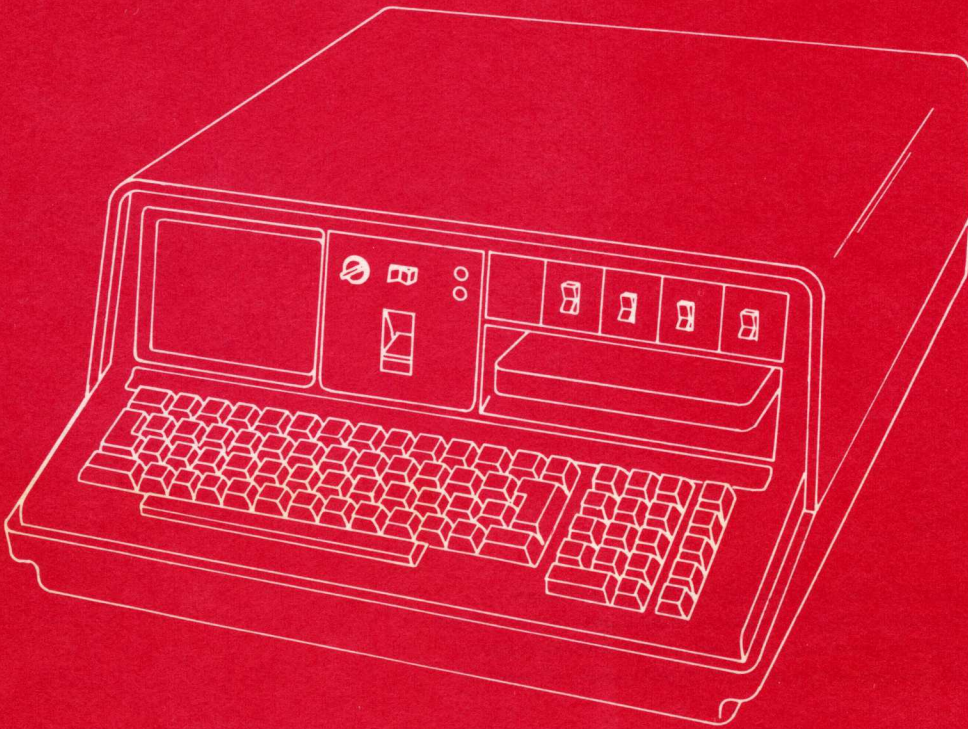




IBM 5100
BASIC Reference Manual

5100



IBM 5100
Portable Computer
BASIC Reference Manual

Preface

This manual contains specific information about the IBM 5100 Portable Computer and the BASIC language.

Prerequisite Publication

IBM 5100 BASIC Introduction, SA21-9216

Related Publications

IBM 5100 BASIC Reference Card, GX21-9218

Fourth Edition (July 1977)

This is a major revision of, and obsoletes, the previous edition SA21-9217-2 and Technical Newsletters SN21-0268 and SN21-0273. Changes to text and illustrations are indicated by a vertical line at the left of the change. Changes are continually made to the specifications herein; any such changes will be reported in subsequent revisions or technical newsletters.

Request for copies of IBM publications should be made to your IBM representative or the IBM branch office serving your locality.

A form for readers' comments is at the back of this publication. If the form is gone, address your comments to IBM Corporation, Publications, Department 245, Rochester, Minnesota 55901.

CHAPTER 1. OPERATION	1	CHAPTER 4. BASIC STATEMENTS	57
IBM 5100 Overview	1	Statement Lines	57
Display Screen	2	BASIC Statements	58
Keyboard	3	CHAIN	60
5100 Special Keys	4	CLOSE	61
5100 Switches	5	DATA	62
5100 Indicators	6	DEF, FNEND, RETURN	64
Editing Input Lines	6	Single Line Function	64
5100 Storage Capacity	7	Multiline Function	64
 		DIM	67
CHAPTER 2. SYSTEM COMMANDS	8	END	69
AUTO Command	10	FNEND	70
GO Command	11	FOR and NEXT	71
LIST Command	13	GET	73
LOAD Command	15	GOSUB and RETURN	75
Keyboard Generated Data Files	18	GOTO	77
Function Keys	18	IF	78
MARK Command	21	Image	80
MERGE Command	23	INPUT	81
PATCH Command	25	LET	83
RD= Command	29	NEXT	85
RENUM Command	30	OPEN	86
REWIND Command	32	Negative File Numbers	87
RUN Command	33	PAUSE	88
SAVE Command	35	PRINT	89
UTIL Command	37	Print Zones	90
File Types	38	Spacing of Printed or Displayed Values	90
 		Standard Output Formats for Printing or Displaying	91
CHAPTER 3. DATA CONSTANTS AND VARIABLES	40	Display Line Operation	92
BASIC Character Set	40	Print Line Buffer Operation	96
Alphabetic Characters	40	PRINT USING and Image	98
Numeric Characters	40	Conversion of Data Reference Values	99
Special Characters	41	Format Specifications	100
Use of Blanks	41	PUT	103
Arithmetic Data	41	READ	105
Arithmetic Data Formats	42	REM	107
Arithmetic Constants	44	RESET	108
Internal Constants	44	RESTORE	109
Arithmetic Variables	44	RETURN	110
Character Data	45	STOP	111
Character Constants	45	USE	112
Character Variables	45	Matrix Operations	113
Arrays	45	MAT Assignment Statements	113
Declaring Arrays	47	MAT Assignment (Scalar Value)	114
Redimensioning Arrays	48	MAT Assignment (Simple)	116
Arithmetic Arrays	48	MAT Assignment (Addition and Subtraction)	118
Character Arrays	49	MAT Assignment (Matrix Multiplication)	120
Summary of Naming Conventions	49	MAT Assignment (Scalar Multiplication)	122
System Functions	49	MAT Assignment (Identity Function)	124
Expressions	51	MAT Assignment (Inverse Function)	126
Arithmetic Expressions and Operators	51	MAT Assignment (Transpose Function)	128
Character Expressions	54	MAT GET	130
Relational Expressions	55	MAT INPUT	132
Array Expressions	56	MAT PRINT	134
Substring Function	56	Spacing of Displayed Values	134
		Display/Print Line Operation	135

MAT PRINT USING	137
MAT PUT	140
MAT READ	141
CHAPTER 5. MORE INFORMATION ABOUT	
YOUR 5100	143
5100 BASIC Compatibility With IBM 370/VS BASIC	143
Tape Cartridge Handling and Care	144
Data Security	144
Storage Considerations	145
APPENDIX A. BASIC CHARACTERS	
APPENDIX B. BASIC ERROR MESSAGES AND	
 OPERATOR RECOVERY	
APPENDIX C. SETUP PROCEDURES	
Environment	157
5100 Setup Procedure	157
IBM 5106 Auxiliary Tape Unit Setup Procedure	163
Cleaning the Tape Head	164
IBM 5103 Printer Setup Procedures	165
Installing the 5103 Printer Stacker	167
APPENDIX D. 5103 PRINTER	
How to Insert Forms	170
How to Adjust the Copy Control Dial For Forms	
Thickness	173
How to Replace a Ribbon	173
APPENDIX E. HEXADECIMAL REPRESENTATIONS .	
INDEX	
177	

IBM 5100 OVERVIEW

The IBM 5100 (Figure 1) is a portable computer designed for direct use by the user for solving problems. The 5100 has a display screen, a combined alphameric and numeric keyboard, a tape unit, switches, and indicator lights. 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 5100.

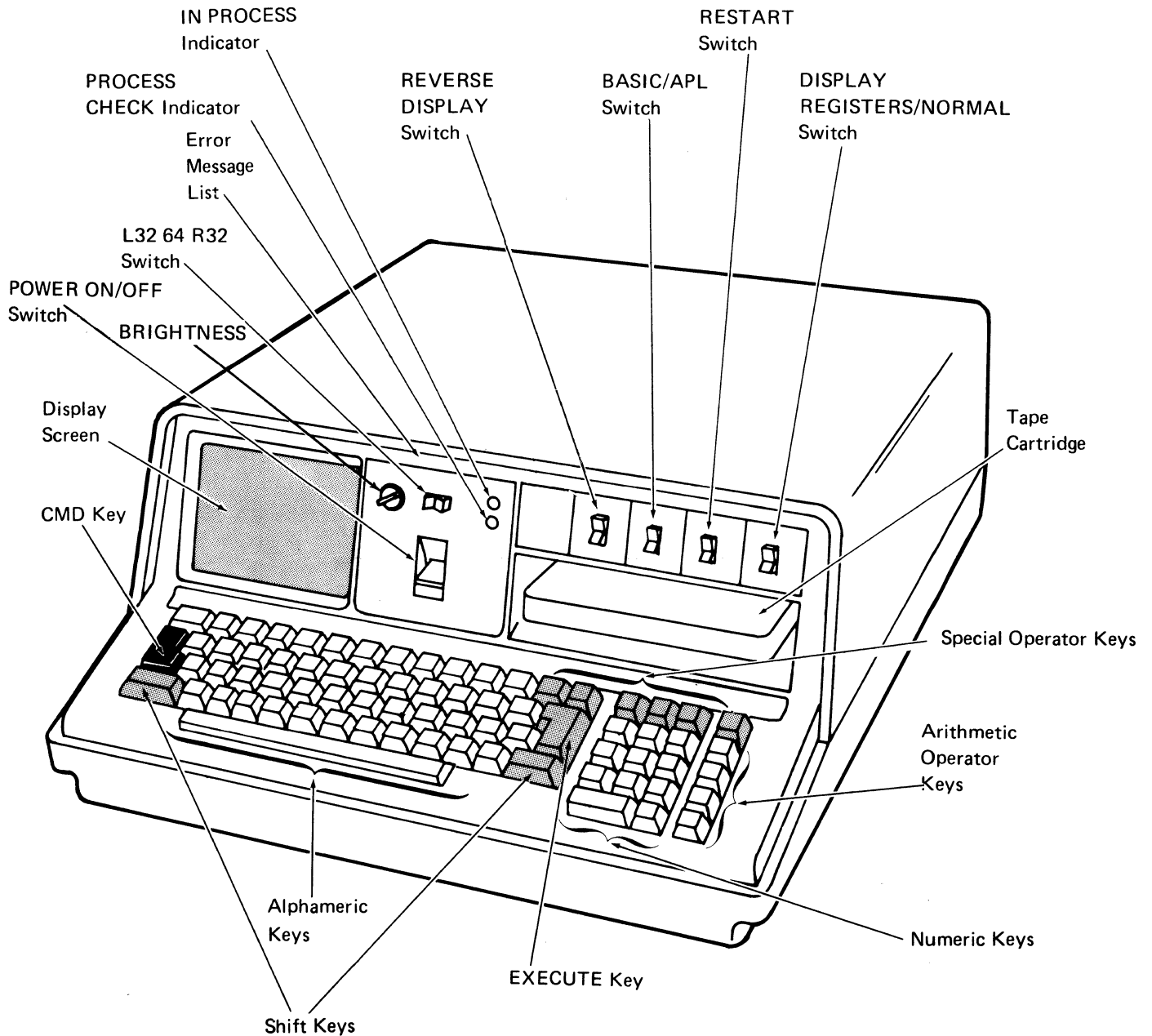


Figure 1. The IBM 5100 Portable Computer

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 corner (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 display (except the bottom line) are moved up so information can be entered on line 1 again.

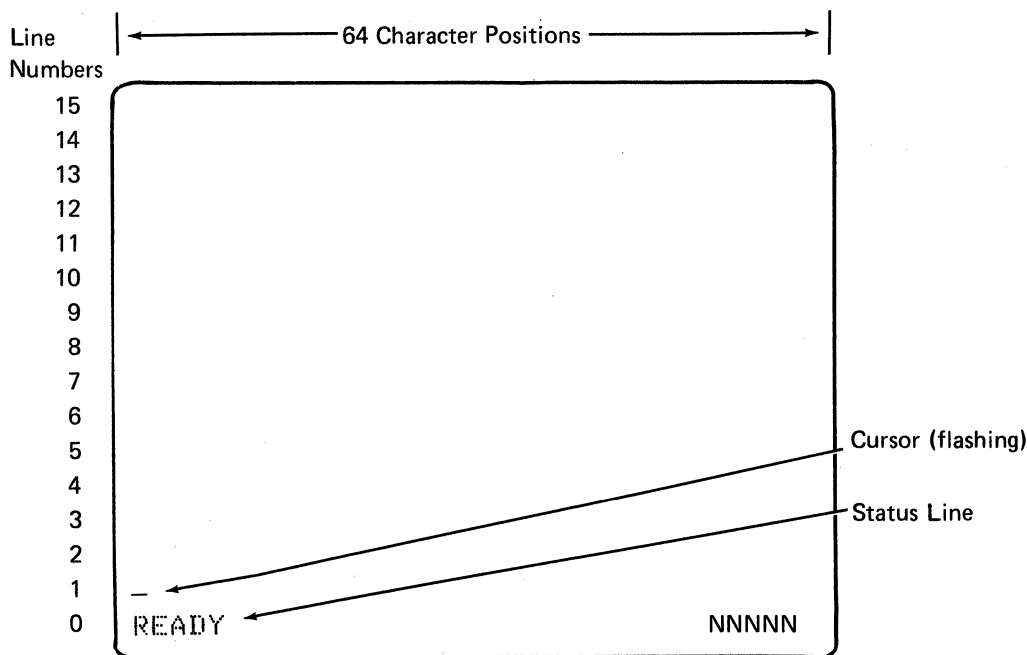


Figure 2. The 5100 Display Screen

KEYBOARD

The 5100 console (Figure 3) has a combined alphameric and numeric keyboard. Numeric data can be entered conveniently by using the calculator arrangement of numeric keys to the right of the alphameric keys. This data can also be entered 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.

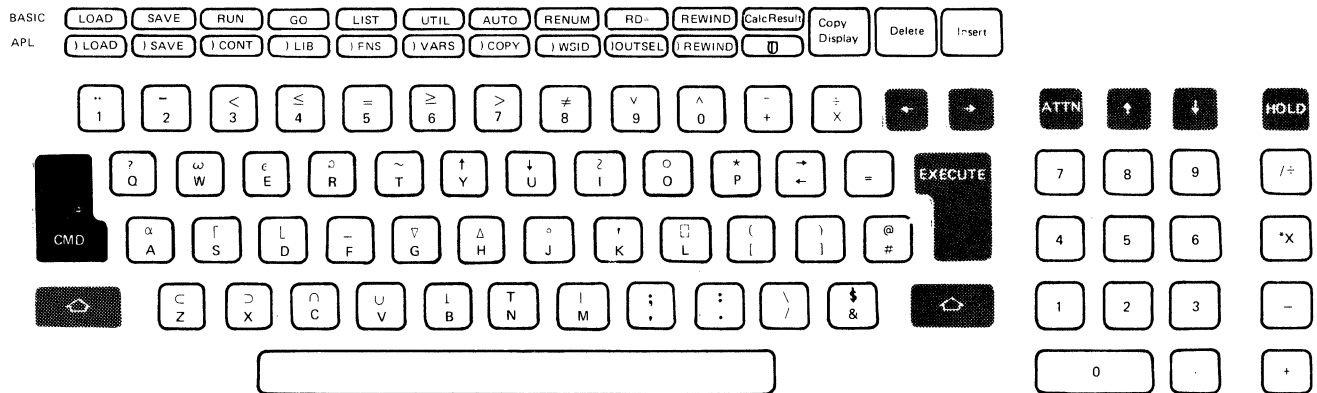


Figure 3. Combined BASIC/APL Keyboard

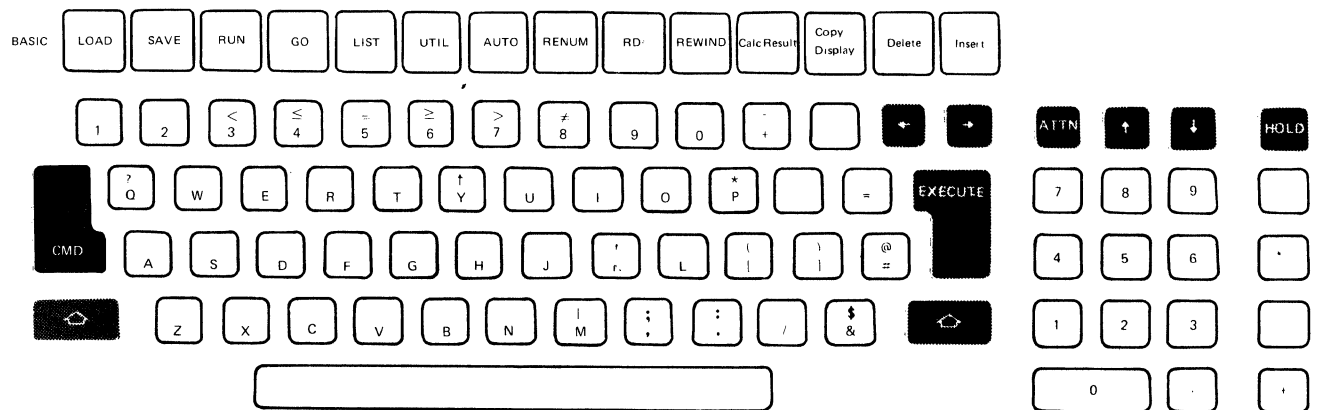


Figure 4. BASIC-Only Keyboard

5100 Special Keys

The following keys have special functions relating to 5100 operations:



(Attention) – You can press the ATTN key to stop system operations. While you are entering data at the keyboard, you can press ATTN to blank everything from, and including, the cursor position to the right on the input line of the display screen. You can then continue entering information. In addition, ATTN and HOLD are the only keys that are active when the display screen flashes to indicate detection of an error. After you press ATTN to stop the flashing display screen, all keys are active and you can proceed.



(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 EXECUTE. 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 HOLD; this activates a function restricted to use by service personnel. Holding down the CMD key and pressing a key with a BASIC statement keyword on the front will enter the keyword beginning at the current position of the cursor.



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




When you press HOLD, all processing stops unless the display screen is blank. Processing resumes when you press HOLD 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.



(Shift) – While you hold down a shift key, you can select an upper-case symbol for input.



(Copy Display—when CMD is held down) – When you hold down CMD 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 5100 is in a hold state, or anytime the 5100 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 line, the cursor moves to the rightmost position of the line. 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 line, the cursor returns to the first position of the line. 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 from 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 line contains a character in the last (64th) 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).



(Scroll Down) – When you press this key, each display line moves down one line position (except the status line).

5100 SWITCHES

The following switches are located above the 5100 keyboard on the console:

POWER ON/OFF – The power switch turns the 5100 on and off. When power is turned on, the 5100 becomes operable in approximately 10-15 seconds.

RESTART – This is a spring-returned switch that reinitializes the 5100 to its power on state. In the dual language 5100, the setting of the BASIC/APL switch determines which language is initialized.

BASIC/APL – This switch appears only on the dual language 5100, 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 pressing REVERSE DISPLAY.

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 must remain in the **NORMAL** position during 5100 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 each character.

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 each character.

5100 INDICATORS

The 5100 console has two indicators:

PROCESS CHECK – When this indicator lights, a system malfunction has been detected, and further operations are not normally possible. Press **RESTART**. 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 **EXECUTE** to enter the line into the 5100, 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, 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 EXECUTE 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 EXECUTE. 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 EXECUTE. To delete several successive lines, enter the number of the first line, enter DEL, enter the number of the last line, then press EXECUTE. For example, to delete lines 20 through 90 in a program, enter:

```
20 DEL 90
```

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

5100 STORAGE CAPACITY

The base 5100 (Model B1) 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 12,000 bytes, while 4,000 bytes are required for internal purposes. The storage capacity is increased in the following models of the 5100:

Model B2 — 32K

Model B3 — 48K

Model B4 — 64K

In these models, all additional storage is allocated to the user work area. For example, on the Model B4, the user work area is approximately 60,000 bytes.

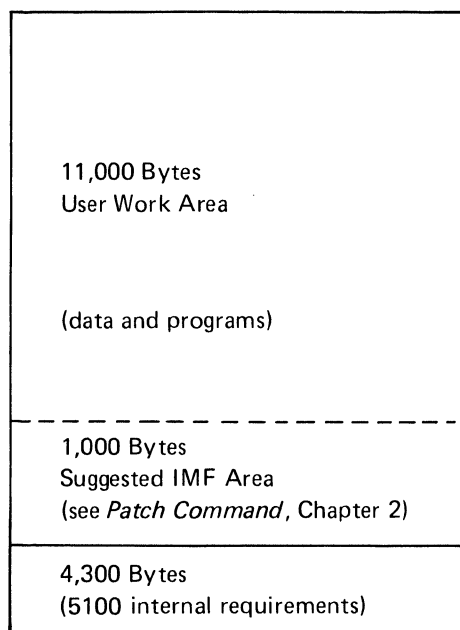


Figure 5. Storage Allocation for a 16K 5100

Chapter 2. System Commands

Some of the 5100 system commands are listed above the top row of numeric keys on the typewriter-like alphanumeric keyboard (Figure 4). These commands allow you to control tape and printer operations, such as storing a program on tape, loading a program into the 5100 from tape, 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 MARK, MERGE, and PATCH, which are not listed). The latter operation inserts the keyword with a single keystroke, thus preventing possible keying errors and providing faster operation. The system commands are used to direct the 5100 to perform the following types of system operations:

- Program Execution — Start or resume execution of a BASIC program or command.
- File Operation — Load or save programs or data on tape, or mark tapes.
- 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 keywords and major functions of the BASIC system commands are:

LOAD	Load work area
RUN	Run BASIC program
GO	Resume interrupted processing
SAVE	Save work area
MERGE	Merge work area
RENUM	Renumber statements
AUTO	Automatic line numbering
LIST	Display or print work area contents
UTIL	Perform utility functions (print/display a tape directory or invoke communications operations)
MARK	Mark tape file
PATCH	Apply internal machine fix (IMF), tape recovery, or tape copy
REWIND	Rewind tape
RD=	Specify rounding position of printed numeric values

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 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.

AUTO [line-num [,increment]]

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 EXECUTE, the next statement number is generated and displayed. The syntax of the AUTO command is as shown:

AUTO [line-num [,increment]]

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, a command word, for example, or a statement number other than the displayed number is entered in the input line. In this case, another AUTO command must be entered in order to continue automatic numbering. Another way to terminate AUTO numbering is to press the scroll up key, which will display a new unnumbered line.

Example

The following examples show AUTO commands:

AUTO (then press EXECUTE)

In this example, the display screen will show statement number 0010. After you press EXECUTE 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 EXECUTE)

$\text{GO [line-num] } \left[\left. \begin{array}{l} \text{RUN} \\ \text{STEP} \\ \text{TRACE [,PRINT]} \end{array} \right\} \right] \text{ [,RD=n]}$
--

GO COMMAND

You can use the GO command to do the following:

1. Resume processing of a MARK command operation that was halted.
2. Resume processing of a BASIC program that was halted by:
 - a. a PAUSE statement,
 - b. executing statements in step mode (see *RUN Command*),
 - c. some error conditions, or
 - d. your pressing the ATTN key.
3. End a MARK command operation or BASIC program before it reaches its normal termination point.

You can resume processing by pressing EXECUTE on a blank line. 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 also 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:

$$\text{GO [line-num] } \left[\left. \begin{array}{l} \text{RUN} \\ \text{STEP} \\ \text{TRACE [,PRINT]} \end{array} \right\} \right] \text{ [,RD=n]}$$

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 or where the command was interrupted.

RUN specifies that processing is to continue in normal mode.

STEP specifies that processing is to continue in step mode.

TRACE specifies that processing is to continue in trace mode.

Note: If neither *RUN*, *STEP*, or *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 indicates the number of digits to the right of the decimal point that will initiate rounding. *n* can be 1 to 13 and is initialized to 6. If not specified, *n* retains its last value.

Notes About GO

- The statement number entry is valid only during execution of a BASIC program.
- When the input line is blank, pressing EXECUTE causes the same operation as a GO command, except when GO is entered in response to an error indicating an attempt to mark a file already marked. In this case, 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 EXECUTE)

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

GO TRACE (then press EXECUTE)

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

GO 620, STEP (then press EXECUTE)

LIST [PRINT] [{KEYx }
{line-num}]

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 contents must be BASIC, DATA, or a KEY group (which requires a different LIST syntax). The syntax of the LIST command is as shown:

LIST [PRINT] [{KEYx }
{line-num}]

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 for printing. This entry is not valid for a KEYx group. For displaying, up to the first 14 lines are displayed.

When the listing is specified to the display, the line specified in the LIST command appears on line 2 of the display screen. The preceding 13 lines (less, if less than 14 are defined) are displayed on the lines above line 2. You can use the scroll up and scroll down keys (↓ and ↑) 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 5100 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 listing data lines longer than 118 characters (bytes), the excess over 118 characters is not displayed.

Example

The following examples show several LIST commands:

- To display the first group of work area lines:

LIST (then press EXECUTE)

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

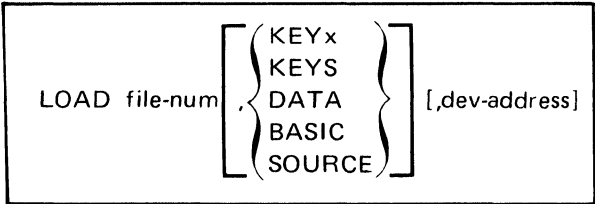
LIST 250 (then press EXECUTE)

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

LIST PRINT, 250 (then press EXECUTE)

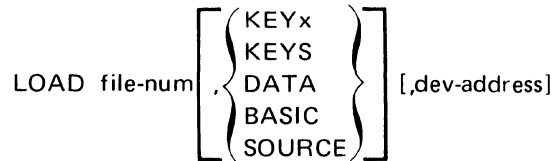
- To display the first 14 lines in the group associated with function key 5:

LIST KEY5 (then press EXECUTE)



LOAD COMMAND

The LOAD command causes a previously saved file to be loaded into the work area from tape 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:



where:

file-num is an integer value. An entry of 0 indicates that a new file will be entered from the keyboard. An entry greater than 0 indicates the file number of the file (saved on tape) to be loaded.

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

dev-address is the address code of the device containing a saved file to be loaded. The default is the built-in tape unit (address E80). Auxiliary tape unit device address is E40.

Entering a LOAD command of LOAD0 clears the entire work area, including all key groups. A LOAD 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 files already exist in storage, the loaded KEYx files will replace the old ones. The work area is assigned the file type specified in the LOAD command. Line 0 will display READY when in a BASIC program file type, DATA for a data file type, and KEYx for key file type. 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 of file definition 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 loading from the keyboard (KEYx is valid). Note that the LOADx, DATA command uses a 324-character buffer.

A LOAD nonzero entry specifies that a saved file is to be loaded from tape 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. 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 or SOURCE)
- Last line number (for BASIC or SOURCE)
- KEYx numbers (for KEYx file types)
- Amount of unassigned work area remaining (in bytes)

If the new file type is KEYx, the new key file is added to the work area. If corresponding KEYx files already exist in storage, they are replaced by the new keys. For example, if a new KEY6 function is loaded from tape, it will replace the existing KEY6 function.

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 loading from the keyboard (LOAD0).
- When loading a general exchange (type 02) file, SOURCE is assumed unless DATA is specified. This file type is used primarily for the Communications feature. Also see *Negative File Numbers* in Chapter 4.
- If file type is omitted, BASIC is assumed if loading from the keyboard, and BASIC or SOURCE is assumed if 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.
- The format for entering data after a LOAD0, DATA is identical to data in a DATA statement. LOAD0, DATA also provides automatic line numbering and the keyword DATA for each line. The display shows:

```
0010 DATA _
```

- If execution of a LOAD command is interrupted (when you press ATTN) while loading a saved file, operation is terminated and the contents of the work area are cleared.
- Calculator statements are invalid if DATA is specified (LOADx, DATA).
- LOAD0, DATA automatically provides a 4-digit statement number and the keyword DATA on the input line in the data file.

Notes About LOAD SOURCE

- When a syntax error occurs during a LOAD SOURCE command you must choose one of the following options:
 1. Correct the line and press EXECUTE to continue.
 2. 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 not completed until the READY message appears or the operation is terminated by other than a syntax error.

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

or

LOAD6

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 5100 responds with automatic line numbering (starting with 0010) followed by **DATA**. You can then enter numeric and character constants, separated by commas, just as you would for a standard **DATA** statement (see *DATA*, Chapter 4). The end of a data file line is indicated when you press **EXECUTE**. If you make a keying error, the display screen will flash and the keyboard will lock. Simply press **ATTN**, then correct the error, which is indicated by an up arrow (**↑**). A typical data file line is:

```
0110 DATA 312.41, 'JONES', 419.21, 'BALANCE'
```

After all data file lines are entered into the work area, they can be saved on tape with the **SAVE** command. Data file lines are saved without line numbers or the keyword **DATA**. When data file lines are listed, the keyword **DATA** is displayed.

Data in a keyboard-generated data file is accessed by **GET** or **MAT GET** statements during program execution. With line numbers and keyword **DATA** removed, the data file is a continuous string of data items.

A saved data file can be edited by first loading it back into the work area with a **LOAD** command. When saved on tape, the line numbers and **DATA** are again removed.

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 EXECUTE)
```

Now, you must define a key group, which is the statements associated with the function. When you press the **CMD** key and the 6 key, these are the statements that will be executed. One of four 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 **EXECUTE**).

- 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 specified key group is referenced in GOSUB). The DATA, END, STOP, DEF, and FNEND statements are not permitted. Thus READ, MAT READ should not be entered. Also, a key group should not contain a reference to a user-defined function. The last BASIC statement must be RETURN or CHAIN. The RETURN statement must not contain an expression. When referenced in GOSUB (statement 999x), the program branches to and seeks the REM key group. If the REM key group is not found, the program seeks statement 999x in the program.
- 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.

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 built-in tape unit to be rewound each time the CMD key and the 6 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 CMD is pressed.

- 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.

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 CMD is pressed.

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 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 K-characters,files,starting file [,dev-address]

MARK COMMAND

You can use the MARK command to initialize one or more tape 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 used in the file are displayed. The ATTN key is not active during mark operations. If you try to mark a file that is already marked, an error message is displayed (Appendix B). To continue, enter GO. Note that you must use the scroll up key to 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 files following the re-marked file cannot be used again.

The syntax of the MARK command is as shown:

```
MARK K-characters,files,starting file [,dev-address]
```

where:

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

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 drive in which the file resides. The default is address E80.

Notes About MARK

- *If an existing tape file is re-marked, the original information on the re-marked file and all files following it cannot be used again.*
- The MARK command must be entered character-by-character from the keyboard.
- The MARK command can be issued any time the specified tape unit 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. Note that each numeric expression requires an average of 16 storage positions (bytes), while each character expression requires 21 storage positions less one position for each trailing blank.

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 bytes on the tape residing in the built-in tape unit (E80).

```
MERGE file-num [, [from line-num] , [through line-num] , [new line-num] [, dev-address]]
```

MERGE COMMAND

The MERGE command allows you to merge all or part of a saved type 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 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 tape file are added to the work area lines in line number sequence. If a line from the tape 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 B).

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 file increment. After the merge is completed, the display shows the READY message, along with the number of unused bytes in the work area. The syntax of the MERGE command is as shown:

```
MERGE file-num [, [from line-num] , [through line-num] , [new line-num] [, dev-address]]
```

where:

file-num is the number of the saved file 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 tape unit in which the saved file resides. The default is the built-in tape unit (E80).

Notes About MERGE

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

MERGE 6,2,200, ,E80

← Omitted new line-num

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.

PATCH COMMAND

The PATCH command allows you to apply internal machine fixes (IMFs) to the 5100. The internal machine fixes are supplied by IBM on the customer support cartridge. The PATCH command provides you with the ability to:

- Copy IMFs to a new cartridge.
- Load IMFs to storage and then return to APL or BASIC.
- Display the EC (engineering change) version of each interpreter module.
- Perform tape recovery when tape read (007) errors occur on files with the following file types:
 - Data exchange (file type 01)
 - General exchange (file type 02)
 - BASIC source (file type 03)
 - APL internal data (file type 08)
- Copy the contents of one tape cartridge to another tape cartridge.

The copy IMFs, load IMFs, display EC version, and tape recovery programs are provided on a cartridge supplied with the 5100. This cartridge contains the following files:

- File 1 contains:
 - Copy IMFs program
 - Load IMFs program
 - Display EC version program
- File 2 contains IMFs for 5100
- File 3 contains tape recovery program
- File 4 contains tape copy program

To request one of these programs:

- Insert the cartridge in the built-in tape unit.
- Enter PATCH.
- Press EXECUTE.

There are no parameters for the PATCH command. When the PATCH command is entered and executed, the following options are displayed:

```
ENTER OPTION NO.  
1. COPY IMF TAPE  
2. LOAD IMFs  
3. DISP EC VER.  
4. KEY-ENTER IMF  
5. END OF JOB  
6. TAPE RECOVERY  
7. TAPE COPY PGM  
_
```

(cursor flashing)

The display shows the options available with each option identified by an option number.

To initiate a selected option:

- Enter the option number.
- Press EXECUTE.

Entering an option number other than those displayed causes the options to be redisplayed.

Once you have entered an option number, additional prompts, if any, are displayed for the option selected.

Option 1 – Copy IMF Tape

The copy IMF tape option allows you to copy the following files from the cartridge:

- File 1, which contains:
 - Copy IMF_s program
 - Load IMF_s program
 - Display EC version program
- File 2, which contains IMF_s for the 5100. IMF_s are copied from file 2 as follows:
 - Copy all IMF_s that apply to BASIC or to APL.
 - Copy all IMF_s that apply to the 5100 on which the copy IMF tape program is running.
 - Copy IMF_s by problem number.
 - Copy IMF_s by problem number that apply to the 5100 on which the copy IMF tape program is running.

- File 3, which contains the tape recovery program, and File 4, which contains the tape copy program, are not copied. These files can only be copied by service personnel.

Note: The cartridge onto which files 1 and 2 are to be copied must be marked. To determine what size files to mark, use the UTIL command.

Copying IMFs allows the creation of a cartridge containing only the IMFs that apply to your 5100.

The copy IMF tape program issues prompts and waits for you to respond. Respond as required to the prompts issued by the program.

Option 2 – Load IMFs

The load IMFs option allows you to load IMFs into storage and then return to the language in which your 5100 is running.

IMFs can be loaded as follows:

- Load all IMFs that apply to the 5100 on which the load IMFs program is running.
- Load IMFs by problem number. The IMFs are loaded into storage only if they apply to the 5100 on which the load IMFs program is running.

The IMFs will occupy storage in the work area (work space), so IMFs should not be applied if the problem does not affect your operation or if the problem can be circumvented by a BASIC statement or command. IMFs remain in the work area until power is turned off or RESTART is pressed. Also, if IMFs are loaded into storage, the performance of your 5100 may be reduced significantly.

The load IMFs program issues prompts and waits for you to respond. Respond as required to the prompts issued by the program.

Option 3 – Display EC Version

The display EC version option is primarily for use by service personnel. This option displays each interpreter module EC version and module identification. Four digits are displayed for each module. The first two digits are the module identification, the next two digits are the EC version.

The program issues prompts and waits for you to respond. Respond as required to the prompts issued by the program.

Option 4 – Key-Enter IMF

The key-enter IMF option allows service personnel to enter an IMF from the keyboard. After being entered from the keyboard, the IMF is copied on the tape. It may then be loaded into storage with option 2 (load IMFs).

Option 5 – End of Job

Selection of this option causes a return to the language in which your 5100 is operating. After the completion of options 1, 3, and 4, the program returns to display all options. Selection of option 5 can then be made.

Option 6 – Tape Recovery

The tape recovery option allows you to recover data from a file or files on which read errors are occurring. The tape recovery program can be used on the following file types:

- Data exchange (file type 01)
- General exchange (file type 02)
- BASIC source (file type 03)
- APL internal data (file type 08)

Tape recovery will recover as much data as possible in the file. Some of the data in the record where the read errors occur cannot be recovered. Some of the data preceding and following the record on which read errors occur may not be recovered.

Tape recovery assumes that the cartridge containing files, which cause read errors, is inserted in the internal tape unit.

The tape recovery program issues prompts and waits for you to respond. Respond as required to the prompts issued by the program.

Option 7 – Tape Copy Program

The tape copy option allows you to copy the contents (up to the end of marked tape) of one cartridge to another cartridge. Tape copy can utilize the auxiliary tape drive, if available. Tape copy also marks the tape being copied to.

Tape copy also allows you to recover most files that have bad header records that cause an ERROR 007 ddd. During the tape copy, if a file is encountered that has a bad header record, tape copy issues prompts to assist you in correcting the bad header record.

Note: If the bad header record occurs after the last marked file, enter 255 (hex FF) for the file type when the appropriate prompt is issued. File type FF indicates the next file to be marked.

Tape copy issues prompts and waits for you to respond to each prompt.

RD=n

RD= COMMAND

You can use the RD= command to specify the number of digits to the right of a decimal point that will be printed or displayed. 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. The range of rounding (n) is 1 to 13 digits.

RD=

RENUM [KEYx] [,first line-num,increment]

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 in GOTO, IF, PRINT USING, GOSUB, GET, and MAT GET statements 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:

```
RENUM [KEYx] [,first line-num,increment]
```

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. If a beginning statement number is not specified, an increment cannot be specified.

Notes About RENUM

- You must enter both a beginning statement number and increment, or you can enter neither.
- 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 example shows the execution of a RENUM command:

RENUM 20,10 (then press EXECUTE)

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 END	0080 END

REWIND [dev-address]

REWIND COMMAND

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

```
REWIND [dev-address]
```

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 E80.

Example

A sample REWIND command is as shown:

```
REWIND E40
```



```
RUN {STEP  
TRACE [,PRINT]} [,P=D] [,RD=n]
```

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 from tape 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 5100 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:

```
RUN {STEP  
TRACE [,PRINT]} [,P=D] [,RD=n]
```

where:

STEP specifies step-by-step execution.

TRACE specifies continued 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).

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 13 and is initially set to 6 at power on.

Notes About RUN

- The RUN command will be rejected by the 5100 if the work area file type is DATA.
- The RUN command initializes all variables and array elements to zeros and 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.

Example

Some sample RUN commands are as shown:

1. To begin normal execution of a program:

RUN (then press EXECUTE)

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

RUN TRACE, PRINT (then press EXECUTE)

SAVE file-num $\left[\begin{array}{l} \{ \text{KEYx} \\ \text{,KEYS} \\ \} \text{SOURCE} \end{array} \right] [,\text{dev-address}] [,\text{'file ID'}]$
--

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 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 B) and the file is reinitialized to unused status. To save the 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:

$$\text{SAVE file-num } \left[\begin{array}{l} \{ \text{KEYx} \\ \text{,KEYS} \\ \} \text{SOURCE} \end{array} \right] [,\text{dev-address}] [,\text{'file ID'}]$$

where:

file-num is an integer specifying the file in which the contents of the work area are to be saved.

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

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

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

dev-address is the address of the tape drive in which the specified file resides. The default is address E80.

'file ID' is up to 17 characters enclosed in single quotation marks that you can use to identify the file containing the saved data. This can be referenced in OPEN statements (see *OPEN*, Chapter 4.)

Notes About SAVE

- If KEY_x, 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 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. If the interruption occurs after the file has been changed, the file is reinitialized to unused status.
- IMFs are not stored with the SAVE command. You must apply IMFs with the PATCH command before you load the work area with any saved program requiring IMFs.

Example

A sample SAVE command is as shown:

SAVE 4, KEY8

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

UTIL { MODE COM [PRINT,] [DIR [integer]] [,dev-address] }

UTIL COMMAND

You can use the UTIL command to list a directory of a tape cartridge or to change the 5100 operating mode from BASIC to communications. You need some space in the work area in order to list a tape cartridge directory. If adequate work space is not available during execution of a UTIL command, an error message (Appendix B) is displayed. You must then clear the work area (with a LOAD0 command, for example) and reissue a UTIL command to continue. Information about each file on the tape cartridge is printed or displayed on one line per file. If you specify a list to the printer, all file information is printed without interruption. You can interrupt the listing to either the display or printer by pressing the ATTN key, which will terminate the command, or by pressing HOLD and pressing HOLD 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-bytes 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 512-byte 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 or cartridge to avoid loss of data due to defective areas on the tape.
- The first and last statement numbers for BASIC and SOURCE files.
- Key numbers saved in the file for KEYS or KEY (0-9) files.

File Types

Valid file types listed during the UTIL command are:

Type	File
0	Marked or unused file
1	Data exchange file
2	General exchange file
3	BASIC source file
4	BASIC work area file
5	BASIC KEYS file
6	APL continued file
7	APL SAVE file
8	APL internal data
16	Patch/tape recovery/tape copy
17	Diagnostic
18	IMFs
72	Tape storage dump

The syntax of the UTIL command is as shown:

$$\text{UTIL} \left\{ \begin{array}{l} \text{MODE COM} \\ \text{[PRINT,] [DIR [integer]] [,dev-address]} \end{array} \right\}$$

where:

MODE COM specifies that communications be invoked. This must be the only entry in a UTIL command invoking communications.

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.

dev-address specifies the address of the tape unit from which the listing is to be made. The default is address E80.

Example

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

```
UTIL PRINT, DIR1
```

The resulting printed output is shown in the following example:

User Identification	Allocated Storage (in K)	First Statement Number	Last Statement Number
001 LISTEK	4 010,005 0	0010,2450	
002 PRUG 7	3 008,002 0	0020,3680	
003	5 003,002 0		
004	0 003		
005	5 003,002 0		
006 LIST DATA	1 030,014 0		

File Number

File Type

Unused Storage (in K)

Defective Areas on Tape (* indicates more than nine defective areas)

Defined Function Keys

1 5 5 8

Chapter 3. Data Constants And Variables

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 uppercase 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 or AND sign
?	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 *PRINT USING and IMAGE*, Chapter 4), which specifies the format of printed or displayed data.

Underscore

Note that the underscore (uppercase F) is a valid overstrike character in the 5100.

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 5100 BASIC programs is greater than or equal to $.1E-99$ and less than $1E+99$.

Precision

In BASIC, the precision of an integer or fixed-point number is the maximum number of significant digits it can contain. The precision of a floating-point number is the number of significant digits to the left of the E (see *Floating-Point Format*). The precision of numbers in the 5100 is 13 digits.

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 ($\geq .1E-99$ and $< 1E+99$). Note, however, that the physical length of values you enter is not limited, although entries exceeding 13 digits will be truncated on the right to the 5100 precision of 13 digits. 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*, 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 six internal constants are:

Constant	Name	Value
Pi (π)	&PI	3.141592653590
Natural log	&E	2.718281828459
Square root of 2	&SQR2	1.414213562373
Centimeters per inch	&INCM	2.540000000000
Kilograms per pound	&LBKG	0.453592370000
Liters per gallon	&GALI	3.785411784000

The internal constant names can only be used as parts of arithmetic expressions, and can be preceded by a plus or minus sign, for example:

$2 * \&PI$ (then press EXECUTE)

The result is 6.283185.

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*, Chapter 4) for a program that is chained to (see *CHAIN*, 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 using 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'  
'A B'
```

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 single character constant is limited only by the maximum number of characters on an input line, which is 64 (including quotation marks).

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 \$\$.

When the program is executed, the initial value of character variables is set to 18 blank characters. The only exception is that variables assigned to the common storage area are not initialized (see *USE*, Chapter 4) for a program that is chained to (see *CHAIN*, Chapter 4). These variables are initialized to blanks by a *LOAD* or *RUN* command (see Chapter 2).

When character constants are assigned to character variables, the values of the constants are adjusted to a length of 18. Longer constants are truncated on the right, and shorter constants are left-justified and padded with blanks to 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:

$$\begin{aligned} A(15) &= 22.4 \\ B(3,20) &= 66.6 \end{aligned}$$

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 255.
- An array can be redimensioned in a MAT assignment statement, a MAT READ statement, a MAT INPUT statement, or a MAT GET 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*, 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$` can refer to either a simple character variable or a character array, or both. The name `D$(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 5100 BASIC language includes system functions that perform a number of commonly used mathematical operations. In addition, you can define and name your own functions by using the `DEF` statement (see `DEF`, `FNEND`, `RETURN`, 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
```

Each of the system functions except STR has a single argument (optional with RND), which can be a valid arithmetic expression (explained later in this chapter) and produces 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 included in the 5100 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
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
INT(x)	Integral part of x
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
RAD(x)	Number of radians in x degrees
RND[(x)]	Random number between 0 and 1 ²
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 for z characters
TAN(x)	Tangent of x radians

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

²Note that the random number of a constant will always produce the same result, but can be used to start a different set of numbers. RND without an argument gives you a random number from within the same set of numbers

Figure 8. 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. Expressions can also be formed 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
 $X^3 / (-6)$
 $X + Y + Z$
 $\text{SIN}(R)$
 -6.4
 $-(X - X^2 / 2 + X)$

The value of an arithmetic expression is obtained by performing the implied operations on the specified data items.

The five arithmetic operators are:

Symbol	Meaning
\uparrow or $**$	Exponentiation
$*$	Multiplication
$/$	Division
$+$	Addition
$-$	Subtraction

Note that the system stores $**$ as \uparrow .

The positive/negative operators are:

- + Positive
- Negative

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

Exponentiation: The expression $A \uparrow B$ 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 $A \neq 0$ and $B=0$, $A \uparrow B$ is evaluated as 1.
5. If $A=0$ and $B > 0$, $A \uparrow B$ 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:

1. Following a left parenthesis and preceding an arithmetic expression.
2. 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. \uparrow or $**$	
3. Positive + and Negative -	
4. * and /	
5. Addition + and Subtraction -	Lowest

An expression is evaluated by being reduced 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
$-A \uparrow 2 + B / C * 2.5$	$-4 \uparrow 2 + 6 / 2 * 2.5$ $- 16 + 6 / 2 * 2.5$ $-16 + 6 / 2 * 2.5$ $-16 + 3 * 2.5$ $-16 + 7.5$ -8.5
$(-A \uparrow 2) + B / C * 2.5$	$(-4 \uparrow 2) + 6 / 2 * 2.5$ $- 16 + 6 / 2 * 2.5$ $-16 + 6 / 2 * 2.5$ $-16 + 3 * 2.5$ $-16 + 7.5$ -8.5
$-A \uparrow (2 + B / C) * 2.5$	$-4 \uparrow (2 + 6 / 2) * 2.5$ $-4 \uparrow (2 + 3) * 2.5$ $-4 \uparrow 5 * 2.5$ $- 1024 * 2.5$ $-1024 * 2.5$ -2560
$-A \uparrow ((2 + B) / C) * 2.5$	$-4 \uparrow ((2 + 6) / 2) * 2.5$ $-4 \uparrow (8 / 2) * 2.5$ $-4 \uparrow 4 * 2.5$ $- 256 * 2.5$ $-256 * 2.5$ -640

Character Expressions

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

D\$(4)
 A\$
 STR(A\$,1,3)=B\$

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 13 digits.

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 the collating sequence (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 $\text{\textcircled{0}}$ and BALL, where $\text{\textcircled{0}}$ is a blank character.

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 of numeric data only. 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*, Chapter 4.)

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 the content of B\$ (CONTROL) with the content of A\$ (PRODUCTION) after the 11th character, which is a blank. In statement 30, A\$ is the expression to which characters will be included, 12 is the first position to be filled by the additional characters, and 7 is the number of characters to be included.

- 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.

Note that the length of the operand on the left of the equal sign should be shorter than or equal to the length of the operand on the right of the equal sign.

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 nonexecutable 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 read/write storage size of the IBM 5100 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 5100 BASIC program is from 1 through 9999. You do not have to enter preceding zeros (0020, for example) because the 5100 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:

10 LET A = 2+2

Statement
Number BASIC
 Statement

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 at the end of the chapter under *Matrix Operations*.

BASIC STATEMENTS

The statements used in the BASIC language for the 5100 are listed below. A brief description of each statement is included.

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

CLOSE – Deactivates open files.

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

DEF – Defines an arithmetic function to be used in the program.

DIM – Specifies the size (dimensions) of an array.

END – Ends a program.

FNEND – Ends an arithmetic function defined in a DEF statement.

FOR – Begins a loop.

GET – Assigns values from a file to variables.

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.

INPUT – Assigns values from the keyboard to variables during program execution.

LET – Assigns values to variables.

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

MAT GET – Assigns values from a file to elements of an array.

MAT INPUT – Assigns values from the keyboard to elements of an array.

MAT PRINT (FLP) – Displays or prints the values of all elements of an array.

MAT PRINT USING (FLP) – Displays or prints the values of all elements of an array in a format defined in an image statement.

MAT PUT – Writes the values of all elements of an array into a tape file.

MAT READ – Assigns values from the internal data table (see *DATA*) to elements of an array.

NEXT – Ends a loop (see *FOR and NEXT*).

OPEN – Activates files for input or output.

PAUSE – Interrupts program execution.

PRINT (FLP) – Displays or prints the values of specified variables, expressions or constants.

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

PUT – Writes the values of specified variables into a tape file.

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

REM – Inserts comments or remarks into a program.

RESET – Repositions a tape file to its beginning or to its end.

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

RETURN – Ends a current subroutine.

STOP – Ends a program.

USE – Saves variables to be used by successive programs.

CHAIN 'dev-address',arith-exp

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:

```
CHAIN 'dev-address',arith-exp
```

where:

'dev-address' is the address of the device containing the next program to be loaded and executed. Device address can be 'E80' (built-in tape unit) or 'E40' (auxiliary tape unit).

arith-exp is a simple expression or constant.

Upon execution of the CHAIN statement, the expression is evaluated and truncated to an integer. This integer identifies the file containing the new program on the device specified. A simple constant can also be used. 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' (built-in tape unit) is loaded and executed after the current program is terminated.

Note 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 statement).

CLOSE file ref [,file ref] . . .

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:

```
CLOSE file ref [,file ref] . . .
```

where:

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

Notes About CLOSE

- If a 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.

Example

A sample CLOSE statement is as shown:

```
0020 CLOSE FL2, FL9
```

DATA	$\left\{ \begin{array}{l} \text{arith-con} \\ \text{char-con} \end{array} \right\} \left[\left\{ \begin{array}{l} \text{arith-con} \\ \text{char-con} \end{array} \right\} \right] \dots$
------	--

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:

$$\text{DATA } \left\{ \begin{array}{l} \text{arith-con} \\ \text{char-con} \end{array} \right\} \left[\left\{ \begin{array}{l} \text{arith-con} \\ \text{char-con} \end{array} \right\} \right] \dots$$

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.)

DATA statement character constants containing less than 18 characters are padded on the right with blanks to a length of 18 before being assigned to character variables. Character constants containing more than 18 characters are truncated on the right to a length of 18 before being assigned to character variables.

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.

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 FNfunction-name [(arith-var [,arith-var] . . .)] [=arith-exp]

FNEND [comment]

RETURN [arith-exp]

DEF, FNEND, RETURN

The DEF statement is not executable, but informational. This statement allows the user to define an arithmetic function for use later 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

```
DEF FNfunction-name [(arith-var [,arith-var] . . .)] [=arith-exp]
```

where:

function-name is any character of the extended BASIC alphabet. This character with FN is the name of the defined function.

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

arith-exp is an arithmetic expression that can be invoked by reference in another statement during program execution. A sample single line function DEF statement is as shown:

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

Multiline Function

```
DEF FNfunction-name [(arith-var [,arith-var] . . .)] [=arith-exp]
```

where:

function-name is any character of the extended BASIC alphabet. This character with FN is the name of the function.

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

RETURN [arith-exp]

where:

arith-exp is an arithmetic expression that specifies the value of the user-defined function to the referencing function.

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 arithmetic 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 be arithmetic and must match the number 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 arithmetic expression in a RETURN statement.

DEF

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.
- The arithmetic variables have a special meaning in the DEF statement. Consequently, it is possible to have a variable with the same name as a simple arithmetic 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.

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)
```

```
DIM array-name (rows [,columns]) [,array-name (rows [,columns])]...
```

DIM

The DIM statement allows you to explicitly specify the size of arrays. The syntax of the DIM statement is as shown:

```
DIM array-name (rows [,columns]) [,array-name (rows [,columns])]...
```

where:

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

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

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 18 blank characters.

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 255.

Example

A sample DIM statement is as shown:

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

The result of the preceding statement is:

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

Array A has 4 rows of 2 columns each:

0	0
0	0
0	0
0	0

END [comment]

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:

```
END [comment]
```

where:

comment is optional.

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.

Note About END

The END statement is optional. If omitted, END is assumed by the system to follow the highest-numbered statement in the program.

Example

A sample END statement is as shown:

```
0910 END PG4
```

END

FNEND [comment]

FNEND

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

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:

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

```
NEXT control-var
```

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 (or 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 (or 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

- 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 optional STEP arithmetic expression is omitted in the FOR statement, the increment value is automatically set to +1.

- If the initial value to be assigned to the control variable is greater than (or 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.

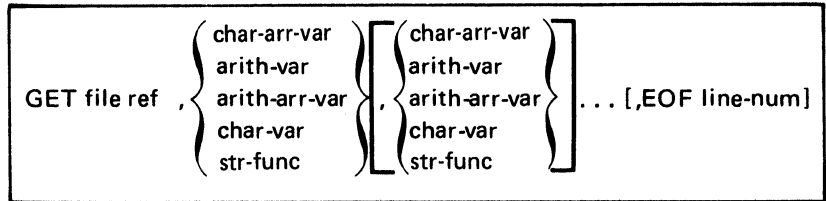
Example

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.

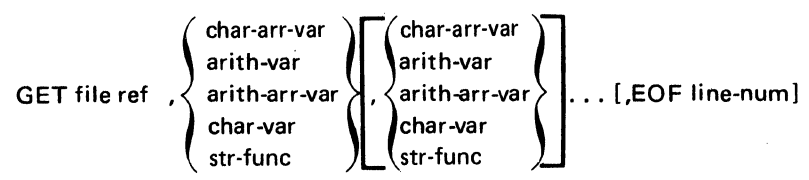
```
10 FOR J = A TO B STEP C
.
.
.
150 FOR K = 1 TO 100
.
.
.
250 NEXT K
.
.
.
300 NEXT J
```



GET

The GET statement allows you to assign values from a specified 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:



GET

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, or a substring reference. Only one variable is required. If more than one is entered, they must be separated by commas.

EOF line-num is the number of the statement to which the program will branch when the end of the file is reached. If this branch is taken, the file is closed and must be reopened before subsequent access.

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.

Notes About GET

- A file currently activated as an output file cannot be specified in a GET statement. It must first be closed, then reopened (in an OPEN statement as an input file).
- Each value assigned in a GET statement must be of the same data type (arithmetic or character) as the corresponding variable.
- A GET statement referencing a currently closed file causes program execution to be terminated.

Example

A sample GET statement is as shown:

```
0090 GET FL4, X, Y, Z, A(4), A(5), D$, EOF 620
```

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

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

RETURN

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

where:

line-num is a string of statement numbers separated by commas. 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.

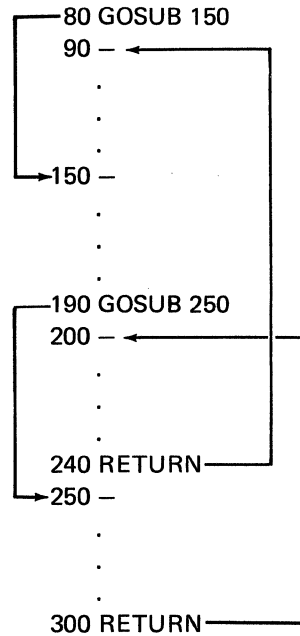
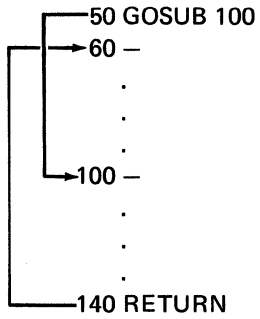
Execution of a computed GOSUB statement causes the arithmetic expression to be evaluated and control transferred to the statement whose numeric position in the list of statement numbers (reading left to right) is equal to the truncated integer value of the expression. Thus, an expression with a value of 2.75 would cause control to be transferred to the second statement in the list. If the expression has a 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.

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.

Example

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



The following example shows a GOSUB to a keys function:

```
100 GOSUB 9994  
.  
.  
.  
9994 PRINT 'IN MAIN PROGRAM'  
9995 RETURN
```

```
LOAD0,KEY4  
9994 REM  
0010 PRINT 'IN KEYGROUP 9994'  
0020 RETURN
```

An arrow points from the '9994' in the first code block to the '9994 REM' line in the second code block.

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

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 string of statement numbers separated by commas. 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 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
```

$\text{IF } \left\{ \begin{array}{l} \text{arith-exp rel-opr arith-exp} \\ \text{char-exp rel-opr char-exp} \end{array} \right\} \left[\begin{array}{l} \{ \& \} \\ \{ \} \end{array} \right] \left\{ \begin{array}{l} \text{arith-exp rel-opr arith-exp} \\ \text{char-exp rel-opr char-exp} \end{array} \right\} \left\{ \begin{array}{l} \text{THEN} \\ \text{GOTO} \end{array} \right\} \text{line-num}$

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:

$$\text{IF } \left\{ \begin{array}{l} \text{arith-exp rel-opr arith-exp} \\ \text{char-exp rel-opr char-exp} \end{array} \right\} \left[\begin{array}{l} \{ \& \} \\ \{ | \} \end{array} \right] \left\{ \begin{array}{l} \text{arith-exp rel-opr arith-exp} \\ \text{char-exp rel-opr char-exp} \end{array} \right\} \left\{ \begin{array}{l} \text{THEN} \\ \text{GOTO} \end{array} \right\} \text{line-num}$$

where:

arith-exp and *char-exp* are arithmetic or character expressions or a string function. Only one pair of expressions is required.

rel-opr is a relational operator. Only one operator is required.

& is the symbol entered for AND. | is the symbol entered for 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.

Before being compared, a character constant containing less than 18 characters is padded on the right with blanks to a length of 18. A character constant containing more than 18 characters is truncated on the right to a length of 18 before comparison. A character constant containing no characters (null) is compared as 18 blank characters.

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 for equal must compare exactly to 13 digits.

Example

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
```

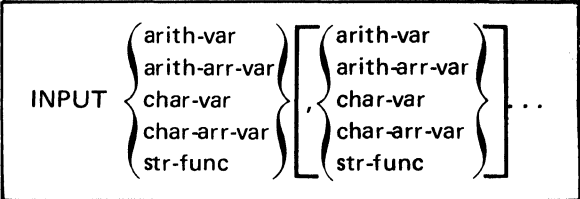
```
80 IF A$ = 'JOB' & B$ = 'DATE' GOTO 100
```

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.

$$: \left\{ \begin{array}{l} \text{char-string} \\ \text{print-image} \end{array} \right\} \dots \text{print-image} \left\{ \begin{array}{l} \text{char-string} \\ \text{print-image} \end{array} \right\} \dots$$

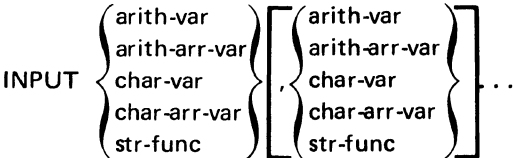
IMAGE

The image statement is used to control formatting of printed or displayed data. For a complete description of the image statement, see *PRINT USING and Image* or *MAT PRINT USING*.



INPUT

The INPUT statement allows you to assign values to variables from the keyboard while your program is being executed. The syntax of the INPUT statement is as shown:



INPUT

where:

var are simple arithmetic or character variables, subscripted references to an array element, or substring references. The rows and columns references to an array element must be enclosed in parentheses. Only one variable is required in an INPUT statement, although many variables can be specified.

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 INPUT statement. When the complete list has been entered, press EXECUTE to resume program execution.

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.

Character constants that contain less than 18 characters are padded on the right with blanks to a length of 18 before being assigned to character variables. Character constants containing more than 18 characters are truncated on the right to a length of 18 before being assigned. Character constants containing no characters (null) are assigned as 18 blank characters.

Notes About INPUT

- Each value entered must be of the same data type (character or arithmetic) as the corresponding variable reference in the INPUT statement. Data types can be mixed in the same statement.
- Each value entered must be separated from the next value by a comma. Two consecutive commas are ignored. To end the series, press EXECUTE.
- A character constant must be enclosed in single quotation marks.
- The number of values entered at execution time must be equal to the number of variable references specified in the 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.

Example

A sample INPUT statement is as shown:

```
60 INPUT A$, B, X, Y(X)
```

```
'YES', 18.6, 8, 1.3597E-6
```

When line 60 is executed, you can enter the list of values shown. The values are then assigned to the corresponding variables.

$$\boxed{
 \begin{array}{l}
 \left. \begin{array}{l}
 \left\{ \begin{array}{l} \text{arith-var} \\ \text{arith-arr-var} \end{array} \right\} \left[\begin{array}{l} \text{arith-var} \\ \text{arith-arr-var} \end{array} \right] \dots = \text{arith-exp} \\
 \left\{ \begin{array}{l} \text{char-var} \\ \text{char-arr-var} \\ \text{str-func} \end{array} \right\} \left[\begin{array}{l} \text{char-var} \\ \text{char-arr-var} \\ \text{str-func} \end{array} \right] \dots = \left\{ \begin{array}{l} \text{char-exp} \\ \text{char-con} \end{array} \right\}
 \end{array} \right.
 \end{array}
 }$$

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:

$$\left. \begin{array}{l}
 \left\{ \begin{array}{l} \text{arith-var} \\ \text{arith-arr-var} \end{array} \right\} \left[\begin{array}{l} \text{arith-var} \\ \text{arith-arr-var} \end{array} \right] \dots = \text{arith-exp} \\
 \left\{ \begin{array}{l} \text{char-var} \\ \text{char-arr-var} \\ \text{str-func} \end{array} \right\} \left[\begin{array}{l} \text{char-var} \\ \text{char-arr-var} \\ \text{str-func} \end{array} \right] \dots = \left\{ \begin{array}{l} \text{char-exp} \\ \text{char-con} \end{array} \right\}
 \end{array} \right.$$

where:

var are scalar variable names, subscripted 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 expressions containing less than 18 characters are padded on the right with blanks to a length of 18 before being assigned to character variables. Character constants containing more than 18 characters are truncated on the right to a length of 18 before being assigned. Character constants containing no characters (null) are assigned as 18 blank characters.

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 (').
- 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).

Example

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)
```

```
50 LET A$ = X'0201130903'
```

After execution of statement 10, the character variable Z\$ will contain the word CAT followed by 15 blank characters.

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.

After execution of statement 50, the variable A\$ will contain the hexadecimal constant listed (see Appendix E for hexadecimal representations).

NEXT control-var

NEXT

For a complete description of the NEXT statement, see *FOR and NEXT*.

NEXT

<pre>OPEN file ref,'dev-address' [,file-num] [, { 'user ID' }] , { IN } { char-var } { OUT }</pre>

OPEN

The OPEN statement allows you to:

- Identify an attached device with a file reference code.
- Allocate space for the file.
- Specify the file number.
- Specify the identification to be used with the file.
- Specify the file usage (input or output).

The syntax of the OPEN statement is as shown:

```
OPEN file ref,'dev-address' [,file-num] [ , { 'user ID' } ] , { IN }
                                     { char-var } { OUT }
```

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 GET, PUT, RESET, CLOSE, PRINT, PRINT USING, MAT PRINT, MAT PUT, MAT GET, and MAT PRINT USING statements.

'dev-address' is the address of the device referenced by the file reference code. Valid addresses are: 'E80' for the built-in tape unit, 'E40' for the auxiliary tape unit, and '500' for the printer.

file-num is a constant or numeric variable. The value of the file number is then used to access the corresponding file on the device specified. Decimal values for variables are truncated to an integer. See *Negative File Numbers*.

char-var is a character variable or character constant enclosed in single quotation marks. The first 17 characters provide the identification field in the header record of a file being opened for output. If you open a file for input, and specify a character variable for identification, the user ID from the file is placed in the character variable when OPEN is executed.

IN or OUT indicates whether the file is to be used for input (IN) or output (OUT).

A sample OPEN statement is as shown:

```
OPEN FL1, 'E80', 3, IN
```

In this example, file reference code FL1 will be assigned to reference file 3 on the built-in tape unit for input.

Negative File Numbers

Negative file numbers can be used to create a type 2 general interchange (source or data) file that can then be processed by BASIC or APL. The following rules control the use of negative file numbers.

- The negative number must first be assigned to a variable. Negative constants cannot be entered directly in the OPEN statement as file numbers. For example:

```
10 F = -1
20 OPEN FL2, 'E80', F, OUT
```

- Each statement in the created file (source or data) must end with hexadecimal X'E3'. This specification is the internal code recognized as end of line by the 5100. The X'E3' is provided automatically if PUT or MAT PUT is used to write the file. The X'E3' must be specified by the program if PRINT is used to write the file.

The following example shows a single file creation with a negative file number:

```
10 F = -1
20 OPEN FL2, 'E80', F, OUT
30 A$ = X'E3'
40 PRINT FL2, '0090 PRINT &PI, &E'; A$;
50 CLOSE FL2
```

In this example, file reference FL2 will contain file number -1, which consists of the single statement to display &PI and &E. Hexadecimal constant X'E3' is assigned to A\$, which ends the source statement (40).

Notes About OPEN

- An OPEN statement must be issued before a GET, MAT GET, PRINT, MAT PRINT, PRINT USING, MAT PRINT USING, PUT, or MAT PUT statement that references the file reference code (FL0-FL9) specified in the OPEN statement.
- If a file is already open, the OPEN statement causes an error message.
- Once a file is open, do not remove the tape cartridge until the file is closed.
- A quote cannot be embedded in a character constant used for a user ID.

```
PAUSE [comment]
```

PAUSE

The PAUSE statement allows you to interrupt program execution to perform calculator operations. The syntax of the PAUSE statement is as shown:

```
PAUSE [comment]
```

where:

comment is optional.

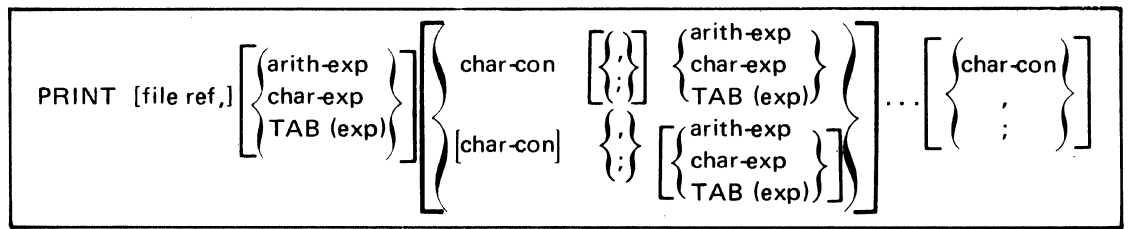
When a PAUSE statement is encountered during program execution, execution is interrupted and the message:

```
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*, Chapter 2) or alter a calling statement (see *DEF*, *FNEND*, *RETURN*) while a program is interrupted.

The following statement would cause the message PAUSE 0080 to be displayed and processing to be suspended until a GO command is issued:

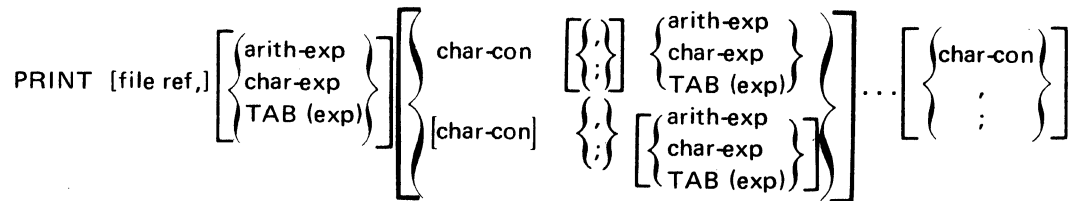
```
80 PAUSE
```



PRINT

The PRINT statement causes the values of specified scalar expressions to be displayed on the display screen, printed by the printer, or written to a file. When using the 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 PRINT statement is as shown:



PRINT

where:

file ref is FLP (printer) or FLO to FL9 (logical file). This entry is optional. FLO 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*). 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.

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.

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 is printed or displayed in its own print zone.

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 or Semicolon Delimiter): For arithmetic expressions, 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 (b represents a blank)
2-4	6	b7.3bb
5-7	9	b17.357bb
8-10	12	-45.63927bb
11-13	15	b1.73579E-23bbb
14-16	18	-8.92270493115bbbb
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*, Chapter 2).

Figure 9. Packed Print Zone Lengths for Arithmetic Expressions

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.

Character Variables

The actual characters (excluding trailing blanks) are printed or displayed.

PRINT

Arithmetic Expressions

I-Format: The integer, consisting of a sign (blank or minus) and up to 13 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 13 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 $1E-1$ 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 13 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 are transferred to the display screen, an internal character-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 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 using the TAB function.

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)		The record is printed or displayed and the pointer is set to the start of the next record.
	Null (end of statement)		

Figure 10 (Part 1 of 3). Positions of Display or Print Line-Position Pointer

PRINT

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.

PRINT

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 non-integer, 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:

$$X = n - L * \text{INT}((n-1)/L)$$

For example, if line length = 64 and the TAB(n) = 70, the integer value $((70-1)/64) = 1$ is multiplied by 6(70-64). 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 using 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 $\text{\textcircled{b}}$ represents a blank character, which always appears in the sign position of a positive number.

Value Given	Value Printed
0123	$\text{\textcircled{b}}$ 123
00123	$\text{\textcircled{b}}$ 123
-1234.5	-1234.5
135999999999	$\text{\textcircled{b}}$ 135999999999

Note About PRINT

If there is no printer attached to the 5100, executing the PRINT FLP statement stops the program unless the P = D (printer = display) option was specified in the RUN command.

Example

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	
110 PRINT 'NAME', TAB (30) 'ADDRESS'	NAME—25 blanks—ADDRESS
120 PRINT 'A', TAB (30) 'B', TAB (40) 'C'	A—29 blanks—B—9 blanks—C
130 END	

$\text{PRINT USING [file ref,] line-num } \left[\left\{ \begin{array}{l} \text{arith-exp} \\ \text{char-exp} \end{array} \right\} \left[\left\{ \begin{array}{l} ; \\ ; \end{array} \right\} \left\{ \begin{array}{l} \text{arith-exp} \\ \text{char-exp} \end{array} \right\} \right] \dots \left[\left\{ \begin{array}{l} ; \\ ; \end{array} \right\} \right]$
$: \left\{ \begin{array}{l} \text{char-string} \\ \text{print-image} \end{array} \right\} \dots \text{print-image} \left\{ \begin{array}{l} \text{char-string} \\ \text{print-image} \end{array} \right\} \dots$

PRINT USING AND IMAGE

The PRINT USING statement and its associated image statement allow you to display specified scalar values in a format of your own choosing. The PRINT USING statement specifies the values to be displayed and the line number of the image statement to be used. The image statement specifies the format of the line to be displayed. The syntax of the PRINT USING and image statements is as shown:

$\text{PRINT USING [file ref,] line-num } \left[\left\{ \begin{array}{l} \text{arith-exp} \\ \text{char-exp} \end{array} \right\} \left[\left\{ \begin{array}{l} ; \\ ; \end{array} \right\} \left\{ \begin{array}{l} \text{arith-exp} \\ \text{char-exp} \end{array} \right\} \right] \dots \left[\left\{ \begin{array}{l} ; \\ ; \end{array} \right\} \right]$
$: \left\{ \begin{array}{l} \text{char-string} \\ \text{print-image} \end{array} \right\} \dots \text{print-image} \left\{ \begin{array}{l} \text{char-string} \\ \text{print-image} \end{array} \right\} \dots$

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 that defines how data is to be formatted when printed or displayed.

exp is an arithmetic or character value separated by semicolons or commas. These values are incorporated (edited) into the format of the image statement specified. A comma or semicolon can also follow the last expression.

print-image is the format specification (see *Format Specifications*).

When a PRINT USING statement is executed, the specified expressions are evaluated, and their values are edited in order of appearance in the PRINT USING statement into the corresponding format specifications in the specified image statement. A string of character constants will be printed exactly as entered in the image statement described in the following text.

Conversion of Data Reference Values

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 negative, 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 floating-point number, either rounding the value or extending it with zeros and adjusting the exponent according to the format specification.

E-Format: The value of the expression is converted to a floating-point number, rounding the value or extending it with zeros according to the format specification.

If 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, 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* 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 consecutive 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	Ø12
###	1.23	ØØ1
##.##	123	*****
##.##	1.23	Ø1.23
##.##	1.23456	Ø1.23
##.##	.123	ØØ.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

PRINT
USING

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 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. This allows the printing of lines longer than 64 characters.

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.

If the end of the image 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 statements permitted in a BASIC program is only limited by available storage.
- If the 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 5100, executing the PRINT USING FLP statement stops the program, unless the P = D (printer = display) option was specified in the RUN command.
- Image statements are nonexecutable and can be placed anywhere in a BASIC program, either before or after the PRINT USING statements that refer to them.

Example

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
```

PUT file ref, $\left\{ \begin{array}{l} \text{arith-exp} \\ \text{char-exp} \end{array} \right\} \left[\left\{ \begin{array}{l} \text{arith-exp} \\ \text{char-exp} \end{array} \right\} \right] \dots$
--

PUT

The PUT statement allows you to write the values of specified variables into a particular file. The file must have already been opened with an OPEN statement. The syntax of the PUT statement is as shown:

$$\text{PUT file ref, } \left\{ \begin{array}{l} \text{arith-exp} \\ \text{char-exp} \end{array} \right\} \left[\left\{ \begin{array}{l} \text{arith-exp} \\ \text{char-exp} \end{array} \right\} \right] \dots$$

where:

file ref is FL0 to FL9 to identify the file into which variable values will be written.

exp are function references or subscripted array references representing the values to be written. Only one value is required. If more than one is specified, the values must be separated by commas.

When a PUT statement is executed, the specified scalar reference 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 PUT statement will be the first value assigned from the file when the file is referenced in a GET statement. When the buffer becomes full, the contents are written out as a record. Tape records are 512 bytes in length. As many bytes as possible are used in each record.

A file can only be activated by an OPEN statement and is deactivated by a CLOSE statement or at the end of program execution.

Notes About PUT

- A file currently activated as an input file cannot be specified in a 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 PUT statement referring to a currently closed file causes program execution to be terminated.

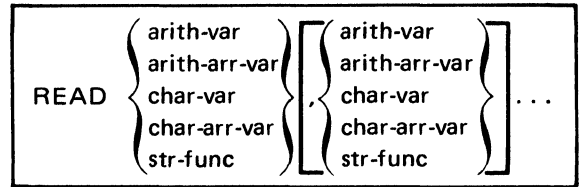
Example

A sample PUT statement is as shown:

20 PUT FL1, Z3, 5*X-7,A,D\$,9.005

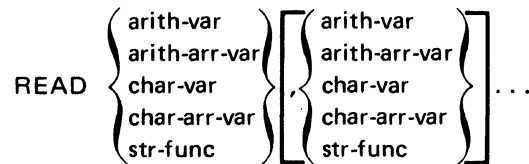
In this example, the specified values will be written into file FL1.

PUT



READ

The READ statement allows you to assign values to variables and array elements from the internal data table created by DATA statements. The syntax of the READ statement is as shown:



where:

var are simple arithmetic or character variables, substrings, or subscripted references to a single array element. Only one variable is required.

When a READ statement is executed, successive values from the data table are assigned to the variables in the READ statement from left to right beginning at the current position of the data table pointer. The data table pointer can be returned to the beginning of the table by the RESTORE statement.

Subscripts of array variables in the READ statement are evaluated as they occur; thus, an assigned variable in a READ statement can be used subsequently as the subscript of another variable in the same statement.

Notes About READ

- 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 there are no DATA statements in the program, the READ statement causes an error.
- If the data table is exhausted and unassigned variables still remain in the READ statement, an error message is displayed.
- A READ statement in a program assigned to one of the function keys will cause an error.

Example

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.

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:

```
REM [comment]
```

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*, Chapter 2.

Example

A sample REM statement is as shown:

```
10 REM THIS PROGRAM DETERMINES THE COST PER UNIT
```

```
RESET file ref [END] [,file ref [END]] ...
```

RESET

The RESET 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:

```
RESET file ref [END] [,file ref [END]] ...
```

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 described below.

When a RESET statement is executed, the specified 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, 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 and OPEN statements. The same is true of files to be used for output while open for input in the same program.

Note About RESET

If a file specified in a RESET statement is not currently active, the file reference in the RESET statement is ignored.

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

The RESTORE statement allows you to begin assigning values beginning with the first item in the first DATA statement (see *DATA*) of the program according to the next READ statement executed. The syntax of the RESTORE statement is as shown:

```
RESTORE [comment]
```

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 executing the following statements, 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 [arith-exp]

RETURN

For the use of the RETURN statement in the creation of subroutines, see the GOSUB and RETURN statements. Also see *Function Keys*.

For the use of the RETURN statement with a multiline function definition, see the *DEF, FNEND, RETURN*.

STOP [comment]

STOP

The STOP statement allows you to terminate program execution. The syntax of the STOP statement is as shown:

```
STOP [comment]
```

where:

comment is one or more characters.

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.

Note About STOP

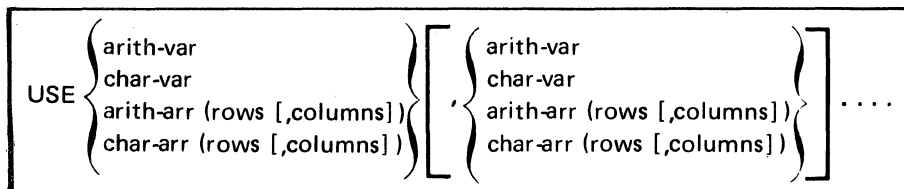
A STOP statement can appear anywhere in a BASIC program.

Example

A sample STOP statement is as shown:

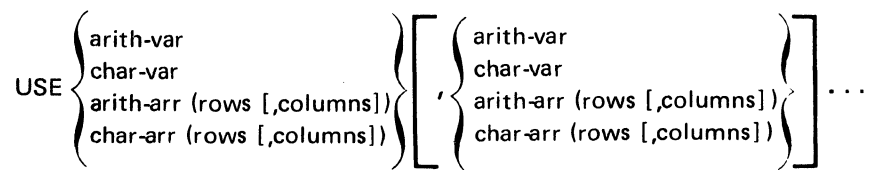
```
110 STOP
```

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. The values of these variables are not changed when one program CHAINs to another program (see CHAIN statement). The syntax of the USE statement is as shown:



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.

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).
- 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.
- Only one USE statement can be entered in a program and it must be the first statement executed that contains a variable reference. 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.

MATRIX OPERATIONS

In the 5100, 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 previously 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

```
MAT array-name [(rows[,columns])] = (scalar-exp)
```

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:

```
MAT array-name [(rows[,columns])] = (scalar-exp)
```

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*).

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 is less than 18 characters, it is padded on the right with blanks to a length of 18 before being assigned. If there are more than 18 characters in the expression, only the leftmost 18 characters are assigned to the array elements.

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.

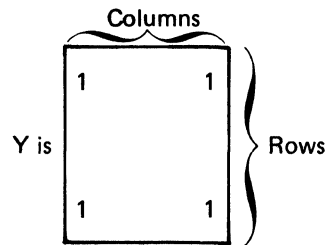
Example

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 array-name [(rows, [columns])] = array-name

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:

MAT array-name [(rows, [columns])] = array-name

where:

array-name is the name of the array.

rows, columns are the redimension specifications for the first array (see *Redimensioning Arrays*, 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 $B = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ and $A = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$

then, after statement 100:

$B = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$

MAT matrix-name [(rows [,columns])] = matrix-name $\left\{ \begin{array}{c} + \\ - \end{array} \right\}$ matrix-name

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:

MAT matrix-name [(rows [,columns])] = matrix-name $\left\{ \begin{array}{c} + \\ - \end{array} \right\}$ matrix-name

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*, 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 =

a	b
c	d

and Z =

e	f
g	h

, then X =

a+e	b+f
c+g	d+h

MAT

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:

MAT matrix-name [(rows ,columns)] = matrix-name * matrix-name

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 that for $i = 1,2,\dots,p$ and for $j = 1,2,\dots,n$:

$$C(i,j) = \sum_{k=1}^m A(i,k) * B(k,j)$$

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*, 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:

If X =

a	b
c	d

 and Y =

e	f
g	h

, then Z =

a*e+b*g	a*f+b*h
c*e+d*g	c*f+d*h

MAT matrix-name [(rows[,columns])] = (arith-exp)*matrix-name

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:

MAT matrix-name [(rows[,columns])] = (arith-exp)*matrix-name

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*, 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 =

a	b
c	d

, then Y =

4*a	4*b
4*c	4*d

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:

MAT matrix-name [(rows, columns)] = IDN

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*, 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:

MAT matrix-name [(rows, columns)] = INV (matrix-name)

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*) can be used to determine if a given matrix has an inverse. The inverse of matrix A exists if $DET(A) \neq 0$.

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 if the result is 1E-20 or less.
- If redimension specifications are included, the rules under *Redimensioning Arrays*, 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 END
```

The resulting values are:

If Y is

1	1
1	2

, then X is

2	-1
-1	1

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:

MAT array-name [(rows, columns)] = TRN (array-name)

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*, 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

a	d
b	e
c	f

, then B is

a	b	c
d	e	f

```
MAT GET file ref, array-name [(rows [,columns])] [,array-name [(rows [,columns])]] ... [,EOF line-num]
```

MAT GET

The MAT GET statement allows you to assign a value to each element of an array from a specified file without referring to each individual array element. The specified file must have already been opened with an OPEN statement (see *OPEN*). You can also use the MAT GET statement to redimension arrays. The syntax of the statement is as shown:

```
MAT GET file ref, array-name [(rows [,columns])] [,array-name [(rows [,columns])]] ... [,EOF line-num]
```

where:

file ref is FL0 to FL9 to represent the file specified in the OPEN statement.

array-names are names of character or numeric arrays.

rows, columns are the dimensions of the arrays named.

EOF line-num is the number of the statement to which the program will branch when the end of the file is reached. If this branch is taken, the file is closed and must be reopened before subsequent access.

When a MAT GET statement is executed, 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.

A file can only be activated by an OPEN statement and is deactivated by a CLOSE statement or at the end of program execution.

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 and then reopened.
- If the input file is exhausted before a specified array is filled, program execution is terminated unless the EOF line number is specified.
- If the redimension specifications are included, the rules under *Redimensioning Arrays*, Chapter 3, must be followed.

Example

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.

```
MAT INPUT array-name [(rows [,columns])] [,array-name [(rows [,columns])]] ...
```

MAT INPUT

The MAT INPUT statement allows you to assign values to array elements from the keyboard without specifying each array element individually. The MAT INPUT statement can also be used to redimension arrays. The syntax of the statement is as shown:

```
MAT INPUT array-name [(rows [,columns])] [,array-name [(rows [,columns])]] ...
```

where:

array-names are character or numeric arrays.

rows, *columns* are the dimensions of the arrays named. Only one array name is required and rows, columns is optional.

When a MAT INPUT statement is executed, it causes a flashing question mark to be displayed on line 1 of the display screen, and program execution is interrupted. You must then enter a list of values that will be assigned row-by-row to elements of the specified arrays.

After each line of input (64 characters maximum) has been keyed, it must be processed by pressing EXECUTE. If the array specified has not been filled, the question mark is displayed again to indicate that more input is required.

Character constants that contain less than 18 characters are padded on the right with blanks to a length of 18 before being assigned to the array element. Character constants that contain more than 18 characters are truncated on the right before being assigned. Character constants containing no characters (null) are assigned as 18 blank characters.

Notes About MAT INPUT

- Each value entered must be of the same type (character or numeric) as the corresponding array element.
- Each value entered must be separated from the next value by a comma.
- The last value entered on a line can be followed by a comma or a blank.
- If the number of data items entered exceeds the number of array elements referred to, the excess data items are ignored.
- Blanks within *numeric* data items are ignored.
- Character data must be enclosed in single quotation marks.
- If redimension specifications are included, the rules under *Redimensioning Arrays*, Chapter 3, must be followed.

Example

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).

MAT PRINT [file ref,] array-name $\left\{ \begin{array}{c} ; \\ , \end{array} \right\}$ array-name] . . $\left[\begin{array}{c} ; \\ , \end{array} \right]$

MAT PRINT

The MAT PRINT statement causes the elements of a specified numeric or character array to be displayed or printed without referring to each array element individually. When using the MAT PRINT statement, the format of all displayed or printed values is standardized, but the spacing between values on the same line can be controlled. The syntax of the statement is as shown:

MAT PRINT [file ref,] array-name $\left\{ \begin{array}{c} ; \\ , \end{array} \right\}$ array-name] . . $\left[\begin{array}{c} ; \\ , \end{array} \right]$

where:

file ref is FLP (for the printer) or FL0 to FL9 for the logical file. FL0 to FL9 must have been previously referenced in an OPEN statement.

array-names are arithmetic or character arrays, and the *comma* or *semicolon* are delimiters that specify the spacing between array elements. Only one array name need be specified.

When a MAT PRINT statement is executed, each array element is converted to a specified output format and displayed. See *Conversion of Data Reference Values* under the PRINT statement.

Each array is displayed by rows; the first row of each array begins at the start of a new line and is separated from 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 Displayed Values*.

Spacing of Displayed Values

The converted value of each array element is 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.

Full Print Zones (Comma Delimiter): Each array element uses one full print zone of 18 character positions.

Packed Print Zones (Semicolon Delimiter): For numeric arrays, the length of the packed print zone is determined by the length of each converted value, including the sign, digits, decimal point, and the exponent, as shown in Figure 11.

Length of Converted Data Item (Character)	Length of Packed Print Zone (Character)	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

Figure 11. Packed Print Zone Lengths for Arithmetic Array Elements

Display/Print Line Operation

As the values of array elements are transferred to the display screen or printer buffer, an internal line-position pointer is maintained to keep track of the next available position where a character (or digit) can be placed.

If the array delimiter is a comma, the line-position pointer is moved past any remaining positions in the full print zone after transferring 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 transferring the element values to the display screen. See *Print Line Buffer Operation* under *PRINT*.

Notes About MAT PRINT

- Null delimiters are not permitted in a MAT PRINT statement except as the final delimiter.
- If your 5100 does not have a printer, the MAT PRINT FLP statement stops the program unless P = D (printer = display) option was specified in the RUN command.

MAT
PRINT

Example

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 [file ref,] line-num, array-name $\left[\begin{array}{c} \{ \} \\ \{ : \} \end{array} \right] [\text{array-name}] \dots \left[\begin{array}{c} \{ \} \\ \{ : \} \end{array} \right]$

: $\left[\begin{array}{c} \{ \text{char-string} \} \\ \{ \text{print-image} \} \end{array} \right] \dots \text{print-image} \left[\begin{array}{c} \{ \text{char-string} \} \\ \{ \text{print-image} \} \end{array} \right] \dots$

MAT PRINT USING

The MAT PRINT USING statement and its associated image statement allow you to display or print all elements of a specified array, in a format of your choice, without specifying each individual array element. In the MAT PRINT USING statement, you can specify the values to be displayed or printed and the line number of the image statement to be used to format the display lines. The syntax of the MAT PRINT USING statement is as shown:

MAT PRINT USING [file ref,] line-num, array-name $\left[\begin{array}{c} \{ \} \\ \{ : \} \end{array} \right] [\text{array-name}] \dots \left[\begin{array}{c} \{ \} \\ \{ : \} \end{array} \right]$

where:

file ref is FLP (for the printer) or FL0 to FL9 for the logical file.

line-num is the number of the image statement.

array-names are numeric or character arrays whose elements are to be edited into the format specified in the image statement. Only one array name is required. Array names can be separated by a comma or semicolon. The last array name can also be followed by a comma or semicolon.

The syntax of the image statement is:

: $\left[\begin{array}{c} \{ \text{char-string} \} \\ \{ \text{print-image} \} \end{array} \right] \dots \text{print-image} \left[\begin{array}{c} \{ \text{char-string} \} \\ \{ \text{print-image} \} \end{array} \right] \dots$

where:

char-string is a string of characters that are *not* enclosed in single quotation marks.

print-image is format specifications (see *Format Specifications* under *PRINT USING and IMAGE*). Only one print-image is required in the image statement.

When a MAT PRINT USING statement is executed, the specified array references are evaluated, and their elements are edited, in the order of their appearance in the MAT PRINT USING statement, into the corresponding format specifications in the specified image statement. A string of character constants will be displayed or printed exactly as entered in the image statement.

Each array element is displayed by row according to the format specifications defined by the associated image statement.

When displayed, the first row of each array begins at the start of a new line and is separated from the preceding line by two blank lines. Each succeeding array row starts at the beginning of a new line and is separated from the preceding row by one blank line.

If the number of elements in a row of an array in a MAT PRINT USING statement exceeds the format specifications in the image statement, the delimiter between array references controls the output:

- If the delimiter is a comma or blank, and the end of the image format specifications is reached, the current line is displayed or printed, and the output continues on a new line using the start of the image statement.
- If the delimiter is a semicolon, and the end of the image format specifications is reached, the output continues on the same line using the start of the image statement. This allows the printing of lines longer than 64 characters.

After the last row of the last array is printed or displayed, the delimiter following that reference in the MAT PRINT USING statement causes the following:

- If the delimiter is a comma or blank, the current line is printed or displayed and the next output will start on a new line.
- If a semicolon is at the end of the array reference list, a new line is not forced after the last data printed or displayed. Thus, the next displayed or printed data will be on the same line.

Notes About MAT PRINT USING

- If your 5100 does not have a printer, the MAT PRINT USING FLP statement stops the program, unless the P = D option was specified in the RUN command.
- The image statement must contain at least one conversion specification.
- 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.

Example

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 PRINT
USING

MAT PUT file ref, array-name [,array-name] . . .

MAT PUT

The MAT PUT statement allows you to place the value of each element of an array into a specified file, without referring to each individual array element. The file must have already been opened with an OPEN statement. The syntax of the MAT PUT statement is as shown:

```
MAT PUT file ref, array-name [,array-name] . . .
```

where:

file ref is FL0 to FL9 to identify the file defined in the OPEN statement.

array-name is the name of a character or numeric array. Only one array is required..

When a MAT PUT statement is executed, the values of the specified array elements are entered into a buffer for the specified file, beginning at the current file position. The file is arranged sequentially so that the first value entered will be the first value assigned from the file when it is referred to by a MAT GET (or GET) statement. When the buffer becomes full, the contents are written out as a record.

A file can only be activated 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 first, then reopened as an output file, or a RESET END statement must be issued (see *RESET*).
- If space in the output file is exhausted before all items in the output list are placed in the file, program execution is terminated.
- A MAT PUT statement referring to a currently closed file causes program execution to be terminated.
- A MAT PUT statement writes one matrix row in each record on the tape. When accessed with a LOADx, DATA command (which uses a 324-character buffer), the matrix data per row cannot exceed 324 characters (including commas).

Example

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 array-name [(rows [,columns])] [array-name [(rows [,columns])]] ...
```

MAT READ

The MAT READ statement allows you to assign values from the data table (established by DATA statements) to the elements of an array without referring to each individual array element. You can also use the MAT READ statement to redimension arrays. The syntax of the MAT READ statement is as shown:

```
MAT READ array-name [(rows [,columns])] [array-name [(rows [,columns])]] ...
```

where:

array-name is the name of a numeric or character array. Only one array name is required.

rows, columns are the optional redimension specifications for the array.

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.

Notes About MAT READ

- Before being used in a MAT READ statement, arrays must have been previously defined, either implicitly, or explicitly in a USE or DIM statement.
- If the data table is exhausted and unassigned 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*, Chapter 3, must be followed.

Example

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 of 18 characters each, array **B** is redimensioned to 12 rows, and then values from the data table are assigned to arrays **A\$**, **A**, and **B**.

5100 BASIC COMPATIBILITY WITH IBM 370/VS BASIC

The BASIC language used in the 5100 differs from IBM 370/VS BASIC in the following areas:

- Lowercase alphabetic characters are not available in the 5100.
- Double quotation marks are not available in the 5100.
- The 5100 provides only one precision (13 digits of precision).
- The 5100 provides hexadecimal constants, which are enclosed in single quotation marks and preceded by X.
- The 5100 allows entry of a TAB (expression) within a PRINT statement.
- The 5100 allows you to direct formatted output to either the printer, or the display screen or a tape file. This capability is provided through an additional parameter (FLP) in PRINT, MAT PRINT, PRINT USING, and MAT PRINT USING statements. An entry of FLO-FL9 in this position directs output to the file referred to in a corresponding OPEN statement.
- The 5100 allows you to specify file description information which is provided by the operating system (in 370/VS BASIC).
- The 5100 allows you to CHAIN to a specified program without initializing the data area reserved by variables specified in a USE statement.
- The 5100 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 5100 provides the TAB(x) function, but does not provide the following VS BASIC functions: CEN(x), CLK, CNT, CPU, DAT(x), DOT(x,y), FAH(x), IDX(x,y), JDY(x), KLN(x), KPS(x), LEN(x), MAX(x,y,z), MIN(x,y,z), NUM(x), PRD(x), RLN(x), SUM(x), and TIM.
- The 5100 does not provide the ascending (ASORT) and descending (DSORT) array assignment functions.
- The 5100 requires statement numbers with values up to 9999, and permits special functions to be assigned by the user to numbers 9990-9999.
- The 5100 does not provide the following BASIC statements: DELETE FILE, EXIT, FORM, READ FILE, REREAD FILE, REWRITE FILE, and WRITE FILE.

- The 5100 does not provide the EXIT, CONV, or IOERR parameters in the MAT GET statement, the ELSE parameter in the IF statement, the EXIT and IOERR parameters in the MAT PUT statement, or the ALL parameter in the OPEN statement.
- The 5100 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.

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 sunlight.
- Do not touch or clean the tape surface.
- If a data cartridge has been exposed to a temperature drop exceeding 30° F 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 a 151 error (end of tape). Press ATTN to continue.
 3. Use the REWIND command to rewind the tape.

DATA SECURITY

The 5100 provides a number of functions that allow you to protect important or sensitive data. The magnetic tape cartridge can easily be locked in a desk much the same as any other confidential record. You must remember that cartridges containing sensitive data are *your* responsibility. You can rewrite (OPEN for output and CLOSE) a tape file to make that data inaccessible to the 5100. You can also use the following BASIC statements to erase data from a tape file:

```
10 DIM A(100)
20 OPEN FLO, 'E80',1,OUT
30 MAT PUT FLO,A
40 GOTO 30
```

This program will write zeros into all positions, then display an error message to indicate end of file. In addition to the security advantages of the tape cartridge, you can eliminate all data currently in the 5100 work area by entering LOAD0, by pressing RESTART, or by turning the power off.

Remember also that the write protect arrow on the tape cartridge ensures that important information on the tape will not be destroyed.

STORAGE CONSIDERATIONS

The following list shows how many bytes of storage are required for each data type that can be in the work area:

Data Type	Number of Bytes Required
Character variable	22
Character array	18 times the number of elements plus 8
Numeric variable	12
Numeric array	8 times the number of elements plus 8

Because the 5100 work area contains a fixed amount of storage, it is a good practice to conserve as much storage as possible.



Appendix A. BASIC Characters

The following chart lists all the characters available on the 5100, BASIC-only keyboard, and their relative value, lowest to highest:

Character	Sequence Number	Character	Sequence Number
(blank)	65	=	127
'	76	↑	139
<	77	≤	141
(78	[174
+	79	≥	175
	80]	190
&	81	≠	191
!	91	A	194
\$	92	B	195
*	93	C	196
)	94	D	197
;	95	E	198
-	97	F	199
/	98	G	200
,	108	H	201
>	111	I	202
?	112	J	210
:	123	K	211
#	124	L	212
@	125	M	213
'	126	N	214

Character	Sequence Number	Character	Sequence Number
O	215	Y	233
P	216	Z	234
Q	217	0	241
R	218	1	242
\	225	2	243
S	227	3	244
T	228	4	245
U	229	5	246
V	230	6	247
W	231	7	248
X	232	8	249
		9	250

Appendix B. BASIC Error Messages And Operator Recovery

The following list contains all the 5100 error codes. Any detected error will deactivate keys on the keyboard and cause the display screen to flash. To stop the flashing display, press ATTN. Error codes below 100 are I/O errors and are displayed as 3-digit numbers, followed by three blanks and one of the following device addresses: 500 for the printer, E80 for the built-in tape unit, or E40 for the auxiliary tape unit. Error code 800 is for UTIL MODE COM when the device is not attached. Refer to *Serial I/O Adapter* for other valid device addresses. Error codes above 100 are simply displayed as 3-digit numbers. Unless otherwise specified, 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.
- 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 EXECUTE.

Note that if errors are detected during execution of a program involving tape operations, you should enter GO END to ensure that tape files are properly closed. In this situation, *do not* attempt a LOAD, MERGE, or REWIND operation because these commands can cause incorrect operation when the program is restarted.

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.

Error	Meaning	Recovery
002	Invalid command for device, or invalid command sequence for device.	Enter GO END to end the program. Check OPEN and CLOSE statements for valid device addresses. Valid device addresses are 500, E80, or E40. The error can also be caused by attempting to open or CHAIN to file 0, for example: OPEN FL1, 'E80', 0, OUT. File values such as .5 can also cause the error. If OPEN or CHAIN statements were incorrect, correct them and run the program again. If the OPEN and CHAIN statements are correct, retry the operation. If the error persists, call the service personnel.
003	Hardware error, or tape cannot be correctly positioned.	If running a BASIC program, enter GO END to end the program and then retry the operation causing the error. If the error occurs on a LOAD or SAVE command, retry the command. If the error persists, call the service personnel.
004	Hardware error, unmarked tape, lack of power on the second tape, erased or magnetically damaged tape, or movement of tape beyond end of marked tape.	<p>If the error occurs due to uneven winding of the tape, the tape should be moved to its extremes as described under <i>Tape Cartridge Handling and Care</i>. If the error occurs during an operation, retry. If the error occurs while attempting a command or opening a file, rewind and retry. If the retry does not work in either case, call the service personnel.</p> <p>If the error occurs during execution of a BASIC program, enter GO END to end the program. Then rewind the tape and retry the operation causing the error. If the error occurs on a LOAD or SAVE command, retry the command. If the error recurs, issue the REWIND command (after issuing GO END, if required) and retry the operation. If the error recurs, call the service personnel.</p>
005	Tape cartridge not inserted.	<p>Insert tape cartridge. If the error occurs on a command operation, reenter the command. If the error occurs during program execution:</p> <ul style="list-style-type: none"> – Insert tape cartridge and enter GO END to end the program. – Insert tape cartridge and enter GO x to continue, where x is the statement number of the OPEN statement. <p>If the error recurs, call the service personnel.</p>
006	Tape is file-protected.	<p>If you want to write to tape, turn the SAFE switch on the cartridge off of the safe position and:</p> <ol style="list-style-type: none"> 1. If the error occurs on a command operation, reenter the command. 2. If the error occurs when opening a file during program execution: <ul style="list-style-type: none"> – Enter GO END to end the program. – Enter GO x to continue, where x is the statement number of the OPEN statement. <p>If the error still recurs, call the service personnel.</p>

Error	Meaning	Recovery
007	A CRC (cyclic redundancy check) error has occurred.	<p>If the error occurs during program execution, enter GO END to end the program. Retry the operation causing the error. If the error recurs and the error is on tape, the tape file is probably bad. Use the tape recovery program if the file type is 1, 2, 3, or 8.</p> <p>If the error recurs, call the service personnel.</p>
008	The next expected physical record cannot be found in sequence.	<p>Enter GO END to end the program. Retry the operation. If the error recurs, copy the files following the file in error onto another cartridge; then re-mark the first cartridge from the file in error for as many files as you want.</p> <p>This error could occur if the tape was removed during a previous write operation or MARK command.</p>
009	End of data has been reached during a read from a tape file. Note that more data can be added to the file.	<p>Enter GO END to end the program. You should check the program logic to correct it from trying to get too much data. Optionally, the EOF statement number may be designated in GET or MAT GET statement. The EOF statement number specifies the statement number to which the program branches when end of file is encountered. Correct the program logic and/or add the EOF clause to GET, MAT GET statements.</p>
010	End of file has occurred during a read from a tape when trying to read beyond the last record in the file, or during a write when trying to write beyond the last record in the file.	<p>Enter GO END to end the program. If the error occurred on the SAVE command, the file is not large enough. Either save the data in a larger file or mark a file large enough to contain the data; then perform the SAVE operation.</p> <p>If the error occurred during program execution, you should check program logic to correct it from trying to get too much data. Optionally, the EOF statement number may be designated in GET or MAT GET statements. The EOF statement number is the number to which the program branches when end of file is encountered. Correct program logic and/or add EOF clause to GET, MAT GET statements.</p> <p>If the error occurs when using PUT, MAT PUT, the file is not large enough to contain the data. Either specify a larger file or mark a larger file, change appropriate statements in your program and run the program.</p> <p>If the error occurs when using RESET END, it may indicate that the file is empty. In this case, the file should be opened for output instead of using RESET END. If the file is not empty when this error occurs on a RESET END, it indicates that there is no room to add records to this file.</p>

Error	Meaning	Recovery
011	Specified file number cannot be found.	Be sure the correct cartridge is loaded. Then specify the correct file number.
012	End of physical tape has been encountered. Note that this could occur when a UTIL command is issued for a new tape that is at a position before the two holes in the tape.	Issue a REWIND command, then retry the UTIL command. If error recurs, insert another tape and retry. If error again recurs, call the service personnel.
013	Specified device is not attached, or a UTIL MODE COM command was entered and communications is not available.	Probably specified wrong device address or specified invalid command. Specify correct device address after issuing GO END to end program.
014	A subdevice on a device address (such as tape) cannot be selected. This could mean that you have specified an invalid device address or device number, or that a hardware error, or programming error has occurred.	Ensure that the correct device address was specified, then retry the operation, or enter GO END to end the program.
050	Printer has detected end of forms.	Insert new forms. If error occurs on a command operation, reenter command. If error occurs 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.
051	Printer is not ready.	Turn on the printer. If the error occurs on a command operation, reenter the command. If the error occurs during program execution, enter GO x where x is the statement number displayed.
052- 059 ddd	Printer error.	Be certain that both the forms tractor and pressure roll paper feed are not engaged simultaneously. Retry the operation. If the error occurs again, call your service representative.
100	Syntax error in statement—statement not accepted.	D
101	Invalid arithmetic constant—line not accepted.	D
102	Incomplete character constant—statement not accepted.	D
103	Unbalanced parentheses in expression—statement not accepted.	D
104	Missing delimiter between character constants—statement not accepted.	D
105	Syntax error in STR function—statement not accepted.	D
106	Statement following LOAD0,KEYx was not a header statement—statement not accepted.	Reenter LOAD0,KEYx then enter new header statement.

Error	Meaning	Recovery
107	Invalid line number for key group header specified—statement not accepted.	Reenter LOAD0,KEYx then enter new header statement.
108	Syntax error in key group header—statement not accepted.	Reenter LOAD0,KEYx then enter new header statement.
109	Statement type not allowed in key group—line not accepted.	D
110	Invalid command syntax or attempt to re-mark file with CRC error from beyond CRC error—command not accepted.	D or issue REWIND and remark from a file before CRC error.
111	Invalid use of command: 1. No place to go on GO command. 2. No statements to RENUM. 3. No program to RUN. 4. No statement to LIST.	D
112	Unable to locate line number or next lower line number in LIST.	D
113	Function key not loaded or defined as null—request terminated.	Define function for the key before retrying.
114	Next line generated by AUTO will exceed 9999—AUTO is terminated.	
115	RENUM will cause a line >9989.	Change to a smaller increment or starting number and retry.
116	Referenced statement number in a BASIC statement not found during RENUM—command terminated. Line number of statement with invalid reference is displayed.	Correct the invalid reference in the program.
117	Device in command does not support requested function—command not accepted.	D
118	End of file reached before completion of SAVE—command terminated.	See <i>SAVE Command</i> , in Chapter 2.
119	Input line exceeds 64 bytes—line is truncated.	See <i>LOAD Command</i> , in Chapter 2.
120	Array dimensions are not within the range of 1 to 255.	D
121	File type of tape file does not match work area type or attempt was made to access a file beyond the last marked file.	Mount correct tape, then D.

Error	Meaning	Recovery
122	Data record greater than 324 characters encountered during LOADx, DATA—work area is cleared.	Use BASIC program with GET and PRINT statements to view the data.
150	Mark command to file already marked.	Enter GO only in positions 1 and 2 of the input line.
151	End of tape reel reached before completion of MARK—the last marked record and K-bytes are indicated.	
152	Insufficient work area for the last command or statement.	Users' option.
153	DIM statement contains array already defined either implicitly or explicitly.	A
154	Values for redimension are zero, negative, or greater than 255.	A
155	New dimensions for an array exceed space allocated by DIM or USE or by implicit allocation.	A
156	Dimensions incorrect for function used, and product array specified.	A
157	IDN function specified for a matrix that is not square.	A
158	Array referenced in a MAT statement not previously defined in a DIM or USE or by implicit definition.	A
159	Incorrect type of data from file or keyboard.	A
160	Data item in DATA statement not the same type (numeric or character) as the variable in the READ list.	A
161	Too few DATA statements for the READs issued.	A
162	Incorrect format for numeric or character data from keyboard or file.	A
163	OPEN specified to a file currently open.	A
	Command issued for a file open in the program.	B
164	OPEN specifies input for output device or output for input device.	A

Error	Meaning	Recovery
165	Multiple files opened to the same device.	A
166	Attempt to open an APL password-protected file.	A
167	The file type on the file is not a valid input file type.	A
168	Output statement specified to a file open for input, or input statement specified to a file open for output, or the file was not opened, or PUT was specified to a file for which a PRINT was already issued, or vice versa.	A
169	I/O started during evaluation of function referred to in an I/O statement.	A
170	PRINT USING specifies a data item and there are no conversion specifications in the image statement.	A
171	Statement number displayed references a statement not in the program.	A
172	FNEND statement not found for a DEF statement.	A
173	No matching NEXT statement for a FOR, or the control variable did not match.	A
174	NEXT statement executed before the matching FOR statement.	A
175	RETURN statement without a value encountered when no GOSUB is active.	A
176	RETURN statement with value encountered when no user function is active.	A
177	USE statement encountered after the first data reference.	A
178	Attempt to reference beyond the eighteenth character in a STR (string) function.	C
200	Data overflow has occurred and a value of $\pm.9999999999999999E99$ was assumed.	B
201	Data underflow has occurred and a value of 0 has been assumed.	B
220	Statement too complex, or too many nested function calls.	C

Error	Meaning	Recovery
221	Work area filled trying to allocate space for a variable.	C
222	Insufficient work area to calculate a determinant.	C
223	Too many subscripts used for array reference.	C
224	Subscript outside range of dimensions—too large or negative.	C
225	Nonsquare matrix for DET function, or undefined matrix.	C
226	Reference to an undefined user function.	C
227	Incorrect number of parameters specified on a user-defined function reference.	C
228	Argument for LOG is zero or negative.	C
229	Argument for square root is negative.	C
230	Argument for arc sine or arc cosine is greater than 1 or less than -1.	C
231	Power function; attempt to raise 0 to 0 power or 0 to negative power.	C
232	Power function; attempt to raise a negative number to a noninteger power.	C

ENVIRONMENT

The 5100 Portable Computer and associated units are designed for these environments:

Operating Environment		Nonoperating Environment
Dry bulb temperature	60°-90° F (15°-32° C)	50°-105° F (10°-43° C)
Relative humidity	8%-80%	8%-80%
Maximum wet bulb temperature	73° F (23° C)	80° F (27° C)

You should not expose the machine to extreme temperatures for an extended time. For example, do not store the machine in the car trunk when the weather is very warm or very cold. If you must expose the machine to extreme temperatures, the machine should be acclimated to the operating environment before turning it on:

- If the machine was exposed to heat, allow it to cool enough so that you can place your hand on the surface without discomfort before turning it on.
- If the machine was exposed to cold and there is no frost or moisture on the external surfaces or visible on the internal parts, the machine can be turned on.
- If the machine was exposed to cold and there is frost or moisture on the external surfaces or visible on the internal parts, acclimate the machine for 8 hours after the frost and moisture disappears. This is to make sure all internal moisture evaporates before turning the machine on; otherwise, the machine may be damaged.

5100 SETUP PROCEDURE

After you have placed the 5100 where you intend to use it, make sure the red POWER ON/OFF switch (located on the front panel) is in the OFF position. Plug the power line into a *grounded* electrical outlet.

Note: For proper operation, the 5100 must be plugged into a grounded outlet.

Set the POWER switch to ON, and be sure that the fan is operating:

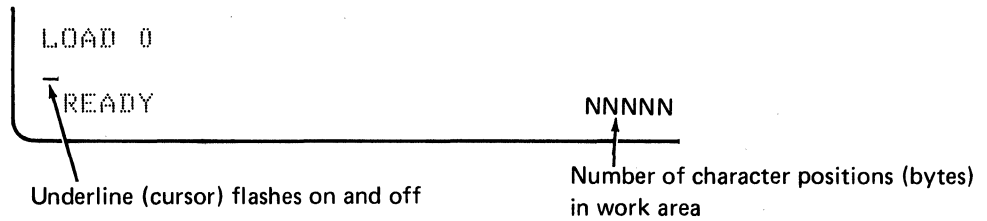
- If your machine location is not too noisy, you should hear the fan motor operating.
- If you are not sure, hold a light piece of paper near the air intake on the back of the machine. The loose end of the paper should be pulled toward the machine.

If the fan does not appear to be operating, check your power outlet. If it is OK, set the POWER switch to OFF and call for service. Do not continue with these instructions.

If the fan is operating, wait for about 20 seconds and your 5100 will be ready for operation.

BASIC Checkout Procedure

- 1. After power has been on 20 seconds, the display screen should show:





If the display screen does not show the above information, check the following top panel switches:

- Turn the BRIGHTNESS control to get the best character definition.
- Set the DISPLAY REGISTERS switch to the NORMAL position.
- Set the L32 64 R32 switch to the center (64) position.
- Set BASIC/APL switch (on the combined machines only) to the BASIC position.
- If the information displayed is not as shown above, press the RESTART switch. This recycles a portion of the power-on sequence. If the information displayed is still not as shown above (after the 20-second delay), call for service.

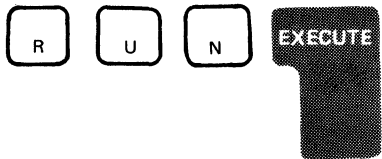
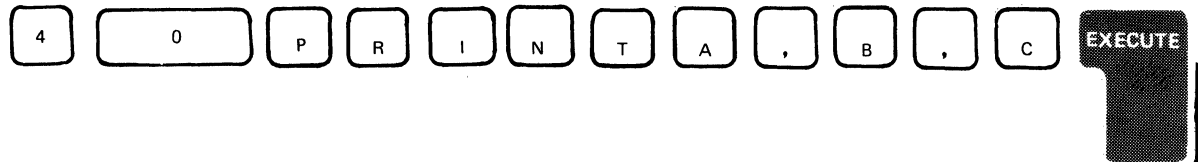
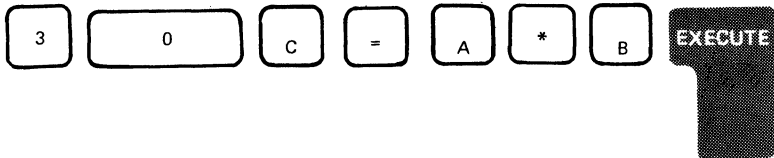
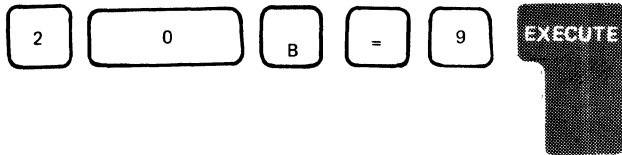
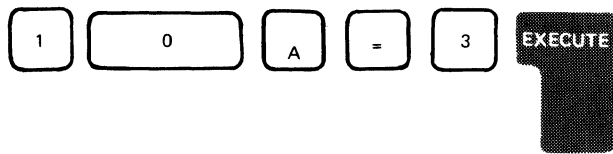
- 2. If the display screen does not show the correct results in the remaining steps of this procedure, press RESTART once, go back to step 1, and try again. If the correct result is still not shown, call for service.

Enter the data shown by the drawings below. The data will be displayed as the keys are pressed.

If you make a keying error, you can press the backspace key  (above EXECUTE) to backspace the cursor and then press the correct key. If you pressed EXECUTE and the line was incorrect, press  (next to ATTN)

once, and reenter the line. If the error causes the screen to flash, press ATTN before reentering the line.

Press the following keys in sequence line by line:



- 3. Below the lines of the test program that you just entered, the answer of 27 will appear beside the constants 3 and 9 (the program multiplies them).

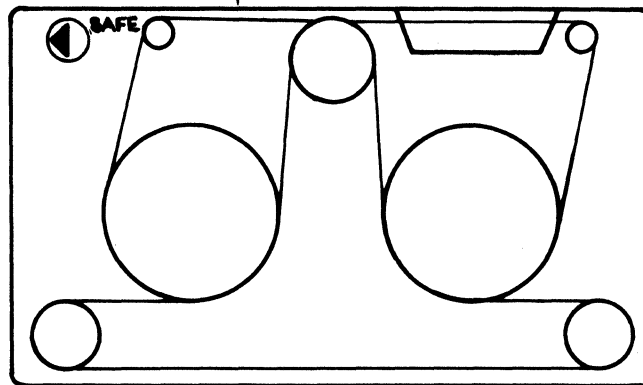
```
LOAD 0
10A=3
20B=9
30C=A*B
40PRINTA,B,C
RUN
3          9          27
-
READY
```

- 4. If you checked out the tape operation under the APL checkout procedures, insert the tape cartridge into the 5100 and go to step 6.

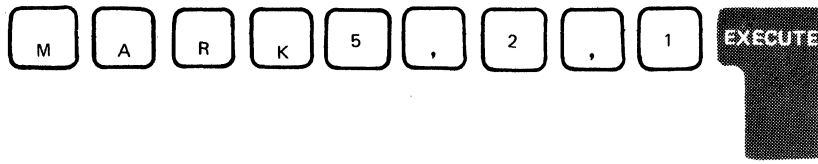
Remove an unused or scratch tape cartridge from its package. Check that the arrow is pointing away from the word **SAFE** as shown in the illustration. Insert a coin or screwdriver into the slot if you must turn the triangular arrow away from the word **SAFE**.

Note: DO NOT USE any of the prerecorded tape cartridges that were shipped with your machine.

This edge goes into the machine first.

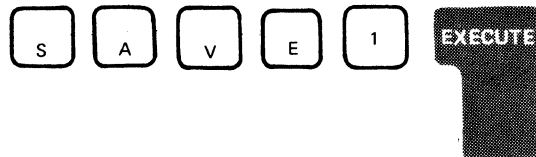


- 5A. Insert the tape cartridge into the 5100 (metal bottom down), and press it in until it seats firmly. Then press the following keys:

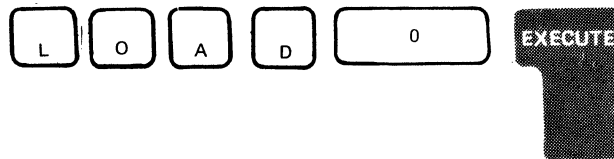


- 5B. The previous step initialized the tape to hold information. When the display screen again shows READY in the lower left corner, go to step 6. (If an error code of 150 is displayed, the tape is already marked. To re-mark the tape, press the following keys: **ATTN** G O **ATTN** EXECUTE When READY is shown, continue.)

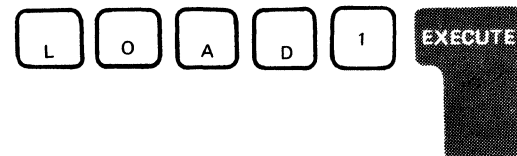
6. Press the following keys:



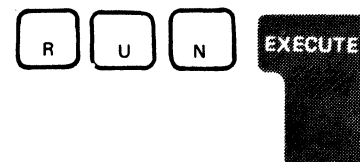
7. The last step wrote the program onto tape, but the program is still recorded in the read/write storage area. To prove it can be read from tape, the program must be erased from storage. To do this, press the following keys:



8. To read the program from tape into the 5100, press the following keys:



9. The tape will move and when READY is again displayed, press these keys:



The display screen will again show:

```
  RUN
   3           9           27
  -
  READY
```

This completes the BASIC checkout procedure.

- 10. Check to see that you received all the items on the Contents Checklist.

If any of the preceding items are missing, notify your IBM marketing representative.

If the words above the top row of numeric keys are labeled on the left with:

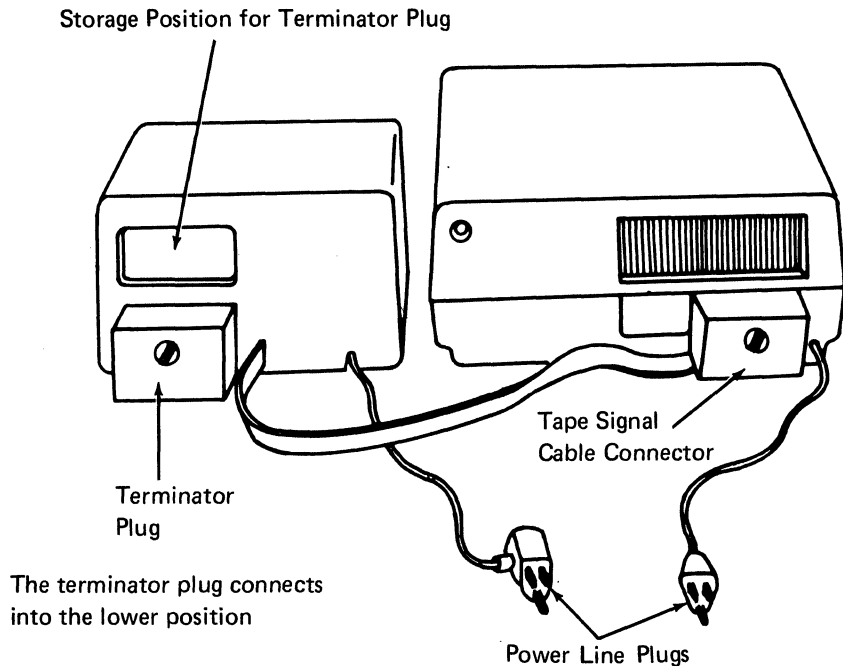
- APL , you have an APL machine without the communications feature.
- BASIC , you have a BASIC machine without the communications feature.
- BASIC }
APL } , you have a combined machine.
- COMM }
BASIC }
APL } , you have a combined machine with the communications feature.
- COMM }
APL } , you have an APL machine with the communications feature.
- COMM }
BASIC } , you have a BASIC machine with the communications feature.

If you have not checked out APL on a combined machine, set the BASIC/APL switch to the APL position, press RESTART, and go to the APL checkout procedure in Appendix A of the *IBM 5100 APL Reference Manual, SA21-9213*. If you already did the APL checkout procedures, continue with step 11.

- 11. If the auxiliary tape unit is to be installed, unpack the tape unit and proceed to the *Auxiliary Tape Unit Setup Procedure* which follows. After installing the auxiliary tape unit, return to step 12.
- 12. If the printer is to be installed, unpack the printer and proceed to the *Printer Setup Procedure* which comes later in this appendix. After installing the printer, return to step 13.
- 13. If your 5100 is equipped with any other feature, use the manual supplied with that feature to set up and check out the feature, then return to step 14 in this manual.
- 14. When the preceding devices or features are installed, or if none are, begin reading the *IBM 5100 BASIC Introduction* to learn how to operate your 5100.

IBM 5106 Auxiliary Tape Unit Setup Procedure

- 1. Set the 5100 and auxiliary tape unit power switches to OFF.
- 2. Remove the shipping tape from the signal cable (flat cable) and connect the signal cable into the back of the 5100. Make sure the connector fits squarely. Turn the knob in a clockwise direction until the connectors fit together firmly.

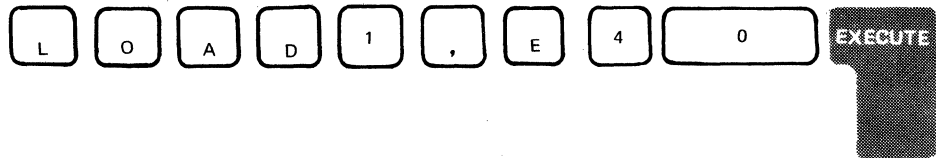


- 3. Check that the terminator plug is in place on the rear panel as shown in the preceding diagram.
 - 4. Remove the shipping tape from the power line and plug the power line into a grounded electrical outlet.
 - 5. Set the auxiliary tape unit POWER switch to ON, and be sure that the fan is operating.
 - a. If your location is not too noisy, you should hear the fan motor operating.
 - b. If you are not sure, hold a light piece of paper near the air intake on the left side of the tape unit. The loose end of the paper should be pulled toward the tape unit.
- If the fan does not appear to be operating, check your power outlet. If it is OK, set the POWER switch to OFF and call for service. Do not continue with these instructions.
- 6. Set the 5100 POWER switch to ON and continue to the checkout procedure.

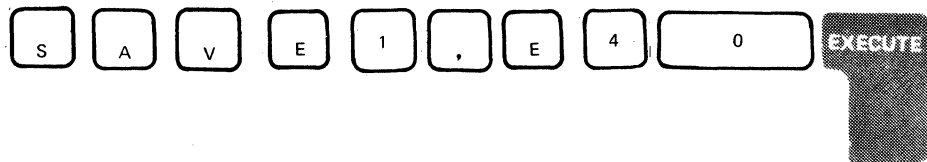
Auxiliary Tape Unit Checkout Procedure

Note: The following steps assume you are using the same cartridge that you used to check the 5100. If you are not, write any program onto the cartridge in the auxiliary tape unit and read it back.

- 1. Insert a tape cartridge into the auxiliary tape unit after checking that the arrow is pointing away from the word **SAFE**.
- 2. Press the following keys to read in the program that was stored on tape during the 5100 checkout procedure:



- 3. The tape will move and when **READY** is again displayed, press these keys:



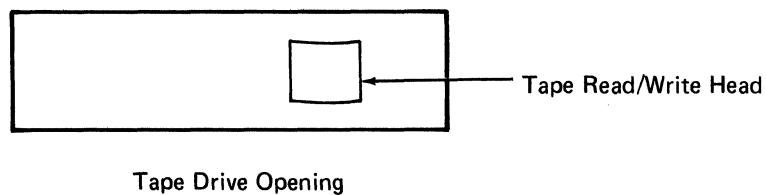
- 4. **READY** is again displayed to verify that the program was written back to tape and was checked by the 5100.

This completes the checkout procedure for the auxiliary tape unit.

Return to step 12 of the 5100 checkout procedure.

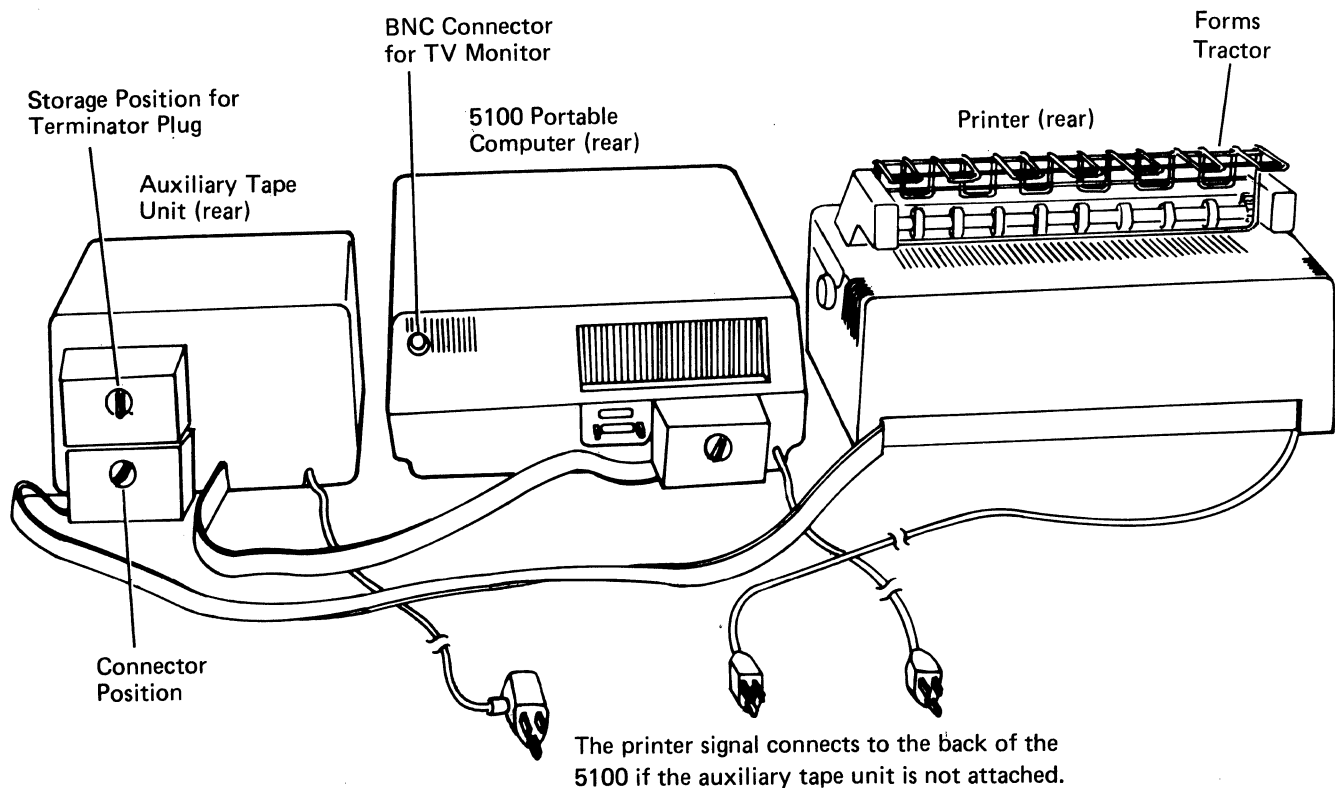
Cleaning the Tape Head

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 alcohol, to clean the tape oxide from the tape head. Then, wipe the tape head dry.

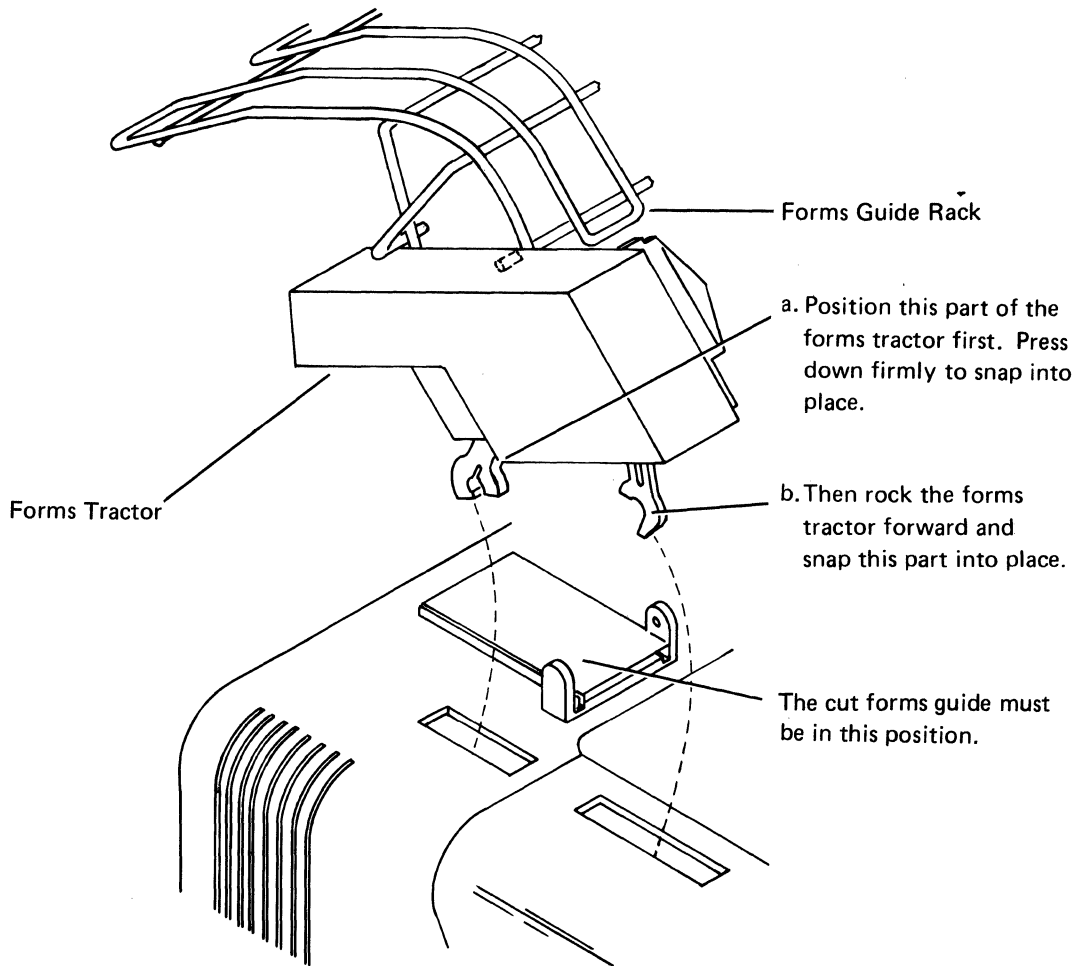


IBM 5103 Printer Setup Procedures

- 1. Set the 5100, 5103, and 5106 (if attached) POWER switches to OFF.
- 2. If you have an auxiliary tape unit, remove the terminator plug from the bottom position and insert it into the top position (storage position).
- 3. Remove the shipping tape from the printer signal cable (flat cable) and connect the signal cable to the back of the auxiliary tape unit, if it is attached, or to the back of the 5100. Make sure the connectors fit squarely. Turn the knob in a clockwise direction until the connectors fit together firmly.
- 4. Remove the shipping tape from the printer power line and plug the power line into the back of the auxiliary tape power plug or into a grounded electrical outlet.



- 5. Unpack the forms tractor and set it in place on top of the printer as shown in the drawing.



- 6. Insert paper in the printer. Use the printer information in this manual if you need help in inserting the paper (see Appendix D).
- 7. Set both the printer and 5100 POWER switches to ON and continue on to the checkout procedure.

Printer Checkout Procedure

Press several alphameric keys to display some information. Then, hold down the CMD key



word strip. The printer will provide a copy of the information on the display screen.

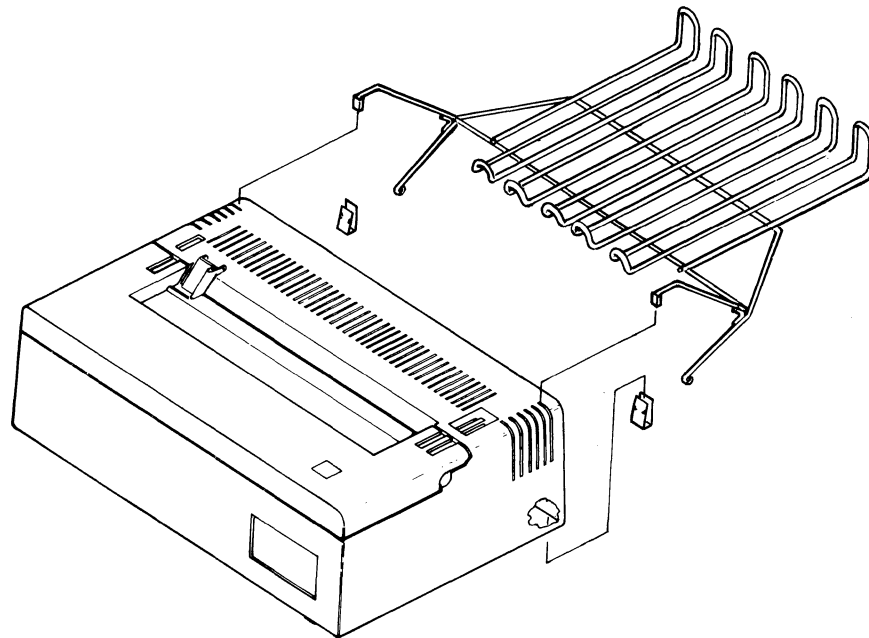
Return to step 13 of the 5100 checkout procedures.

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.



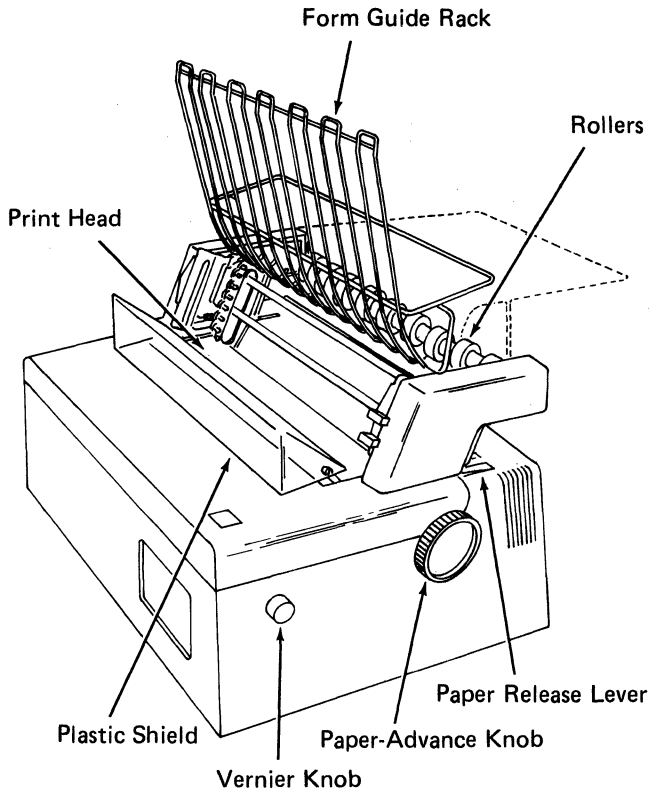


The 5103 Printer has the following characteristics:

- Bi-directional 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

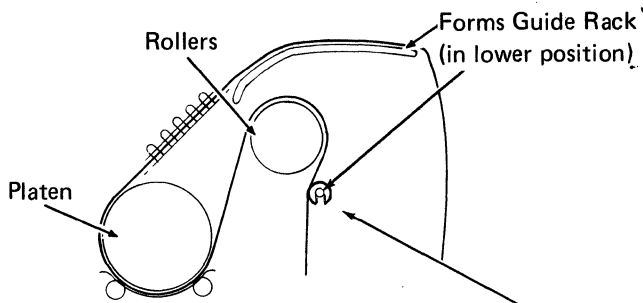


1. Pivot the plastic shield forward.
2. Push the print head to the extreme left position.
3. For singlepart forms, pivot the form guide rack up and forward to a vertical position. For multipart forms, leave the forms guide rack in the horizontal position.

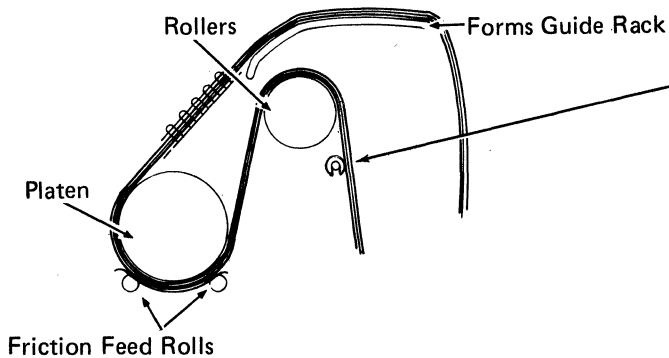
The diagrams below and to the left show the proper forms path for singlepart and multipart forms.

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.

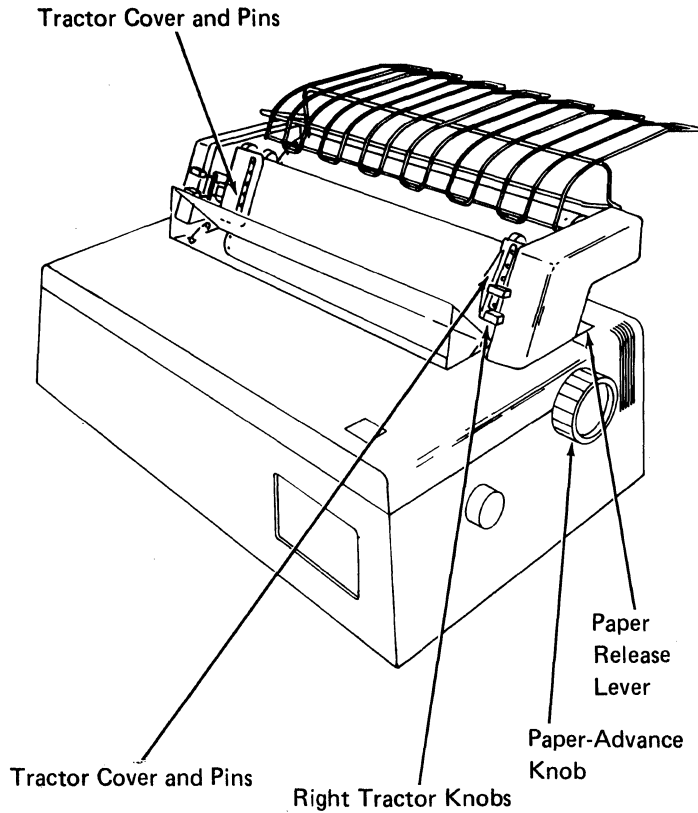
Forms Path for Singlepart Forms



Forms Path for Multipart Forms



Position these guides at the edge of the paper.



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. Close the plastic shield.
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. If you are installing the printer, return to step 7 of the *Printer Installation Procedure*.

CAUTION

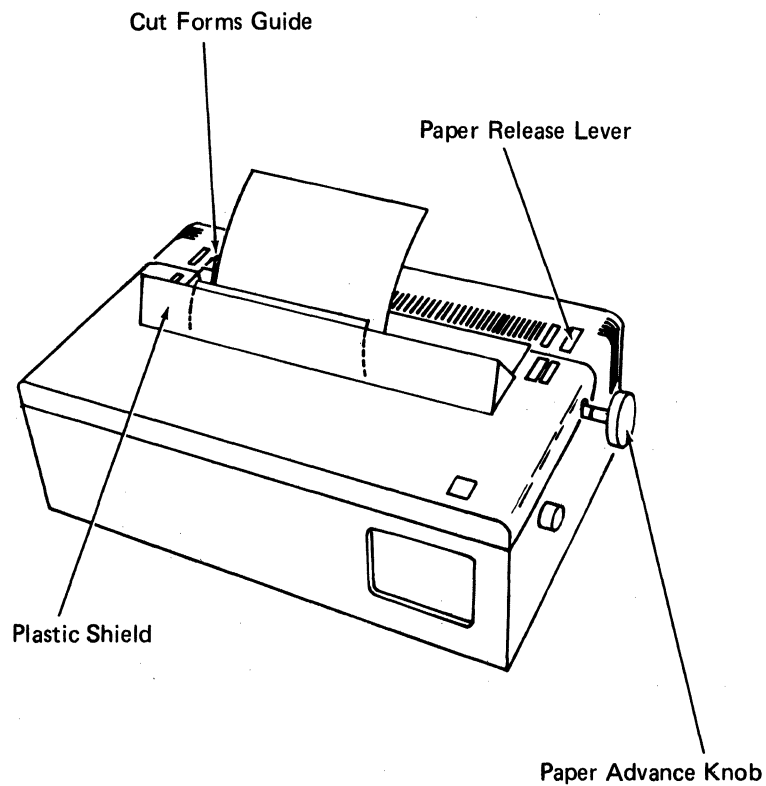
The switch that senses end of forms is deactivated when the friction feed rolls are engaged. Thus, the print wires could hit the base platen if no forms are in the printer.

Cut Forms

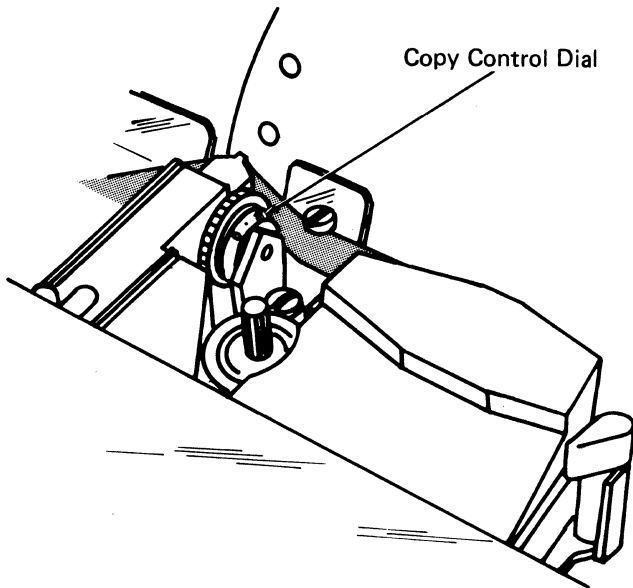
1. Remove the forms tractor by tilting it back and lifting it off.
2. Move the cut forms guide forward.
3. Pivot the plastic shield forward.
4. Push the print head to the extreme left position.
5. Push the paper release lever to the rear to activate the friction feed rolls.
6. Place the form in position 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. Close the plastic shield.

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.

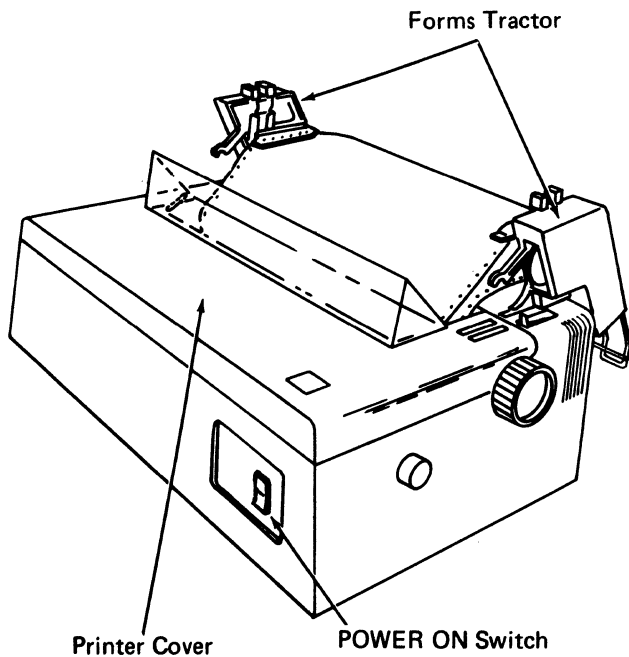


How to Adjust the Copy Control Dial For Forms Thickness

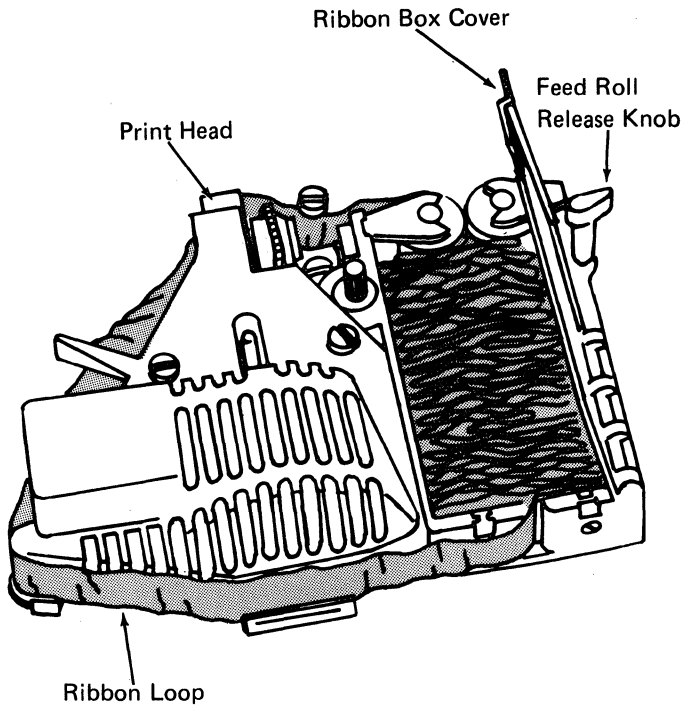


1. If you are using singlepart forms, set the copy control dial on 0.
2. If you are using multipart forms and the last sheet is not legible, rotate the copy control dial toward 0 one click at a time to obtain the legibility you desire.
3. If you are using multipart forms and the ribbon is smudging the first sheet, rotate the copy control dial toward 8 one click at a time until smudging stops.

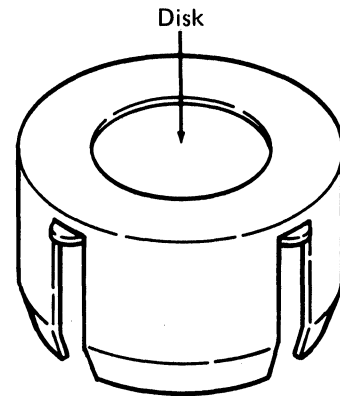
How to Replace a Ribbon (Part Number 1136653)



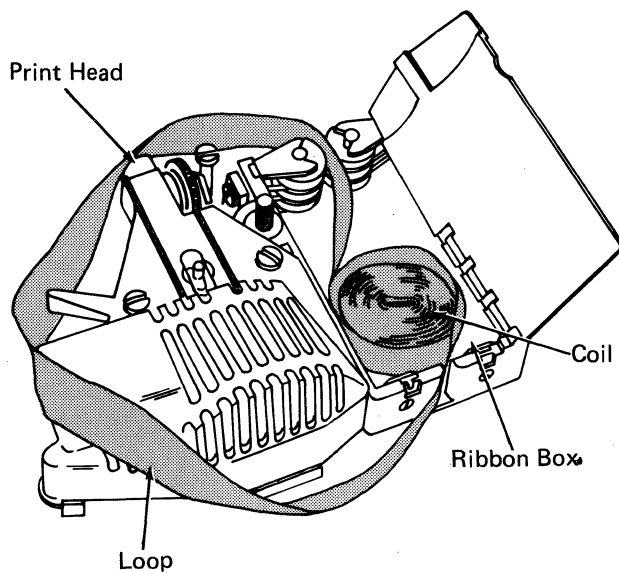
1. Turn off power to the printer.
2. Tilt the forms tractor back by lifting both sides at the front.
3. Open the printer cover by sliding it forward.



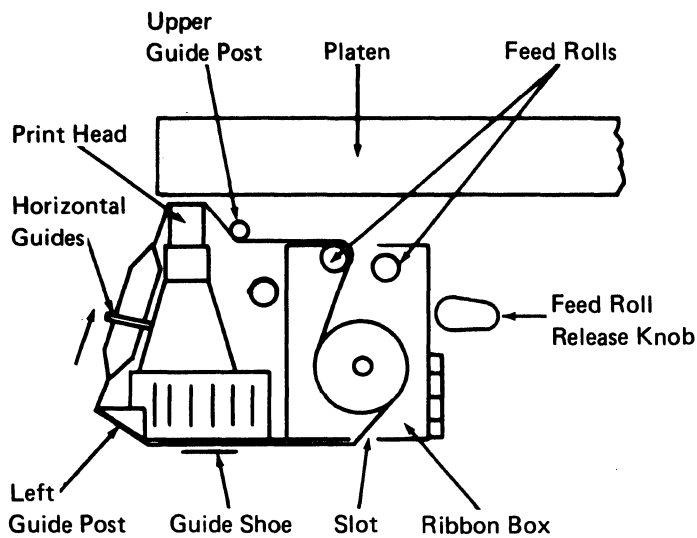
4. Be sure that the print head is to the extreme left.
5. Turn the feed roll release knob counterclockwise until it points to the right.
6. Open the ribbon box cover.
7. Put on the gloves supplied with the new ribbon.
8. Remove the old ribbon from the guides being careful to disengage it from the clip 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.

Appendix E. Hexadecimal Representations

These are the characters that the 5100 can display along with their hexadecimal representations. The second 128 characters are the same as the first 128 with the addition of underscores.

X'00'	=		X'2B'	=	,	X'56'	=)	X'81'	=	A	X'AC'	=	.	X'D7'	=	:
X'01'	=	A	X'2C'	=	.	X'57'	=	;	X'82'	=	B	X'AD'	=	0	X'D8'	=	;
X'02'	=	B	X'2D'	=	α	X'58'	=	:	X'83'	=	C	X'AE'	=	1	X'D9'	=	0
X'03'	=	C	X'2E'	=	ι	X'59'	=	φ	X'84'	=	D	X'AF'	=	2	X'DA'	=	0
X'04'	=	D	X'2F'	=	η	X'5A'	=	θ	X'85'	=	E	X'AO'	=	3	X'DB'	=	0
X'05'	=	E	X'30'	=	λ	X'5B'	=	ϑ	X'86'	=	F	X'AP'	=	4	X'DC'	=	0
X'06'	=	F	X'31'	=	ε	X'5C'	=	θ	X'87'	=	G	X'BA'	=	5	X'DD'	=	0
X'07'	=	G	X'32'	=	ι	X'5D'	=	χ	X'88'	=	H	X'BB'	=	6	X'DE'	=	0
X'08'	=	H	X'33'	=	∇	X'5E'	=	λ	X'89'	=	I	X'BC'	=	7	X'DF'	=	0
X'09'	=	I	X'34'	=	Δ	X'5F'	=	∏	X'8A'	=	J	X'BD'	=	8	X'E0'	=	0
X'0A'	=	J	X'35'	=	ι	X'60'	=	!	X'8B'	=	K	X'BE'	=	9	X'E1'	=	0
X'0B'	=	K	X'36'	=	°	X'61'	=	∞	X'8C'	=	L	X'BF'	=	0	X'E2'	=	0
X'0C'	=	L	X'37'	=	'	X'62'	=	∞	X'8D'	=	M	X'BO'	=	0	X'E3'	=	0
X'0D'	=	M	X'38'	=	∏	X'63'	=	θ	X'8E'	=	N	X'BP'	=	0	X'E4'	=	0
X'0E'	=	N	X'39'	=	ι	X'64'	=	∅	X'8F'	=	O	X'BA'	=	0	X'E5'	=	0
X'0F'	=	O	X'3A'	=	τ	X'65'	=	∞	X'90'	=	P	X'BB'	=	0	X'E6'	=	0
X'10'	=	P	X'3B'	=	o	X'66'	=	∞	X'91'	=	Q	X'BC'	=	0	X'E7'	=	0
X'11'	=	Q	X'3C'	=	*	X'67'	=	∞	X'92'	=	R	X'BD'	=	0	X'E8'	=	0
X'12'	=	R	X'3D'	=	?	X'68'	=	∞	X'93'	=	S	X'BE'	=	0	X'E9'	=	0
X'13'	=	S	X'3E'	=	p	X'69'	=	∞	X'94'	=	T	X'BF'	=	0	X'EA'	=	0
X'14'	=	T	X'3F'	=	Γ	X'6A'	=	∞	X'95'	=	U	X'CO'	=	0	X'EB'	=	0
X'15'	=	U	X'40'	=	∞	X'6B'	=	∞	X'96'	=	V	X'C1'	=	0	X'EC'	=	0
X'16'	=	V	X'41'	=	↓	X'6C'	=	∞	X'97'	=	W	X'C2'	=	0	X'ED'	=	0
X'17'	=	W	X'42'	=	u	X'6D'	=	∞	X'98'	=	X	X'C3'	=	0	X'EE'	=	0
X'18'	=	X	X'43'	=	∅	X'6E'	=	∞	X'99'	=	Y	X'C4'	=	0	X'EF'	=	0
X'19'	=	Y	X'44'	=	∅	X'6F'	=	&	X'9A'	=	Z	X'C5'	=	0	X'F0'	=	0
X'1A'	=	Z	X'45'	=	↑	X'70'	=	@	X'9B'	=	0	X'C6'	=	0	X'F1'	=	0
X'1B'	=	0	X'46'	=	c	X'71'	=	#	X'9C'	=	1	X'C7'	=	0	X'F2'	=	0
X'1C'	=	1	X'47'	=	∧	X'72'	=	\$	X'9D'	=	2	X'C8'	=	0	X'F3'	=	0
X'1D'	=	2	X'48'	=	'	X'73'	=	%	X'9E'	=	3	X'C9'	=	0	X'F4'	=	0
X'1E'	=	3	X'49'	=	'	X'74'	=	A	X'9F'	=	4	X'CA'	=	0	X'F5'	=	0
X'1F'	=	4	X'4A'	=	<	X'75'	=	∅	X'A0'	=	5	X'CB'	=	0	X'F6'	=	0
X'20'	=	5	X'4B'	=	∞	X'76'	=	ö	X'A1'	=	6	X'CC'	=	0	X'F7'	=	0
X'21'	=	6	X'4C'	=	∞	X'77'	=	ü	X'A2'	=	7	X'CD'	=	0	X'F8'	=	0
X'22'	=	7	X'4D'	=	∞	X'78'	=	∞	X'A3'	=	8	X'CE'	=	0	X'F9'	=	0
X'23'	=	8	X'4E'	=	v	X'79'	=	∞	X'A4'	=	9	X'CF'	=	0	X'FA'	=	0
X'24'	=	9	X'4F'	=	∞	X'7A'	=	∞	X'A5'	=	∞	X'D0'	=	0	X'FB'	=	0
X'25'	=	/	X'50'	=	<	X'7B'	=	∞	X'A6'	=	∞	X'D1'	=	0	X'FC'	=	0
X'26'	=	+	X'51'	=	\	X'7C'	=	∞	X'A7'	=	∞	X'D2'	=	0	X'FD'	=	0
X'27'	=	x	X'52'	=	-	X'7D'	=	∞	X'A8'	=	∞	X'D3'	=	0	X'FE'	=	0
X'28'	=	†	X'53'	=	+	X'7E'	=	∞	X'A9'	=	∞	X'D4'	=	0	X'FF'	=	0
X'29'	=	[X'54'	=	→	X'7F'	=	∞	X'AA'	=	∞	X'D5'	=	0			
X'2A'	=]	X'55'	=	(X'80'	=	∞	X'AB'	=	∞	X'D6'	=	0			

- ABS (x) absolute value of x 50
 ACS (x) arc cosine of x 50
 addition 51, 118
 adjusting forms thickness 173
 alphabetic characters 40, 143
 AND symbol (&) 41, 78
 APL continued file 38
 APL internal data 25, 28, 38
 APL SAVE file 38
 APL symbols 3, 6
 arithmetic arrays 48, 114
 arithmetic constants 44
 arithmetic data 41, 99, 105
 arithmetic data formats 42
 arithmetic format selection 43
 arithmetic hierarchy 53
 arithmetic operators 51
 arithmetic variables 44, 112
 array element values 67, 114
 array elements 45, 114
 array expressions 46, 51, 56
 array size 67, 114
 arrays 45, 114
 ASN (x) arc sine of x 50
 assigning array elements to a file 140
 assigning values to array elements 62, 114
 ATN (x) arc tangent of x 50
 ATTN key 4, 18
 AUTO command 10
 automatic statement numbering 10, 18
 auxiliary tape unit 32, 163
- backspace key 5, 159
 BASIC character set 40, 147
 BASIC file type 15, 36
 BASIC keys file 38
 BASIC source file 25, 28, 38
 BASIC statement keywords 3
 BASIC statements 57
 BASIC work area file 38
 BASIC/APL switch 5, 158
 blank characters 9, 41, 99, 132
 BRIGHTNESS switch 5, 158
 byte storage considerations 145
- capacity, storage 7
 centimeters per inch (&INCM) 44
- CHAIN statement 60, 66, 112, 143
 changing execution mode 11
 character arrays 49, 132
 character constants 19, 45, 62, 81, 132
 character data 45, 105
 character expressions 54, 89
 character sequence 147
 character variables 45, 62, 112
 checkout procedures 158
 cleaning the tape head 164
 clearing the work area 15, 37, 144, 161
 CLOSE statement 61, 130
 CMD function keys 18
 CMD key 4
 command keywords 8
 common storage area 112
 communications 38
 comparisons 55, 78
 compatibility to IBM 370/VS BASIC 143
 computed GOSUB 75
 computed GOTO 77
 control variable 71
 copy display 4
 copy IMFs to a cartridge 25, 27
 COS (x) cosine of x radians 50
 COT (x) cotangent of x radians 50
 CSC (x) cosecant of x radians 50
 cursor 2, 158
 customer support cartridge 25
 cut forms 171
- data constants 40
 data exchange file 25, 28, 38
 DATA file type 15, 36
 data security 144
 DATA statement 62, 105, 109, 141
 data table pointer 62, 105, 141
 debugging 33
 decimal point 29
 declaring arrays 47, 113
 DEF statement 64, 88
 DEG (x) degrees in x radians 50
 DEL function 7
 delete function 5
 deleting statement lines 7
 DET (x) determinant of an arithmetic array 50, 126
 device address 15, 32, 86
 diagnostic file 38
 DIM statement 67
 DIR 38
 directory, tape 37

display EC levels of modules 25, 28
DISPLAY REGISTERS/NORMAL switch 6, 158
display screen 2
displaying array contents 134
displaying work area contents 13
division 51

E-format 92, 99
E-format numbers 43
editing 6
editing function key groups 20
end of file 35, 130, 144
end of job 28
END statement 69, 111
entering IMFs from the keyboard 28
environment 157
EOF line number 73
erasing a tape 144
error correction 6, 149, 159
error messages 149
executable statements 57
EXECUTE key 4
execution modes 33
EXP (x) natural exponential of x 50
exponentiation 51
expressions 51

F-format numbers 42, 92, 99
file numbers 15, 86
file reference 61, 73, 108, 134, 143
file types 15, 28, 38
fixed-point format 42
flashing question mark 81, 132
floating-point format 43
FLP file reference 98, 134, 137, 143
FNEND statement 64, 70
FOR statement 71
format specifications 98, 137
forward space key 5
full print zones 90
function keys 18, 30, 143

general exchange file 25, 28, 38
GET statement 18, 73, 103
GO command 11, 21
GO END command 11, 149
GOSUB statement 19, 75, 78
GOTO statement 66, 77

HCS (x) hyperbolic cosine of x 50
hexadecimal constants 84, 143
hexadecimal representations 176
HOLD key 4, 37
HSN (x) hyperbolic sine of x 50
HTN hyperbolic tangent of x 50

I-format 92, 99
I-format numbers 42
IF statement 78
IMAGE statement 80, 137
IMFs 25, 38
IN PROCESS indicator 6
indicators 6
initializing tape files 21, 161
initializing variables 44
input files 11, 61, 86, 103, 108, 130, 140
INPUT statement 81
insert function 5
inserting printer forms 170
installing the 5103 printer stacker 167
INT (x) integral part of x 50
integer format 42
internal constants 44
internal data table 62
internal requirements 7

keyboard 3
keyboard generated data files 18
KEYS file type 15
KEYx file type 13, 15, 20, 30
kilograms per pound (&LBKG) 44

LET statement 83
LGT (x) logarithm of x to base 10 50
line number 10, 13, 23, 57, 88
LIST command 13
listing tape contents 37
liters per gallon (&GALI) 44
LOAD command 15, 144, 161
load IMFs in work area 25, 27
LOG (x) logarithm of x to base e 50
loops 71
LTW (x) logarithm of x to base 2 50
L32 64 R32 switch 6

magnitude 42
MARK command 21
marked or unused file 37
MAT (addition and subtraction) assignment 118
MAT (identify function) assignment 124
MAT (inverse function) assignment 126
MAT (matrix multiplication) assignment 120
MAT (scalar) assignment 114
MAT (scalar multiplication) assignment 122
MAT (simple) assignment 116
MAT (transpose function) assignment 128
MAT assignment statements 113
MAT GET statement 130
MAT INPUT statement 132
MAT PRINT statement 134
MAT PRINT USING statement 137
MAT PUT statement 140
MAT READ statement 62, 141

matrix operations 113
MERGE command 23
merging tape and work area contents 23
MODE COM 37
multiline user-defined functions 64
multiplication 51, 120, 122

naming conventions, summary of 49
natural log (e) 44
negative file numbers 87
nested loops 72
NEXT statement 71
nonexecutable statements 57, 64
normal program execution 33
NULL function keys 18
numeric arrays 113
numeric characters 40
numeric keys 3

one-dimensional arrays 67
OPEN statement 86, 130, 140, 143
operating environment 157
OR symbol (|) 41, 78
output files 61, 86, 103, 130, 140
overview 1

packed print zones 90, 135
parentheses 53
PATCH command 25
PATCH file 38
PAUSE statement 88
PI (π) 44
positive/negative operators 52
POWER ON/OFF switch 5, 144, 157
precision 42, 143
print array contents 134
print line buffer 96, 135
print or display line delimiters 90, 93, 134, 138
print or display output formats 91, 135
PRINT statement 89
PRINT USING statement 98
print zones 90, 134
printer 165, 169
printer stacker 167
printing work area contents 13
PROCESS CHECK indicator 6
PUT statement 18, 103

RAD (x) radians in x degrees 50
random numbers 50
RD=command 29
re-marking a tape file 21, 161
READ statement 62, 105, 109
recovery from errors 149
redimensioning arrays 48, 114
relational expressions 51, 55, 78

REM function keys 19
REM statement 107
RENUM command 30
renumbering statements 23, 30, 88
replacing printer ribbon 173
RESET statement 108
RESTART switch 5, 144, 158
RESTORE statement 62, 105, 109, 141
resume processing (GO command) 11
RETURN statement 64, 75, 110
REVERSE DISPLAY switch 5
REWIND command 21, 32
ribbon part number 173
RND (x) random number between 0 and 1 50
rounding 11, 29, 33
RUN command 33

SAVE command 18, 35, 161
scalar operations 56, 114
scroll down key 5, 13
scroll up key 5, 13, 159
SEC (x) secant of x radians 50
setup procedures 157
SGN (x) sign of x (-1, 0, or +1) 50
shift keys 4
simple GOSUB 75
simple GOTO 77
SIN (x) sine of x radians 50
single line user-defined functions 64
SOURCE file type 15
spacing of printed or displayed values 90, 134
special characters 41, 144
SQR (x) square root of x 50
square root of 2 (&SQR2) 44
statement lines 30, 57, 143
statue line 2
step program execution 33
STOP statement 66, 69, 111
storage available 2, 7
storage capacity 7, 21, 37
storage considerations 145
STR (substring) function 50, 56
substring function 56
subtraction 51, 118
switches 5
syntax 9
syntax error 9
system commands 8
system functions 49

TAB function 96, 143
TAN (x) tangent of x radians 50
tape cartridge 37, 144, 160
tape cartridge removal 87
tape copy program 25, 28
tape directory 37
tape head cleaning 164
tape recovery program 25, 28
tape storage dump 38
THEN operator 78

trace program execution 33
two-dimensional arrays 67
TXT function keys 19

underscore character 41
USE statement 60, 112, 143
user file identification 35, 37, 86
user-defined arithmetic functions 64
UTIL command 37

variables 40

5100 setup procedure 157
5103 printer 169
5103 printer setup procedures 165
5103 printer stacker installation 167
5106 auxiliary tape unit setup procedure 163

READER'S COMMENT FORM

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

Note: All comments and suggestions become the property of IBM.

● No postage necessary if mailed in the U.S.A.

Check if reply is requested.

Name _____

Address _____

Cut Along Line

Fold

Fold

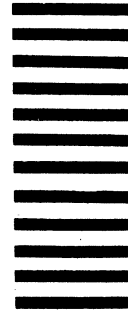
FIRST CLASS
PERMIT NO. 40
ARMONK, N. Y.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
General Systems Division
Development Laboratory
Publications, Dept. 245
Rochester, Minnesota 55901



IBM 5100 BASIC Reference Manual Printed in USA SA21-9217-3

Fold

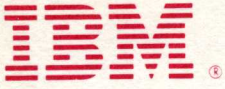
Fold



International Business Machines Corporation

General Systems Division
5775D Glenridge Drive N. E.
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)



International Business Machines Corporation

General Systems Division
5775D Glenridge Drive N. E.
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 5100 BASIC Reference Manual Printed in USA SA21-9217-3

SA21-9217-3

102727121