

PATENT SPECIFICATION

(11) 1 444 145

1 444 145

- (21) Application No. 3348/76 (22) Filed 6 July 1973
(62) Divided out of No. 1 444 141
(31) Convention Application No. 269 899
(32) Filed 7 July 1972 in
(33) United States of America (US)
(44) Complete Specification published 28 July 1976
(51) INT CL² G06F 3/02
(52) Index at acceptance

G4A 10EX 15A2 16G 17B4 17P 3W10W2 3W2 5A 5B 6G
6H 6MX 8C 9C 9X KB



(54) ELECTRONIC DATA PROCESSING APPARATUS

(71) We, HEWLETT-PACKARD COMPANY, a corporation organized and existing under the laws of California, United States of America, of 1501 Page Mill Road, Palo Alto, California, United States of America, do hereby declare the invention, for which we pray that a patent may be granted to us, and the method by which it is to be performed, to be particularly described in and by the following statement:—

This invention is concerned with improvements in or relating to electronic data processing apparatus and is especially concerned with electronic calculators of the type disclosed in the complete specification of our co-pending United Kingdom patent application No. 32269/73 (Serial No. 1,444,141). The present invention is also related to the inventions disclosed in our co-pending United Kingdom patent application Nos. 3345/76 to 3347/76 (Serial Nos. 1,444,142 to 1,444,144) and 3349/76 to 3352/76 (Serial Nos. 1,444,146, 1,444,147, 1,444,148 and 1,444,149).

In the complete specification of our copending U.K. patent application No. 32269/73 (Serial No. 1,444,141), there is disclosed an electronic calculator comprising:

keyboard input means including a plurality of alphameric keys for entering one or more BASIC language statements into the calculator, an execute control key for initiating execution of a BASIC language statement entered into the calculator, and a store control key for initiating storage of a BASIC language statement entered into the calculator;

memory means, coupled to said keyboard input means, for storing a program of BASIC language statements entered into the calculator;

processing means coupled to said keyboard input means and said memory means, said processing means being responsive to actuation of said execute control key, following entry of a BASIC language statement into the calculator, for executing that BASIC language statement, said processing means being further responsive to actuation of said store control key, following entry of a BASIC language statement into the calculator, for storing that BASIC language statement as part of a program in said memory means; and

output means, coupled to said memory means, for providing an output indication of the result of execution of a BASIC language statement.

In the complete specification of our copending U.K. patent application No. 3345/76 (Serial No. 1,444,142), there is disclosed electronic data processing apparatus comprising:

keyboard input means for entering lines of one or more alphameric characters each into said electronic data processing apparatus;

first memory means for storing lines of one or more alphameric characters each;

second memory means, coupled to said keyboard input means and first memory means, adapted to store a single line of one or more alphameric characters entered into said electronic data processing apparatus from said keyboard input means or recalled from said first memory means; and

display means, coupled to said second memory means, adapted to visually display the single line of one or more alphameric characters stored in said second memory means;

(12)

said display means including means for visually displaying a cursor for designating any character position of the displayed line of one or more alphameric characters;

said keyboard input means including cursor control means for controlling the position of said cursor; and

said electronic data processing apparatus including logic means, coupled to said keyboard input means, second memory means, and display means, for positioning said cursor to designate any character position of the displayed line of one or more alphameric characters in response to actuation of said cursor control means.

In the complete specification of our copending U.K. patent application No. 3346/76 (Serial No. 1,444,143), there is disclosed electronic data processing apparatus comprising:

keyboard input means, including a plurality of numeric keys, for entering lines of one or more alphameric characters each into said electronic data processing apparatus;

first memory means for storing lines of one or more alphameric characters each;

second memory means, coupled to said keyboard input means and first memory means, for storing a line of one or more alphameric characters entered into said electronic data processing apparatus from said keyboard input means or recalled from said first memory means; and

display means, coupled to said second memory means, for visually displaying the contents thereof;

said keyboard input means including a memory recall control key for recalling a designated line of one or more alphameric characters from said first memory means to said second memory means;

each line of one or more alphameric characters stored in said first memory means being associated with a separate line number; and

said electronic data processing apparatus including logic means, coupled to said keyboard input means and first and second memory means, for transferring a line of one or more alphameric characters from said first memory means to said second memory means in response to sequential actuation of said memory recall control key and one or more of said numeric keys designating the line number associated with the line of one or more alphameric characters to be transferred.

In the complete specification of our copending U.K. patent application No. 3347/76 (Serial No. 1,444,144), there is disclosed electronic data processing apparatus including:

keyboard input means, including a plurality of alphameric keys, for entering lines of one or more alphameric characters each into said electronic data processing apparatus;

memory means for storing lines of one or more alphameric characters each, every stored line of one or more alphameric characters being associated with a separate line number;

buffer storage means, coupled to said keyboard input means and memory means, for storing a line number and for storing a line of one or more alphameric characters entered into said electronic data processing apparatus; and

display means, coupled to said buffer storage means, for visually displaying the contents thereof;

said keyboard input means including a control key for designating an automatic line numbering mode; and

said electronic data processing apparatus further including a line number counter for storing a current line number, temporary storage means for storing a line number increment and logic means coupled to said keyboard input means and buffer storage means, said logic means being responsive to actuation of said control key, followed by actuation of one or more alphameric keys designating both a starting line number and a line number increment, for storing the starting line number in both said buffer storage means and said line number counter and for storing the line number increment in said temporary storage means, said logic means being further responsive to completion of each entry of a line of one or more alphameric characters into the electronic data processing apparatus for combining the contents of said line number counter and said temporary storage means and for storing the result in both said line number counter and said buffer storage means.

In the complete specification of our copending U.K. patent application No.

03349/76 (Serial No. 1,444,146), there is disclosed an electronic desk-top calculator comprising:

keyboard input means with a plurality of alphameric keys for entering BASIC language commands into the calculator and with an execute key for initiating execution of BASIC language commands entered into the calculator;

storage means, coupled to said keyboard input means, for storing BASIC language commands entered into the calculator; and

processing means, coupled to said keyboard input means and storage means, for immediately executing a BASIC language command in response to entry of that BASIC language command into the calculator followed by actuation of said execute key.

In the complete specification of our copending application No. 3350/76 (Serial No. 1,444,141), there is disclosed electronic data processing apparatus comprising: keyboard input means for entering alphameric information into the apparatus; X-Y plotter means including a pen for graphically plotting selected alphameric information; memory means, coupled to said keyboard input means, for storing specified X and Y offset values; and processing means coupled to said keyboard input means, X-Y plotter means, and memory means, said processing means being responsive to an offset command entered into the apparatus for storing specified X and Y offset values into said memory means, said processing means being further responsive to subsequent plot commands, including specified X and Y coordinates, for summing each specified set of X and Y coordinates with the X and Y offset values stored in said memory means and for enabling plotting of the coordinates resulting from such summation.

In the complete specification of our copending application No. 3351/76 (Serial No. 1,444,148), there is disclosed electronic data processing apparatus comprising: keyboard input means with a plurality of keys for entering alphameric information into the data processing apparatus;

a modem for transmitting and receiving alphameric information over telephone lines; and

logic means coupled to said keyboard input means and modem, said logic means being responsive to entry of a terminal control command for selecting one of a plurality of baud rates at which alphameric information is to be transmitted from or received by the data processing apparatus.

In the complete specification of our copending application No. 3352/76 (Serial No. 1,444,149) there is disclosed an electronic calculator comprising:

keyboard input means for entering statements of one or more alphameric characters each into the calculator;

memory means, coupled to said keyboard input means, for storing statements entered into the calculator;

processing means, coupled to said keyboard input means and said memory means, for executing statements entered into the calculator or stored in said memory means;

magnetic tape reading and recording means for reading information from and recording information onto an external tape; and said processing means further including

logic means coupled to said keyboard input means, memory means, and magnetic tape reading and recording means, said logic means being responsive to execution of a mark statement, entered from said keyboard input means or stored in said memory means, for electrically partitioning the external magnetic tape into one or more files prior to statements of one or more alphameric characters being transferred to the external magnetic tape.

The present invention provides electronic data processing apparatus comprising: keyboard input means for entering one or more lines of one or more alphameric characters each into said electronic data processing apparatus; keyboard buffer means, coupled to said keyboard input means, for storing a line of one or more alphameric characters entered into said electronic data processing apparatus; and read-write memory means, coupled to said keyboard input means and keyboard buffer means, for storing one or more lines of more than three alphameric characters each entered into said electronic data processing apparatus; said keyboard input means including a plurality of keys, each representative of a single alphabetic or numeric character, for entering a representation of every alphabetic character of a selected alphabet and every numeric character from zero to nine into said electronic data processing apparatus; said electronic data processing apparatus including read-only memory

100

means storing routines and subroutines for recognizing particular strings of alphameric characters entered into said electronic data processing apparatus as being commands; and said electronic data processing apparatus including processing means, coupled to said keyboard input means, read-write memory means, and read-only memory means, for selectively performing the routines and subroutines stored in said read-only memory means and for executing the commands recognized thereby.

In electronic data processing apparatus as set forth in the last preceding paragraph said keyboard input means may include one or more additional keys each representative of a fixed string of alphameric characters representing a command.

In electronic data processing apparatus as set forth in the last preceding paragraph but one said keyboard input means may further include a plurality of definable keys, and control means for associating designated alphameric characters with a designated definable key as defined by the user; said read-write memory means may be arranged to store groups of alphameric characters associated with said definable keys; said electronic data processing apparatus may further comprise display means, coupled to said keyboard buffer means, for visually displaying the contents thereof; and said processing means may further comprise logic means coupled to said keyboard input means, said read-write memory means, and keyboard buffer means, said logic means being responsive to actuation of said control means, a selected one of said definable keys, and selected ones of said alphameric keys for associating the alphameric characters designated by said selected ones of said alphameric keys with the selected one of said definable keys and for storing those alphameric characters in said read-write memory means, said logic means thereafter being responsive to actuation of the selected definable key for transferring the alphameric characters associated therewith from said read-write memory means to said keyboard buffer means.

In electronic data processing apparatus as set forth in the last preceding paragraph but two said keyboard input means may include a key for designating a line termination character that may also be associated with the selected definable key; and said logic means may be responsive to actuation of a definable key, whose associated group of alphameric characters includes the line termination character, for immediately terminating the group of alphameric characters associated with that definable key.

In electronic data processing apparatus as set forth in the last preceding paragraph or the last preceding paragraph but one the alphameric characters associated with any of the definable keys can represent an executable command; and said logic means may be responsive to actuation of such a definable key for immediately executing the command represented by the alphameric characters associated with that definable key.

The illustrated preferred embodiment of this invention is a calculator which comprises a keyboard input unit, a magnetic tape cassette reading and recording unit, a solid state output display unit, an optional external output printer unit, an input-output control unit, a memory unit, and a central processing unit to provide an adaptable programmable calculator having manual operating, automatic operating, program entering, magnetic tape reading, magnetic tape recording, and alphanumeric display and print modes. The keyboard input unit includes a group of data keys for entering numeric data into the calculator, a group of control keys for controlling the various modes and operations of the calculator and the format of the output display, a group of alphanumeric keys arranged as a typewriter keyboard for entering statements, and a group of user-definable keys. All of the data and alphanumeric keys and some of the control keys may also be employed for programming the calculator.

The magnetic tape cassette reading and recording unit includes a reading and recording head, a drive mechanism for driving a magnetic tape past the reading and recording head, and reading and recording drive circuits coupled to the reading and recording head for bidirectionally transferring information between the magnetic tape and the calculator as determined by keyboard commands or commands which are part of a stored program.

The input-output control unit includes a sixteen-bit universal shift register serving as an input-output register into which information may be transferred serially from the central processing unit or in parallel from the keyboard input and magnetic tape cassette reading and recording units and from which information may be transferred serially to the central processing unit or in parallel to the solid

15

5

1,444,145

5

state output display, magnetic tape cassette reading and recording, and output printer units. It also includes control logic responsive to the central processing unit for controlling the transfer of information between these units. The input-output control unit may also be employed to perform the same functions between the central processing unit and peripheral units including, for example, an external printing unit, a digitizer, a marked card reader, an X—Y plotter, an external magnetic tape unit, a disc, a typewriter, and a modem. A plurality of peripheral units may be connected at the same time to the input-output control unit by simply plugging interface modules associated with the selected peripheral units into receptacles provided therefor in a rear panel of the calculator housing.

The memory unit includes a modular random-access read-write memory having a dedicated system area and a separate user area for storing program statements and/or data. The user portion of the read-write memory may be expanded without increasing the overall dimensions of the calculator by the addition of a program storage module. Additional read-write memory made available to the user is automatically accommodated by the calculator, and the user is automatically informed when the storage capacity of the read-write memory has been exceeded.

The memory unit also includes a modular read-only memory in which routines and subroutines of assembly language instructions for performing the various functions of the calculator are stored. These routines and subroutines of the read-only memory may be expanded and adapted by the user to perform additional functions oriented toward the specific needs of the user. This is accomplished by simply plugging additional read-only memory modules into receptacles provided therefore in a side panel of the calculator housing. Added read-only memory modules are automatically accommodated by the calculator and are accessed by the calculator through a series of mnemonic tables. These tables contain mnemonics which are additions to the calculator's programming language.

Plug-in read-only memory modules include, for example, a matrix module, a string variables module, a plotter module, an extended input-output module, and a terminal module. The matrix module makes available to the user standard BASIC language matrix functions plus an additional function which returns the determinant of a previously defined square matrix. The string variables module makes available to the user standard BASIC language string variables operations. The plotter module enables the user to conveniently plot and label on an external X—Y plotter. The extended input-output module allows the calculator to be used with a wide variety of peripheral input-output units. The terminal module facilitates interfacing the calculator with a modem for communicating, for example, with remotely located time-sharing computer systems. It further allows free text editing and storage.

The memory unit further includes a pair of recirculating sixteen-bit serial shift registers. One of these registers serves as a memory address register for serially receiving information from an arithmetic-logic unit included in the central processing unit, for parallel addressing any memory location designated by the received information back to the arithmetic-logic unit. The other of these registers serves as a memory access register for serially receiving information from the arithmetic-logic unit, for writing information in parallel into any addressed memory location, for reading information in parallel from any addressed memory location, and for serially transferring information to the arithmetic-logic unit. It also serves as a four-bit parallel shift register for transferring four bits of binary-coded-decimal information in parallel to the arithmetic-logic unit.

The central processing unit includes four recirculating sixteen-bit serial shift registers, a four-bit serial shift register, the arithmetic-logic unit, a programmable clock, and a microprocessor. Two of these sixteen-bit serial shift registers serve as accumulator registers for serially receiving information from and serially transferring information to the arithmetic-logic unit. The accumulator register employed is designated by a control flip-flop. One of the accumulator registers also serves as a four-bit parallel shift register for receiving four bits of binary-coded-decimal information in parallel from and transferring four bits of such information in parallel to the arithmetic-logic unit. The two remaining sixteen-bit serial shift registers serve as a program counter register and a qualifier register, respectively. They are also employed for serially receiving information from and serially transferring information to the arithmetic-logic unit. The four-bit serial shift register serves as an extend register for serially receiving information from

either the memory access register or the arithmetic-logic unit and for serially transferring information to the arithmetic-logic unit.

The arithmetic-logic unit is employed for performing one-bit serial binary arithmetic, four-bit parallel binary-coded-decimal arithmetic, and logic operations. It may also be controlled by the microprocessor to perform bidirectional direct and indirect arithmetic between any of a plurality of the working registers and any of the registers of the read-write memory.

The programmable clock is employed to supply a variable number of shift clock pulses to the arithmetic-logic unit and to the serial shift registers of the input-output, memory, and central processing units. It is also employed to supply clock control signals to the input-output control logic and to the microprocessor.

The microprocessor includes a read-only memory in which a plurality of microinstructions and codes are stored. These microinstructions and codes are employed to perform the basic instructions of the calculator. They include a plurality of coded and non-coded microinstructions for transferring control to the input-output control logic, for controlling the addressing and accessing of the memory unit, and for controlling the operation of the two accumulator registers, the program counter register, the extend register and the arithmetic-logic unit. They also include a plurality of clock codes for controlling the operation of the programmable clock, a plurality of qualifier selection codes for selecting qualifiers and serving as primary address codes for addressing the read-only memory of the microprocessor, and a plurality of secondary address codes for addressing the read-only memory of the microprocessor. In response to a control signal from a power supply provided for the calculator, control signals for the programmable clock, and qualifier control signals from the central processing and input-output control units, the microprocessor issues the microinstructions and codes stored in the read-only memory of the microprocessor as required to process either binary or binary-coded-decimal information entered into or stored in the calculator.

In the keyboard mode, the calculator is controlled by keycodes sequentially entered into the calculator from the keyboard input unit by the user. The solid state output display unit displays either the alphanumeric representation of the keys as they are depressed or a numeric representation of output data or alphanumeric user instructions or program results. An external output printer unit may be controlled by the user to selectively print a numeric representation of any numeric data entered into the calculator from the keyboard input unit, a numeric representation of any result calculated by the calculator, or a program listing on a line-by-line basis of the statements entered.

When the calculator is in the keyboard mode, it may also be operated in a print-all printing mode. The output printer unit then prints out each program line as it is entered by the user.

In the program running mode, the calculator is controlled by automatically obtaining an internal representation of the program statements stored in the user storage section of the read-write memory. During automatic operation of the calculator, data may be obtained from the memory unit as designated by the program, from the keyboard input unit while the operation of the calculator is stopped for data either by the program or by the user, or from the magnetic tape cassette unit as designated by the program.

When the calculator is in the program running mode, the user may also selectively employ a trace mode to check the execution of the program line-by-line in order to determine whether the program, as entered into the calculator, does in fact carry out the desired sequence of statements.

In the program entering mode, statements are sequentially entered by the user into the calculator from the keyboard input unit and are translated into an internal stored format which consists of a series of operation codes and operand names and are thereafter stored as statements of a program in the user storage section of the read-write memory.

The magnetic tape cassette reading and recording unit may be employed by the user to separately load either data, BASIC language programs, assembly language programs, or sets of user-definable key definitions into the calculator from an external magnetic tape cassette.

The magnetic tape cassette reading and recording unit may also be employed by the user to separately record either data, BASIC language programs, or sets of user-definable key definitions stored in the user section of the read-write memory onto an external magnetic tape cassette. Programs, or portions thereof, may be coded by the user as being secure when they are recorded onto an external

magnetic tape cassette. The calculator detects such programs when they are reloaded into the calculator and prevents the user from re-recording them or obtaining any listing or other indication of the individual program steps contained in the secured portions of such programs.

Description of the Drawings.

Figure 1 is a front perspective view of an adaptable programmable calculator according to the preferred embodiment of this invention.

Figure 2 is a rear perspective view of the adaptable programmable calculator of Figure 1.

Figures 3A—B are a simplified block diagram of the adaptable programmable calculator of Figures 1 and 2.

Figures 4A—F are a memory map of the memory unit employed in the adaptable programmable calculator of Figures 1—3.

Figure 4' is a diagram showing the arrangement of Figures 4A—F.

Figure 5 is a plan view of the keyboard input unit employed in the adaptable programmable calculator of Figures 1—3.

Figure 6 is an overall firmware block diagram for the adaptable programmable calculator.

Referring to Figures 1 and 2, there is shown an adaptable programmable calculator 10 including both a keyboard input unit 12 for entering information into and controlling the operation of the calculator and a magnetic tape cassette reading and recording unit 14 for recording information stored within the calculator onto one or more external tape cassettes 16 and for subsequently loading the information recorded on these and other similar magnetic tape cassettes back into the calculator. The calculator also includes a solid state output display unit 18 for displaying alphanumeric information stored within the calculator. All of these input and output units are mounted within a single calculator housing 24 adjacent to a curved front panel 26 thereof.

A plurality of peripheral input and output units including, for example, a line printer, a digitizer, a marked card reader, an X—Y plotter, a typewriter, a teletypewriter, an extended read-write memory unit, a magnetic disc reading and recording unit, and a modem for connecting the calculator via telephone lines to a remotely located computer, may be connected to the calculator at the same time by simply inserting interface modules 30 (Figure 3) associated with the selected peripheral units into any of four receptacles 32 provided therefor in a rear panel 34 of the calculator housing. As each interface module 30 is inserted into one of these receptacles, a spring-loaded door 38 at the entrance of the receptacle swings down allowing passage of the interface module. Once the interface module is fully inserted, a printed-circuit terminal board 40 contained within the interface module plugs into the mating edge connector mounted inside the calculator. If any of the selected peripheral units require AC line power, their power cords may be plugged into either one of two AC power outlets 42 provided therefor at the rear panel of the calculator housing 24.

Referring to the simplified block diagram shown in Figures 3A—B, it may be seen that the calculator also includes an input-output control unit 44 (hereinafter referred to as the I/O control unit) for controlling the transfer of information to and from the input and output units, a memory unit 46 for storing and manipulating information entered into the calculator and for storing routines and subroutines of basic instructions performed by the calculator, and a central processing unit 48 (hereinafter referred to as the CPU) for controlling the execution of the routines and subroutines of basic instructions stored in the memory unit as required to process information entered into or stored within the calculator. The calculator also includes a bus system comprising a S-bus 50, a T-bus 52, and an R-bus 54 for transferring information from the memory and I/O control units to the CPU, from the CPU to the memory and I/O control units, and between different portions of the CPU. It further comprises a power supply for supplying DC power to the calculator and peripheral units employed therewith and for issuing a control signal POP when power is supplied to the calculator.

The I/O control unit 44 includes an input-output register 56 (hereinafter referred to as the I/O register), associated I/O gating control circuitry 58, and input-output control logic 60 (hereinafter referred to as the I/O control). I/O register 56 comprises a universal sixteen-bit shift register into which information may be transferred either bit-serially from the CPU 48 via T-bus 52 or in parallel from keyboard input unit 12, magnetic tape cassette reading and recording unit 14,

and peripheral input units 28 such as the marked card reader via twelve input party lines 62. Information may be transferred from I/O register 56 either bit-serially to CPU 48 via S-bus 50 or in parallel to magnetic tape cassette reading and recording unit 14, solid state output display unit 18, output printer unit 20, and peripheral output units 28 such as the X—Y plotter or the typewriter via sixteen output party lines 64.

I/O gating control circuitry 58 includes control circuits for controlling the transfer of information into and out of I/O register 56 in response to selected I/O qualifier control signals from CPU 48, and selected I/O control instructions from I/O control 60. It also includes an interrupt control circuit 65, a peripheral control circuit 66, a printer control circuit 68, and a display control circuit 69 for variously controlling the input and output units and issuing control signals QFG and EBT to I/O control 60 via two output lines 71 and 72. These last mentioned control circuits variously perform their control functions in response to control signal POP from the power supply, I/O qualifier control signals from CPU 48, I/O control instructions from I/O control 60, and control signals from keyboard input unit 12. Interrupt control circuit 65 initiates the transfer of information into I/O register 56 from keyboard input unit 12 or interrupting peripheral input units 28 such as the marked card reader and issues a qualifier control signal QNR to CPU 48 via output lines 73. Peripheral control circuit 66 enables interface modules 30 plugged into the calculator to respond to information from I/O register 56, control associated peripheral units 28, transfer information to and/or receive information from associated peripheral units 28, and in some cases initiate the transfer of information to I/O register 56 from the interface modules themselves. Printer control circuit 68 and display control circuit 69 enable output display unit 18, and output printer unit 20, respectively, to respond to information from I/O register 56.

When a basic I/O instruction obtained from memory unit 46 is to be executed, CPU 48 transfers control to I/O control 60 by issuing a pair of I/O microinstructions PTR and XTR thereto. In response to these I/O microinstructions from CPU 48, control signal POP from the power supply, control signals QFG and EBT from I/O gating control circuitry 58, and I/O qualifier and clock control signals from CPU 48, I/O control 60 selectively issues one or more I/O control instructions to gating control circuitry 58 as required to execute the basic I/O instructions designated by CPU 48 and issues control signals, TTX, XTR, QRD, and SCB to CPU 48 via output lines 74—77. The I/O qualifier control signals issued to I/O control 60 and gating control circuitry 58 by CPU 48 are derived from the basic I/O instruction to be executed. Those qualifier control signals issued to I/O control 60 designate the specific I/O control instructions to be issued by I/O control 60, while those issued to gating control circuitry 58 designate selected control circuits to be employed in executing the basic I/O instruction.

Memory unit 46 includes a modular random-access read-write memory 78 (hereinafter referred to as the RWM), a modular read-only memory 80 (hereinafter referred to as the ROM), a memory address register 82 (hereinafter referred to as the M-register), a memory access register 84 (hereinafter referred to as the T-register), and control circuitry 85 for these memories and registers. The RWM 78 and ROM 80 comprise MOS-type semiconductor memories. As shown in the memory map of Figures 4A—F the basic RWM 78 contains a dedicated system storage section of 256 sixteen-bit words extending from address 1400 to address 1777 and a separate user program and/or data storage section of 1792 sixteen-bit words extending from address 40400 to address 43777. All addresses on the memory map are represented in octal form.

An optional 2048 sixteen-bit words of RWM may be made available to the user at address 44000 to address 47777. This is accomplished by removing a top panel 90 of the calculator housing shown in Figure 1, and inserting an additional printed circuit board containing the optional memory. The additional RWM is automatically accommodated by the calculator.

As shown in the more detailed memory map of Figures 5A—B of copending application 32269/73, the RWM dedicated system storage section includes 52 words (addresses 1400—1463) containing information in the form of mnemonic variables which is employed by the firmware routines shown in Figure 6. A more detailed description of these mnemonic variables is given on page 21 of the calculator basic system firmware listing located in the specification of application 32269/73 (Serial No. 1,444,141). Addresses 1466—1477 and 1701—1737 are used as temporary storage by the various routines shown in Figure 6. Addresses 1500—1550 comprise a 41-word buffer used to contain the input characters during

5 syntax analysis. Addresses 1551—1621 comprise a 41-word buffer used by the
single line display refresh routine of Figure 6. These 41 words along with 42
additional words (addresses 1622—1673) are used as a syntax buffer during syntax
analysis. Four of these words (addresses 1622—1625) are used as temporary
10 registers by several of the statement execution routines of Figure 6. Eight words
(addresses 1630—1637) are used as two temporary floating point number registers
by the formula evaluation routines of Figure 6. Addresses 1640—1677 comprise 32
words which are used by the statement execution routines of Figure 6. Eight words
(addresses 1744—1747 and 1754—1757) are employed as "AR1" and "AR2" four-
15 word working registers for performing binary-coded-decimal arithmetic. An
additional eight words (addresses 1740—1743 and 1750—1753) are employed as
working data registers "X₀" and "Y₀" for implementation of the trigonometric
functions. These sixteen words (addresses 1740—1757) are used as temporary
storage registers by all the routines of Figure 6 except the statement execution and
formula evaluation routines. A variable-length "system subroutine stack"
(addresses 1760—1772) is employed for storing return addresses required by
programs stored in ROM 80. Four words (addresses 1773—1776) are used to store
20 the result of the latest keyboard computation. The last word in the system RWM
(address 1777) is used to store a pointer indicating the next available location for
the return address of the next subroutine call within the basic system. A complete
assembly language description of the system RWM is included in pages 21—24 of
the calculator basic system firmware listing which is to be found in the
aforementioned application 32269/73 (Serial No. 1,444,141).

25 As shown in the memory map of Figures 4A—F and the more detailed
memory map of Figure 6 of application 32269/73 (Serial No. 1,444,141), user
program and/or data storage section of RWM 78 contains 1760 words available to
the user (as user addresses 40440—43777) for storing programs and/or data, 20
words dedicated for use by the interrupt routine of Figure 6 and also of Figures
117A—B of application 32269/73 (Serial No. 1,444,141), and 12 words available for
30 use by plug-in read-only memory modules. An additional 2048 sixteen-bit words
may be made available to the user (as user addresses 44000—47777).

Also, as shown in the memory map of Figures 4A—B, the basic ROM 80
contains 7680 sixteen-bit words extending from address 0000 to address 1377, from
address 2000 to address 16777, and from address 40000 to address 40377. Routines
35 and subroutines of basic instructions for performing the basic functions of the
calculator and constants employed by these routines and subroutines are stored in
these portions of ROM 80. An additional 8192 sixteen-bit words of ROM may also
be added at addresses 20000—37777 in steps of 512 and 1,024 words. This is
accomplished by simply inserting plug-in ROM modules 92 into receptacles
40 provided therefor within the calculator which are accessible through a door in the
left panel of the calculator housing as illustrated in Figure 1. As each plug-in ROM
module 92 is inserted into one of these receptacles a printed circuit terminal board
96 contained within the plug-in ROM module plugs into a mating edge connector
45 mounted inside the calculator. A handle pivotally mounted at the top end of each
plug-in ROM module 92 facilitates removal of the plug-in ROM module once it
has been fully inserted into one of the receptacles.

Routines and subroutines of basic instructions (and any needed constants) for
enabling the calculator to perform many additional functions are stored in each
50 plug-in ROM module 92. The user himself may therefore quickly and simply adapt
the calculator to perform many additional functions oriented toward his specific
needs by simply plugging ROM modules of his own choosing into the calculator.
Added plug-in ROM modules are automatically accommodated by the calculator.

KEYBOARD OPERATIONS

55 All operations performed by the calculator may be controlled or initiated by
the keyboard input unit and/or by keycodes entered into the calculator from the
keyboard input unit, the magnetic tape cassette reading the recording unit, or
peripheral input units such as the marked card reader and stored as program steps
in the program storage section of the RWM. An operational description of the
60 keyboard input unit is therefore now given with specific reference to the
perspective view of the calculator as in Figure 1 and the plan view of the keyboard
as in Figure 5, except as otherwise indicated.

Line Switch

An on-off line switch 182, which may be considered as part of the keyboard

20

10

1,444,145

10

input unit, controls the application of power to the calculator and hence initiation of the control signal POP from the power supply.

5

5

10

10

As shown in Figure 2, the calculator may be operated at 240, 220, 120, or 100 volts +5%, -10% as determined by a pair of line voltage selector switches mounted at rear panel 34 of the calculator housing and at a line frequency within the range of 48 to 66 Hertz. The calculator is provided with a 6-amp fuse and either a 1-amp fuse for operation at a line voltage of 220 or 240 volts +5%, -10% or a 2-amp fuse for operation at a line voltage of 100 or 120 volts +5%, -10%. It is also provided with a three-conductor power cable 184 which, when plugged into an appropriate AC power outlet, grounds the calculator housing. The maximum power consumption of the calculator is 150 voltamps. No more than a total of 610 voltamps may be drawn from AC power outlets 42 provided for peripheral units.

EXECUTE

15

15

20

20

The EXECUTE (often referred to as EXEC) key, when pressed, will perform the indicated operations previously keyed in, if any, and display the result of any arithmetic statements on the 32-character display. (Although the display is only 32 characters, an 80-character line can be keyed in with automatic scrolling both for programs and for keyboard operations.) Most keys, when pressed, immediately cause their mnemonic to be displayed. However, pressing certain keys, such as PRT ALL (to be discussed later), allows a particular mode to be in effect till that mode is overridden.

FIXED N, FLOAT N

25

25

Immediately after either turn-on or SCRATCH EXEC, the user read/write memory area is cleared; numerical calculations that are executed will be displayed in float-nine notation. The values 2 and 12 in float-nine notation would appear as 2.000000000E+00 and 1.200000000E+01, respectively. Float 9 refers to the nine digits succeeding the decimal point; E symbolizes $\times 10$ raised to the power of the two digits following the E.

NOTE

30

30

In this text, individual keyboard operations will be identified by being underlined; e.g., 3 + 2 EXEC. (On the display would appear 5.000000000E+00.)

35

35

You can specify the desired notation by pressing FIXED N or FLOAT N followed by the appropriate number from 0 through 11. The designated 'N' indicates the number of digits to be displayed to the right of the decimal point after execution.

For example:

FIXED N 7 EXEC.

40

40

then 123456.7 EXEC, displays 123456.7000000

or, 123456789.123456789 EXEC, displays 123456789.1230000†

FLOAT N 5 EXEC.

then 123.4 EXEC displays 1.23400E+02

or, 1234567 EXEC displays 1.23457E-01††

45

45

In fixed-n notation, a maximum of 12 digits will be displayed to the left of the decimal point; beyond that value the calculator reverts to float-n notation, with the number of digits displayed to the right of the decimal point determined by the particular fixed-n.

50

50

The FIXED N and FLOAT N keys are not programmable; that is, they can be used only in keyboard operations. Output formatting under program control is accomplished by a format statement, which is discussed below.

The calculating range of the calculator is: $-9.9999999999 \times 10^{99}$ through -10^{-99} , 0, and 10^{-99} through $9.9999999999 \times 10^{99}$.

†The machine calculates to 12 significant digits regardless of the display length.

††Since only five digits can be displayed to the right of the decimal point the fifth digit is rounded.

CLEAR, DELETE LINE

Pressing either CLEAR or DELETE LINE will erase whatever previously had been displayed during keyboard operations. Pressing either key will cause ← (the Lazy T) to appear on the far left of the display, indicating that the calculator is available for new inputs. The difference between the two keys occurs when a stored program line is displayed; DELETE LINE will erase the line from the stored program, whereas CLEAR will only erase the display and not affect the program line itself. (A program line can also be deleted by keying in the line number followed by END OF LINE.)

RECALL

Pressing RECALL during keyboard operations allows the last line that was executed to be recalled to the display; pressing RECALL during program-inputting operations allows the last program line that was input to be recalled to the display. If an error message appears on the display when an attempt is made either to execute a line in keyboard mode or to input a program line into memory, pressing RECALL will allow the original line to be reviewed for editing purposes.

RESULT

Pressing CLEAR RESULT EXEC displays the numerical value of the last arithmetic statement that was executed. The RESULT key not only allows the result of the previously executed statement to be reviewed, but also can function as an "accumulator" during arithmetic operations; e.g., by pressing

$2 \div 4$ EXEC (in FIXED 2 notation) displays 6.00, then pressing
 $3 \div$ RESULT EXEC displays 9.00, then pressing
 $4 \div$ RESULT RESULT EXEC displays 22.00.

Keying in either R E S or R E S U L T has the same effect as pressing the RESULT key.

PRINT ALL

By pressing PRINT ALL, you can determine whether or not the calculator is in print all mode. If ON appears, both the expression and the result of any executed calculation will be typed on whichever printing device is plugged into the calculator; if OFF appears, the information will appear only on the display. Pressing the PRINT ALL key a second time causes the alternative mode to be in effect. In the print all mode, each program statement is printed when END OF LINE is pressed. All error messages are also printed.

BACK, FORWARD, INSERT

The BACK and FORWARD keys can be used to edit expressions on the display. Successive presses of the BACK key will move a blinking cursor to the desired location within the display. Editing can then be performed at this location. The FORWARD key performs the same function as the back key, but in the opposite direction.

At the location of the blinking cursor, the following editing can be performed:

1. A character can be inserted by pressing INSERT. (This opens up a space to the left of the cursor, thereupon moving the cursor to the location of the space; additional presses of the INSERT key will open up more spaces, with the cursor always positioning itself in the left-most space, thus allowing for the immediate insertion of more than one character.)
2. A character can be changed by overscoring it with another character.
3. A character can be deleted by pressing either the space bar or SHIFT INSERT, the only difference being that pressing SHIFT INSERT will close up the space of the deleted character.

Holding either BACK or FORWARD down for approximately 1.5 seconds will cause the cursor to move in rapid succession in the chosen direction.

Once a line is appropriately edited, it can be immediately executed without having to move the cursor to the end of the line.

→, ←

When a line that is greater than 32 characters (80 characters maximum) is being input, the characters to the left of the display are pushed out of the display

region to make room for the additional characters. To view the beginning of the input, press → (the right arrow); this operation moves the characters in the display to the right. Pressing ← (the left arrow) performs the reverse operation. Either arrow key, when held down for approximately 1.5 seconds, will repeat its operation in rapid succession.

ARITHMETIC

There are five basic numerical operators: add (+), subtract (−), multiply (*), divide (/), and exponentiate (^). The order of execution, known as the hierarchy, is identical to the BASIC hierarchy described below. The BASIC functions described below are available on the Calculator simply by keying in the appropriate mnemonics. In addition, the value of π can be obtained by keying in PI.

When operating on trigonometric functions, the calculator assumes the angle to be in radians unless otherwise stated. To express an angle in either degrees or grads, first clear the display, then press either DEG EXEC or GRAD EXEC, respectively; then key in the expression. To revert to radians, clear the display, then press RAD EXEC. Radians, degrees, and grads are also programmable commands.

VARIABLES

The simple "scalar" variables are A through Z and A0 through Z9 (286 total). Simple variables can be used in keyboard operations; e.g., pressing A 3 = 7 EXEC assigns the value of 7 to A3. Unless the value of A3 is then changed or erased from memory, pressing 4 * A 3 EXEC will display 28. If a variable is undefined, any attempt to use this variable in an expression (other than by assigning a value to it) will result in an error message.

An array is an ordered collection of numerical data. An array (subscripted) variable can have either one or two dimensions as indicated by the subscripts, which are presented as numbers within parentheses. A(m) is a one-dimensional array (or column vector) when m designates the row of the element; A(m, n) is a two-dimensional array where m designates the row and n designates the column of the element.

The maximum size of an array is limited by the available calculator memory. At normal 12-point precision, the effective memory limitations would be approximately a (30, 30) array.

Arrays should be referenced in either a common statement or a dimension statement before use in a program; if not, the program defaults to either a 10 element array for a singly subscripted array, or a 10 by 10 array for doubly subscripted arrays.

TAPE CASSETTES

The present Calculator is capable of operating with 10 tape cassettes: One cassette is available through the built-in (internal) cassette drive. Four peripheral cassette drives can be connected directly to the calculator through the four I/O slots in the rear panel.

Five more peripheral cassette drives can be added if the user has an I/O expander box (in all, 11 peripheral devices can be added by having an expander box).

SYNTAX

The tape cassette commands have general syntactical rules which will be briefly described. The general command form, with minor variations, is:

COMMAND [UNIT] [FILE], or
COMMAND [UNIT] (FILE) [LNX, [LNX,]]

Brackets [] indicate that the enclosed information is optional; parentheses () indicate that the enclosed information is required with the particular command.

COMMAND
UNIT

- The various commands will be individually discussed.
- Individual units are referenced by the number sign, #, followed by the select code; the internal cassette is designated by #10 (if no select code is given, the internal cassette is assumed); the nine peripheral cassettes are identified by #1 through #9.

FILE

- Files within individual cassettes are identified by file numbers; if no file is identified by number, the scratch-pad (or default) file, file 0, is assumed.

LN_X₁ — indicates the beginning line number to be affected by the command.
 LN_X₂ — indicates either the line number where program execution is to begin following the command implementation, or the ending line number to be accessed in the command. In any command whose syntax allows LN_X₁ and LN_X₂, in order for LN_X₂ to be designated, LN_X₁ must have been specified.

All information to be input following the command must be separated by commas.

In keyboard mode, press **EXEC** after fulfilling the syntax requirements of a particular command to implement the command.

PREPARING A FRESH CASSETTE

To prepare a fresh cassette (one which has no markings), first open the cassette door by pressing downward on the switch on the far right of the keyboard; then insert the cassette (print-side up) into the slot built into the door; be certain that the tape is wound around the left spindle — if it is not, but you wish to prepare that side of the cassette anyway, simply close the door and press **REWIND**.

To store information onto the cassette, first mark the files that are to be used.

PROGRAMMABLE COMMANDS

The following tape cassette commands are programmable: mark, store, load, merge, store key, load key, store data, load data, load binary, rewind, find, and tlist. These commands can also be used in the keyboard mode. The secure command, on the other hand, can be used only in the keyboard mode.

MARK

The mark command produces the designated number of files and defines the file lengths (by the number of 16-bit words per file).

Syntax: **MARK [UNIT] (No. of FILES) (LENGTH)**

e.g., **MARK #10, 5, 1000 EXEC** — since unit #10 identifies the internal cassette, including it is superfluous; the '5' indicates the number of files to be made available; the '1000' signifies the number of 16-bit words per file.

Successive files on the tape can be marked with different word lengths as shown in the following example; **MARK 3, 1000 EXEC** then, **MARK 2, 2000 EXEC** will mark files 0, 1, 2 of the internal cassette with 1000-word lengths and will make files 3, 4 of the same cassette with 2000-word lengths.

The length of a file can be changed; however, changing it will affect all the files following it by distorting their contents. To change the length of a particular file, first position the cassette at that file by using **FIND** (**FIND** will be discussed more thoroughly, later); then mark the file with the desired length. For Example: to mark file 12 of the internal cassette with 1000 words, press **FIND 12 EXEC** then press, **MARK 1, 1000 EXEC**

To mark files not previously marked (virgin files), first perform a **TLIST**; **TLIST** will be discussed thoroughly later; but for now, merely knowing that **TLIST [UNIT] EXEC** will list all the marked files, is sufficient. To mark virgin files, it is first necessary to mark the last file listed in **TLIST**†. As previously discussed, this file is located by the find command. Beginning with this file, successive virgin files can be marked by the methods previously discussed.

To mark the beginning of a tape, be certain that the tape is completely rewound.

STORE

The store command will take the program line numbers in read/write memory, and put them in the designated tape location where they will be saved.

Syntax: **STORE [UNIT] [FILE] or
 STORE [UNIT] (FILE) [LN_X₁] [LN_X₂]**

†The number of files referenced in **TLIST** is always one more than the number marked by the user.

e.g.,

STORE #2, 3 EXEC; select code #2 designates which cassette is being accessed; the program will then be stored on that cassette in file number 3.

In the store command, LNX_1 and LNX_2 are used to store a portion of the program: LNX_1 specifies the beginning line number to be stored; LNX_2 specifies the ending line number to be stored. If LNX_2 is not specified, the last line to be stored is assumed to be the highest-numbered program line.

For LNX_2 to be given, it is always necessary to have LNX_1 ; this is true for any cassette command. In addition, whenever LNX_1 is to be given, the file to be accessed must be identified; e.g.,

STORE 3, 60, 150 EXEC; file 3 of the internal cassette is located; program lines 60 through 150 in memory will then be stored into file 3.

Note

Once a cassette has been marked, any file that has been marked can be accessed by giving its designated file number in a command; e.g., if five files are marked, information can be stored in file 4 even though file 3 is a virgin (empty) file, by pressing **STORE 4 EXEC**.

LOAD, LINK

The load command will take a program that is stored on a cassette and put it into the memory area.

Syntax: **LOAD [UNIT] [FILE]** or,
LOAD [UNIT] (FILE) [LNX_1] [LNX_2]

e.g.,

LOAD EXEC assumes the internal cassette, default file (file 0) is to be loaded into the calculator; any program previously in the memory will be erased.

LOAD 5, 40, 10 EXEC locates file 5 on the internal cassette; the entire program on this file is renumbered beginning at line number 40, and then the program is loaded into memory; program execution is initiated at line number 10. All program line numbers beginning at line 40, that were previously in memory, will be erased and replaced by the program being loaded; if the memory previously had line numbers 10, 20, 30, it will retain them.

In both the load and the merge command, the use of LNX_1 and LNX_2 can have various results. Rules for predicting the results are given at the conclusion of the merge command discussion.

By substituting **LINK** for **LOAD** in the previous syntax, the user can implement the link command. This command operates identically to the load command with one exception:

During program execution, if a load statement is encountered, the calculator functions as though **RUN EXEC** were pressed — that is, the old symbol table is destroyed and a new symbol table is built; on the other hand, if a link command is encountered, the calculator functions as though **CONT EXEC** were pressed — that is, all variables retain their previous values.

MERGE

The merge command attempts to take program line numbers from the cassette and position them in read/write memory in front of the program currently there, between consecutive line numbers in the program currently there, or behind the program currently there. However, if any line number of the program to be entered matches a line number currently in the program, an error will result. In addition, if the line numbers of the two programs are interwoven, an error will occur; e.g., if the program currently in memory has line numbers 10, 20, 30, 40 and if the program to be merged has line numbers 15, 25, 35, the merge command will cause an error message to occur even though no two line numbers matched.

Syntax: **MERGE [UNIT] [FILE]** or,
MERGE [UNIT] (FILE) [LNX_1] [LNX_2]

e.g.,

MERGE #2, 1, 200, 100 EXEC — file 1 of the cassette with select code #2 is located; the entire program on this file is renumbered beginning at line number

200 (LNX₁) and then it is combined with the program currently in memory. Following implementation of the command, program execution will begin at line number 100 (LNX₂).

5 Either the merge or the load command can be used for stacking programs in read/write memory. The major difference the two commands is as follows: LOAD will erase the line numbers previously in memory, beginning at the designated LNX₁; MERGE, on the other hand, will retain all line numbers previously in 9830A memory. 5

10 In both the load and the merge command, LNX₁ and LNX₂ are predictable. Regardless of the mode, LNX₁ rennumbers the program line numbers of the accessed file to begin at LNX₁; the spacing between consecutive line numbers remains the same; all GO TO statements, etc. are properly adjusted to reflect the new line numbers; this program is then loaded into user memory. 10

15 In program mode:
1. If LNX₂ is given, program execution will continue at LNX₂.
2. If LNX₂ is not given, program execution will continue either at the next higher line number of the original program or at LNX₁, whichever comes first. 15

20 In keyboard mode:
1. If LNX₂ is given, program execution will begin at LNX₂.
2. If LNX₂ is not given, the calculator will halt after loading in the program. 20

STORE KEY, LOAD KEY

The store key command will take all user definable keys (upper left-hand region of the keyboard) that have been defined and put them on a cassette file, which will be tagged as a key file.

25 The load key command takes the user definable information from the cassette and positions it in memory such that each user definable key performs the same operation that it previously did before being stored on tape. 25

Syntax: STORE KEY [UNIT] (FILE)
LOAD KEY [UNIT] (FILE)

30 Spacing may arbitrarily be left between the words STORE and KEY and the words LOAD and KEY. In general, the HP Basic language ignores blank spaces (except, of course, in a quote field where every character and space is duplicated). 30

STORE DATA, LOAD DATA

35 The store data command takes a block of data from memory and puts it on a cassette. Normally, only arrays can be stored using this command; however, if no array is specified in the comand, all data in the common statement of the program can be stored. The calculator allows simple variables in the common statement, as well as arrays; in fact, the common statement can accept simple variables, array variables, integer arrays and variables, and split arrays and variablest. 35

40 The load data command takes the data that was previously stored on a cassette file, and loads it into mainline memory. If an array has been stored, then LOADDATA must specify an array; if LOADDATA does not specify an array, an error will result. If, on the other hand, the common area will be retrieved by the load data command (in this case, no particular array can be specified in the command, lest an error occur). 40

Syntax: STOREDATA [UNIT] (FILE) [ARRAY]
LOAD DATA [UNIT] (FILE) [ARRAY]

e.g.,

50 STORE DATA 6, B EXEC will locate file 6 on the internal cassette; then the B array in the current program will be stored in this file. (A simple variable cannot be stored in this manner). 50

LOADDATA 6, B EXEC can then retrieve the B array and load it into memory whenever it is needed; the following command could be given, also:

55 LOADDATA 6, C EXEC; this would retrieve the B array and load it into memory in place of the C array, provided B and C are the same size and type. 55

†The common statement acts like a dimension statement with the additional feature that data in common is saved from program to program.

Assume the common statement in a program looks like this: 1 COM A(8) , B(5,5), D3, E.

Pressing: STORE DATA 2 EXEC will store all the common statement variables into file 2 of the internal cassette. Both the array variables and the simple variables will be retrieved by pressing: LOAD DATA 2 EXEC.

Pressing: LOAD DATA 2, A EXEC is illegal and causes an error since no particular array can be retrieved from a file if it was stored in common.

In all store and load data commands, the file to be accessed must be identified, even if it is the default file (file 0).

LOAD BIN

The load binary command will transfer binary information — assembly language program — from the cassette to the user memory. The assembly language program may be a system diagnostic, an I/O subroutine, or a simulated "option block" designed to perform some specific function.

The assembly language program cannot be listed or displayed.

Syntax: LOAD BIN [UNIT] (FILE)

Note

Files on cassettes are tagged as: program files, key files, data files, or binary files. If an attempt is made to load from a particular file and the load command incorrectly identifies the file tag, an error will occur; e.g., pressing LOAD KEY 1 EXEC when file 1 is a program file, causes an error message to appear.

REWIND

Pressing the REWIND key, located on the right-hand side of the keyboard, immediately rewinds the internal tape cassette to the clear leader.

To rewind any other cassette, REWIND must be typed in, followed by the select code of the particular cassette. (The internal cassette can also be rewound by typing in REWIND EXEC.)

REWIND must be typed if the command is to be used in the programming mode.

Syntax: REWIND [UNIT]

e.g.,

REWIND #2 EXEC will rewind the cassette with select code #2.

FIND

The find command (previously mentioned in conjunction with the mark command) is used to locate a particular file. While the cassette is searching for the file number, a "lazy T" appears on the display. During this interval, the cassette is searching under interrupt control, thus returning control of the calculator keyboard to the user. This feature allows for the execution of one portion of a large program, while another portion is being found — thereby improving access time.

When the specified file is found, the cassette tape halts.

Syntax: FIND [UNIT] (FILE)

e.g.,

FIND #3, 2 EXEC causes the cassette with select code #3 to search until file number 2 is located.

TLIST

Beginning with the current tape location, this command reads all subsequent file identifiers and prints out information concerning each file.

Syntax: TLIST [UNIT]

The information for each file, on the designated cassette, is printed out on one line. There are no column headers identifying the information in each line; the assumed headers are:

27

17

1,444,145

File No.	File Type* (Code No.)	Absolute File Size (in words)	Actual File Size (in words)	Program Line Nos. (Beginning) (Ending) (LNX ₁) (LNX ₂)	Common Area (in words)
----------	--------------------------	-------------------------------------	-----------------------------------	--	---------------------------

*The code numbers identifying the file types are as follows:

- | | | |
|---|--------------------|---|
| 5 | 1 binary | 5 |
| | 2 data | |
| | 3 program (source) | |
| | 4 key | |

10 In addition, if the file is secured, the number 2 appears in front of the code number (this applies only to binary, source, and key files); e.g., if "24" appears in the second column, the file is a secured key file.

If a file is a data file, LNX₁, is superfluous; in this case the data is described in this column as:

- | | | |
|----|---------------------|----|
| 15 | 0 full precision | 15 |
| | 1 split precision | |
| | 2 integer precision | |
| | 3 common | |

If the file is not a program file, the last two columns will contain no information.

20 NON-PROGRAMMABLE COMMANDS 20

SECURE

25 This command has the capability of concealing program lines from potential users; that is, your program could be given to another, and that person could load and run it — however, he would not be able to fetch particular program lines for viewing purposes; nor could he store the program on any other cassette file.

An attempt to fetch a secured program line will result in the line number appearing on the display, followed by an *; attempts to list the program will result in the secured line numbers appearing followed by an *.

30 If any lines in a program are secured, the entire program is considered to be secured; that is, even though certain program statements are visible, none of the program can be reproduced onto another cassette file.

Syntax: SEC [LNX₁ [LNX₂]] or SECURE [LNX₁ [LNX₂]]

It is therefore, possible to secure specific lines within a program; e.g., pressing:

35 SEC 30, 80 EXEC, followed by STORE 2 EXEC secures lines 30 through 80 of the program in memory and then stores both the secured and unsecured portions of the program into file 2 of the internal cassette.

40 When a program is initially secured, it can still be reproduced onto as many files as necessary; however, once the program is scratched from memory, (even though it can be loaded back into memory) it cannot be reproduced onto any cassette files.

45 When program lines are secured, the entire calculator is in the secured mode. Therefore, after the secured program is stored away, the user should press SCRATCHA before inputting other programs — thus, avoiding "secured program" errors.

User definable keys (when not being used as typing aids) can be secured too. Just press FETCH (particular key) SEC EXEC and the designated key will be secured.

50 Note 50

To protect all the information on a particular cassette, break one of the tabs on the top of the cassette; this makes the cassette inaccessible for further storage.

PROGRAMMING

The programming language of the Calculator is, with minor variations, BASIC as described below.

Instructions to the computer within a program are provided by program statements. Each statement in Basic has an associated line number which must appear in the left-most portion of the statement.* Statement line numbers appear in ascending order with 9999 being the largest possible line number.

A program statement, which has been correctly keyed in, can be stored in read/write memory by pressing the END OF LINE (EOL) key. This key is situated in an area corresponding to the carriage return/line feed key on a teletype (TELETYPE is a Registered Trade Mark) keyboard.

AUTO

As previously mentioned, statement line numbers can be automatically input. In its simplest form, pressing AUTO # EXEC causes line number 10 to immediately appear on the display awaiting the program statement. The line numbers of additional statements will be in ascending order with a spacing of ten between consecutive line numbers. The AUTO # (AUTO) syntax and the examples to follow present some of the alternatives in automatic line numbering.

Syntax: AUTO # [LN_X, [Spacing]]

LN_X is the beginning line number to be automatically input; the desired spacing between lines can then be input if LN_X is given. If no spacing is indicated, a spacing of 10 is assumed.

e.g.,	<u>AUTO # 30 EXEC</u>	<u>AUTO # 40, 2 EXEC</u>	<u>AUTO # EXEC</u>
	30	40	10
	40	42	20
	50	44	30
	:	:	:

When AUTO # is pressed, AUTO appears in the display.

Note

Although the CLEAR and DELETE LINE keys have previously been discussed, the following point should be made. If a program line currently being keyed in is found to be totally unacceptable (not worth salvaging by using the editing keys), it can be erased by using either the CLEAR or the DELETE LINE keys. However, if the program line numbers have been automatically input, pressing CLEAR not only erases the entire display but also eliminates the AUTO # mode, whereas pressing DELETE LINE erases only the program statement without affecting the line number itself.

PROGRAM VIEWING

- There are two methods of viewing a program that is currently in memory:
1. List it on a printing device.
 2. Bring it line-by-line to the display.

LIST

The list command has two specific applications: it can be used to provide a total listing of the programs in read/write memory, or it can be used to indicate the read/write memory available for inputting; e.g.,

LIST EXEC lists all program lines that are in memory on the user's standard printing device.

*The length of a statement, including the line number and including appropriate spacing, can be up to 80 characters.

LIST #3 EXEC lists all program lines that are in memory on the peripheral with select code #3.

LIST 9999 EXEC causes the number of 16-bit words available in memory to be displayed.

If a specific program line is in the display, pressing ↓ (down arrow) displays the next higher-numbered program line; ↑ (up arrow), on the other hand, displays the next lower-numbered program line. When no program line is currently in the display, pressing ↓ would display the successively higher line from the one most recently displayed, whereas ↑ would display the successively lower line.

FETCH

In addition to viewing a program line-by-line, specific program lines can be immediately brought to the display by using the fetch command.

Syntax: **FETCH [LN_X]**

Where LN_X is the specific line number to be accessed;

e.g.,

FETCH EXEC always displays the lowest-numbered program line.

FETCH 300 EXEC displays line 300 if it exists; if line 300 is not available (and there are other higher-numbered lines), the line immediately higher than 300 will be displayed; if there are no line numbers as high as 300, the highest-numbered line available in memory will be displayed.

PROGRAM EDITING

Any displayed program line can be edited by using the **BACK**, **FORWARD**, and **INSERT** keys.

The **RECALL** key, previously discussed, need only be briefly mentioned. A program line is keyed in; if an error message appears on the display when **EOL** is pressed, the program line can be reviewed by pressing **RECALL**. Appropriate editing can then be performed on the program line. The previously input line can always be recalled by pressing **RECALL** whether or not an error appears.

CLEAR and **DELETE LINE** have already been thoroughly discussed. However, to reiterate, there are two methods of deleting a program line currently in memory:

1. If the line is currently in the display, pressing **DELETE LINE** will erase it from memory (**CLEAR** only clears the display).
2. Any line in memory can be immediately deleted by keying in the appropriate line number followed by **EOL**.

DELETE

The delete command (to be distinguished from **DELETE LINE**) can selectively delete program lines.

Syntax: **DELETE [LN₁ [LN₂]]** or **DEL [LN₁ [LN₂]]**

Where LN₁ is the first line to be deleted and LN₂ is the last line to be deleted:

e.g.,

DELETE EXEC deletes all program lines;

DELETE 40 EXEC deletes all statements beginning at line number 40;

DELETE 50, 80 EXEC deletes all lines numbered 50 through 80.

*It should be noted that if a program line has more than 32 characters, the first 32 characters will be displayed; by pressing ←, the rest of the line can be viewed as the display is scrolled to the left. At the end of the line ← appears.

SCRATCH

The scratch command can erase a variety of things from memory:

e.g.,

SCRATCH EXEC erases all program lines and variables;

SCRATCH A EXEC erases everything from memory — program lines, user definable keys, variables (identical to turning the calculator off, then on again);

SCRATCH K EXEC erases all user definable keys (user definable keys will be discussed later);

SCRATCH V EXEC erases all variables;

SCRATCH (particular UD key) erases the particular user definable key that was pressed — pressing EXEC is not required in this case.

Scratch can be accessed either by keying in the seven letters or by pressing SCRATCH.

RENUMBER

The renumber command will take all the program line numbers and renumber them.

Syntax: RENUMBER [LN₁, [SPACING]] or REN [LN₁, [SPACING]]

Where LN₁ will be the new line number of the first program statement, and SPACING will be the spacing between consecutive line numbers:

e.g.,

REN EXEC will renumber all statements by numbering the first statement '10' with a spacing of 10 between statements;

RENUMBER 30 EXEC rennumbers the first statement '30' with a spacing of 10 between statements;

REN 45, 20 EXEC rennumbers the first statement '45' with a spacing of 20 between statements.

All statements in the program that reference another line number are appropriately corrected with the renumber command; e.g., GO TO 80 would be corrected to reference the line that replaced 80.

PROGRAM DEBUGGING**NORMAL, TRACE**

The TRACE key can be used to determine the order of statement execution for a program that is currently running. Pressing TRACE during program execution causes the line numbers to be printed in the order in which they are accessed; then pressing NORMAL reverts the calculator to the normal mode. Thus, when a program is running, both TRACE and NORMAL are immediate execute keys.

When no program is running, to revert to either trace or normal mode requires pressing the appropriate key followed by EXEC. Trace mode can be set up to trace specific line numbers.

Syntax: TRACE [LN₁, [LN₂]]

Where LN₁ is the first line number to be traced, and LN₂ is the last line number to be traced;

e.g.,

TRACE 20 EXEC will trace beginning at line number 20 when the program is running.

TRACE 50, 60 EXEC will trace beginning at line number 50 and ending at line number 60, each time these line numbers are executed in the program.

STOP

The stop command can be a statement within a program (to be discussed later) and can be used as a debugging tool.

As a debugging tool, STOP is extremely valuable. A program that is running can be stopped at any time by pressing STOP (the current line number of the program will be displayed). If any program lines are then edited, the program must

be rerun from the beginning, using the RUN key. If no editing has been performed, the program can continue where it left off if the CONT key (to be discussed in detail later) is pressed.

While a program is stopped, the values of variables can be checked to determine if the program is doing what was intended; e.g., pressing A EXEC would determine the present value of the simple variable A.

While a program is stopped, STOP[†] can have another function. Pressing STOP displays STOP; then keying in either one line number or two line numbers separated by a comma — indicates that the calculator should stop program execution at these line numbers. Pressing CONT EXEC will then start program execution; e.g., STOP 80 EXEC then CONT EXEC or RUN EXEC will cause the program to halt at line number 80. Once a program is running under these conditions, there is one way to revert to normal program execution: After the program has halted, press STOP EXEC, then CONT EXEC, and the program will no longer stop at the given line numbers.

STEP

After a program is halted by a stop command, execution can continue by pressing STEP. STEP is always immediate execute; it causes the program to execute the appropriate statement and then to halt. Therefore, after each statement is executed, it can be checked to ensure that it performed the required function. When a particular function is considered satisfactory, either CONT or STEP can be pressed; CONT will execute the rest of the program while STEP will execute the next program statement only.

RUN, CONTINUE

Pressing RUN EXEC causes a program to begin execution at the first statement regardless of whether the program had previously been halted by a stop command; however, if the program had been halted by a stop command, pressing CONT EXEC will begin program execution where it had previously halted.

As previously mentioned, if a program is edited after it has been halted, program execution is reinitialized at the first line by RUN EXEC[†]. However, if, during the halt, the values of variables are changed or other internally-programmed conditions are changed, then CONT EXEC must be pressed to keep these newly adjusted conditions intact. The following things can be done to the program while it is halted if, upon completion, the program is executed by pressing CONT EXEC:

1. Variables can be changed; e.g., B = 5 EXEC sets the simple variable B equal to 5.
2. Angular units in trigonometric functions can be changed to measure in radians, degrees, or grads, depending on the user's requirements; e.g., DEG EXEC will assume all angles to be in degrees.
3. Write, print, and display statements can be input from the keyboard (these statements will be discussed later).
4. The data pointer, which indicates the next datum to be encountered, can be reset to the beginning of the data by typing RESTORE EXEC.
5. The program can go to a particular statement and be available for execution there; e.g., GO TO 80 EXEC sets the program line counter to 80 for either step-by-step or continuous execution. (If continuous execution is desired at line number 80, the continue command can be employed, as discussed below.) IF ... THEN can also access a particular line number.
6. Any calculator-keyboard statements can be executed.

A halted program can be executed beginning at any line number by using the continue command; e.g., CONT 95 EXEC will continue execution starting at line number 95.

A program can be run beginning at any line number; e.g., RUN 110 EXEC will begin program execution at line number 110.

[†]When a key is not being used as an immediate execute key, whether the key itself is pressed or whether the letters displayed on the key are individually keyed in, is generally arbitrary.

[†]The initialize command, to be discussed later, is also legitimate.

NOTE

The major difference between RUN and CONT is that RUN initializes all variables in the program and reverts to all normal program modes, while CONT neither affects any variables nor affects any current program modes.

PROGRAM STATEMENTS

LET

As in Basic LET A = 6 is a legitimate statement; however, the implied let statement is also allowed. Thus, A = 6 is the same as LET A = 6.

GO TO, GO SUB

The GO TO and GO SUB statements are the same as in Basic, however, each statement has one additional feature called respectively, the computed GO TO and the computed GO SUB. In either case an expression is evaluated and the rounded integer value of the expression is determined; the integer then acts as a pointer to a particular line number from a parameter listing in the statement. Some examples should explain this feature more clearly:

(In all examples, the present value of T will be 2.)

20 GO TO T*2—3 OF 250, 350, 450

Since the integer value of the expression is one, the first parameter following "OF" will be accessed (line number 250).

80 GO SUB T+2.5 OF 130, 260, 330, 370, 490

The rounded integer value of the expression is 5; therefore, the subroutine beginning at line number 490, the fifth parameter following "OF", will be accessed. (Decimal values of .5 and above are always rounded) to the next higher integer value.)

Any legitimate expression can be used; if the rounded value is either less than 1 or greater than the number of parameters following "OF", then the line number following the GO TO or GO SUB statement is executed.

PRINT

The calculator has one feature in the print statement not generally available in BASIC. Alphabetic information in a quote field can be printed in either upper or lower case letters. Printing in lower case is just the opposite of that on a regular typewriter; with shift or shift lock pressed, letters inside the quote field will be printed in lower case (the display, however, will still appear in upper case). If the "at" symbol (@) is required, press SHIFT RESULT.

Quote fields in both the format and write statements also have this "lower case" feature available.

DISPLAY

The display statement performs the same function as a print statement; the difference is that the information appears on the display rather than on a printing device. Thus, when a permanent record of the information is desired, the print statement should be used. The syntax used is DISP.

FORMAT, WRITE

The format statement is a means of structuring program printouts in a specified manner. The write statement defines the variables, constants, etc. that will appear on the printout; it also determines the device to be printed upon and the particular format statement to be followed. The following examples should adequately explain the features in both the format and the write statements:

e.g.,

5 FORMAT F10.2
6 WRITE (15, 5) .7

printout will be ^^^^^^0.70 where the "^" indicates a blank space.

23

5

10

15

F10.2 — the "F" refers to fixed-point format; "10" refers to the total field width reserved for the printout "2" refers to the number of digits to the right of the decimal point. Excess space to the right of the decimal point will be filled with zeros; one space is reserved for the decimal point; two spaces are reserved to the left of the decimal point, one for a digit preceding the decimal point, another for a sign (however, only minus signs are printed).

5

(15, 5) — in the write statement, the information within parentheses is required. "15" refers to the printing device to be used (select code 15 refers to the standard printer); "5" refers to the line number of the format statement that is being accessed (in this case line number 5). The format statement, which is being accessed, can appear anywhere in the program listing.

10

7 — in the write statement, the information following the right parenthesis is to be printed according to the designated formats. The value is always right-justified within its field width.

15

e.g.,

20 A = 62.4

25 FORMAT E12.2, F7.3, X, E8.1, /, 2F4.0, 3X, E9.0

20

260 WRITE (15, 25) 231, -61, -12.4, A, "A IS THE VALUE OF A", .4, A+3

20

printout will be:

^^^^2.31E+02-61.000^-1.2E+01

 ^^62.4 IS THE VALUE OF A^^^0^^^7.E+01

25

The format statement, the write statement, and the printout all have individual fields referenced; interlinking fields are represented by corresponding reference numbers.

25

Since the write statement references the format statement numbered 25, values in the write statement will be formatted as specified in this format statement.

30

REFERENCES

30

1 E12.2 indicates exponential (floating-point) notation with a field width of 12 and two digits to the right of the decimal point. The printout displays the exact form. Remember the field width must be large enough to include a leading sign, the decimal point and E±XY. Since printouts from the write statement are right justified and since "231" takes up only 8 of the 12 character field width, the four blank spaces are to the left of the value.

35

2 The value "-61" in the write statement totally fills the F7.3 field, therefore there is no space between this value and the previous value in the printout. Since F7.3 indicates three digits to the right of the decimal point, zeros are supplied in this case.

40

3 "X" indicates a space between values; therefore, the values supplied for the F7.3 and E8.1 formats will be separated in the printout by at least one space.

4 The value, -12.4 totally fills up E8.1, in fact, the last digit is suppressed since, with this field designation, only one digit can follow the decimal point.

34
24

5 5 “/” tells the printing device to skip one carriage return to the beginning of the next line.

6 5 2F4.0 specifies two consecutive fixed-point formats of F4.0 to be used for values in the write statement. The first format is for the value of A, which from program line number 20 is 62.4; when a fixed-point format specifies zero digits to the right of the decimal point, the value supplied is rounded to be an integer and the decimal point is suppressed — 62, in this case. Since the carriage return had previously been specified, this value is printed on the beginning of the next output line.

10 10 7 Quote fields are printed in the sequence in which they occur. Since this quote field is in the write statement, it is printed immediately after 62.4 is printed in F4.0 format. Note the space in front of “IS”; without this space, the printout would read 62IS instead of 62 IS. Quote fields can appear in either the format or write statements.

15 15 8 The value, .4 is also to be printed in F4.0 format. The rounded integer value of .4 is 0 — hence, the printout.

20 20 9 “3X” indicates there should be three spaces between the values supplied for the F4.0 and the E9.0 formats.

20 20 10 Expressions can be specified in write statements. The value of A+3 is 65.4; however, with this format, it will be rounded to 7.E+01. The decimal point is not suppressed in floating-point notation when zero is specified as the number of digits to the right of the decimal point.

25 25 If there are more values presented in the write statement than there are formats in the referenced format statement, the formats will be repeated; e.g.,

25 25 10 FORMAT F6.2, E10.2

25 25 20 WRITE (15, 10) 18, 21, 19.3, 29.6, .71

30 30 printout will be:

30 30 18.00 2.10E+01

30 30 19.30 2.96E+01

30 30 .71

35 35 After the first two values are printed according to the specified formats, the output printer's carriage return is activated, then two more values are printed according to the same two specified formats, etc.

35 35 Formatting Rules:

35 35 In fixed-point format, Fm.n, m stipulates total field width and n stipulates the number of digits to the right of the decimal point. If n>0, the minimum field width allowable is m = n + 3; e.g., F4.1 for a value of -0.6 would print -0.6, which takes up the total field width of 4. If n = 0, the minimum field width is m = 2; e.g., F2.0 for a value of -7 would print -7, thus taking up the allotted field width.

40 40 In floating-point format, Em.n, m and n are the same as in fixed-point format. However, the minimum field width allowable is always m = n + 7.

40 40 The following are all allowed in format statements:

45 45 Fm.n — fixed-point formats;

45 45 Em.n — floating-point formats (often called exponential or scientific notation);

45 45 X — space;

45 45 / — carriage return (for printing device);

45 45 “****” — quote field;

50 50 B — binary format (where write statement could have octal number, the binary equivalent would be output).

All of the above can be duplicated any number of times by leading the symbol with the appropriate number.

25

The following are all allowed in write statements: constants, variables, expressions, and quote fields. It should be noted that a write statement can be input from keyboard mode; that is, the write statement can reference the line number of a format statement in memory without being in the program itself.

5

The maximum width of both the fixed-point and the floating-point fields is 9,999. However, the programmer is effectively restricted by the allowable characters per line of the printing device.

5

10

The information in the format statement must be separated by commas. The information in the write statement must be separated either by commas or semicolons; generally it makes no difference. However, one additional feature of the write statement is that it can perform the identical operations as the print statement; the benefit is the ability of the write statement to select the device to be printed upon. To write on a punched tape photoreader with select code 2, the write statement could be set up in the following manner:

10

15

30 WRITE (2, *) A; B; C, D

15

The "*" indicates that no format statement is referenced; thus, WRITE acts like a PRINT statement: data will be left justified, semicolons pack the output fields, commas spread out the fields, etc.

20

P TAPE

As in BASIC, PTAPE causes the computer to read in a program from the punched tape photoreader. If the photoreader select code is 5, then pressing either PTAPE #5 or PTA #5 will perform this task unless no photoreader is connected to the calculator. If a photoreader is not connected, the calculator will wait till one is hooked up to complete the command. During this time the display will be blank.

20

25

During the implementation of this command, lines being loaded into memory have their syntax checked; if a line is in error, it will be rejected — thus, only those lines with correct syntax are loaded into the calculator. To obtain a record of the rejected lines, it is necessary to put the calculator in the print-all mode prior to pressing PTA #5; in print-all mode, all rejected lines are printed.

25

30

Note

To punch information onto paper tape, use the list command as discussed earlier.

30

35

MULTILINE FUNCTIONS

Multiline functions serve the same purpose as single-line functions with the added capability of being able to describe more sophisticated functions.

35

In single-line functions the general form of the defining function is:

statement number DEF FN single letter A to Z (simple variable) = expression

40

In multiline functions, the general form is the same aside from the equal sign and the expression; for in a multiline function, the expression can be spread out over many statement numbers. Thus, an extremely complicated expression can appear in a more simplified manner; additional flexibility is also gained in that the value of any variable within the expression can be computed for a given value of the argument of the function; e.g.,

40

10 W = .5
20 Y = 2
30 PRINT FNA (3)
40 STOP
50 DEF FNA (X)
60 Z = X+2 + Y+2
70 Q = Z + 3
80 D = Q/W
90 RETURN D
100 STOP

When this program is run, D will return a value of 32. The return statement returns the result of FNA (X) which, in this case, is equal to D. Line 40, the stop statement is needed to keep the calculator from trying to re-execute lines 50 through 90 after D is printed; without STOP in line 40, an error occurs in line 60 since a second pass beginning at line 50, would be made with X undefined.

36

26

1,444,145

26

Any variable evaluated in the expression can be returned by the return statement, and multiple return statements are allowed; e.g.,

10	X = 3	100	RETURN Z	
20	INPUT Y	110	Q = Z↑2	
5	30 WRITE (15,900) FNG (Y)	120	IF Q ≤ 100 THEN 150	5
	40 END	130	PRINT "Q ="	
	50 DEF FNG (Y)	140	RETURN Q	
	60 Y = Y + 1	150	PRINT "Z IS"	
	70 Z = Y↑2*X	160	RETURN Z	
10	80 IF Z ≤ 100 THEN 110	900	FORMAT F12.1	10
	90 PRINT "Z ="	1000	END	

In this example, a variable can be returned from three different lines (100, 140, 160) depending on the initial value of Y; Z can be returned from both lines 100 and 160 if the value of Y meets certain criteria.

15 Caution must be taken as to the placement of the statement that calls the function (in both examples, statement 30), and it is generally advisable to put a stop statement immediately after this statement (as in line 40 of both examples); otherwise, an undesirable loop may develop.

If correctly entered, the function of a function can be evaluated.

20 STOP, END

STOP was previously discussed, with emphasis on its program debugging capabilities. Now it will be discussed as a program statement, emphasizing the differences between it and the end statement.

25 When the program encounters a stop statement, it halts and is waiting; if CONT EXEC is then pressed, the program will continue with the statement following the STOP. This is not true if an end statement is encountered; the program will halt, but if CONT EXEC is pressed, the calculator will revert back to the lowest-numbered statement in memory. Therefore, the stop statement should be used between stacked programs that are to be run sequentially. When STOP is used in this manner, the values of simple variables can be passed from program to program.

A program should be terminated by encountering either a STOP or an END. The highest-numbered program statement need not be an end statement.

INPUTTING DATA

35 Program data can be input in three ways: the input statement, the read and data statements, and the initialize command. The input statement and the read and data statements are thoroughly discussed below.

INITIALIZE

40 The INITIALIZE key, when pressed, allocates storage space in memory for array variables. After the required data is keyed in, the program can be executed by pressing CONT EXEC; remember — RUN EXEC erases the values of all variables, thereby requiring all variables to be defined in the program itself.

45 Simple variables can always be input in keyboard mode without using the INITIALIZE key, as long as CONT EXEC is pressed to run the program. Since array variables can be input in keyboard mode by using the initialize command, it is not necessary to define any variables in the program itself. It is still necessary, however, to identify arrays in either a dimension or a common statement.

†The simple variable is a dummy variable which indicates where the actual argument of the function is used in the defining expression.

USER-DEFINABLE KEYS

There are ten User-Definable Keys (UDK) in the upper left-hand block of the keyboard. There are, however, effectively 20 accessible UDK's since each key can be accessed normally or with the shift key held down.

To enter UDK mode, press FETCH (particular UDK); the display will then read, KEY indicating the mode. To exit from UDK mode, press CLEAR END EXEC; the scratch command can also be used to exit from UDK mode — this command will, of course, erase certain information in the process. UDK mode is automatically exited when certain sequences are followed; these cases will be discussed later.

The user-definable keys can be used effectively in three ways:

1. to represent text (where text can be used as a typing aid);
2. to represent functions (where different values can be passed to the function);
3. To represent programs.

REPRESENTING TEXT

If a key represents text, merely pressing the key will immediately display the text without erasing anything that was previously on the display. Thus, commonly used words and phrases can be put on keys to serve as typing aids. Text can be put on a key in the following manner.

First, access a key by pressing FETCH (particular UDK). Then, press * followed by a character string† and finally EOL. Besides inputting the character string, pressing EOL in this sequence takes the user out of UDK mode. Any time the programmer wishes to use a character string, he must press the key into which the desired character string was input.

If, for example, a key was accessed by FETCH (particular UDK); then * FORMAT F10.2,X,E10.1 EOL was input. If subsequently program line number 60 needed this format, pressing 60 (particular UDK) EOL will put line number 60 into memory with the required format.

A typing-aid key can be used as an immediate execute command if an * is placed both in front of the text and following the text:

e.g., FETCH (UDK) * LOADDATA #4, 6, B * EOL

This command will be immediately executed whenever the UDK is pressed.

To use a key that has text, merely press the key. Pressing FETCH (particular UDK) will display the * with the text; however, text can be edited if the fetch command is used. Pressing FETCH and then * will erase the old character string and then wait for new text to be input.

REPRESENTING FUNCTIONS

A user definable key can be used to represent functions — either single or multiline. In either case, after the key is accessed, the function must be preceded by a line number — input either manually or automatically.

After a key has been accessed, the following function could be input:

10 DEF FNA (X) = 7 * X - 3 EOL

Whenever a value is to be passed to X (the argument of the function), first press the appropriate key; the display will read FNA. Then key in the appropriate value of the argument, which can be either a constant or an expression (e.g., 20), and press EXEC; the value of the function will then be displayed (in this case, 137). The same result could have been achieved by using the fetch command; however, FNA would not appear automatically on the display; it would have to be keyed in along with the argument; e.g., FETCH (UDK) FNA 20 EXEC would also display 137.

If a multiline function, DEF FNB (Z), has been input, pressing the appropriate UDK causes FNB to be displayed; as before, passing a value to the argument, Z, and then pressing EXEC will compute and display the value of the function.

†The maximum length for the character string (including the *) is 80 characters.

Functions in mainline memory and in a UDK can be called, regardless of the current operating mode.

If a function in the calculator is defined in more than one place, the first function found with the designated name will be accessed. If the user is in UDK mode, the calculator will search for the function in the following order:

1. The current UDK program will be checked.
2. The first line of each key (in the order defined) will be checked.
3. Mainline memory will be checked.

If the user is not in UDK mode, the order of the search will be steps 3 and 2, respectively.

REPRESENTING PROGRAMS

A UDK can be used to represent an entire program. Programming rules in UDK mode are consistent with those discussed above. There is one restriction, however; if a common statement is used, its size must be less than or equal to the size of the common statement in the mainline program — for there is only one common area allocated to memory.

To run a program that is represented in a UDK, it is advisable to press RUN (particular UDK) or FETCH (UDK), then INIT (particular UDK). The program can be continued merely by pressing the (particular UDK); but if there are array variables in the program, pressing only the key will cause these variables to be undefined (similar in this respect to the continue command, which neither destroys the old symbol table nor builds a new one). After executing the program, the calculator will exit from UDK mode.

Programs represented on a UDK generally use only simple variables for the obvious ease of handling.

To list program lines on a particular UDK, press:

LIST (particular UDK);

pressing LIST EXEC in the UDK mode will list the program lines on the key currently being accessed. To selectively list particular lines on the UDK, it is first necessary to FETCH a key; then use the list command as discussed below.

To load a program from a cassette file onto one particular key, first FETCH the key, then give the load command followed by EXEC. Program lines on a particular key can be stored onto the cassette in the same manner; text (as a typing aid), however, cannot be stored in this manner.

STORE KEY and LOAD KEY can be used for any UDK regardless of the information on the key. The use of these keys is discussed below.

STATEMENTS

This is a BASIC statement:

10 INPUT A,B,C,D,E

COMMENTS

A statement contains a maximum of 80 characters.
A statement may also be called a line.

STATEMENT NUMBERS

Each BASIC statement begins with a *statement number* (in this example, 20):

20 LET S=(A+B+C+D+E)/5

COMMENTS

The number is called a *statement number* or a *line number*. The statement number is chosen by you, the programmer. It may be any integer from 1 to 9999 inclusive.

Each statement has a unique *statement number*. The computer uses the numbers to keep the statements in order.

Statements may be entered in any order; they are usually numbered by fives or tens so that additional statements can be easily inserted. The computer keeps them in numerical order no matter how they are entered. For example, if statements are input in the sequence 30,10,20, the computer arranges them in the order: 10,20,30.

INSTRUCTIONS

The statement then gives an *instruction* to the computer (in this example, PRINT):

30 PRINT S

COMMENTS

Instructions are sometimes called *statement types* because they identify a type of statement. For example, the statement above is a "print" statement.

OPERANDS

If the instruction requires further details, *operands* (numeric details) are supplied (In this example, 10; on the previous page, "S"):

40 GO TO 10

COMMENTS

The *operands* specify what the instruction acts upon; for example, what is PRINTed, or where to GO.

A PROGRAM

The sequence of BASIC statements given on the previous pages is called a *program*.

10 INPUT A,B,C,D,E

20 LET S=(A+B+C+D+E)/5

30 PRINT S

The last statement in a program, as shown here, is an *END* statement.

40 GO TO 10

50 END

COMMENTS

The last (highest numbered) statement in a program must be an *END* statement.

The *END* statement informs the computer that the program is finished.

FREE-FORMAT LANGUAGE

BASIC is a "free format" language — the computer ignores extra blank spaces in a statement. For example, these three statements are equivalent:

```
30 PRINT S
30 PRINT  S
30PRINTS
```

COMMENTS

When possible, leave a space between words and numbers in a statement. This makes a program easier for people to read.

TERM: SIMPLE VARIABLE

DEFINED IN BASIC AS: A letter (from A to Z); or a letter immediately followed by a digit (from 0 to 9).

EXAMPLES:

A0	B
M5	C2
Z9	D

COMMENTS

Variables are used to represent numeric values. For instance, in the statement:
10 LET M5 = 96.7

M5 is a variable; 96.7 is the value of the variable M5. There is one other type of variable in BASIC, the array (subscripted) variables; its use is explained in Section IV.

TERM: NUMBER

DEFINED IN BASIC AS: A decimal number (the sign is optional) between 1E-99 and 9.999999999999999E+99

Zero is included in this range.

EXAMPLES:

-10008	5	3.14159	10E+37
126.257	0	10 E37	10E-37
16.01	.06784	-10 E37	1.0E+2

TERM: E NOTATION

DEFINED IN BASIC AS: A means of expressing numbers having more than six decimal digits, in the form of a decimal number raised to some power of 10.

EXAMPLES:

1.000000E+06 is equal to 1,000,000 and is read: "1 times 10 to the sixth power" (1×10^6).
 1.020000E+04 is equal to 10,200
 1.020000E-04 is equal to .000102

COMMENTS

"E" notation is used to print numbers having more than six significant digits. It may also be used for input of any number.

When entering numbers in "E" notation, leading and trailing zeroes may be omitted from the number; the + sign and leading zeroes may be omitted from the exponent.

The precision of numbers is 12 decimal digits.

TERM: EXPRESSION

DEFINED IN BASIC AS: A combination of variables, constants and operators which evaluates to a numeric value.

EXAMPLES: $(P + 5)/27$
(where P has previously been assigned a numeric value.)

$Q - (N + 4)$
(where Q and N have previously been assigned numeric values.)

TERM: ARITHMETIC EVALUATION

DEFINED IN BASIC AS: The process of calculating the value of an expression.

5

THE ASSIGNMENT OPERATOR

5

SYMBOL: =

EXAMPLES: 10 LET A = B2 = C = 0
20 LET A9 = C5
30 LET Y = (N-(P+5))/T
40 LET N5 = A + B2
50 LET P5 = P6=P7=A=B=98.6

GENERAL FORM: LET variable = expression

PURPOSE

Assigns an arithmetic or logical value to a variable.

COMMENTS

10

When used as an assignment operator, = is read "takes the value of," rather than "equals". It is, therefore, possible to use assignment statements such as:

10

LET X = X+2

15

This is interpreted by BASIC as: "LET X take the value of (the present value of) X, plus two."

Several assignments may be made in the same statement, as in statements 10 and 50 above.

15

See Section V, "Logical Operations" for a description of logical assignments.

RELATIONAL OPERATORS

SYMBOLS:

= # <> > < >= <=

EXAMPLES:

```

100 IF A=B THEN 900
110 IF A+B > C THEN 910
120 IF A+B < C+E THEN 920
130 IF C>=D*E THEN 930
140 IF C9<=G*H THEN 940
150 IF P2#C9 THEN 950
160 IF J <> K THEN 950

```

PURPOSE

Determines the logical relationship between two expressions, as

5

equality: =
 inequality: # or <>
 greater than: >
 less than: <
 greater than or equal to: >=
 less than or equal to: <=

5

10

10

COMMENTS

NOTE: It is not necessary for the novice to understand the nature of logical evaluation of relational operators, at this point. The comments below are for the experienced programmer.

15

15

Expressions using relational operators are logically evaluated, and assigned a value of "true" or "false" (the numeric value is 1 for "true," and 0 for false).

When the = symbol is used in such a way that it might have either an assignment or a relational function, BASIC assumes it is an assignment operator. For a description of the assignment statement using logical operators, see Section V, "Logical Operations."

20

20

ARITHMETIC OPERATORS

SYMBOLS:

↑ * / + -

EXAMPLES:

```

40 LET N1 = X-5
50 LET C2 = N13
60 LET A = (B-C)/4
70 LET X = ((P12)-(Y*X))/N+Q

```

PURPOSE

Represents an arithmetic operation, as:

25

exponentiate: ↑
 multiply: *
 divide: /
 add: +
 subtract: -

25

30

30

COMMENTS

The "-" symbol is also used as a sign for negative numbers. It is good practice to group arithmetic operations with parentheses when unsure of the exact order of precedence. The order of precedence (hierarchy) is:

35

35

↑
* /
+ -

with ↑ having the highest priority. Operators on the same level of priority are acted upon from left to right in a statement. See "Order of Precedence" in this Section for examples.

43

33

1,444,145

33

ARITHMETIC OPERATORS, Continued

The symbols + and - are also used to indicate unary plus and unary minus. For example, negative numbers may be expressed in a statement without using parenthesis:

5

```
10 LET A1 = -B
20 LET C2 = D ++E
30 LET B5 = B ---C
```

5

See "Order of Precedence" in this section for examples of how unary + and unary - are interpreted.

10

THE AND OPERATOR

10

SYMBOL:

AND

EXAMPLES:

```
60 IF A9 < B1 AND C # 5 THEN 100
70 IF T7 # T AND J = 27 THEN 150
80 IF P1 AND R > 1 AND N AND
  V2 THEN 10
90 PRINT X AND Y
```

PURPOSE

Forms a logical conjunction between two expressions. If both are "true," the conjunction is "true"; if one or both are "false," the conjunction is "false."

15

NOTE: It is not necessary for the novice to understand how this operator works. The comments below are for experienced programmers.

15

COMMENTS

The numeric value of "true" is 1, of "false" is 0.

20

All non-zero values are "true." For example, statement 90 would print either a 0 or a 1 (the logical value of the expression X AND Y) rather than the actual numeric values of X and Y.

20

Control is transferred in an IF statement using AND, only when all parts of the AND conjunction are "true." For instance, example statement 80 requires four "true" conditions before control is transferred to statement 10.

25

See Section V, "Logical Operations" for a more complete description of logical evaluation.

25

THE OR OPERATOR

SYMBOL:

OR

EXAMPLES:

```
100 IF A > 1 OR B < 5 THEN 500
110 PRINT C OR D
120 LET D = X OR Y
130 IF (X AND Y) OR (P AND Q)
  THEN 600
```

PURPOSE

Forms the logical disjunction of two expressions. If either or both of the expressions are true, the OR disjunction is "true"; if both expressions are "false," the OR disjunction is "false."

30

NOTE: It is not necessary for the novice to understand how this operator works. The comments below are for experienced programmers.

30

COMMENTS

The numeric values are: "true" = 1, "false" = 0.

35

All non-zero values are true; all zero values are false.

35

Control is transferred in an IF statement using OR, when either or both of the two expressions evaluate to "true."

40

See Section V, "Logical Operations" for a more complete description of logical evaluation.

40

THE NOT OPERATOR

SYMBOL:

NOT

EXAMPLES:

```
30 LET X = Y = 0
35 IF NOT A THEN 300
45 IF (NOT C) AND A THEN 400
55 LET B5 = NOT P
65 PRINT NOT (X AND Y)
70 IF NOT (A=B) THEN 500
```

PURPOSE

Logically evaluates the complement of a given expression.

NOTE: It is not necessary for the novice to understand how this operator works. The comments below are intended for experienced programmers.

COMMENTS

If $A = 0$, then $\text{NOT } A = 1$; if A has a non-zero value, $\text{NOT } A = 0$.

The numeric values are: "true" = 1, "false" = 0; for example, statement 65 above would print "1", since the expression $\text{NOT } (X \text{ AND } Y)$ is true.

Note that the logical specifications of an expression may be changed by evaluating the complement. In statement 35 above, if A equals zero, the evaluation would be "true" (1); since A has a numeric value of 0, it has a logical value of "false," making $\text{NOT } A$ "true."

See Section V, "Logical Operations" for a more complete description of logical evaluation.

ORDER OF PRECEDENCE

The order of performing operations is:

↑ **highest precedence**
NOT unary + unary -
 * /
 + -
Relational Operators
AND
OR **lowest precedence**

COMMENTS

If two operators are on the same level, the order of execution is left to right, for example:

$5 + 6 * 7$ is evaluated as: $5 + (6 * 7)$
 $7 / 14 * 2 / 5$ is evaluated as: $(7 / 14) * 2$
 5

Parentheses override the order of precedence in all cases, for example:

$5 + (6 * 3)$ is evaluated as: $5 + 18$
 and $3 + (6 + (2 * 2))$ is evaluated as: $3 + (6 + 4)$

Unary + and - may be used; the parentheses are assumed by BASIC. For example:

$A + + B$ is interpreted: $A + (+B)$
 $C - + D - 5$ is interpreted: $C - (+D) - 5$

Leading unary + signs are omitted from output by BASIC, but remain in program listings.

STATEMENTS

Statements are instructions to the calculator. They are contained in numbered lines within a program, and execute in the order of their line numbers. Statements cannot be executed without running a program. They tell the calculator what to do while a program is running.

Here are some examples mentioned in Section I:

LET
PRINT
INPUT

Do not attempt to memorize every detail in the "Statements" subsection; there is too much material to master in a single session. By experimenting with the sample programs and attempting to write your own programs, you will learn more quickly than by memorizing.

THE LET STATEMENTEXAMPLES:

10 LET A = 5.02
20 LET X = Y7 = Z = 0
30 LET B9 = 5* (X+2)
40 LET D = (3*C2+N)/(A*(N/2))

GENERAL FORM:

statement number LET variable = number or expression or variable ...

PURPOSE

Used to assign or specify the value of a variable. The value may be an expression, a number, or a variable.

COMMENTS

The assignment statement must contain:

1. A statement number,
2. LET is optional
3. The variable to be assigned a value (for example, B9 in statment 30 above),
4. The assignment operator, an = sign,
5. The number, expression or variable to be assigned to the variable (for example, 5*(X+2) in statement 30 above).

Statement 20 in the example above shows the use of an assignment to give the same value (0) to several variables. This is a useful feature for initializing variables in the beginning of a program.

REMEXAMPLES:

10 REM—THIS IS AN EXAMPLE
20 REM: OF REM STATEMENTS
30 REM——//*****!!!!
40 REM. STATEMENTS ARE NOT
EXECUTED BY BASIC

GENERAL FORM:

*statement number REM any remark or
series of characters*

PURPOSE

Allows insertion of a line of remarks or comment in the listing of a program.

COMMENTS

Must be preceded by a line number. Any series of characters may follow REM.

REM lines are part of a BASIC program and are printed when the program is listed or punched; however, they are ignored when the program is executing.

Remarks are easier to read if REM is followed by a punctuation mark, as in the example statements.

PRINT

This sample program gives a variety of examples of the PRINT statement. The results are shown below.

```

10 LET A=B=C=10
20 LET D1=E9=20
30 PRINT A,B,C,D1,E9
40 PRINT A/B,B/C/D1+E9
50 PRINT "NOTE THE POWER TO EVALUATE AN
  EXPRESSION AND PRINT THE"
60 PRINT "VALUE IN THE SAME STATEMENT."
70 PRINT
80 PRINT
90 REM* "PRINT" WITH NO OPERAND CAUSES THE
  TELEPRINTER TO SKIP A LINE.
100 PRINT "'A' DIVIDED BY 'E9' =":A/E9
110 PRINT "11111", "22222", "33333", "44444", "55555", "66666"
120 PRINT "11111", "22222", "33333", "44444", "55555", "66666"
130 END

```

-----RESULTS-----

```

RUN
10      10      10      20      20
1      20.05
NOTE THE POWER TO EVALUATE AN EXPRESSION AND
PRINT THE VALUE IN THE SAME STATEMENT.

'A' DIVIDED BY 'E9' = .5
11111      22222      33333      44444      55555
66666
111112222233333444445555566666

```

NOTE: The "." and ";" used in statements 110 and 120 have very different effects on the format.

GENERAL FORM:

```

statement number PRINT expression , expression , ...
or
statement number PRINT "any text" ; expression ; ...
or
statement number PRINT "text" ; expression ; "text" , "text" , ...
or
statement number PRINT any combination of text and/or expressions
or
statement number PRINT

```

PURPOSE

Causes the *expressions* or "*text*" to be output to the PRINTER.
Causes the printer to skip a line when used without an operand.

COMMENTS

Note the effects of , and ; on the output of the sample program. If a comma is used to separate PRINT operands, five fields are printed per printer line. If semicolon is used, up to twelve "packed" numeric fields are output per printer line (72 characters).

Text in quotes is printed literally.

47

5

5

or

where the variables represent numeric expressions.

10

10

15

15

50 GO TO 30

1	2	3	4	5	6	7	8	9	10	11	12
13	14	15	16	17	18	19	20	21	22	23	24

20

20

25

25

50 GO TO 30

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

30

30

A PRINT statement without an operand (statements 70 and 80 in the sample program) generates a *return linefeed*.

78

GO TO AND MULTIBRANCH GO TO

EXAMPLES:

```
10 LET X = 20
:
40 GO TO X+Y OF 410,420,430
50 GOTO 100
80 GOTO 10
90 GO TO N OF 100,150,180,190
```

GENERAL FORM:

statement number GO TO statement number
statement number GO TO expression OF sequence of statement numbers

PURPOSE

GO TO transfers control to the statement specified.

GO TO *expression*... rounds the *expression* to an integer *n* and transfers control to the *n*th statement number following OF.

COMMENTS

GO TO may be written GOTO or GO TO.

Must be followed by the statement number to which control is transferred, or *expression* OF, and a sequence of statement numbers.

GO TO overrides the normal execution sequence of statements in a program.

If there is no statement number corresponding to the value of the *expression*, the GO TO is ignored.

Useful for repeating a task infinitely, or "jumping" (GOing TO) another part of a program if certain conditions are present.

GO TO should not be used to enter FOR-NEXT loops; doing so may produce unpredictable results or fatal errors.

GOSUB...RETURN

EXAMPLE:

```
50 READ A2
60 IF A2<100 THEN 80
70 GOSUB 400
:
380 STOP (STOP frequently precedes the
      first statement of a subroutine,
      to prevent accidental entry.)
390 REM--THIS SUBROUTINE ASKS
      FOR A 1 OR 0 REPLY.
400 PRINT "A2 IS>100"
410 PRINT "DO YOU WANT TO
      CONTINUE";
420 INPUT N
430 IF N #0 THEN 450
440 LET A2=0
450 RETURN
:
600 END
```

GENERAL FORM:

statement number GOSUB statement
number starting subroutine
:
statement number RETURN

PURPOSE

GOSUB transfers control to the specified statement number.

RETURN transfers control to the statement following the GOSUB statement which transferred control.

GOSUB...RETURN eliminates the need to repeat frequently used groups of statements in a program.

MULTIBRANCH GOSUB

EXAMPLES:

```

20 GOSUB 3 OF 100,200,300,400,
   500
60 GOSUB N+1 OF 200,210,220
70 GOSUB N OF 80,180,280,380,
   480,580

```

GENERAL FORM:

statement number GOSUB expression OF sequence of statement numbers...

PURPOSE

GOSUB *expression* rounds the *expression* to an integer *n* and transfers control to the *n*th statement number following OF.

COMMENTS

Subroutines should be exited only with a RETURN statement.

The *expression* indicates which of the specified subroutines will be executed. For example, statement 20, above transfers control to the subroutine beginning with statement 300. The *expression* specifies which statement in the sequence of five statements is used as the starting one in the subroutine.

The *expression* is evaluated as an integer. Non-integer values are rounded to the nearest integer.

If the *expression* evaluates to a number greater than the number of statements specified, or less than 1, the GOSUB is ignored.

Statement numbers in the sequence following OF must be separated by commas.

IF ... THEN

SAMPLE PROGRAM:

```

10 LET N = 10
20 READ X
30 IF X <= N THEN 60
40 PRINT "X IS OVER"; N
50 GO TO 100
60 PRINT "X IS LESS THAN OR
   EQUAL TO"; N
70 GO TO 20
80 STOP

```

GENERAL FORM:

statement number IF expression relational op expression THEN statement number

PURPOSE

Transfers control to a specified statement if a specified condition is true.

COMMENTS

Sometimes described as a conditional transfer; "GO TO" is implied by IF...THEN, if the condition is true. In the example above, if $X \leq 10$, the message in statement 60 is printed (statement 60 is executed).

Since numbers are not always represented exactly in the computer, the = operator should be used carefully in IF...THEN statements. Limits, such as $\leq$, $\geq$, etc. should be used in an IF expression, rather than =, whenever possible.

If the specified condition for transfer is not true, the program will continue executing in sequence. In the example above, if $X > 10$, the message in statement 40 prints.

The relational operator is optional in logical evaluations.

See Section V, "Logical Operations," for a more complete description of logical evaluation.

FOR ... NEXT

EXAMPLES:

```

100 FOR P1 = 1 TO 5
110 FOR Q1 = N TO X
120 FOR R2 = N TO X STEP 2.5
130 FOR S = 1 TO X STEP '
140 NEXT S
150 NEXT R2
160 NEXT Q1
170 NEXT P1

```

Sample Program — Variable Number Of Loops

```

40 PRINT "HOW MANY TIMES DO YOU WANT TO LOOP";
50 INPUT A
60 FOR J = 1 TO A
70 PRINT "THIS IS LOOP"; J
80 READ N1, N2, N3
90 PRINT "THESE DATA ITEMS WERE READ:" N1; N2; N3
100 PRINT "SUM ="; (N1+N2+N3)
110 NEXT J
120 DATA 5, 6, 7, 8, 9, 10, 11, 12
130 DATA 13, 14, 15, 16, 17, 18, 19, 20, 21
140 DATA 22, 23, 24, 25, 26, 27, 28, 29, 30
150 DATA 31, 32, 33, 34
160 END

```

GENERAL FORM:

statement number **FOR** simple variable = initial value **TO** final value

or

statement no. **FOR** simple var. = initial value **TO** final value **STEP** step value

statement number **NEXT** simple variable

NOTE: The same simple variable must be used in both the **FOR** and **NEXT** statements of a loop.

PURPOSE

Allows controlled repetition of a group of statements within a program.

COMMENTS

Initial value, final value and step value may be any expression.

STEP and *step value* are optional; if no step value is specified, the computer will automatically increment by one each time it executes the loop.

How the loop works:

The simple variable is assigned the value of the *initial value*; the value of the simple variable is increased by 1 (or by the *step value*) each time the loop executes. When the value of the *simple variable* passes the *final value*, control is transferred to the statement following the "NEXT" statement.

The *initial*, *final*, and *step values* are all evaluated upon entry to the loop and remain unchanged after entry. For example,

FOR I = 1 TO I + 5

goes from 1 to 6; that is, the final value does not "move" as I increases with each pass through the loop.

For further details on the STEP feature, see "FOR ... NEXT with STEP" in Section III.

Try running the sample program if you are not sure what happens when FOR ... NEXT loops are used in a program.

NESTING FOR...NEXT LOOPS

Several FOR...NEXT loops may be used in the same program; they may also be nested (placed inside one another). There are two important features of FOR...NEXT loops:

1. FOR...NEXT loops may be nested.

Range of loop A1 → 10 FOR A1 = 1 TO 5
Range of loop B2 → 20 FOR B2 = N TO P
Range of loop C3 → 30 FOR C3 = X TO Y STEP R
80 NEXT C3
90 NEXT B2
100 NEXT A1

2. The range of FOR...NEXT loops may not overlap. The loops in the example above are nested correctly. This example shows improper nesting.

The range of loops
I and J overlap

10 FOR I = 1 TO 5
30 FOR J = 1 TO N
50 NEXT I
90 NEXT J

READ, DATA AND RESTORE

Sample Program using Read and Data

```

15 FOR I=1 To 5
20 READ A
40 LET X=A^2
45 PRINT A; " SQUARED =";X
50 NEXT I
55 DATA 5.24,6.75,30.8,72.65,89.72
60 END

```

5 Each data item may be read only once in this program. TSB keeps track of data with a "pointer." When the first READ statement is encountered, the "pointer" indicates that the first item in the first DATA statement is to be read; the pointer is then moved to the second item of data, and so on.

10 In this example, after the loop has executed five times, the pointer remains at the end of the data list. To reread the data, it is necessary to reset the pointer. A RESTORE statement moves the pointer back to the first data item.

Sample Program Using READ, DATA and RESTORE

```

20 FOR I=1 To 5
30 READ A
40 LET X=A^2
50 PRINT A; "SQUARED =";X
60 NEXT I
80 RESTORE
100 FOR J=1 To 5
110 READ B
120 LET Y=B^4
130 PRINT B; "TO THE FOURTH POWER =";Y
140 NEXT J
150 DATA 5.24,6.75,30.8,72.65,89.72
160 END

```

GENERAL FORM:

statement number READ variable , variable , ...
statement number DATA number or string , number or string , ...
statement number RESTORE
statement number RESTORE statement number

PURPOSE

15 The READ statement instructs TSB to read an item from a DATA statement. The DATA statement is used for specifying data in a program. The data is read in sequence from first to last DATA statements, and from left to right within the DATA statement.

The RESTORE statement resets the pointer to the first data item, allowing data to be re-read.

20 RESTORE followed by a statement number resets the pointer to the first data item, beginning at the specified statement.

COMMENTS

25 READ statements require at least one DATA statement in the same program. Items in a DATA statement must be separated by commas. String and numeric data may be mixed.

DATA statements may be placed anywhere in a program. The data items will be read in sequence as required.

DATA statements do not execute; they merely specify data.

30 The RUN command automatically sets the pointer to the first data item. If you are not sure of the effects of READ, DATA, and RESTORE, try running the sample programs.

WAIT

EXAMPLE: 900 WAIT (1000)
990 WAIT (3000)

GENERAL FORM: *statement number* **WAIT** (*expression*
max. value of 32767)

PURPOSE

Introduces delays into a program. WAIT causes the program to wait the specified number of milliseconds (maximum 32767 milliseconds) before continuing execution.

COMMENTS

The time delay produced by WAIT is not precisely the number of milliseconds specified because there is no provision to account for time elapsed during calculation or terminal-computer communication.

One millisecond = 1/1000 second.

TERM: ROUTINE

DEFINED IN BASIC AS: A sequence of program statements which produces a certain result.

PURPOSE

Routines are used for frequently performed operations, saving the programmer the work of defining an operation each time he uses it, and saving computer memory space.

COMMENTS

A routine may also be called a program, subroutine, or sub-program. The task performed by a routine is defined by the programmer.

TERM: STRING

DEFINED IN BASIC AS: 0 to 255 printer characters enclosed by quotation marks (one line on a teleprinter terminal).

COMMENTS

Sample strings: "ANY CHARACTERS!*/_—" "
"TEXT 1234567..."

Quotation marks may not be used within a string. Strings are used only in PRINT statements.

The statement number PRINT and quotation marks are not included in the 65 character count. Each statement may contain up to 72 characters. Maximum string length is 72 characters minus 6 characters for "PRINT", two for the quotation marks, and the number of characters in the statement number.

TERM: FUNCTION

DEFINED IN BASIC AS: The mathematical relationship between two variables (X and Y, for example) such that for each value of X there is one and only one value of Y.

COMMENTS

The independent variable in a function is called an argument; the dependent variable is the function value. For instance, if X is the argument, the function value is the square root of X, and Y takes the value of the function.

TERM: ARRAY OR MATRIX

DEFINED IN BASIC AS: An ordered collection of numeric data (numbers).

COMMENTS

Arrays are divided into columns (vertical) and rows (horizontal):

C	ROWS
O	
L	
U	
M	
N	
S	

Arrays may have one or two dimensions. For example,

1.0
2.1
3.2
4.3

is a one-dimensional array, while

6 , 5 , 4
3 , 2 , 1
0 , 9 , 8

is a two-dimensional array.

Array elements are referenced by their row and column position. For instance, if the two examples above were arrays A and Z respectively, 2.1 would be A(2); similarly, 0 would be Z(3,1). The references to array elements are called subscripts, and set apart with parentheses. For example, P(1,5) references the fifth element of the first row of array P; 1 and 5 are the subscripts. In X(M,N) M and N are the subscripts.

TERM: WORD

DEFINED IN BASIC AS: The amount of computer memory space occupied by the two teleprinter characters.

COMMENTS

Numbers require two words of memory space when stored as numbers. When used within a string, numbers require 1/2 word of space per character in the number.

SUBROUTINES AND FUNCTIONS

The following pages explain BASIC features useful for repetitive operations — subroutines, programmer-defined functions and standard functions.

The programmer-defined features, such as GOSUB, FOR...NEXT with STEP, and DEF FN become more useful as the user gains experience and learns to use them as shortcuts.

Standard mathematical and trigonometric functions are convenient timesavers for programmers at any level. They are treated as numeric expressions by BASIC.

FOR...NEXT WITH STEP

EXAMPLES:

```
20 FOR I5 = 1 TO 20 STEP 2
40 FOR N2 = 0 TO -10 STEP -2
80 FOR P = 1 TO N STEP X5
90 FOR X = N TO W STEP
   (N+2-V)
   :
```

GENERAL FORM:

statement no. **FOR** *simple var.* = *expression* **TO** *expression* **STEP** *expression*

PURPOSE

Allows the user to specify the size of the increment of the FOR variable.

COMMENTS

The step size need not be an integer. For instance,

```
100 FOR N = 1 TO 2 STEP .01
```

is a valid statement which produces approximately 100 loop executions, incrementing N by .01 each time.

A step size of 1 is assumed if STEP is omitted from a FOR statement.

A negative step size may be used, as shown in statement 40 above.

GENERAL MATHEMATICAL FUNCTIONS

EXAMPLES:

```
642 PRINT EXP(N); ABS(N)
652 IF RND (0) >=.5 THEN 900
662 IF INT (R) # 5 THEN 910
672 PRINT SQR (X); LOG (X)
```

GENERAL FORM:

The general mathematical functions may be used as expressions, or as parts of an expression.

PURPOSE

Facilitates the use of common mathematical functions by pre-defining them as:

ABS (*expression*) the absolute value of the expression;
EXP (*expression*) the constant *e* raised to the power of the expression value (in statement 642 above, *e*^N)

INT (*expression*) the largest integer ≤ the expression;

LOG (*expression*) the logarithm of the positively valued expression to the base *e*;

RND (*expression*) a random number between 1 and 0; the expression is a dummy argument;

SQR (*expression*) the square root of the positively valued expression.

COMMENTS

The RND function is restartable; the sequence of random numbers using RND is identical each time a program is RUN.

TRIGONOMETRIC FUNCTIONS

EXAMPLES:

```
500 PRINT SIN(X); COS(Y)
510 PRINT 3*SIN(B); TAN (C2)
520 PRINT ATN (22.3)
530 IF SIN (A2) <1 THEN 800
540 IF SIN (B3) = 1 AND SIN(X) <1
    THEN 90
```

PURPOSE

Facilitates the use of common trigonometric functions by pre-defining them, as:

SIN (*expression*) the sine of the expression
 COS (*expression*) the cosine of the expression
 TAN (*expression*) the tangent of the expression
 ATN (*expression*) the arctangent of the expression

COMMENTS

The function is of the value of the expression (the value in parentheses, also called the argument).

The trigonometric functions may be used as expressions or parts of an expression.

The angle of the trigonometric functions can be specified as radians, degrees, or grads by executing a RAD, DEG, or GRAD statement. The calculator assumes radians if not specified.

THE TAB AND SGN FUNCTIONS

EXAMPLES:

```
500 IF SGN (X) # 0 THEN 800
510 LET Y = SGN(X)
520 PRINT TAB (5); A2; TAB (20)
    "TEXT"
530 PRINT TAB (N),X,Y,Z2
540 PRINT TAB (X+2) "HEADING",
    R5
```

GENERAL FORM:

The TAB and SGN may be used as expressions or parts of an expression. The function forms are:
 or parts of an expression. The function forms are:

TAB (*expression indicating number of spaces to be moved.*)
SGN (*expression*)

PURPOSE

TAB (*expression*) is used only in a PRINT statement, and causes the terminal typeface to move to the space number specified by the expression (0 to 71). The *expression* value after TAB is rounded to the nearest integer. Expression values greater than 71 cause a return linefeed to be generated.

SGN (*expression*) returns a 1 if the expression is greater than 0, returns a 0 if the expression equals 0, returns a -1 if the expression is less than 0.

WHAT WE CLAIM IS:—

1. Electronic data processing apparatus comprising:
 keyboard input means for entering one or more lines of one or more alphanumeric characters each into said electronic data processing apparatus;

TRIGONOMETRIC FUNCTIONS

EXAMPLES:

```
500 PRINT SIN(X); COS(Y)
510 PRINT 3*SIN(B); TAN (C2)
520 PRINT ATN (22.3)
530 IF SIN (A2) <1 THEN 800
540 IF SIN (B3) = 1 AND SIN(X) <1
    THEN 90
```

PURPOSE

Facilitates the use of common trigonometric functions by pre-defining them,

as:
 SIN (*expression*) the sine of the expression
 COS (*expression*) the cosine of the expression
 TAN (*expression*) the tangent of the expression
 ATN (*expression*) the arctangent of the expression.

COMMENTS

The function is of the value of the expression (the value in parentheses, also called the argument).

The trigonometric functions may be used as expressions or parts of an expression.

The angle of the trigonometric functions can be specified as radians, degrees, or grads by executing a RAD, DEG, or GRAD statement. The calculator assumes radians if not specified.

THE TAB AND SGN FUNCTIONS

EXAMPLES:

```
500 IF SGN (X) # 0 THEN 800
510 LET Y = SGN(X)
520 PRINT TAB (5); A2; TAB (20)
    "TEXT"
530 PRINT TAB (N),X,Y,Z2
540 PRINT TAB (X+2) "HEADING",
    R5
```

GENERAL FORM:

The TAB and SGN may be used as expressions or parts of an expression. The function forms are:
 or parts of an expression. The function forms are:

TAB (*expression indicating number of spaces to be moved*)
SGN (*expression*)

PURPOSE

TAB (*expression*) is used only in a PRINT statement, and causes the terminal typeface to move to the space number specified by the expression (0 to 71). The *expression* value after TAB is rounded to the nearest integer. Expression values greater than 71 cause a return linefeed to be generated.

SGN (*expression*) returns a 1 if the expression is greater than 0, returns a 0 if the expression equals 0, returns a -1 if the expression is less than 0.

WHAT WE CLAIM IS:—

1. Electronic data processing apparatus comprising:
 keyboard input means for entering one or more lines of one or more alphameric characters each into said electronic data processing apparatus;



(ST)

keyboard buffer means, coupled to said keyboard input means, for storing a line of one or more alphameric characters entered into said electronic data processing apparatus;

read-write memory means, coupled to said keyboard input means and keyboard buffer means, for storing one or more lines of more than three alphameric characters each entered into said electronic data processing apparatus;

said keyboard input means including a plurality of keys, each representative of a single alphabetic or numeric character, for entering a representation of every alphabetic character of a selected alphabet and every numeric character from zero to nine into said electronic data processing apparatus;

said electronic data processing apparatus including read-only memory means storing routines and subroutines for recognizing particular strings of alphameric characters entered into said electronic data processing apparatus as being commands; and

said electronic data processing apparatus including processing means, coupled to said keyboard input means, read-write memory means, and read-only memory means, for selectively performing the routines and subroutines stored in said read-only memory means and for executing the commands recognized thereby.

2. Electronic data processing apparatus as in Claim 1 wherein said keyboard input means includes one or more additional keys each representative of a fixed string of alphameric characters representing a command.

3. Electronic data processing apparatus according to claim 1 wherein:

said keyboard input means further includes a plurality of definable keys, and control means for associating designated alphameric characters with a designated definable key as defined by the user;

said read-write memory means is arranged to store groups of alphameric characters associated with said definable keys;

said electronic data processing apparatus further comprises:

display means, coupled to said keyboard buffer means, for visually displaying the contents thereof; and

said processing means further comprises:

logic means coupled to said keyboard input means, said read-write memory means, and keyboard buffer means, said logic means being responsive to actuation of said control means, a selected one of said definable keys, and selected ones of said alphameric keys for associating the alphameric characters designated by said selected ones of said alphameric keys with the selected one of said definable keys and for storing those alphameric characters in said read-write memory means, said logic means thereafter being responsive to actuation of the selected definable keys for transferring the alphameric characters associated therewith from said read-write memory means to said keyboard buffer means.

4. Electronic data processing apparatus according to Claim 3 wherein:

said keyboard input means includes a key for designating a line termination character that may also be associated with the selected definable key; and

said logic means is responsive to actuation of a definable key, whose associated group of alphameric characters includes the line termination character, for immediately terminating the group of alphameric characters associated with that definable key.

5. Electronic data processing apparatus according to either one of Claims 3 and 4 wherein:

the alphameric characters associated with any of the definable keys can represent an executable command; and

said logic means is responsive to actuation of such a definable key for immediately executing the command represented by the alphameric characters associated with that definable key.

ERIC POTTER & CLARKSON,
Kingsway House,
Kingsway,
London, WC2 B6 QX.

