TEXAS INSTRUMENTS

TM 990

UNIVERSITY BASIC

User's Manual

MICROPROCESSOR SERIES™

PREFACE

The University BASIC User's Manual describes Texas Instruments' University BASIC which is an interactive programming language used with the TM990/189 University Board microcomputer.

The following manuals present additional information relative to the use of University BASIC, the TM990/189 microcomputer, and the POWER BASIC family of software products.

- POWER BASIC Reference Manual, MP308, 1602061-9701
- TM990 POWER BASIC Elementary Tutorial Manual, MP311
- TM990/189 Microcomputer User's Guide, MPB06, 1602004-9701
- 9900 Family Systems Design and Data Book, LCC4400, 97049-118-NI
- TM990 Introduction To Microprocessors: Hardware and Software, MPB30, 1602008-9701
- Software Development Handbook, MPA29
- Color Video Using TMS9918 and University BASIC Application Report, MP723

TABLE OF CONTENTS

APPENDICES

LIST OF ILLUSTRATIONS

LIST OF TABLES

SECTION I

OVERVIEW

1.1 GENERAL

University BASIC* is a member of the POWER BASIC** family of software products. Designed as a training tool for students and engineers wishing to become familiar with the POWER BASIC language and TM990 modules, University BASIC offers the user an excellent opportunity to learn through "hands-on" experience. The TM990/189 microcomputer is a standalone system that can be used as an aid in learning microcomputer fundamentals and interfacing, as well as demonstrating the 9900 family 16-bit architecture.

This chapter provides general information concerning the use of University BASIC and conventions employed in the manual. For further information concerning other POWER BASIC family members, consult the TM990 POWER BASIC Reference Manual, MP308.

1.2 University BASIC

University BASIC requires the TM990/189 CPU board and a suitable power supply such as the TM990/519, and will support a single user on an RS-232-C compatible terminal. University BASIC is an integer BASIC that supports many (but not all) POWER BASIC commands and statements. It includes several found in this product only, such as the TONE command which allows sound capability. Color video commands are also available, and are dealt with in detail in the Color Video Using TMS9918 and University BASIC Application Report, MP723. Installation procedures for University BASIC may be found in Section II of this manual, with additional information provided in the TM990/189 Microcomputer User's Guide. Figure 1-1 on the following page shows a typical University BASIC system configuration.

1.3 INTERACTION WITH University BASIC

Interaction with University BASIC involves user input of a series of program statements and commands and user responses to program-generated requests for input. The user may enter program statements or invoke commands required to examine, debug, or run the program. Each statement or command is completed by entering a carriage return.

---

*Trademark of Texas Instruments
**Trademark of Texas Instruments

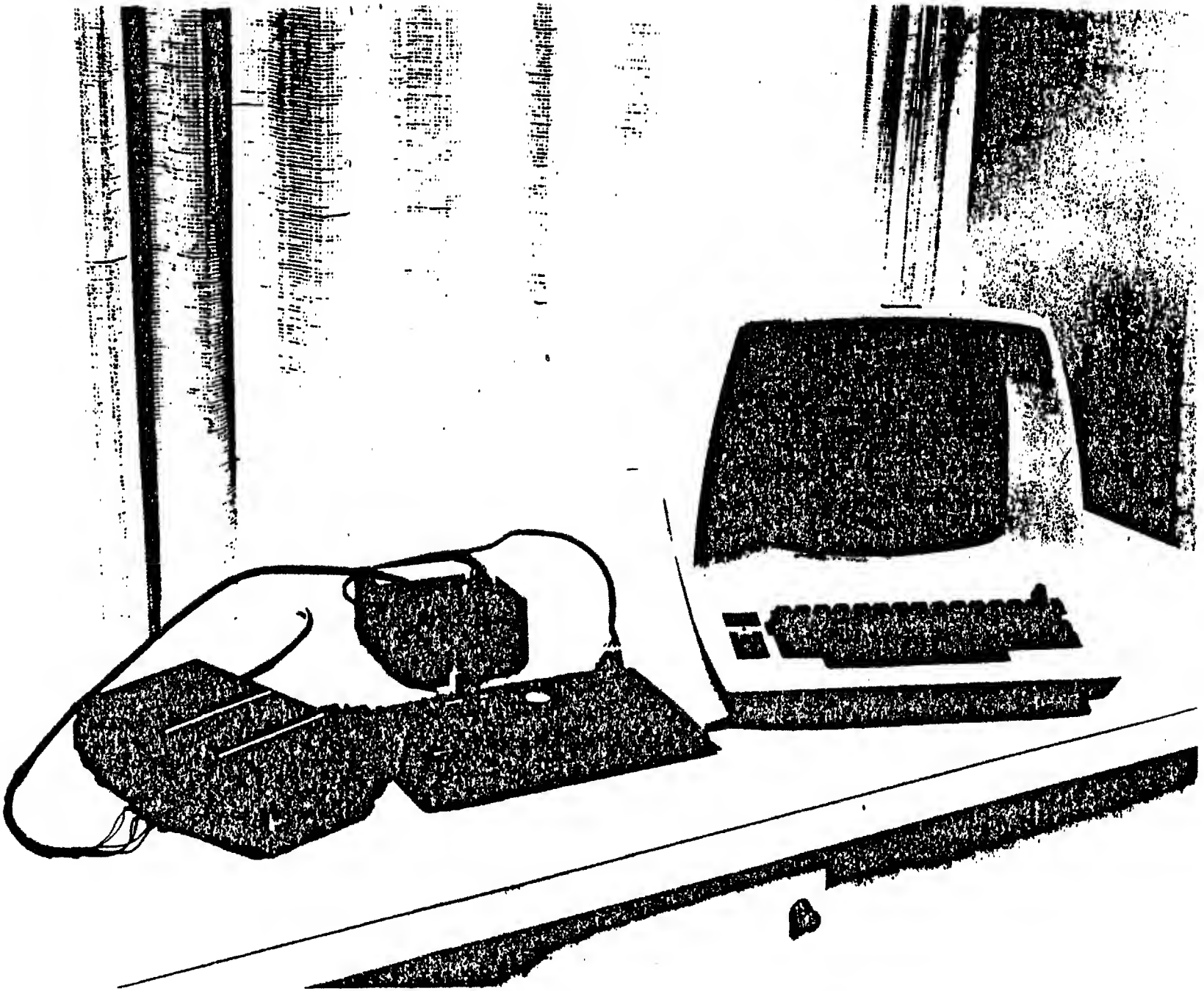


FIGURE 1-1. TYPICAL UNIVERSITY BASIC SYSTEM CONFIGURATION

The carriage return terminates and enters the line, advances one line, and waits for further keyboard input. Each program statement is stored as it is entered, and the program may be listed at any time during its generation or at its completion by using the LIST command. At any time the user may halt execution or terminate a statement/command by striking the escape key on a terminal, or the Shift 9 key on the TM990/189 board keypad.

1.4 CONVENTIONS USED IN THIS MANUAL

The following conventions are used to describe the statements, commands, and examples in this manual:

Numeric values for command parameters are integers unless otherwise specified.

Angle brackets (<>) indicate essential elements of user-supplied data in statements, commands, and examples:

```
10 LET <variable> = <expression>
```

Braces ({}) indicate a choice between two or more possibilities (alternative items), one of which must be included.

```
10 {PRINT} <expression>
   {  ;  }
```

Brackets ([]) enclose optional items.

```
[10]  [LET] A=3:B=4
```

Items in capital letters must be entered exactly as shown.

Items in lower-case letters are user-supplied characters.

1.5 PACKAGING AND RELEASE MEDIA

University BASIC is released pre-programmed in EPROM which may be directly installed in the EPROM sockets of the TM990/189 board. The release package consists of:

- 2 EPROM's containing the University BASIC code:
  - One 2532 EPROM (4K x 8 bytes)
  - One 2715 EPROM (2K x 8 bytes)
- The University BASIC User's Manual
- Color Video Using TMS9918 and University BASIC Application Report

Figure 1-2 depicts the system memory map.

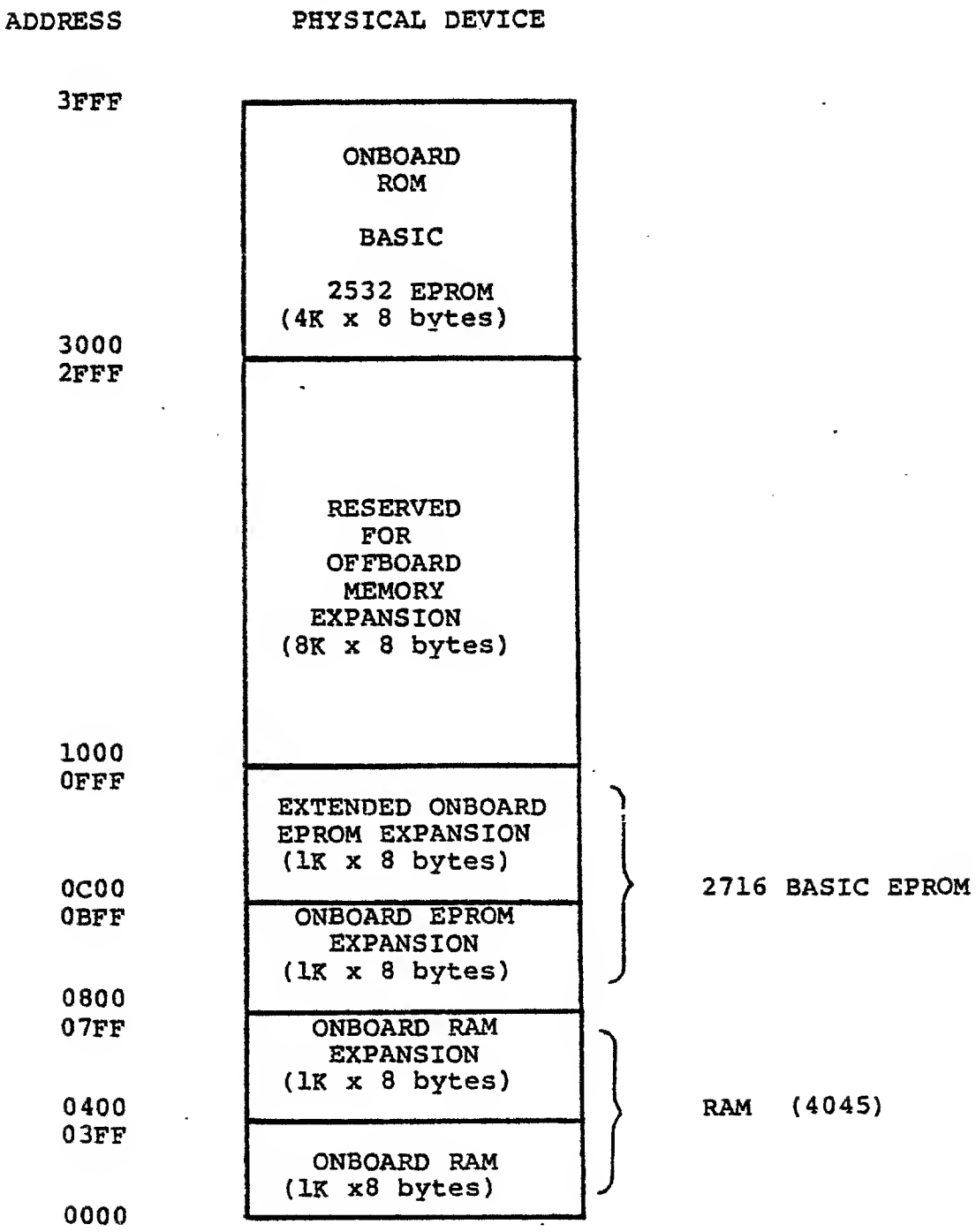


FIGURE 1-2. SYSTEM MEMORY MAP

1.6 MANUAL ORGANIZATION

The manual is organized into three major segments, the first containing an introduction to University BASIC and a discussion of the installation procedures as well as a simple example program. The second segment contains a functional description of University BASIC as well as general programming information that describes conventions of the language. The third segment covers statement syntax and provides specific examples along with more advanced demonstration programs.

SECTION II

INSTALLATION

2.1 GENERAL

System configuration and University BASIC EPROM insertion will be covered in this section, and a simple example program provided to verify system operation. Refer to the TM990/189 Microcomputer User's Guide for instructions on setting-up and initial operation of the TM9900/189 microcomputer.

2.2 MINIMUM CONFIGURATION

The minimum configuration for University BASIC includes:

- TM990/189 Microcomputer
- Power supply (TM990/519 or equivalent)

2.3 MAXIMUM CONFIGURATION

The maximum configuration may include:

- Any EIA RS-232-C or 20 mA or TTY compatible data terminal. (e.g., Lear Siegler, ASR745, 763 etc.)
- On-board memory expansion of up to 1k bytes (RAM)
- Off-board memory expansion through the Bus Expansion Interface
- Audio Cassette Interface

The TM990/189 K1 I/O expansion kit provides the necessary components to add options to the TM990/189 microcomputer. These options include:

- Asynchronous Communication port
- On-board relay for audio cassette interface
- Off-board CRU expansion

2.4 EPROM INSERTION

The user should carefully remove the TMS4732 (UNIBUG) EPROM from the socket marked U33 and replace it with the TMS 2532 EPROM.

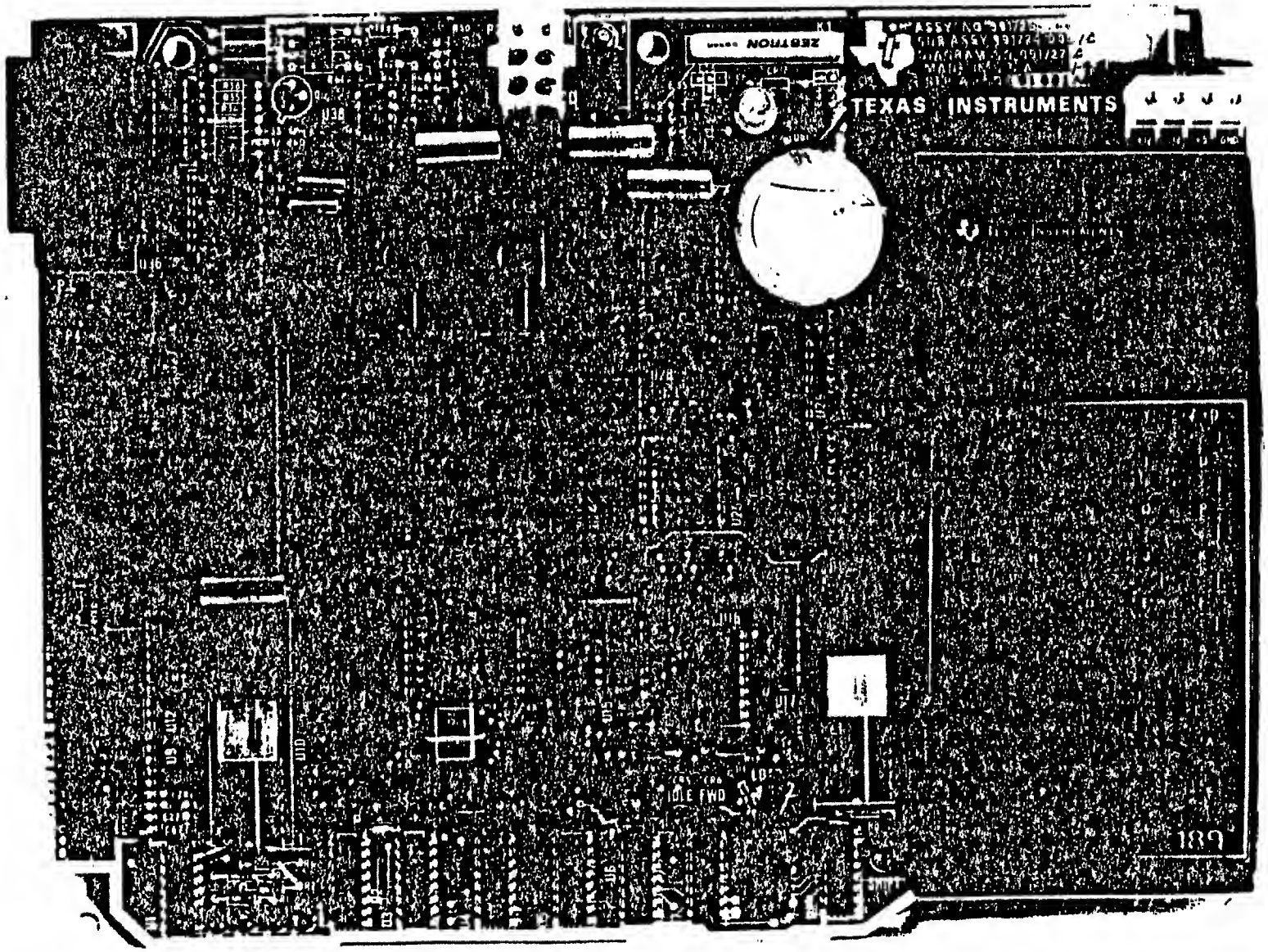


FIGURE 2-1.^^UNIVERSITY BASIC ON THE TM990/189 MICROCOMPUTER

After following the modification procedure found in Section 9, page 9-1 of the TM990/189 Microcomputer User's Guide, install the TMS 2716 EPROM in the socket marked U32. Refer to Figure 2-1 for appropriate socket markings.

2.5 INPUT/OUTPUT OPTIONS

2.5.1 EIA Communications

Devices that are RS-232-C compatible may be added to provide asynchronous serial communications between the chosen device and the TM990/189 microcomputer through installation of the Serial Communications Interface components listed in Table 2-1. The interface resides at CRU base address >0800 through >083E. The schematics in Appendix A, page A-10 of the TM990/189 Microcomputer User's Guide depict signals available at 25-pin connector P3, and show the interconnection of the components in Table 2-1.

TABLE 2-1. DESCRIPTION OF MATERIAL - EIA OPTION

ITEM	QTY	DESCRIPTION	INSTALL AT
1	1	TMS9902	U31
2	1	75188	U37
3	1	RES,10K ,10%,0.25W	R34
4	2	RES,3.3K ,10%,0.25W	R35,R36
5	1	Connector,AMP206584-2 or CANNON DBP-25S-AA	P3

Instructions concerning the hookup for an external terminal can be found in Section II, 2.5.3.4 of the TM990/189 Microcomputer User's Guide.

2.5.2 TTY Communications

Installation of the Serial Communications Interface components shown in Table 2-1, coupled with the board modifications presented in Table 2-2 allow the capability to perform asynchronous serial communications with devices having a 20 mA current loop interface. (Refer to Appendix A, page A-10 of the TM990/189 Microcomputer User's Guide for both schematics showing signals available at the 25-pin connector P3, and the components in Table 2-1). The serial

interface resides at R12 CRU base addresses >0800 through >083E. The modification procedures entails the addition of a jumper wire between E25 and E26.

Table 2-2. DESCRIPTION OF MATERIAL - TTY OPTION

ITEM	QTY	DESCRIPTION	INSTALL AT
1	1	TMS9902	U31
2	1	75188	U37
3	1	2N2905	Q12
4	1	1N914B	CR9
5	1	RES,33K ,10%,0.25W	R32
6	1	RES,3.3K ,10%,0.25W	R33
7	1	RES,10K ,10%,0.25W	R34
8	1	RES,330 ,10%,0.5W	R37
9	1	RES,560 ,10%,0.5W	R38
10	1	RES,2.7K ,10%,0.25W	R39
11	1	CONNECTOR,AMP 206584-2 or CANNON DBP-25S-AA	P3

Section II, paragraph 2.5.3.4 of the TM990/189 Microcomputer User's Guide explains the procedure for an external terminal hookup.

2.5.3 Audio Cassette Interface

Audio cassette interface circuitry and optional user-installed recorder control circuitry is provided by the TM990/189. Programs written to be run on this microcomputer can be stored on cassette (via the SAVE command) and loaded when required, thus freeing on-board memory between-times. Section II, subsection 5.9 of the TM990/189 Microcomputer User's Guide will instruct the user in the the hookup and operation of an audio cassette recorder.

2.6 SIMPLE EXAMPLE PROGRAM

Apply power to the board and activate the LOAD switch. (This causes University BASIC to begin execution.) "READY" will appear on the TM990/189 keypad. If an external data terminal is used, enter UNIT followed by the correct numerical assignment (refer to Section V, subsection 5.9 for these output device assignments), press the RETURN key on the keypad, and the external data terminal will now be configured to accept data from the keyboard.

The user may now enter the following sequence of commands and statements to verify that University BASIC has powered-up correctly.

```
100 FOR J=1 TO 25
110 FOR I=1 TO 25
120 TONE I,J
130 NEXT I
140 NEXT J
150 PRINT "CONGRATULATIONS!"
160 STOP
```

The program may be executed by typing in the RUN command and entering a carriage return. Upon completion, the program size and variable storage used may be displayed by entering the SIZE command with a carriage return.

```
SIZE
USED:118
FREE:1364      (Fully Populated)
```

All variables may now be cleared by typing the NEW command and entering a carriage return.

SECTION III

GENERAL PROGRAMMING INFORMATION

3.1 GENERAL

General programming information about the University BASIC language is covered in this section. Language features such as syntax, editing commands, and error listings will be presented.

3.2 University BASIC Language

Communication with the microcomputer is accomplished using a programming language; in this case, University BASIC. University BASIC is composed of commands and statements. Commands are used to list, save, load, and execute the user's program. Commands begin with the command name and are executed immediately upon entry; they are never preceded by line numbers. In general, commands are used to immediately alter the computer's operating environment in preparation for entering a program, debugging a program, or executing a program.

Statements in University BASIC programs are designed to perform a task or solve a problem. Statements begin with a line number and may be displayed with the LIST command and modified by retyping the whole line. A sequence of previously entered statements is called a program. The user may abort (or modify) a command or statement entered by: 1) NOT using the carriage return key at the end of the line, but backspacing (Shift 'M') and retyping the line on a terminal keyboard, or 2) using the shift key and pressing the M key on the TM990/189 keypad.

3.3 THE UNIVERSITY BASIC PROGRAM

A University BASIC program consists of one or more lines, each uniquely identified by a line number in the range 0 to 32,767. Each line contains at least one statement of the form:

```
<line number> <statement>
```

More than one statement may appear on one line by separating the statements with a colon. (Refer to subsection 3.7 for exceptions to this rule).

```
<line number> <statement 1>:<statement 2>
```

The last statement on a line must be contained on that line, and cannot be continued on the following line. Each line must be terminated with a carriage return character. Note that no more than 80 characters may appear on one line.

Automatic line numbering may be implemented by using the line feed key or the Shift A on the TM990/189 keypad. Auto-line numbering is initialized to begin at statement number 10 and increments by 10 between subsequent numbers.

To initiate auto-line numbering when generating a program, either:

- Enter a CNTR A (or shift A) as the first character of the line (to which University BASIC responds with line number 10), or
- Enter the first (starting) statement number and the associated statement and terminate the line with a CNTR A (or Shift A) entry.

Auto-line numbering may be terminated by entry of a carriage return at the end of the statement, or by striking the escape key.

University BASIC programs are executed beginning with the lowest numbered line and continue sequentially through the program unless directed otherwise by another statement, or until the last statement has been executed. Line numbers are also used in associating program editing activities with a particular statement line in the program.

3.4 SOURCE STATEMENT FORMAT

3.4.1 Line Number Field

The line number field is the first field of any program line and consists of a decimal integer between 1 and 32,767 inclusive.

3.4.2 Statement Field

The statement field follows the line number in a program line and contains one or more University BASIC statements separated by colons (:). Each statement is comprised of a University BASIC keyword followed by a number of constants and/or variables separated by University BASIC operators.

3.4.3 Decimal Integer Constants

A 'constant' is a quantity or data representation that does not change in value. In University BASIC, a decimal integer constant is any integer between -32767 and 32767, inclusive.

3.4.4 String Constants

A string constant is a string of characters enclosed within double quotes. For example:

```
PRINT "UNIVERSITY BASIC"
```

3.5 VARIABLES

A 'variable' is a representation of a quantity or the quantity itself, which can assume any of a given set of values. University BASIC supports: 1) simple numeric variables, 2) numeric array variables, 3) simple string variables, and 4 string array variables. The two numeric variable formats are used extensively in University BASIC statements and arithmetic operation, while the two string variable formats are used for string-character manipulation, text, and output. If any string variable is referenced and has not previously been defined, the string variable will be defined as a null string.

3.5.1 Simple Variables

Names for simple numeric variables must begin with a capital letter (A-Z) and may be followed by one capital letter or a number in the range 0-9. Variable names may not begin with a University BASIC keyword. A maximum of 120 distinct variables is allowed. Attempts to define more than this will result in error number 05, "TOO MANY VARIABLES".

Examples are:

```
Valid names: A, A0, A8
```

3.5.2 Numeric Array Variables

The same rules given for formation of simple numeric variable names apply to numeric array variables, with the additional specification that numeric array variables must appear in a DIM statement which is executed before the first reference to the variable (see DIM statement, Section V, subsection 5.3). Numeric array variables must always appear with a subscript. The subscript distinguishes an array variable from a simple variable of the same name (i.e., PRINT A and PRINT A(0) refer to two completely separate variables). Parenthesis should be used when entering a reference to an array variable. There is no limit placed upon the number of numeric array variables allowed in University BASIC.

3.5.3 Simple String Variables

Simple string variables are handled in the same way numeric array variables are with the exception that the reference must be preceded by a dollar sign ($). Internally, string data is stored left-justified and delimited by a null character (a zero byte). Characters are normally represented as 8-bit ASCII (normal 7-bit ASCII with the 8th bit set to zero). If the 8th bit is set to one, the interpreter will treat the character the same. However, note that a character with the 8th bit on is NOT equal to the same character with the 8th bit off! A simple variable in University BASIC is composed of 16 bits, or 2 (8-bit) bytes. Thus, a maximum of 1 character should be stored in a simple string variable of University BASIC; longer strings should be stored in string arrays (dimensional string variables) as explained below.

NOTE

> Any operation which attempts to place more than the maximum number of characters in a string variable will result in overwriting of data immediately following the string variable.

3.5.4 String Array

The same rules given for the formation of numeric array variables apply to string array variables with the added requirement that the name must be preceded by a dollar sign ($). The dollar sign, however, is omitted when defining array variables with the DIM statement.

3.6 OPERATORS AND EXPRESSIONS

An expression is a list of variables and constants separated by 'operators'. An operator, in this sense, is a symbol which indicates the action to be performed on an item called an operand. There are three types of University BASIC operators and expressions: arithmetic, logical, and relational. These are listed and briefly described in the following subsections.

3.6.1 Arithmetic Operators

The following is a list of valid arithmetic operators, the precedence of which is given in subsection 3.6.4:

+	addition
-	subtraction
*	multiplication
/	division
+	unary plus
-	unary minus

3.6.2 Arithmetic Expressions

An arithmetic expression is any valid sequence of numbers or variables, properly balanced; no two numbers or variables can be adjacent, and no two binary operators can be adjacent. For example:

An expression may consist of a single operand:

```
DIM A(8)
```

A sequence of operands may be combined by arithmetic operators:

```
X*Y

A*B-W/Z
```

Any expression may be enclosed in parentheses and considered to be a basic operand:

```
(X+Y)/Z

(A+B)*(C+D)
```

Any expression may be preceded by a plus or minus sign:

```
+X

-(A+B)
```

3.6.3 Relational Operators

The relational operators are all binary operators that operate on two arithmetic expressions. They return values of 1 (TRUE) or 0 (FALSE). Relational operators consist of the following:

=	exactly equal
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to

<> not equal

3.6.4 Expression Evaluation

Expressions are evaluated left to right if the operators are of equal precedence, and there are no parentheses. If there are parentheses in the expression, the sub-expression within the innermost parentheses is evaluated first. Not all operators have equal precedence; operators which are operated on by an operator of high precedence are evaluated before operations of low precedence..

The precedence of operators is:

1. Expressions in parentheses
2. Negation
3. *,/
4. +,-
5. <=,<>
6. >=,<
7. =,>

3.7 MULTIPLE STATEMENTS ":"

A colon terminates a University BASIC statement and can therefore be followed by another statement on the same line. This saves memory, speeds execution, and allows for better program segmentation. All University BASIC statements may be preceded and followed by a colon in multiple statement lines with the exception of the GOTO, GOSUB, IF-THEN, REM, FOR, NEXT statements The NEXT statement should not be preceded by another statement (i.e., should be the first statement of the line), and the REM, GOSUB , RETURN, and FOR statements should not be followed by any statements on the same line.

3.8 KEYBOARD MODE

University BASIC executes statements in either "execution" mode or "keyboard" mode. In keyboard mode, statement numbers are not entered, only one line is executed at a time, and control is returned to the user after its execution. This line may contain multiple statements properly separated by a single colon.

The system recognizes two kinds of inputs: statements and commands. See Section IV for University BASIC commands; Section V for statements. One and only one command may be executed per line with no statements on the line.

In execution mode, the program counter moves through the program executing statements. Execution mode is entered by RUN or GOTO and returns to keyboard mode after any error, "STOP", when all statements have been executed, or when the escape key is entered.

The following examples illustrate one line calculations in keyboard mode. Note that ";" is equivalent to PRINT. The user must terminate each entry line with a carriage return and University BASIC responses are underlined for clarity.

Example:

```
PRINT 12*12  144
J=1:A=2:M=3:B=4:;J,A,M,B   1 2 3  4
```

The following types can only be executed in keyboard mode. They can only be entered one command per line and cannot be entered in a program:

```
LIST
NEW
RUN
SAVE
SIZE
```

3.9 ERRORS AND ERROR LISTING

The first run of a new program may be free of errors and give the correct answers. It is more common, however, that errors will be present and will have to be corrected. Errors will be of two types: errors of form (syntax, arithmetic, structure, or grammatical errors) which prevent the running of the program, and logical errors in the program which cause the computer to produce either the wrong answers or no answers.

Errors of form cause the error code and statement number in which the error occurred to be printed, and program execution stops. Logical errors are often much harder to uncover, particularly when the program gives answers that appear to be nearly correct. In either case, after the errors are discovered, they can be corrected by changing lines, by inserting new lines, or by deleting lines from the program. A line is changed by typing it correctly with the same line number; a line is inserted by typing it with a line number between those of two existing lines; and a line is deleted by typing its line number and pressing the carriage return key. A line can be inserted only if the original line numbers are not consecutive numbers. For this reason, most programmers will start out using line numbers that are multiples

of five or ten to leave space for the inevitable changes and corrections.

Corrections can be made at any time before or after a run. Since the computer sorts lines (and arranges them in order), a line may be retyped out of sequence. Simply retype the offending line with its original line number. If, after examining a program, the errors are not obvious and there are no grammatical errors, insert temporary PRINT statements to verify the machine is computing correctly.

University BASIC displays error code numbers corresponding to the appropriate error message to indicate which error has occurred. Errors are reported in two formats, the first of which displays both the error code number and the statement number in which the error occurred:

```
*ERR XX AT YYYY
```

where XX is the error code or error message and YYYY is the statement number.

This error format is displayed whenever errors are encountered during program execution, and program execution will be terminated at the offending statement. The error format displays the statement line in which the error occurred. The offending statement line or other segments of the program may then be edited to correct the reported error.

The second format displays only the error code or error message when an error occurs. These types of errors are detected during keyboard mode statement execution, during statement or command entry, or during program LOADing from cassette. They indicate that the most recently entered statement or command, on the most recently LOADed statement is in error. For further details, refer to Appendix A.

The following error codes and error messages may be issued by the University BASIC package:

```
01 = SYNTAX ERROR
02 = UNMATCHED PARENTHESIS
03 = INVALID LINE NUMBER
04 = ILLEGAL VARIABLE NAME
05 = TOO MANY VARIABLES
06 = ILLEGAL CHARACTER
07 = NOT IMPLEMENTED IN THIS RELEASE
08 = STACK OVERFLOW
09 = STACK UNDERFLOW
10 = STORAGE OVERFLOW
11 = NEXT W/O FOR
12 = NOT IMPLEMENTED IN THIS RELEASE
13 = NO SUCH LINE NUMBER
14 = EXPECTING STRING VARIABLE
15 = SUBSCRIPT OUT OF RANGE
16 = TOO MANY SUBSCRIPTS
17 = SQUARE ROOT OF NEGATIVE #
18 = INTEGER OVERFLOW
19 = DIVISION BY ZERO
20 = ILLEGAL DELIMITER
21 = NOT IMPLEMENTED IN THIS RELEASE
22 = TIMEOUT ERROR
```

SECTION IV

UNIVERSITY BASIC COMMANDS

4.1 GENERAL

University BASIC recognizes two kinds of input: statements and commands. Commands direct and control system functions which include initiating computer operation, storing data, and listing programs. Commands cause immediate computer interaction thereby allowing operator control. Any command may be entered once University BASIC has been initialized. An error messsage is generated when an improper or illegal entry is attempted.

4.2 LIST COMMAND

Forms:

```
                    LIST
<line - number>     LIST
```

The LIST command displays all or any portion of the current program. Entering only the command forces the entire program to be listed. By entering a line number, specific portions of te program can be listed. The line number specifies the starting line number where listing of the program is to begin. The starting line number need not be an existing line number. University BASIC will begin listing at the first line number greater than or equal to the starting line number and terminate listing at either the last line number of the program or when the user enters the ESCape key.

Example:

```
LIST
```

results in a listing of an entire program, while

```
100 LIST
```

lists all the lines from 100 through end of program, inclusive.

4.3 SAVE COMMAND

Form:

SAVE

The SAVE command writes the source form of the entire University BASIC program currently in memory to the cassette device. The program remains in memory after the SAVE and can be deleted by the NEW command (reference subsection 4.5). The program may be retrieved by the LOAD command at some future date (subsection 4.4).

The SAVE command will result in the user's program being stored on audio cassette.

NOTE

> The audio cassette device service routines cannot be interrupted during the saving of a program since each bit of the data bytes have a specified minimum and maximum pulse width for reliable data storage and retrieval. Therefore, all interrupts are masked at the CPU whenever a SAVE is being performed.

4.4 LOAD COMMAND

When a user program has been properly "SAVEd" on cassette (Section 4.4.1), or burned into EPROM (Section 4.4.2), the LOAD command transfers the user program into memory.

Form:

```
LOAD          (Loads from Cassette)
LOAD <exp>    (Loads from designated address on EPROM)
```

4.4.1 LOAD

Only those statements having statement numbers will be written to cassette, and statements in memory which have the same statement number as the program on cassette will be overwritten when the cassette is loaded.

To load a program from cassette, the tape drive must be readied and in the playback mode. If an error occurs while loading from cassette, the error message and the statement number where the error occurred will be printed. When an error occurs, the loading procedure is terminated and University BASIC returns to the keyboard mode. University BASIC automatically stops the cassette transport on a error. Note that all statements on the cassette tape prior to the occurrence of the error will have been successfully loaded and need not be entered again. The remainder of the statements on the cassette may be loaded by again

entering the "LOAD" command after manually stopping the cassette transport. When loading is complete, it automatically returns to the keyboard mode and awaits command/statement entry. Program listing, editing, and execution may then proceed.

NOTE

> The audio cassette device service routines cannot be interrupted during loading of a program since each bit of the data bytes has a specified minimum and maximum pulse width for reliable data storage and retrieval. Therefore, all interrupts are masked at the CPU whenever a LOAD is being performed.

4.4.2 LOAD <exp>

The LOAD <exp> command allows EPROM resident programs to be quickly loaded into University BASIC programs space. The argument specifies an index into a table of addresses located at memory address >1004. Each address in this table consists of two bytes beginning at >1004 and points to the first byte of the user program in EPROM (e.g., LOAD 0 points to the address at location >1004, LOAD 1 points to the address at location >1006, and LOAD 2 points to >1008, etc.

NOTE

> To accomodate larger programs, the load command does a program purge while preserving all variables and variable values. The LOAD command can be placed in a program to chain to another program module. Therefore, the load command can also be executed as a program statement. Execution begins with the first statement of the program when such conventions are used.

For example:

```
10    REM   PROGRAM INITIALIZATION
20    ...
999   LOAD 1  (Chains to next module)

10    REM   MAIN PROGRAM
20    ...
30    ...
```

Programs stored in EPROMS must be in ASCII code with each line beginning with a line number, followed by a space (ASCII >20) then followed by statement or statements (just like in keyboard mode) and terminated by a carriage return (ASCII >0D). A null byte must be stored at the end of each program.

Here is an example of a program placed in EPROMS. It can be loaded into University BASIC by doing a "LOAD 0" or a "<line number> LOAD 0".

```
AORG   >0004
DATA   >1018
DATA   0,0,0,0,0,0,0,0
AORG   >0018
TEXT   '10 FOR I=1060 TO 1109'
BYTE   >0D
TEXT   '20 INPUT X:MEM(I)=X'
BYTE   >0D
TEXT   '30 Y=MEM(I)::Y'
BYTE   >0D
TEXT   '40 NEXT I'
BYTE   >0D
TEXT   '50 ;"HENRY WADSWORTH LONGFELLOW"'
BYTE   >0D
TEXT   '60 Z=SUB 1060'
BYTE   >0D
TEXT   '70 ;Z:GOTO 50'
BYTE   >0D
TEXT   '80 STOP'
BYTE   >0D
```

4.5 NEW COMMAND

Form:

```
NEW
```

The NEW command deletes the current user program and clears all variable space, pointers, and stacks. University BASIC responds with "READY" and awaits the entry of a new program. (The program may be retrieved later if it has been SAVEd.)

4.6 SIZE COMMAND

The SIZE command monitors memory usage by listing the current program size and free memory in bytes.

Form:

```
SIZE
```

Example:

```
SIZE
USED:0
FREE:1446
```

```
SIZE
USED:0
FREE:1446
```

4.7 TM990/189 KEYPAD EDIT COMMANDS

The keypad of the TM990/189 microcomputer requires special commands in order to perform some editing functions. The are listed in the following table.

SYNTAX	EXPLANATION
(Shift) 9	Cancel input line or break program execution.
(Shift) M	Backspace.
(Shift) A	Initiate automatic line numbering. (increments of 10)
(Shift) V	Moves keypad display left 6 spaces.
(Shift) W	Moves keypad display left by 1 space.
(Shift) Y	Moves keypad display right 1 spaces.
(Shift Z)	Moves keypad display right 6 spaces.

4.8 UNIVERSITY BASIC COLOR VIDEO COMMANDS

In addition to the commands listed earlier in this section, color video commands are available to the user of University BASIC. These commands will be listed in the table below, but additional information (including full descriptions and demonstration programs) may be obtained in the Color Video Using TMS9918 and University BASIC Application Report, MP723.

TABLE 4-1. VDP COMMANDS

COMMAND NAME	FORMAT
COLOR	COLOR <exp>
MODE	MODE <exp> {,<exp>}
MOVE	MOVE <exp> MOVE <exp>,<exp>,<exp>
PATTERN	PATTERN <exp>,<string>
SPRITE	SPRITE <exp>,<string> SPRITE <exp>,<exp> {,<exp>} SPRITE <exp>,<exp>,<exp>
VDP	VDP(<exp1>) = <exp2> <var> = VDP <exp>

SECTION V

UNIVERSITY BASIC STATEMENTS

5.1 GENERAL

This section discusses the University BASIC program statements. Statement formats are presented and their uses are described.

During program execution, control may pass to any statement. Some statements have no effect on the program when encountered and are called nonexecutable (such as the REM statement); all others are called executable.

Statements form the basis of all functional University BASIC programs. Each statement of a program may occupy only one line; however, numerous statements may appear on each line when delimited by a colon (:). Table 5-1 lists and briefly describes each University BASIC statement.*

TABLE 5-1. UNIVERSITY BASIC STATEMENTS

STATEMENT	FUNCTION	USE
REM (EOL)	Comment Line	Program documentation/explanation
DIM	Size Specifier	Allocates space for array variables
LET	Assignment	Evaluates expressions and assigns value
GOTO (EOL)	Control	Transfers unconditionally
IF-THEN(EOL)	Control	Conditionally executes statement(s)
GOSUB (EOL)	Control	Transfers to University BASIC Subroutines
RETURN	Control	Returns from University BASIC Subroutines
FOR (EOL)	Control	Defines top of loop and loop parameters
NEXT (EOL)	Control	Closes program loop
STOP	Control	Stops program

*NOTE: "EOL" = End of Line

INPUT	I/O	Reads from terminal
PRINT	I/O	Prints on output device
TAB	I/O	Formats output into columns
UNIT	I/O	Designates print output device
BASE	CRU Base Assignment	Sets the CRU base address
CALL	External Subroutine	Transfers to subroutine
TONE	Sound	Allows sound capability
LOAD	Load Program	Chains long programs

5.2 REMark STATEMENT

Form:

```
<line number> REM <text> EOL
```

The REM statement is used to insert remarks (comments) in a program. REM may contain any textual information. It has no effect when encountered in execution. However, its line number may be used as the argument of a GOTO or GOSUB statement.

Examples:

```
10    REM THIS IS A COMMENT

100   REM CHECK FOR X=0
```

5.3 DIMension STATEMENT

Dimension declarations are used to specify the size attributes for subscripted variables within the program.

Form:

```
<line number>  DIM <var(dim[,dim]...)
               DIM <var(dim[,dim]...)
```

The DIM statement dynamically allocates user variable space for array variables. Dimensioned (array) variables must be declared by the DIM statement before the variables are used. Once dimensioned, attempts to redimension an array variable to a larger array size will result in an error message, and attempts to redimension to a smaller size will be disregarded.

Array sizes are specified by indicating the maximum subscript values in parentheses following the array name. Subscripts of

dimensioned variables may be any numeric quantity including constants, simple variables, other dimensioned variables, or even function calls. (Note that only two dimensions per variable may be used). An error will occur if the dimensioned variable requires more variable space than is currently available in the user's partition. Dimensioned variables always use the 0 subscript as the first element in the array.

Examples:

```
10    DIM  A(10),B(10,20)

100   DIM  A1(10),B1(20,30),B9(10,10)
```

The first statement allows for the entry of an array of 11 elements (0-10) into A, and of an array of 11 x 21 elements into the two dimensional array, B. The two remaining statements dimension arrays in a similar manner. String variables must be dimensioned as numeric variables, e.g., $A must be dimensioned as A(10), not $A(10). Thereafter, the dimensioned numeric variable may be referenced as a string variable by preceding the variable with a dollar sign ($). The string array A dimensioned above should be referenced as $A(0) through $A(10).

Example:

```
20   DIM   C(10),D(8)
```

This statement defines C to be a one dimensional array with 11 elements and defines D as a one dimensional array of 9 elements. Hereafter, these arrays may be considered as string arrays by referencing the variables via $C(0) through $C(10) and $D(0) through $D(8).

Strings are stored one character per byte with a null character used to terminate the string. Hence, simple string variables and single array elements which are 2 bytes in length can contain up to one character. Dimensioned string variables can contain up to the number of elements times 2. Therefore, the dimensioned string variable $C can contain up to 21 characters.

5.4 LET STATEMENT

The LET statement assigns a value to a variable where the variable is set equal to an expression consisting of variables and/or constants separated by operators. The variable being evaluated may appear within the expression. The newly calculated value of the variable replaces the old value.

In University BASIC the letters LET may be omitted from the statement so only an equation appears. The LET statement may have either of the following forms:

```
<line number>  LET  <variable> = <expression>
               LET  <variable> = <expression>
<line number>  <variable> = <expression>
               <variable> = <expression>
```

where

variable is a string variable, numeric scalar variable, or array element.

The assignment statement assigns an expression value to a variable. Both variable and the expression must be either string or numeric. The following example illustrates the assignment statement.

```
10 LET A=3:B=4
20 C=6:D=8
30 DIM E(10)
40 $E(0)="STOP"
50 PRINT A,B,C,D,$E(0)
60 STOP

RUN
3     4     6     8     STOP
```

5.5 CONTROL AND COMPUTED TRANSFER STATEMENTS

University BASIC statements are executed sequentially unless altered by control statements. Control may be accomplished by an unconditional branch, subroutine branch, or loop.

5.5.1 Unconditional GOTO Statement

When the computer encounters a GOTO statement, it jumps to the program line number specified in the statement. The program executes the statement at the specified line number and continues in sequence with the statements that follow.

Form:

```
<line number> GOTO  <line number> EOL
              GOTO  <line number> EOL
```

If the "GOTO" is not preceded by a line number (i.e., entered in Keyboard mode), it allows statements to be skipped and execution to begin at the line number specified in the statement.

Examples:

```
          GOTO  200  Begins execution at statement 200    (Keyboard Mode)
100       GOTO  140  Transfers control to statement 140
```

The following program illustrates the GOTO statement:

```
10  INPUT A
20  GOTO 40
30  STOP
40  PRINT A
50  GOTO 40 30
```

The program execution sequence is line numbers 10, 20, 50, 40 and 30 where execution stops.

5.5.2 IF-THEN Statement

The IF statement alters sequential execution of the program depending on the state of the specified condition.

Forms:

```
<line number> IF <expression> THEN <statement(s)> EOL
              IF <expression> THEN <statement(s)> EOL
<line number> IF <expression> <relation> <expression> THEN <statement(s)>
              IF <expression> <relation> <expression> THEN <statement(s)>
<line number> IF <string> <relation> <string> THEN <statement(s)> EOL
              IF <string> <relation> <string> THEN <statement(s)> EOL
<line number> IF <string> THEN <statement(s)> EOL
              IF <string> THEN <statement(s)> EOL
```

The condition may be any variable, numeric expression, relational expression, logical expression, string variable, string relational expression, or function which can evaluate to a zero or non-zero value. Two expressions or strings are compared according to the given relation and a true or false condition results. If only a single expression or string is given, the condition is considered false if the expression is zero or the string is null; otherwise, it is considered true.

If the condition is true, the statement(s) following the THEN clause on the same line will be executed. If the condition is false, the statement on the line following the IF-THEN statement will be the next statement executed. Any University BASIC statement or statements (including GOTO's and other IF-THEN statements) may immediately follow the THEN clause. They cannot extend to the next statement line because statement execution continues at the next statement line when a false condition occurs. The IF and THEN clauses must appear on the same statement line.

Examples:

```
 20 IF A=0 THEN GOTO 100

100 IF I+2 THEN PRINT I

 60 IF $A THEN $B=$A
```

5.5.3 Subroutine (GOSUB AND RETURN) Statements

University BASIC programs may contain internal subroutines. An internal subroutine is a sequence of statements performing a well-defined function or operation within the University BASIC program. Two types of statements govern access to a subroutine: a GOSUB statement for entry into the subroutine and a RETURN statement for return to the calling program.

Forms:

```
<line number>  GOSUB  <line number> EOL
<line number>  RETURN
```

An internal University BASIC subroutine may be invoked from any point within the program by using a GOSUB statement which specifies the entry point of the subroutine as a line number. Execution of the GOSUB statement pushes the address of the statement immediately following the GOSUB statement onto the GOSUB stack for return, and passes execution to the specified line number.

A RETURN statement placed in the subroutine is an exit point from the internal University BASIC subroutine. A RETURN statement should be placed at each logical end of all subroutines. The RETURN statement causes execution to resume at the first statement following the GOSUB statement that transferred to the subroutine. During this transfer, the top return address is removed from the GOSUB stack. All subroutines should be exited only via a RETURN statement so the top return address will always be removed from the GOSUB stack. Unpredictable results occur if a subroutine is exited in any other fashion.

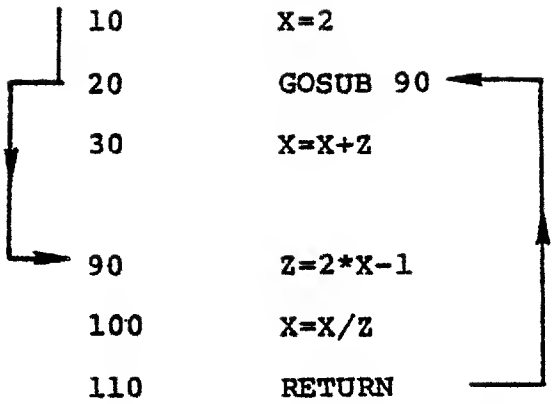


FIGURE 5-1. GOSUB EXAMPLE

In Figure 5-1 GOSUB 90 involves statements on line numbers 90 (start of subroutine), 100, and 110 (end of subroutine). If a GOSUB statement is used, the subroutine it branches to must contain at least one RETURN statement. The example illustrates the simplest use of GOSUB and RETURN. The arrows indicate the flow of control in the program.

A return address (first line number after the call) must be stored for each GOSUB statement until that statement is executed. The program in the following example contains "nested" subroutines (i.e., subroutines within a subroutine) and shows the actual execution sequence. Each GOSUB to a subroutine must be accompanied by at least one RETURN statement per exit path. The nested program and execution sequence of the example demonstrate entry to and exit from a subroutine. Note that subroutines may be only be nested to a level of 8.

A RETURN statement must not be encountered unless a GOSUB statement has been executed.

"Remembering" all the return points by saving them on the GOSUB stack and never removing them can exhaust the available GOSUB stack area. The following program, which calculates N illustrates this problem; its use requires that N return points be remembered.

```
 10    INPUT "N= ";N
 20    GOSUB 100
 30    ;"N=";N,"N1=";N1,"N2="N2,"N3=";N3
 40    STOP

100    GOSUB 200
110    N1=N*10
120    RETURN
200    GOSUB 300
210    N2=N*100
220    RETURN
300    N3=N*1000
310    RETURN
```

5.5.4 FOR/NEXT Loops

FOR and NEXT statements indicate the start and end of an instruction block that is to be repeatedly executed as a set. One variable takes on different values within a specified range; this variable is often used in the computation or evaluation contained in the instruction block. The FOR statement names the variable and stepping values of that variable and also specifies its initial and final values. The NEXT statement closes the program loop. Note that the maximum nesting level possible for FOR-NEXT loops in University BASIC is 4.

The FOR statement may have either of the following forms:

```
<line number> FOR <variable> = <exp> TO <exp> EOL
<line number  FOR <variable> = <exp> TO <exp> STEP <exp> EOL
```

where

variable is a simple numeric scalar variable
expression is a valid University BASIC numeric expression

Note that no pre-test is done by the FOR statement (i.e., it will always execute once).

The NEXT statement has the form:

```
<line number>  NEXT  <variable> EOL
```

where

variable is a simple numeric variable

The simple variable of the NEXT statement must be the same as the FOR statement variable at the beginning of the loop.

Specification of the STEP value is optional. If omitted, a value of +1 is used. The step value may be any constant, variable, or expression which evaluates to a positive or negative value. Negative step intervals can be used to decrease the value of the FOR variable from one pass through the loop to the next. By using a step value of -1, the FOR variable can be made to decrease by integer values during successive loop interactions.

Example:

```
100 FOR J=1 TO 60 STEP 8
110 FOR A=1 TO 30 STEP 6
120 TONE J,A
130 NEXT A
140 NEXT J
150 FOR J=10 TO 1 STEP -1
160 FOR A=-3 TO 12 STEP 2
170 TONE J,A
180 NEXT A
190 NEXT J
200 STOP
```

Note that the step size may be a variable, an expression, a negative number, or a positive number. If expressions are used to specify the initial, final or step-size values, they will be evaluated only once when the FOR loop is entered. Changing any of the values (either the step, initial or final values) within the FOR loop does not affect the number of times the sequence is executed with the exception of the control variable. The control variable is assigned to the initial values when the FOR statement is entered and is incremented (if the STEP value is positive), or decremented (if the step value is negative) after each repetition of the loop sequence. The last repetition of the loop sequence is when the control variable is equal to the final value. When exiting the loop in this manner, the control variable is incremented (or decremented) one step value beyond the final value.

A pre-check is performed so that if the initial value is greater than the final value in the case of positive STEP values, the loop sequence will not be executed. Likewise, if the initial value is less than the final value and the STEP value is negative, the loop sequence will not be executed.

The control variable may be changed within the body of the loop and the latest value of the variable will be used in the exit test; however, this programming practice is not recommended.

The loop continues to be executed as long as the condition:

(step value)*(control variable)<(step value)*(end value)

remains true. If the condition:

(step value)*(start value)>(step value)*(end value)

is true when the FOR statement is first encountered, the loop will not be executed.

When the loop is being executed, the control variable is first set to the initial value and if the end criterion is not true, the loop is executed. The control variable is then incremented by the step value each time the NEXT statement is encountered and executed. The loop terminates with the control variable equal to the last value used in the loop plus the step value.

Example:

```
10   FOR I=1 TO 4 STEP 2
 .
 .
 .
80   NEXT I
90   PRINT "I=";I
RUN
I=5
```

The NEXT statement closes the FOR loop. When it is encountered, the step value is added to the control variable. If the control variable has not gone beyond the end value, control will be returned to the first statement following the FOR which opened the loop. The control variable of the loop to be closed must be specified by the NEXT statement.

FOR loops may be nested; i.e., one FOR loop may contain another which may contain a third, etc. If nested, however, they should not use the same control variable. When two loops are nested, one must be completely contained within the other. Overlapping is not permitted. The following structure is correct:

```
100  FOR I=1 TO 2
110  FOR J=1 TO 2
120  FOR K=1 TO 2
130  PRINT I,J,K
140  NEXT K
150  NEXT J
160  NEXT I
170  STOP
```

while the next two structures are incorrect:

```
10   FOR I=1 TO 2
20   FOR J=1 TO 2
.
.
80   NEXT I
90   NEXT J  (WRONG, loops may not overlap)

10   FOR I=1 TO 2
20   FOR I=1 TO 2
.
.
80   NEXT I
90   NEXT I (WRONG, nested loops may not have the same control
     variable)
```

NOTE: Maximum nesting level is 4.

The following program illustrates nesting:

```
LIST
  10    REM AREA OF A TRIANGLE
  20    FOR B=6 TO 9
  30    FOR H=11 TO 13 STEP 2
  40    A=B*H/2
  50    PRINT "A=";A,"B=";B,"H=";H
  60    NEXT H
  70    NEXT B
  80    STOP
```

This program prints the base, height, and area of triangles with bases 6, 7, 8 and 9, and heights 11 and 13. All combinations are printed: Eight sets of data for the four bases and two height values.

All values of the variable in the inner loop are cycled through while the variable in the outer loop is set to its first value. The outer loop variable is then set to its second value and the inner loop is cycled through again. The program runs through each outer loop value this way.

It is legal to transfer control from within a loop to a statement outside the loop, but it is never advisable to transfer control into a loop from outside. The next two examples illustrate both of these situations.

Valid transfer out of a loop:

```
  20    FOR I=1 TO N
  30    X=X+2*I
  40    IF X>1000 THEN GOTO 100
  50    NEXT I
```

Invalid transfer into a loop:

```
  20    GOTO 50
  30    FOR I=1 TO N
  40    X=X*2*I
  50    Y=Y+Y/2
  60    NEXT I
  .
  .         (WRONG, 50 is inside a loop)
  .
```

However, it is permissable to call a subroutine from within a loop and then return from the subroutine back into the loop. The following example illustrates repetitive calling of a subroutine from inside a loop.

Example:

```
10   FOR I=1 TO N
20   X=2*I-1
30   GOSUB 150
40   Z=Z+Y
50   NEXT I
 .
 .
 .
150  IF X<>12 THEN GOTO 180
160  Y=248
170  RETURN
180  Y=200+4*X
190  RETURN
```

5.5.5 STOP Statement

The STOP statement terminates program execution at the logical end of the program. There may be one or more STOP statements in a University BASIC program, and they may appear anywhere within the program.

Form:

<line number> STOP

The system displays the line where program execution terminated.

Example:

```
100 STOP
STOP at 900
```

5.6 INPUT STATEMENT

The INPUT statement is used for keyboard input from an interactive terminal into variables of the BASIC program.

Form:

<line number> INPUT <variable> {,;} <variable> {,;}

The INPUT statement performs as a READ statement* with the exception that it accesses the numeric constants and strings from the external keyboard.

*Refer to the POWER BASIC Reference Manual, MP308

It provides all translation from character data to the internal formats of the University BASIC system and thus assigns input values to the variables or array elements specified in the input list. All characters are echoed as they are entered. The INPUT statement is extremely versatile and provides a means to 1)input numbers only, 2)input character strings, 3)detect control characters, 4)prompt with character strings, 5)specify maximum number of input characters, 6)specify exact number of input characters, 7)suppress carriage return/line feed, and 8) suppress prompting.

Input variables may be entered in a list separated by carriage returns. Numeric data may be represented as decimal integers. There should be no embedded spa ces within numeric values and all spaces preceding or following numeric data are ignored. For string data input, the string consists of all characters after the prompting character and up to (but not including) the end of the input (carriage return). The string includes all entered blanks and quotes.

The INPUT statement, when used with the comma (,) delimiter prompts the user with a question mark (?) for numeric inputs only, and a colon (:) for character inputs. If an illegal number is entered in response to the question mark prompt, the computer will respond with a double question mark (??) and wait for correct input. The computer will continue to prompt until the user has entered all data requested. When used with the semi-colon (;) delimiter, however, no prompts are given.

In the following examples, a carriage return is represented by (CR) and all user responses are underlined.

Examples:

```
 30 DIM A(1), B(1), Y(1)
 40 INPUT X
 50 INPUT $A(0), $B(0)
 60 INPUT $Y(0). $Z(0)
 70 PRINT X,$A(0), $B(0), $Y(0), $Z(0)
 80 STOP

RUN
 ?256 (cr)
 :BOB (cr):DAN (cr)
 :HI (cr) ?80A (cr) ??80 (cr)
 256          BOB           DAN           HI            80

 STOP AT 80
```

In the program, statement 40 outputs a question mark waiting for numeric input. The user enters the number "256" followed by a carriage return which terminates the INPUT statement of line 40. The variable X is assigned the value of "256". Next it prompts with a colon awaiting character string input. The user enters "BOB" followed by a carriage return. The computer immediately prompts with a colon awaiting the next string input. The user enters "DAN" and a carriage return which terminates this input line. The computer then prompts with a colon and the user inputs "HI" and a carriage return. Next, the computer prompts with a question mark and the user incorrectly enters "80A", an illegal numeric value. Therefore, the computer responds with a double question mark and awaits correct input. The user enters "80" followed by a carriage return which terminates the INPUT statement. The program is then executed, and statement 70 outputs the values read into the variables.

An INPUT statement can be combined with a PRINT statement to prompt user response as follows:

```
20   PRINT "YOUR VALUES OF X, Y, AND Z ARE";
30   INPUT X, Y, Z
40   PRINT X,Y,Z
50   STOP

RUN
YOUR VALUES OF X, Y, AND Z ARE? 50 (cr) ?70 (cr) ?90 (cr)
50        70         90

STOP AT 40
```

Since user prompting for data input is required in most applications, the INPUT statement has been designed to permit string constants to be embedded in the INPUT statement for direct prompting output. The string constants must be enclosed by quotation marks. There may be any number of string constants within the INPUT statement separated from input variables and other string constants by commas or semicolons.

The above example may be performed as follows:

```
20   INPUT "YOUR VALUE OF X IS", X, " Y", Y, " Z",Z
30   STOP

RUN
YOUR VALUE OF X IS? 1 (cr) Y?  2 (cr) Z? 3 (cr)
STOP AT 30
```

Similarly for string input:

```
10   DIM N(5)
20   INPUT "WHAT IS YOUR NAME", $N(0)
30   PRINT "YOUR NAME IS ";$N(0)
40   GOTO 20

RUN

WHAT IS YOUR NAME:  JOHN  (cr)
YOUR NAME IS JOHN
WHAT IS YOUR NAME:
```

A semicolon may be used to perform input formatting. If a semicolon is placed at the end of an INPUT statement line, the carriage return/line feed is suppressed after processing the INPUT statement as the example below illustrates:

```
10   INPUT "INPUT X", X;
20   PRINT " X SQUARED="; X*X
30   INPUT "INPUT Y", Y
40   PRINT "Y CUBED="; Y*Y*Y
50   STOP

RUN

INPUT X?12 (cr) X SQUARED= 144
INPUT Y?3 (cr)
Y CUBED= 27

STOP AT 50
```

In line 10 the semicolon is present at the end of the INPUT statement; therefore, the carriage return/line feed is suppressed after entering the constant 12 so that "X SQUARED= 144" can be output on the same line. In line 30 a semicolon is not present so the carriage return/line feed is performed.

When the semicolon is placed before an assignment variable in the INPUT list, the automatic prompting of a question mark or colon is suppressed. The user may then perform his own prompting in the University BASIC Program by using PRINT statements or placing character strings in the INPUT statement.

Example:

```
 5    DIM N(3)
10    INPUT "WHAT IS YOUR EMPLOYEE NUMBER?", $N(0)
20    INPUT "WHAT IS YOUR EMPLOYEE NUMBER?"; $N(0)
30    STOP

RUN

WHAT IS YOUR EMPLOYEE NUMBER?: 1234 (cr)
WHAT IS YOUR EMPLOYEE NUMBER?1234 (cr)

STOP AT 30
```

In line 10, the INPUT Statement prompted with a colon (:). In line 20 no prompt was issued.

5.7 PRINT Statement

The PRINT statement causes the values of all expressions in the list to be printed on the output terminal. Commas and semicolons are used to separate expressions and provide for print formatting.

Form:

```
                {PRINT}                {,}               {,}         {,}
  line number   {  ;  }   expression   {;}   expression  {;}   ...   {;}
                {PRINT}                {,}               {,}         {,}
                {  ;  }   expression   {;}   expression  {;}         {;}
```

The expression list may contain any numeric variable, numeric expression, string variable, string constant, or any ASCII code which is to be output to the terminal device.

String constants may be printed directly by inserting them in the PRINT statement expression list. String variables are printed by having the variable name preceded with the dollar sign designator. The following example illustrates the output of string constants and string variables.

```
100    DIM N(10)
110    $N(0)= "UNIVERSITY BASIC."
120    PRINT "THE NAME OF THE LANGUAGE IS ";
130    PRINT  $N(0)
RUN

 THE NAME OF THE LANGUAGE IS UNIVERSITY BASIC.

 STOP AT 140
```

To facilitate rapid statement entry in the edit mode, a semicolon (;) may be used in place of the word "PRINT" in any PRINT statement. Upon statement entry, the semicolon is internally translated to the "PRINT" code. Thereafter, listing of the statement will result in output of the word "PRINT". For example:

```
10     PRINT I,J
20     ;X,Y,Z
30     ;'THE SEMICOLON WILL LIST AS "PRINT"
LIST

10     PRINT I,J
20     PRINT X,Y,Z
30     PRINT 'THE SEMICOLON WILL LIST AS "PRINT"
```

In its simplest form the expressions in the output list are separated by commas. In this form, an output line is divided into 8-character print fields starting in columns 1, 9, 17, etc. A comma following an expression in a list is a signal to advance to the next field. Expressions separated by commas are output one expression per print field. This enables output lines to be formatted into ten left-justified columns within the field. Expressions may occupy more than one field, in which case the comma following the expression in the PRINT list advances the print output to the next blank field. If the terminal device does not perform in this manner, output values may be lost at the end of output lines, and the five column output format may be skewed. Printing will continue in as many lines as are required to complete the output list. When the entire output list has been printed, a carriage return/line feed is automatically inserted after the last print item. Subsequent printing begins on the next line. For example, the following statements:

```
10     X=7
20     PRINT X, X+2, X+4
30     PRINT "JACK", "JOHN", "ANDY"
```

would generate

```
7         9         11
JACK      JOHN      ANDY
```

The automatic carriage return/line feed at the end of a PRINT statement may be suppressed by placing a comma at the end of the output list. Subsequent printing will begin in the next field of the same line. For example:

```
10    X = 7
20    PRINT X, X+2, X+4,
30    PRINT "JACK", "JOHN", "ANDY"
40    STOP
```

would generate

```
7 9 17 JACK JOHN ANDY
```

Note that most terminals automatically generate a carriage return and line feed as occurs in the following example:

```
10    FOR I=1 TO 14
20    PRINT I,
30    NEXT I
40    STOP
```

```
RUN
1             2             3             4             5
6             7             8             9             1
11            12            13            14

STOP AT 40
```

More compact printing can be achieved by using semicolons rather than commas as expression separators. When followed by a semicolon, numbers in the output list will print in as many characters as required to print the numbers of the expression plus one blank space added on the left. However, strings in the output list will print in exactly the end of an output list, the last item will print in a short field as just described, and subsequent printing will begin immediately after that field. For example:

```
10   S1=95
20   S2=87
30   S3=92
40   PRINT "SCORES AND NAME:";S1;S2;
50   PRINT S3; "GARY"
```

would generate

```
SCORES AND NAME: 95 87 92 GARY
```

Another example:

```
10   FOR I=1 TO 14
20   PRINT I ;
30   NEXT I
40   STOP
RUN
1  2  3  4  5  6  7  8  9  10  11  12  13  14

STOP AT 40
```

Note that both semicolons and commas may be used to separate expressions in any PRINT statement and that the print position of the next expression will depend on the separator (semicolon or comma) used to delimit the expressions. The following example illustrates the use of both delimiters in a single PRINT statement.

```
10   H=98
20   L=60
30   A=79
40   PRINT "HIGH= ";H,"LOW= ";L,"AVERAGE= ";A
```

would generate

```
HIGH=98          LOW=60   AVERAGE=79
```

When a ";" and "," are mixed, the next variable or string is printed at the next variable blank column location.

A PRINT statement without an expression list is a valid statement. Execution of this statement results in the output of one blank line, as the example following illustrates.

```
10   PRINT "THERE SHOULD BE TWO BLANK LINES BETWEEN HERE AND"
20   PRINT
30   PRINT
40   PRINT "HERE!"
```

would generate

THERE SHOULD BE TWO BLANK LINES BETWEEN HERE AND

HERE!

5.8 TAB Statement

Output formatting can also be controlled by use of the TAB function.

Form:

TAB (<expression>)

The expression in the TAB function specifies the horizontal column position in which the print item following the TAB will begin printing. The TAB function may contain any expression as its argument. The expression is evaluated and its integer portion used. If the result is greater than the line size, the specified print item will be printed on the next output line. If the column specified by the integer part of the expression has already been passed in the current print line, the TAB function will be ignored and the print item will be output at the current position in the print line. The TAB function may be used to format output into columns on the output device.

Examples:

```
10   PRINT "BIG"; TAB(20);"SPACE"
```

will generate

```
BIG                          SPACE
```

while:

```
10   PRINT TAB(20); "SPACE";TAB(1);"BIG"
```

while:

```
                             SPACEBIG
```

In the first example, the string "BIG" is output starting in column 1. The TAB function advances th printer to column 20 and outputs the string "SPACE". In the second example, the TAB function advances the printer to column 20 and outputs the string "SPACE". The TAB (1) attempts to return the printer to column 1 in the print line. Since that column position has already been passed, the string "BIG" is output immediately following "SPACE" (the current position on the print line).

Note that the printing of tabs in the keyboard mode is not supported.

5.9 UNIT Statement

The UNIT statement designates the baud rate at which all subsequent printed output will be sent to the terminal. Note that the output is always sent to the LED's, and the keyboard is always active.

Forms:

```
UNIT <expression>
```

The unit number assignments are as follows:

UNIT	OUTPUT DEVICE
0	University Board
1	110 BAUD
2	300 BAUD
3	1200 BAUD
4	2400 BAUD
5	4800 BAUD
6	9600 BAUD
7	19.2K BAUD

Note that the TM990/189 microcomputer works at all baud rates.

5.10 BASE STATEMENT

The BASE statement sets the CRU base address for subsequent CRU operations.

Form:

```
<line number> BASE  <expression>
              BASE  <expression>
```

The BASE statement evaluates the expression and sets the CRU base address to the result for use by the CRB and CRF functions (reference subsections 7.3.1 and 7.3.2). The CRB function addresses bits within -128 to +127 of the evaluated base address. CRF function transfers bits using the evaluated base address as the starting CRU address.

The CRU provides a maximum of 2048 input and output lines that may be individually selected by a 11-bit address. The 11-bit address used by the CRU instructions is actually located in bits 4 through 14 of a workspace register. The evaluated expression of the BASE statement is loaded into the entire 16-bits of this workspace register. Therefore, the BASE expression should evaluate to twice the actual (physical) CRU base address desired since only bits 4 through 14 are used. The least significant bit of the BASE expression value is ignored for CRU operations. Therefore, all expressions should evaluate to an even number. The range of valid expressions is from 0 to 4095.

Examples:

```
10   BASE 64
20   CRF(0)=-1
30   BASE 100
40   CRB(-1)=0
```

Statement 10 sets the CRU BASE address to 64 (physical address of 32), and statement 20 outputs a 16-bit -1 value. Statement 30 sets the CRU BASE address to 100 (physical address of 50, and statement 40 sets the CRU bit displaced -1 from the base (physical address of 49) to zero.

5.11 CALL STATEMENT

The CALL statement allows the user access to assembly language subroutines.

FORMS:

```
<line number> CALL <expression>
              CALL <expression>
```

where expression is the decimal address of the assembly language subroutine (i.e., the location at which it resides in memory).

The CALL statement initiates a branch and link and execution resumes at the address specified in the call. If the workspaces used are the same as those used by University BASIC, a branch indirect to Register 11 will return control to the calling program at the following line. If user-specified registers are employed, an RTWP returns to the old workspace where a branch indirect to Register 11 will return control to the calling program. Note that only Registers 0 through 7 should be used when in University BASIC workspace.

The following program uses the CALL statement to access an assembly language routine. Note that the RAM area chosen in this example is totally arbitrary and no RAM area that is used by University BASIC should be used for assembly language programming. The following assembly language routine outputs a tone and returns to the calling University BASIC program. This program also puts a value in R0 of the University BASIC workspace, where it can be accessed by the 'SUB' function. The 'MEM' statement is used to load the assembly language program into RAM, where it accesses the routine via 'CALL' and 'SUB' statements.

```
10 FOR I=1060 TO 1109
20 INPUT X: MEM(I)=X
30 Y=MEM(I): PRINT Y
40 NEXT I
50 MEM(1033)=1 :REM PASS VALUE USED BY ROUTINE
60 PRINT "DO CALL"
70 CALL 1060
80 PRINT "DO SUB"
90 Z=SUB 1060
100 PRINT Z :GOTO 80
```

INPUT NUMBERS UNDER 'DECIMAL'

```
                    DECIMAL

0001 0424                    AORG >0424          ;CHANGE WORKSPACE
0002 0424 0420   04 32       BLWP @>042E
     0426 042E   04 46
0003 0428 C020 192 32        MOV  @>0408,R0      ;LOAD FUNCTION VALUE
     042A 0408   04 08
0004 042C 045B   04 91       B    *R11           ;GO BACK TO UBASIC PROGRAM
0005 042E                    AORG >042E          ;START OF PROGRAM
0006 042E 0402   04 02       DATA >0402          ;ADDR OF NEW WP
0007 0430 0432   04 50       DATA >0432          ;ADDR OF PROG. START
0008 0432 020C   02 12       LI   R12,>043C      ;BASE ADDR. TO SPEAKER
     0434        04 60
0009 0436 0201   02 01       LI   R1,>0200       ;LOAD TONE DURATION
     0438 0428   02 00
0010 043A 0202   02 02       LI   R2,>00A0       ;LOAD TONE PITCH
     043C 00A0   00 160
0011 043E 1D00   29 00       SBO  0              ;SET SPEAKER BIT TO ONE
0012 0440 0602   06 02       DEC  R2             ;DO WAIT LOOP
```

```
0013 0442 16FE   22 254 JNE   T1              ;JUMP IF NOT EQUAL TO 0
0014 0444 0202   02 02  LI    R2,>00A0        ;LOAD TONE PITCH AGAIN
     0446 00A0   00 160
0015 0448 1E00   30 00  SBZ   0               ;SET SPEAKER BIT TO ZERO
0016 044A 0602   06 02  DEC   R2              ;DO WAIT LOOP
0017 044C 16FE   22 254 JNE   T2              ;JUMP IF NOT EQUAL TO 0
0018 044E 0601   06 01  DEC   R1              ;DECREMENT DURATION LOOP
0019 0450 16F4   22 244 JNE   L1              ;IF NOT ZERO DO TONE AGAIN
0020 0452 0A13   10 19  SLA   R3,1            ;MULTIPLY UBASIC VARIABLE BY 2
0021 0454 0380   03 128 RTWP                  ;RETURN TO UBASIC WORKSPACE
```

5.12 TONE STATEMENT

The TONE statement provides sound capability in University BASIC.

Forms:

```
<line number>  TONE   <exp1>,<exp2>;[<exp1>,<exp2>]
               TONE   <exp1>,<exp2>;[<exp1>,<exp2>]
```

where <exp1> is the pitch and <exp2> is the length of time.

Example:

```
100 FOR I=1 TO 25
110 FOR J=1 TO 15
120 TONE I,J
130 NEXT J
140 NEXT I
150 PRINT I:PRINT J
160 STOP
RUN
```

Line 120 directs the TONE command to do Tone I for "J" number of clock TICS.

5.13 LOAD STATEMENT

Refer to subsection 4.4.1 for a description of LOAD Statement.

SECTION VI

CHARACTER STRINGS

6.1 GENERAL

ASCII character strings are stored in the same variables as are other University BASIC variables. Variables are designated as containing character strings by program content or semantics. Any variable or array may contain ASCII characters and, in fact, may be filled with ASCII characters and numbers at the same time. String variables are designated by preceding the variable name with a dollar sign. Otherwise, the variable is treated as a number. ASCII characters are stored in contiquous memory locations with a null character terminating the string. A DIM statement will ensure that enough memory for a string variable has been set aside to store all the characters. If this is not done, other contiguous variables may be destroyed.

6.2 CHARACTER ASSIGNMENT

When a string assignment is made the actual characters are moved to the new variable.

Form:

```
$ VAR = <$VAR>
$ VAR =  "<character string>"
```

Characters are transferred one by one until a null byte is found.

Examples:

```
10   $I1="Y"
20   $J0=$J1
30   $N(4,0) = "CHARACTER STRING"
```

A character string is referred to as <$VAR> and implies either a literal string or a dollar sign preceding a variable. $<VAR> implies a character ONLY of the form dollar sign preceding a variable.

ASCII comparisons of the following form are valid:

```
IF <$VAR> <RELATION> <$VAR>  THEN  <BASIC STATEMENT>
```

Examples:

```
100   IF $I1="Y"  THEN GOTO 500
110   IF $N(I,O) =$B(J,O) THEN GOSUB 600
```

A dimensioned string variable can have a byte index into the character string by following the subscripts with a semicolon and the byte displacement. The range of the index is from 1 through the last byte of the ASCII string. $A(0;1) is equivalent to $A(0).

Example:

```
10   DIM A(13)
20   $A(0)="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
30   PRINT $A(0)
40   PRINT $A(0;1)
50   PRINT $A(0;10)
60   STOP

RUN
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
JKLMNOPQRSTUVWXYZ

STOP AT 60

10   DIM A(13),B(13)
20   $A(0)="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
30   $B(0)=$A(0;10)
40   $A(0;2)=$B(0;2)
50   PRINT $A(0), $B(0)
60   STOP

RUN
AKLMNOPQRSTUVWXYZ  JKLMNOPQRSTUVWXYZ

STOP AT 60
```

6.3 CHARACTER CONCATENATION

Strings are concatenated by using the "+" operator.

Form:

```
$<VAR>  =  <$VAR> + <$VAR> +...
```

Concatenation operations may be chained together and the final string will automatically be terminated with a null by University Basic.

```
10   DIM A(10), B(10)
20   $A(0)="ABCDE"
30   $A(0)=$A(0)+"FG"+"HIJK"
40   PRINT $A(0)
50   STOP
```

```
RUN
ABCDEFGHIJK

STOP AT 50
```

6.4 CHARACTER PICK

Characters can be picked from one variable into another by using the assignment operator.

Form:

```
$<VAR> = <$VAR> , <EXP>
```

The expression is evaluated and the resulting number specifies the number of bytes to be assigned. The string is then terminaated with a null. Note that if the expression evaluates to a non-positive value, no character pick will occur.

Example:

```
10   DIM A(10),B(10)
20   $A(0)="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
30   $B(0)=$A(0;4),6
40   $B(0;5)=$A(0),1
50   PRINT $B(0)
60   STOP

RUN
DEFGA

STOP AT 60
```

6.5 CHARACTER REPLACEMENT

Character replacement is very similar to character pick with the exception that a null is not placed at the end of the string.

Form:

```
10    DIM A(10),B(10)
20    $A(0)="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
30    $B(0)=$A(0;4),6
40    $B(0;5)=$A(0);1
50    PRINT $B(0)

RUN
DEFGAI

STOP AT 60
```

6.6 BYTE REPLACEMENT

Individual bytes may be altered by using the numeric equivalent of an ASCII character along with the "%" operator.

Form:

```
$<VAR> = %<EXP> ...
```

The evaluated expression specifies the byte code to replace in the string variable. Byte replacements may be chained together.

Example:

```
10    DIM A(10),B(10)
20    $A(0)=%65%66%0
30    PRINT $A(0)
40    STOP

RUN
AB

STOP AT 40
```

SECTION VII

University BASIC FUNCTIONS

7.1 GENERAL

University Basic includes several predefined mathematical, string, and miscellaneous functions. A function is called by using the following form in any statement where a variable may be used:

function name (argument)

where

function name is a three-letter name
argument may be an expression or variable.

The specified function of the argument replaces the function name in the statement in which it is used. Functions may be used instead of, or in combination with, variables in almost all University Basic statements such as: LET, PRINT, IF, FOR, etc.

7.2 MATHEMATICAL FUNCTIONS

Paragraphs 7.2.1 and 7.2.2 describe the mathematical functions and their associated forms provided by University Basic.

7.2.1 Absolute Value Functon (ABS)

The absolute value function (ABS) obtains the absolute value of a positive or negative number. The argument entered following the function name is the variable name or numeric value for which the absolute value is required. The function returns a non-negative argument unaltered and returns the absolute value of a negative argument.

Example:

```
10   INPUT X
20   PRINT SQR(ABS(X))
30   STOP
```

7.2.2 Square Root Function (SQR)

The square root (SQR) function returns the square root value of the specified argument. The argument entered following the function returns the square root of the argument. An error message (*ERROR 17 AT XXXX) is produced if the argument is negative).

Example:

```
10   INPUT K
20   PRINT SQR(K)
30   STOP
```

Executing the above example produces:

```
? 2
 1
```

NOTE: If the square of the number does not evaluate to an integer, the square root of the next-lowest number (whose square root evaluates to an integer) is returned.

7.3 MISCELLANEOUS FUNCTIONS

The miscellaneous functions described in paragraphs 7.3.1 through 7.3.6 are supported by University Basic.

7.3.1 CRU Single Bit Function (CRB)

A CRU bit, addressed relative to a base displacement, is either read or stored according to program context. The displacement ranges from -128 to +127. (Refer to Section 5, paragraph 5.10 for details on the BASE statement.) The function returns a 1 if the CRU bit is set, and a 0 if not set. Likewise, the selected CRU bit is set to 1 if the assigned value is non-zero and to 0 if the assigned value is zero. For example:

```
CRB(10)=0
```

will clear the tenth bit relative to the base, while

```
CRB(11)=1 or CRB(11)=345
```

will set the eleventh bit on. Also,

```
IF CRB(5) THEN J=4
```

will set J=4 if the fifth bit is 1.

Form:

```
CRB(<exp>)=<exp>
```

7.3.2 CRU Field Function (CRF)

The specified number of bits are transferred to or read from the CRU starting at the address set by the BASE statement. (Refer to Section 5, paragraph 5.10 for details on the BASE statement.) The

specified number of bits ranges from 0 to 15. If zero, all 16 bits will be transferred. For example:

```
CRF(0) = -1
```

transfers 16 bits to the CRU address specified by the BASE statement. While,

```
VAL=CRF(8)
```

reads 8 bits from the CRU base address and stores the result in VAL. (Bit 1 is the least significant bit of VAL).

Form:

```
CRF(<exp>)=<exp>
```

7.3.3 Key Function (KEY)

The key function (KEY) conditionally samples the keyboard in run-time mode. When the argument is zero, the value of the last key struck is returned and the key register is reset. Otherwise, a value of 0 is returned. For example,

```
I = KEY(0)
```

returns the last key struck, or a 0 if none of the keys were struck; while

```
IF KEY(65) THEN PRINT "A"
```

prints "A" if the last key entered was "A".

7.3.4 Delta Time (TIC) Function

The delta time (TIC) function samples a realtime clock and returns the current TIC value minus the expression value. For example:

```
T = TIC(0)
```

obtains current time, and

```
D = TIC(T)
```

calculates elapsed time since the time stored in the variable T (i.e., TIC (T) = TIC (0) - T.

The TIC function utilizes the interval timer of the TMS9901, programmed to generate an interrupt (or TIC) every second when the system clock rate is at 2MHz.

Example:

```
10   $B=%7%0:THIS IS THE CODE FOR CONTROL 'G' (BELL)
20   A=TIC(0)
30   IF TIC(A)<>25 THEN GOTO 30
40   PRINT $B
50   GOTO 20
```

7.3.5 Memory Modification (MEM) Function

The memory modification (MEM) function reads or modifies a memory location (byte) as specified by the argument. For example:

```
M = MEM(65)
```

reads the byte from decimal integer location "65", while

```
MEM(65) = 25
```

stores a 25 at decimal integer location "65".

Example:

```
M = MEM (1)
```

reads the byte from location 1, while

```
MEM (1000) =2
```

stores a 2 at location 1000.

7.3.6 RANDOM NUMBER (RND) FUNCTION

The random number function (RND) is used to generate a psuedo random number between 0 and 32767.

Form: RND <exp>

where exp is a number between 0 and 32767.

Example:

```
Y=RND 10
```

would generate a random number between 0 and 10. Note that the RND function does not support negative numbers.

7.3.7 SUB Function

The SUB function essentially performs in the same manner as the CALL statement by performing a branch and link to the address

7.3.7 SUB Function

The SUB function essentially performs in the same manner as the CALL statement by performing a branch and link to the address specified in the call, but returns the data in Register 0. (Register 0 through Register 4 only should be used.)

The following program uses the CALL statement to access an assembly language routine. Note that the RAM area chosen in this example is totally arbitrary and no RAM area that is used by University BASIC should be used for assembly language programming. The following assembly language routine outputs a tone and returns to the calling University BASIC program. This program also puts a value in R0 of the University BASIC workspace, where it can be accessed by the 'SUB' function. The 'MEM' statement is used to load the assembly language program into RAM, where the program is accessed via 'CALL' and 'SUB' statements.

```
10 FOR I=1060 TO 1109
20 INPUT X: MEM(I)=X
30 Y=MEM(I): PRINT Y
40 NEXT I
50 MEM(1033)=1 :REM PASS VALUE USED BY ROUTINE
60 PRINT "DO CALL"
70 CALL 1060
80 PRINT "DO SUB"
90 Z=SUB 1060
100 PRINT Z :GOTO 80
```

INPUT NUMBERS UNDER 'DECIMAL'

DECIMAL

```
0001 0424                    AORG >0424          ;CHANGE WORKSPACE
0002 0424 0420  04 32        BLWP @>042E
     0426 042E  04 46
0003 0428 C020 192 32        MOV  @>0408,R0      ;LOAD FUNCTION VALUE
     042A 0408  04 08
0004 042C 045B  04 91        B    *R11           ;GO BACK TO UBASIC PROGRAM
0005 042E                    AORG >042E          ;START OF PROGRAM
0006 042E 0402  04 02        DATA >0402          ;ADDR OF NEW WP
0007 0430 0432  04 50        DATA >0432          ;ADDR OF PROG. START
0008 0432 020C  02 12        LI   R12,>043C      ;BASE ADDR. TO SPEAKER
     0434       04 60
0009 0436 0201  02 01        LI   R1,>0200       ;LOAD TONE DURATION
     0438 0428  02 00
0010 043A 0202  02 02        LI   R2,>00A0       ;LOAD TONE PITCH
     043C 00A0  00 160
0011 043E 1D00  29 00        SBO  0              ;SET SPEAKER BIT TO ONE
0012 0440 0602  06 02        DEC  R2             ;DO WAIT LOOP
0013 0442 16FE  22 254       JNE  T1             ;JUMP IF NOT EQUAL TO 0
0014 0444 0202  02 02        LI   R2,>00A0       ;LOAD TONE PITCH AGAIN
     0446 00A0  00 160
```

```
0015 0448 1E00   30 00   SBZ   0          ;SET SPEAKER BIT TO ZERO
0016 044A 0602   06 02   DEC   R2         ;DO WAIT LOOP
0017 044C 16FE   22 254  JNE   T2         ;JUMP IF NOT EQUAL TO 0
0018 044E 0601   06 01   DEC   R1         ;DECREMENT DURATION LOOP
0019 0450 16F4   22 244  JNE   L1         ;IF NOT ZERO DO TONE AGAIN
0020 0452 0A13   10 19   SLA   R3,1       ;MULTIPLY UBASIC VARIABLE BY 2
0021 0454 0380   03 128  RTWP             ;RETURN TO UBASIC WORKSPACE
```

APPENDIX A

UNIVERSITY BASIC ERROR CODES

ERROR	EXAMPLE/EXPLANATION
01 = SYNTAX ERROR	10 LET X>B (incorrect) 10 LET X=B (correct) (Violates language syntax.)
02 = UNMATCHED PARENTHESIS	10 LET X=(A+B (incorrect) 10 LET x=(A+B) (correct) (Parenthesis must be matched.)
03 = INVALID LINE NUMBER	32768 LET X=B (incorrect) (There are only 32767 possible line numbers in University BASIC.)
04 = ILLEGAL VARIABLE NAME	10 X=AB1 (incorrect) 10 X=A1 (correct) (Variable names must begin with a capital letter and may be followed by one capital letter or a number 0-9.)
05 = TOO MANY VARIABLES	A maximmum of 120 distinct variables is allowed. Attempts to define more than this number will result in this error.
06 = ILLEGAL CHARACTER	FOR A 1 TO 10 (incorrect) FOR A=1 TO 10 (correct) (The blank between 'A' and '1' is interpreted as an illegal character.)
07 = EXPECTING OPERATOR	Not implemented in this release.
08 = STACK OVERFLOW	This error occurs when the GOSUB nesting level exceeds 8.
09 = STACK UNDERFLOW	This error occurs when the FOR/NEXT nesting level exceeds 4.
10 = STORAGE OVERFLOW	This error occurs when University BASIC has used all existing RAM.

11 = NEXT W/O FOR	10 PRINT I 20 NEXT I (incorrect) 10 FOR I=1 TO 5 20 PRINT I 30 NEXT I (correct) (A FOR statement always requires a corresponding NEXT statement).
12 = EXPRESSION TOO COMPLEX	Not implemented in this release.
13 = NO SUCH LINE NUMBER	10 J=9 20 PRINT J 30 GOTO 15 (incorrect) 40 STOP (Line 15 does not exist in this program.)
14 = EXPECTING STRING VARIABLE	10 IF $A=X THEN GOTO 20 (incorrect) 10 IF $A=$B THEN GOTO 20 (correct) (The "X" is not a string variable.)
15 = SUBSCRIPT OUT OF RANGE	10 DIM A(10) 20 A(20)=5 (incorrect) 10 DIM A(10) 20 A(10)=5 (correct) (In the first example, A(20) exceeded the range set by A(10)).
16 = TOO MANY SUBSCRIPTS	10 DIM A(10,10,10) (incorrect) 10 DIM A(10,10) (correct) (The DIM statement may have no more than 2 subscripts.)
17 = SQUARE ROOT OF NEGATIVE #	10 X=SQR(-9) (incorrect) 10 X=SQR(9) (correct)
18 = INTEGER OVERFLOW	10 X=32768 (incorrect) 10 X=32767 (correct) (Upper limit of a variable is 32767.)
19 = DIVISION BY ZERO	10 X=10/0 (incorrect) 10 X=10/2 (correct) (Division by zero is prohibited

	in University BASIC.)
20 = ILLEGAL DELIMITER	10 IF X=B; THEN GOTO 30 (incorrect) 10 IF X=B THEN GOTO 30 (correct) (";" is an illegal delimiter in this statement.)
21 = EXPECTING VARIABLE	Not implemented in this release.
22 = TIMEOUT ERROR	This error occurs when improper interface between the microcomputer and the audio cassette takes place.

APPENDIX B

STATEMENT AND COMMAND SUMMARY

The following pages contain a summary of all University BASIC commands and statements, and provide a short description of each.

EDIT MODE COMMANDS

Edit mode commands aid in writing, editing, and debugging University BASIC programs. Note that "#" indicates a key (or keys) on the TM990/189 microcomputer keypad while "*" indicates a terminal keyboard such as the Lear Siegler ADM.

SYNTAX	EXAMPLE/EXPLANATION
(Shift) 9 (#)/ ESC (*)	Cancel input line or break program execution.
(Shift) M (#)/ (Shift) Rubout (*)	Backspace/Delete.
(Shift) J (#) Control A (*)	Initiate automatic line numbering. (Increments of 10)
(Shift) V (#)	Moves keypad display left 6 spaces.
(Shift) W (#)	Moves keypad display left by 1 space.
(Shift) Y (#)	Moves keypad display right by 1 space.
(Shift) Z (#)	Moves keypad display right by 6 spaces.

University BASIC commands direct and control system operations. Commands cause immediate computer interaction thereby allowing operator control. Commmands may only be entered one per line and may not be abbreviated.All letters must be entered in upper case.

SYNTAX	EXAMPLE/EXPLANATION
<ln> LIST	LIST
	List the user's program.(<ln> LIST will list from the specified line number through the end of the program or until the Escape key is entered).
LOAD	LOAD
	Loads a previously recorded University BASIC program from cassette.
LOAD <program #> <ln> LOAD <program #>	LOAD 10 LOAD Loads programs stored in EPROM into RAM.
NEW	NEW
	Clears current user program, variables, pointers, and stacks, and prepares for entry of new progr
RUN	RUN
	Begin program execution at the lowest line number.
SAVE	SAVE
	Records University BASIC program onto cassette.
SIZE	SIZE
	Display current program size, allocated variable variable space, and available memory in bytes.

University BASIC statements form the basis of all University BASIC programs. Statements are typically entered into a program with line numbers and are executed with the RUN command. Statements may also be entered without line numbers if they are to be executed immediately. Numerous statements (with the exception of NEXT, REM, and FOR) may appear on one line if separated by a colon (:).

SYNTAX	EXAMPLE/EXPLANATION
<ln> BASE <(exp)>	BASE (256) Sets CRU base for subsequent CRU operations.
<ln> DIM <var[(dim,dim])><,<var([dim,dim])>	DIM A(10), DOG(3,10) Allocates user variable space for dimensioned or array variables. (Note that only two dimensions may be accepted per DIM statement.)
<ln> FOR <simple var>=<exp> to <exp> STEP <exp> EOL	For I=1 to 10 STEP 2 The FOR statement is used with the NEXT statement to open and close a program Both identify the same control variable. The FOR statement specifies the control variable and assigns the starting, ending, and optionally stepping values. Note that there is no pre-test done, (i.e., it will always execute once).
<ln> NEXT <simple var> NEXT 1 EOL	*(see 'NEXT' statement)
<ln> GOSUB ln EOL	GOSUB 200 Transfer program execution to an internal UBASIC subroutine beginning at the specified line number.
<ln> RETURN	RETURN * (see RETURN statement)
<ln> GOTO ln EOL	GOTO 200

Transfer program execution to the specified line number.

<ln> IF condition THEN statement(s) EOL

IF I=0 THEN I=J : GOTO 200

Causes conditional execution of the statement(s) following THEN. Statements following THEN execute on TRUE condition.

<ln> INPUT {<num-var> / <string-var>}[{,/;} {<num-var> / <string-var>}] ... {,/;}

INPUT I, $B

Places input and numeric values entered from the keyboard into variables in the INPUT list

<ln> [LET] <var> = <exp>

LET A=B*4

Evaluates and assigns values to variables or array elements. The LET is optional.

<ln> NEXT EOL
<simple variable>

NEXT I

Delimits end of FOR loop. The simple variable must match FOR control variable.

<ln> PRINT <exp>[,exp]...

PRINT A, B, $NAM

Print (without formatting) the evaluated expressions to the terminal device.

<ln> REM <text> EOL

REM comment lines for documentation. Inserts comment lines into program.

<ln> RETURN

RETURN

Return from UNIVERSITY BASIC subroutine and remove top address from GOSUB stack.

<ln> UNIT <exp>

UNIT 3

Designate the device(s) to receive all printed output.

```
UNIT = 0  University Board
     = 1  110  Baud
     = 2  300  Baud
     = 3  1200 Baud
     = 4  2400 Baud
     = 5  4800 Baud
     = 6  9600 Baud
     = 7  19.2 k Baud
```

•

<ln> CALL <exp>

CALL

Allows access to assembly language subroutines. <Exp> is the address at which the subroutine resides in memory.

<ln> TONE <expl>,<exp2>;[<expl>,<exp2>]

TONE I,J

Provides sound capability. Expl = pitch and exp2 = length of time.

<ln> LOAD <exp>

Loads from designated address on EPROM.

FUNCTIONS

University BASIC provides several predefined mathematical, string, and system functions which simplify program entry and development Any University BASIC function may be used in any statement where a variable may be used. A function may be called by using "function name (argument)", where the function is the three-letter name and the argument may be any expression or variable. The specified function of the argument replaces the function name in the statement in which it is used.
NJ

SYNTAX	EXAMPLE/EXPLANATION
ABS (<exp>)	A=ABS(B) Absolute value of expression
CRB (<expl>)	A=CRB(-1) Reads CRU bit as selected by the CRU hardware base = exp. Exp is valid over range -127 +128.
CRB (<exp>) =<exp2>	CRB(-4)=0 Set or reset CRU bit as selected by CRU hardware base = expl.If exp2 is non-zero, the bit will be set, else reset. Expl is valid for -127 thru 126.
CRF (<expl>)	A=CRF(4) Read n CRU bits as selected by CRU base where exp evaluates to n. Exp is valid for 0 thru 15. If exp=0, 16 bits will be read.
CRF (<expl>)= <exp2>	CRF(5)=150 Output expl bits of exp2 to CRU lines as selected by CRU BASE. Expl is valid for 0 thru 15. If expl = 0, 16 bits will be output.
MEM (<exp>)	A = MEM 123 A = MEM(167) Read byte from user memory at decimal integer address specified by exp.
MEM (<expl>)=	exp2 MEM(45)=123

	MEM(1000)= 15 Store byte exp2 into user memory at decimal integer address specified by expl.
KEY <exp>	A = KEY(0) IF KEY(65) THEN PRINT "A" Conditionally samples the keyboard in run-time mode. If exp=0, return the value of last key struck and clear key register. (Zero is returned if no other key was struck). If exp is <> 0, compare last struck key with value of exp. If they are the same, a value of 1 is returned and the key register is reset, if not equal, then return a 0.
SUB <exp>	SUB
RND <exp>	A = RND Returns a random number between 0 and 32767.
SQR (<exp>)	A = SQR(B) Square root of expression.
TIC (<exp>)	T1=TIC(0) T2=TIC(T1) Returns the number of time TICs less the expression value. One TIC equals 1 second.

STRINGS

String variables in University BASIC are designated by preceeding the variable name with a dollar sign ($). ASCII character strings are stored in contiguous memory byte locations with a null character terminating the string. Dimensioned string variables may have a byte index following the subscript(s) to indicate a byte position within the specified string. This is indicated by following the subscripts with a semicolon and the byte displacement (e.g., $A(0;5)).

FUNCTION/SYNTAX	EXAMPLE/EXPLANATION
Assignment <string-var>= {<string-var> / <string-constant>}	$A(0)=B(0) $A(0)=B(0;5) Store string into string-var ending string with a null. A dimensioned string variable can have a byte index into the character string by following the subscripts with a semicolon and the byte displacement. (Range of index is 1 - last byte of the ASCII string.)
Concatenate <string-var> {<string-var> / <string-constant>} . {<string-var> / <string-constant>}	$A(0)=$A(o)+"END" Concatenate specified string variables or string constants.

Pick

<string-var1>=<exp>, { <string-var2> | <string-constant2> }

```
$I=$A(0;2),3
$J="ABCDE", 3
```

Pick exp number of characters from string2 into string-var1 ending string with null.

Replace

<string-var>=<exp>; { <string-var2> | <string-constant2> }

```
$B(0;2)=$A(0);1
$B(0;2)="....";2X
```

Replace exp number of characters of string-var1 with string 2.

Byte Assignment

```
$A(0)=% 65% 66% 0
```

<var>=%<exp>...

Alter individual bytes using the numeric equivalent of an ASCII character and the "%" operator.

INPUT OPTIONS

The following options are available for use with the INPUT statement to provide the University BASIC user with enhanced terminal input capability. For additional details on their use, refer to the INPUT statement, Section V.

SYNTAX	EXAMPLE/EXPLANATION
;	INPUT A;B Delimit expressions for multiple inputs
;	INPUT; A INPUT A; Supress prompting if before variable, or CR LF if at end of line.
"STRING"	INPUT "YES or NO?";$A Prompt with string and then get input. Equivalent to : PRINT "YES or NO?";::INPUT;$A

OUTPUT OPTIONS

University BASIC provides the following options for use with the PRINT statement. They provide powerfulprint formatting capability for all user outputdirected to the terminal and/or auxiliary device (see UNIT statement). For additional information on these formatting options, refer to Section 5.

SYNTAX	EXAMPLE/EXPLANATION
;	PRINT A;B PRINT A; Delimits expressions or supresses CR LF if at end of line.
,	PRINT A,B Tab to next print field.
TAB (<exp>)	PRINT TAB (50);A Tab to column specified by exp.
string	PRINT "HI";$A(0) Print string or string variation.

SPECIAL CHARACTER

The following characters have a special meaning when encountered in program statement lines:

CHARACTER	USE
:	Statement separator when entering multiple statements per line.
;	Equivalent to "PRINT" statement

ARITHMETIC OPERATIONS

A=B	Assignment
A-B	Subtraction
B+B, $A+$B	Addition or string concatenation
A*B	Multiplication
A/B	Division
-A	Unary Minus
+A	Unary Plus

RELATIONAL OPERATORS

The relational operators are binary operators that operate on two arithmetic expressions. They return values of 1 (TRUE) or 0 (FALSE).

A=B	TRUE if equal, else FALSE
A<B	TRUE if less than, else FALSE
A<=B	TRUE if less than or equal, else FALSE
A>B	TRUE if greater than, else FALSE
A>=B	TRUE if greater than or equal,else FALSE
A<>B	TRUE if not equal, else FALSE

OPERATOR PRECEDENCE

1. Expressions in parentheses
2. Negation
3. *,/
4. +,
5. <=,<>
6. >=,>
7. =,>

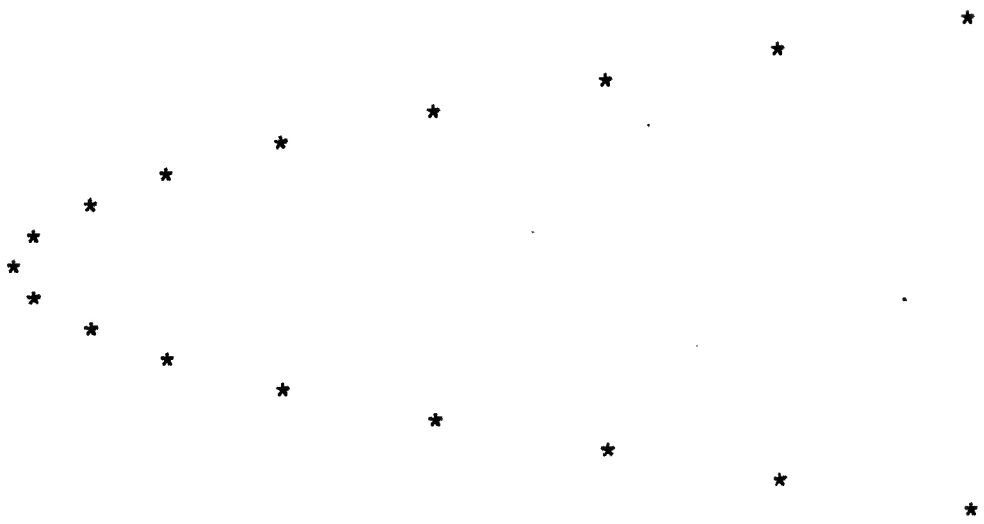
APPENDIX C

SAMPLE PROGRAMS

C.1 PARABOLA

This program demonstrates the use of the TAB function.

```
100 FOR I=-8 TO 8
110 PRINT TAB(I*I)"*"
120 NEXT I
130 STOP
RUN
```



C.2 BLIP

When executed, this program blinks LEDs numbered 2, 3 and 4 on the TM990/189 microcomputer while making tones on the sound disk.

```
10 BASE 0
20 FOR I=17 TO 19
30 CRB(I)=1
40 IF I=17 THEN CRB(19)=0
50 IF I=18 THEN CRB(17)=0
60 IF I=19 THEN CRB(18)=0
70 TONE I-10,80
80 NEXT I
90 GOTO 10
100 STOP
```

C.3 TIME-OF-DAY CLOCK

The following program creates a time-of-day clock.

```
2 DIM A(8),B(2),S(2)
4 INPUT "HOURS"$B(0):GOSUB 300
5 A(0)=S(1):A(1)=S(2)
6 IF A(0)<2 THEN GOTO 220
7 IF A(1)>3 THEN GOTO 210
8 IF A(0)>2 THEN GOTO 210
9 INPUT "MINUTES"$B(0):GOSUB 300
10 A(3)=S(1):A(4)=S(2)
11 IF A(3)>5 THEN GOTO 210
12 IF A(4)>9 THEN GOTO 210
14 INPUT "SECONDS"$B(0):GOSUB 300
15 A(6)=S(1):A(7)=S(2)
16 IF A(7)>9 THEN GOTO 210
18 IF A(6)>5 THEN GOTO 210
20 $A(8)="A":A(8)=((A(8)/256)-39)*256:$A(2)=":":$A(5)=":"
30 B=TIC(0)
40 IF TIC(B)<>1 THEN GOTO 40
50 GOSUB 500
60 IF A(7)=9 THEN GOTO 80
70 A(7)=A(7)+1:GOTO 30
80 A(7)=0:IF A(6)=5 THEN GOTO 100
90 A(6)=A(6)+1:GOTO 30
100 A(7)=0:A(6)=0:IF A(4)=9 THEN GOTO 120
110 A(4)=A(4)+1:GOTO 30
120 A(4)=0:IF A(3)=5 THEN GOTO 140
130 A(3)=A(3)+1: GOTO 30
140 A(3)=0:A(4)=0:IF A(0)>1 THEN GOTO 155
146 IF A(1)=9 THEN GOTO 160
150 A(1)=A(1)+1:GOTO 30
155 IF A(1)>2 THEN GOTO 200
157 GOTO 150
160 A(1)=0:A(0)=A(0)+1:GOTO 30
200 A(0)=0:A(1)=0:A(3)=0:A(4)=0:A(6)=0:A(7)=0:GOTO 30
210 PRINT "ERROR":GOTO 2
220 IF A(1)>9 THEN GOTO 210
230 GOTO 8
300 FOR I=1 TO 2
310 $S(I)=$B(0;I),1
320 S(I)=(S(I)/256)-48
330 NEXT I
340 RETURN
500 PRINT $A(8);
510 PRINT A(0);A(1);$A(2);A(3);A(4);$A(5);A(6);A(7)
520 RETURN
```

APPENDIX D

UNIVERSITY BASIC COLOR VIDEO COMMANDS

D.1 GENERAL

In addition to the commands listed in Appendix B, color video commands are available to the user of University BASIC. These commands are listed below, and their formats provided. Additional information which includes a full description of each command and University BASIC demonstration programs may be obtained in the Color Video Using TMS9918 and University BASIC Application Report, MP723.

D.2 VDP COMMAND/FORMAT LISTING

COMMAND NAME	FORMAT
COLOR	COLOR <exp>
MODE	MODE <exp> [,<exp>]
MOVE	MOVE <exp> MOVE <exp>,<exp>,<exp>
PATTERN	PATTERN <exp>,<string>
SPRITE	SPRITE <exp>,<string> SPRITE <exp>,<exp> [,<exp>] SPRITE <exp>,<exp>,<exp>
VDP	VDP(<exp1>) = <exp2> <var> = VDP <exp>