the digital group
po box 6528 denver, colorado 80206 (303) 777-7133

BUSINESS BASIC

296-001-A-18+3

Business BASIC — Table of Contents

BUSINESS BASIC V1.0

Introduction

Business BASIC is an extension of the original MAXI-BASIC available from the Digital Group. It has been modified, by MicroWorks Inc., so that users of Digital Group systems can more fully utilize their systems including Phidecks, printers, and TV-64 display. Special attention has been paid to the needs of those individuals intending to write business application programs. Included in Business BASIC are many new features not previously available to Digital Group users, such as a complete file handling package, printer drivers with upper and lower case, cursor capability for the TV-64 display and masked conversion of strings to numbers and vice versa. This manual is not designed to teach BASIC; however, experimentation with the programs contained in this manual will allow the uninitiated user to become familiar with the language. It is intended to be used as a reference manual.

Machine Requirements

Business BASIC requires that the system in use contains the PHIMON operating system. No standard operating system is included in Business BASIC, unlike previous releases of BASIC. Pages 1-5 are now used for the file handling routines.

Business BASIC itself occupies 68 pages of memory (17K) and PHIMON occupies 8 pages (2K) plus 4 pages (1K) for the directory buffer. Therefore, the minimum memory configuration is 26K. A standard ASCII keyboard is addressed as input port Ø and a TV-64 display driver is addressed as output port Ø. Full driver routines are provided to operate a Digital Group printer which, if used, must be attached to port 3 (input and output).

Manual Conventions

In this manual certain conventions have been adopted. The term **constant** refers to a numeric constant, i.e., 1, 29, 52736, etc. The term **variable** refers to a numeric variable name, i.e., A, B3, Z9, etc. The term **expr** refers to a numeric constant or a numeric variable name or a valid arithmetic expression, i.e., A+1, A+T5, etc. The term **str constant** refers to an alphanumeric constant, i.e., "TOP", "UNIT2","345", etc. The term **str variable** refers to an alphanumeric variable name, i.e., A$, B8$, R7$, etc. The term **str expr** refers to an alphanumeric constant or an alphanumeric variable name or a valid alphanumeric expression, i.e., A$+B$, R7$+"X", etc Brackets **[ ]** surrounding a parameter indicate that it is optional. All control characters will be denoted by **CTRL-x**, e.g., CTRL-H. CTRL-H is entered by depressing the "CTRL" key and the "H" key at the same time

Usage of Business BASIC

Entering a Program

Each program line is preceded by a statement number used to identify the line number. The valid range for these numbers is zero through 65535. Business BASIC assumes that any line beginning with a line number is to be processed by the program editor; all others are considered commands or direct execution statements. The program editor serves as the user's method of entering and altering a program. If the statement number is valid and the line contains at least one character beyond the statement number, the line is added to the program. If the statement number is equal to an existing statement number, the existing line is replaced by the new line. If there are no characters beyond the statement number, the line with a matching statement number is deleted. An error will be generated if the statement number is invalid, or if the line is longer than the current line length (see LINE command), or if memory becomes full.

Blanks preceeding a statement number are ignored. The first non-numeric character delineates the statement number from the rest of the line. All blanks are ignored, except those contained in a quoted literal. Blanks are inserted automatically in listings for ease of reading. FOR-NEXT loops are also indented for the same reason.

Corrections

Before typing a carriage return, corrections may be made to a line in two ways. First, typing a RUBOUT (left arrow or ASCII DEL on some keyboards) causes Business BASIC to backspace one character. Typing a CTRL-X (ASCII CANCEL) causes Business BASIC to ignore the entire line.

Multiple Statements Per Line

Multiple statements on one line are allowed. The statements should be separated by a colon.

Example: 1Ø PRINT A : GOTO 4Ø

A statement number may occur only at the beginning of a line.

Direct Execution

All statements may be entered without a statement number for immediate execution. This allows the user to examine values and perform other operations without having to run a program. Multiple statements per line may also be entered for direct execution.

Interrupting Execution

To stop a listing or execution of a program, type a CTRL-C (ASCII ETX). If a list is in progress, the output will stop at the end of the line currently being listed. If program execution is in progress, it will terminate at the end of the current statement being executed. Typing CONT (see "Commands") will resume execution of the program.

Commands

There are two types of directives the user may present to Business BASIC: commands and statements. Commands are defined as those directives which act upon the program contained in memory. Statements, on the other hand, are defined as the individual lines of code which make up the program contained in memory. The following are the commands available in Business BASIC:

AUTO [constant 1] [,constant 2]

This command allows automatic numbering of statements as they are being entered into the system. Constant 1 represents the starting statement number. Constant 2 represents the increment to be used between statement numbers. To exit from the automatic numbering mode, type a CTRL-D (ASCII EOT).

Example: AUTO 20,5
(Begins auto statement numbering at line 20 and indicates an increment of 5 between numbers, i.e., 25, 30, 35, etc.)

CLEAR

This command clears the contents of all variables. All dimensioned variables are deleted and all function definitions are removed.

CONT

This command continues execution of a Business BASIC program after execution of a STOP statement or a CTRL-C interruption.

DELAY [constant]

This command causes a delay to be generated between characters on output. The delay generated is the constant times .00266 seconds, where the constant is 0 to 255. (Note: If the constant is omitted the default value 0 is assumed.)

Example: DELAY 100
(causes a .266 second delay between characters on output.)

LINE constant

This command establishes the maximum line length of the system's terminal device. No input or output line longer than the specified number of characters will be allowed. If an attempt is made to enter a longer line on input, a "length" error will be generated. On output a new line will automatically be generated after the specified number of characters have been sent to the terminal device. The range for the constant is 1 to 132. Maximum line length has no effect when a program is being listed on the screen. Line length may be altered from within the program by executing a "FILL 12890,expr" (see "Statements" below).

LIST [constant 1] [,] [constant 2]

This command is used to list the statements comprising the program currently in memory. There are four variations of this command, each performing a different function. The first variation is LIST with no parameters; this will list the entire program starting with the first statement. The second variation is LIST with constant 1; this will list the statement which has a statement number equal to constant 1. The third variation is LIST with constant 1 followed by a comma, which will list the program starting with the statement number equal to constant 1 through the last statement. The final variation is LIST with constant 1 and constant 2 separated by a comma; this will list all statements from the statement number equal to constant 1 to the statement number equal to constant 2, inclusive.

REN [constant 1] [,constant 2]

This command is used to renumber the entire program. Constant 1 is used to indicate the first statement number in the resulting renumbered program. Constant 2 is used to indicate the increment between statements in the resulting program. If constant 1 or constant 2 is omitted a default value of 10 is assumed.

RUN [constant]

This command instructs Business BASIC to begin execution of the program. The constant is an optional statement number at which to begin execution. If the constant is omitted, execution will begin with the first statement in the program. (See also "Loading and Saving Programs".)

SCR

This command is used to clear the program and data storage area in Business BASIC so that a new program may be entered.

Constants

Storage of all numeric constants and all other numeric items is in Binary-Coded-Decimal (BCD). Eight digits of precision are maintained at all times; therefore, rounding off may occur within a program if necessary. The magnitude range of constants is .1E-63 through .99999999E+63.

Variable Names

Simple numeric variable names are denoted by a single letter or a letter followed by a number (e.g., A, R0, Z9, etc.). Each numeric variable occupies eight bytes of memory. Once assigned a value these bytes cannot be released without issuing a CLEAR or SCR command. Numeric array variable names take the same form as simple numeric variable names. The number of bytes required by a numeric array can be calculated by the following formula:

Number of bytes = (Total number of elements * 5) + (Number of dimensions * 2) + 7

Unlike simple numeric variables, the memory occupied by an array can be released by the program. (See UNDIM.)

String variable names take the same format as simple numeric variables with the addition of a "$" (e.g., A$, R0$, Z9$, etc.). The memory requirements for a string variable can be calculated by the following formula:

Number of bytes = Dimension size of string + 7

Memory occupied by strings may be released in the same way as memory occupied by arrays. There are no string arrays supported by Business BASIC.

Function names are in the same format as simple numeric variable names preceded by "FN" (e.g., FNA, FNR0, FNZ9, etc.). Note that the same variable name can be used to refer to a simple numeric variable, a numeric array, a string and a function definition. As an example, the names A0, A0(0,0), A0$, and FNA0 all refer to different items and no relationship between these items is assumed to exist. DATE is a special variable name in Business BASIC, used to contain today's date, and may be used in statements as a string name. (No substring reference is allowed.) If DATE is used in an INPUT statement an automatic prompt, "Enter date: (MM/DD/YY)", will be generated.

Arrays

Arrays in Business BASIC may be dimensioned with any number of dimensions. Indexing of arrays is zero relative, i.e., the first element of an array is element zero. An array established by the statement 10 DIM X(2) would contain three elements: X(0), X(1) and X(2). Care must be taken so that an array is not re-dimensioned without first having been released. (See UNDIM.)

Strings

Strings in Business BASIC may be dimensioned to any size (with the obvious upper limit of available memory). Strings may be referenced in two ways. The first is to reference the entire string; this is done by using the string name alone (e.g., 20 LET B$=A$ . . . this says "Copy the contents of the entire string called A$ into the entire string called B$".). The second is to reference part of the string (hereafter called a sub-string); this is done by using the string name followed by two parameters enclosed in parentheses. The first parameter indicates the starting position in the string and the second indicates the ending position (e.g., 20 LET B$=A$ (4,7) . . . this says "Copy the contents of the fourth through the seventh character of the string called A$ into the entire string called B$".). If only the first parameter is provided, the substring is assumed to begin with the indicated position and continue through the end of the string (e.g., 20 LET B$=A$(4) . . . this says "Copy the contents of the fourth through last character of the string called A$ into the entire string called B$".).

Combining two strings into one (i.e., concatenation) is accomplished by use of the plus sign (e.g., 20 LET B$=A$+C$ . . . this says "Append the contents of the entire string called C$ to the contents of the entire string called A$ and place the result in the entire string called B$". The strings A$ and C$ are unaffected.).

All strings should be initialized before accessing them using substring notation.

```
Example:  10  DIM A$ (10)
          20  FOR I=1 to 10
          30  A$ =A$+ " "
          40  NEXT I
```

When assigning a value to a string, if the result string is not subscripted the result string is replaced by the assigned value.

```
Example:  A$="ABCDEF"
          PRINT A$
          ABCDEF
```

If the result string is subscripted and the value is longer than the result string, the assigned value will be truncated to fit. Further, if the assigned value is shorter than the result substring, the extra characters in the result substring will be unaffected.

```
Example:  A$="ABCDEF"
          PRINT A$
          ABCDEF
          A$(2,3)="WXYZ"
          PRINT A$
          AWXDEF
          A$(2,3)="R"
          PRINT A$
          ARXDEF
```

When using string IF statements, strings of unequal length will not be considered equal. If two strings of different lengths are equal up to the length of the shorter string, the shorter string is assumed to be the lesser in value. Strings in DATA statements and string constants must be enclosed in quotes.

```
Example:  10  LET A$="ABC"
          30  DATA "XYZ","UNIT"
```

Operators

The valid numeric operators for Business BASIC are: "+" for addition, "-" for subtraction, "*" for multiplication, "/" for division and "↑" for exponentiation. When an arithmetic operation is executed, any exponentiation in the statement is done first. Next, any multiplication or division is executed in left to right order. Finally, any addition or subtraction is executed, also in left to right order.

```
Example:  PRINT 1+2*5+4
          15
```

If execution in some other order is needed, parentheses may be used. If a pair of parentheses appears within another pair, execution starts with the innermost pair and moves outward.

```
Example:  PRINT (1+2)*(5+4)
          27
          PRINT (1+2)*(5+4)+4*2
          35
          PRINT (1+2)*((5+4)+4*2)
          51
```

Relational operators are "=" for equal to, "<" for less than, ">" for greater than, "<>" for not equal to, "<=" or "=<" for less than or equal to, and ">=" or "=>" for greater than or equal to. A relational operation assigns a value of 1 for true and zero for false.

```
Example:  PRINT 2=3
          Ø
          PRINT 2<3
          1
          A$="TEST"
          PRINT (A$="TEST")
          1
```

Boolean operators are AND, OR and NOT. Boolean operands are considered true if not equal to zero and false if equal to zero. Results of boolean operations are 1 or zero.

```
Example:  PRINT 1 AND Ø
          Ø
          PRINT 3 OR Ø
          1
```

(Note: The usual usage for the boolean operators is for complex IF statements, but they can be used as above for other purposes such as program analysis of complex logic structures.)

Loading and Saving Programs

Saving source programs is accomplished by the statement:

SAVE#expr,str expr

where expr is the drive number and the string expr contains the program name.

Loading of previously SAVEd programs is accomplished by the statement:

LOAD#expr,str expr

where expr and the string expr have the same usage as in SAVE.

A special form of RUN (see "Commands") allows for LOADing and executing a source file:

RUN#expr,str expr

where expr and the string expr have the same usage as in LOAD or SAVE.

LOAD, SAVE, and RUN are all executable statements.

```
Example:  1Ø  RUN#Ø,"TSTPRG"
          2Ø  LOAD#1,A$
          3Ø  SAVE#3,"TSTPRG"
```

Statements

The statements available in Business BASIC can be broken down into four categories: data handling, program control, general input/output, and file statements. There is one special statement which does not fall into any of these categories, the REM statement.

REM

```
Format: 1Ø  REM [any comment]
or      1Ø  ' [any comment]
```

This statement is used to place remarks in a program. It is ignored during execution of the program, but will appear in any LIST of the program.

```
Example:  1Ø  REM THIS PROGRAM WRITTEN
              OCT. 15
```

Data Handling Statements

CONVERT

```
Format: 2Ø  CONVERT expr TO str variable (mask)
or      2Ø  CONVERT str variable TO variable
```

The CONVERT statement is used to convert numbers to strings and strings to numbers. The mask is used to indicate the format of the resulting conversion. The mask is made up of the following characters:

- \- = Optional minus sign
- # = Character that will be replaced by a digit
- . = Optional decimal point

```
Example:  LET X=234.56789
          CONVERT X TO A$(###.##)
          PRINT A$
          234.57
          CONVERT X TO A$ (-###.##)
          PRINT A$
          Ø234.57
          X=-234.56789
          CONVERT X TO A$ (-###.##)
          PRINT A$
          -234.57
          ET A$="234.56789"
          CONVERT A$ TO X
          PRINT X
          234.56789
```

DIM

```
Format: 1Ø  DIM variable name1(expr1,expr2 . . . exprn),
            variable name2(expr1, . . . ) , . . .
or      1Ø  DIM str variable name1(expr),str variable
            name 2(expr) , . . .
```

The DIM statement is used to establish the maximum size requirements for strings or numeric arrays. There is no limit to the number of dimensions attributed to a numeric array. A string may have only a single dimension. If no DIM exists for a numeric array, it is assumed to be a one-dimensional array of ten elements. Non-dimensioned strings are assumed to have a maximum of ten characters.

Example: 10 DIM X(2,4,5,2,3),C$(23)

FILL

Format: 10 FILL expr1, expr2

The FILL statement is used to place a value into a specific location in memory. The value of expr2 is placed in the location indicated by expr1. Expr1 may have a value from 0 to 65535, expr2 may have a value from 0 to 255.

Example: 10 FILL 65535,255
(places the hex value FF at hex address FFFF)

LET

Format: 10 [LET] variable = expr
or 10 [LET] str variable = str expr

This statement is used to assign the variable named to the left of the equal sign, the value of the expression to the right of the equal sign. The equal sign should be read as "is replaced by". Thus, the first example below would be read "let A be replaced by the value X plus one".

Example: 10 LET A=X+1
20 LET D$="test"

Note: Multiple assignments such as 10 A=B=0 are not allowed. (This statement is treated as a boolean expression.) The proper form would be 10 A=0 : B=0 .

PACK

Format: 10 PACK (mask) str variable FROM expr

The PACK statement is used to compress numeric data into a string variable. The mask is used to indicate the maximum size of the result string and the number of decimal places in the result. The format of the mask is the same as CONVERT. The length of the result string is calculated by the following formula:

Number of bytes = INT((Number of significant digits indicated by mask+1)/2)

This two-for-one compression will require less memory for storage of numbers with a few significant digits and is particularly useful for size reduction of mass storage records. (See also UNPACK.)

```
Example: X=314.159
         PACK (###.##) A$ FROM X
         PRINT LEN(A$)
         3
```

READ AND DATA

Format: 10 READ variable 1,variable 2, . . . variable n
20 DATA constant 1,constant 2, . . . constant n
(The variable and constants may be string as well as numeric.)

The READ and DATA statements are used to enter predetermined data into a program. The READ statement will assign the values contained in the DATA statement to the variables named in the READ statement. The first variable in the first READ statement will be assigned the first value contained in the first DATA statement in the program. Each successive variable contained in a READ statement will be assigned the next value in a DATA statement. As all values in a DATA statement are used, the next DATA statement in the program is used. Care should be taken that the type of variable and the type of the value agree, i.e., a numeric value for a numeric variable and a string value for a string variable, or an error will occur.

```
Example: 10 READ X,A$,B,Z$
         20 DATA 23,"TEST"
         30 DATA 50,"TEST2"
         40 PRINT X,A$,B,Z$
         RUN
            23    TEST    50    TEST2
```

RESTORE

Format: 10 RESTORE [statement number]

The RESTORE statement is used to instruct Business BASIC to begin using a particular DATA statement. If the optional statement number is given, the DATA statement indicated will be used by the next READ statement. If no statement number is given, the first DATA statement in the program will be used by the next READ statement.

```
Example: 10 RESTORE
         90 RESTORE 85
```

UNDIM

Format: 10 UNDIM variable 1, variable 2, . . . variable n
(Variable name 1 through n may be string or numeric array names.)

The UNDIM statement is used to free up the space occupied by strings and/or numeric arrays. The indicated strings and/or numeric arrays are deleted and all other variables are moved, so that all variables will occupy a contiguous section of memory. All data contained in the variables named in the UNDIM will be lost.

```
Example: PRINT FREE(0)
         5825
         DIM A(15),B$(12)
         PRINT FREE(0)
         5717
         UNDIM A,B$
         PRINT FREE(0)
         5825
```

UNPACK

Format: 10 UNPACK (mask) variable FROM str variable

The UNPACK statement is the opposite of the PACK statement. This statement is used to take a PACKed number in a string and turn it back into a numeric variable. The mask is used to indicate the format of the PACKed number. The masks used in the PACK and UNPACK statements should be the same. The format of the mask is the same as CONVERT.

```
Example:  X=314.159
          PACK (###.##) A$ FROM X
          UNPACK (###.##) X FROM A$
          PRINT X
          314.15
```

(Restriction: Do not attempt to UNPACK into a variable named I. To do so will result in an error message because the BASIC interpreter will interpret the "I" and the "F" as an "IF".)

Program Control Statements

END

Format: 10 END

This statement terminates execution of a program. There is no way to CONTinue execution after an END.

```
Example:  9999 END
```

EXIT

Format: 10 EXIT statement number

This statement functions in the same way as a GOTO statement. In the process it terminates the currently active FOR-NEXT loop. It must be used to branch out of a FOR-NEXT loop. Using a GOTO to branch out of FOR-NEXT loops will result in control stack errors. This is because the FOR-NEXT loop places information on the internal control stack and a GOTO will not clear this information. (In earlier versions of Maxi-BASIC, the EXIT cleared ALL active FOR-NEXT loops. This version clears only the currently active innermost loop.)

```
Example:  10 FOR I=1 TO 3
          20 FOR J=1 TO 10
          30 IF J=3 THEN EXIT 60
          40 PRINT I,J
          50 NEXT J
          60 NEXT I
          RUN
          1       1
          1       2
          2       1
          2       2
          3       1
          3       2
```

FOR - NEXT

Format: 10 FOR variable = expr1 TO expr2 [STEP expr 3]
50 NEXT [variable]

These statements are used to establish iterative loops. The FOR statement, during the first iteration has no effect except to assign the value of expr1 to the named variable. Control is then passed to the statement following the FOR. When the NEXT statement is encountered several things occur. First, the value of the optional expr3 (or 1 if no STEP is present) is added to the variable named in the FOR. Then a comparison is made between the named variable and expr2. If the named variable is greater, control is passed to the statement following the NEXT statement. If the variable is not greater, control is passed to the statement following the FOR.

```
Example:  10 FOR I=1 TO 10 STEP 3
          20 PRINT I
          30 NEXT
          RUN
             1
             4
             7
             10
```

FOR-NEXT loops may be multiply nested. Care must be taken so that FOR-NEXT loops are contained totally one within the other. No overlap of FOR-NEXT loops may occur. The variable name in the NEXT is optional. If it is present, a check will be made for proper nesting.

GOSUB

Format: 10 GOSUB statement number

The GOSUB statement is used to pass control to the statement number indicated and establish the linkage necessary to come back to the statement following the GOSUB. (Used with RETURN.)

```
Example:  10 GOSUB 1055
```

GOTO

Format: 10 GOTO statement number

The GOTO statement passes control to the indicated statement number.

```
Example:  10 GOTO 1905
```

IF . . . THEN . . . ELSE

Format: 10 IF condition THEN any statement
[ELSE any statement]

This statement allows conditional testing and selective statement execution based on this conditional testing. The condition, which can be a simple compare or a complex boolean condition using AND, OR and NOT, is evaluated and if it is true, the statement following the THEN is executed. If the condition is false and there is an ELSE clause, it will be executed. If no ELSE clause is present and the condition is false, the next numbered statement will be executed. The statements contained in the THEN or ELSE clause may be any valid statement including another IF. If the statement in the THEN or ELSE clause is a GOTO, the GOTO is optional (only a statement number is necessary).

```
Example:  A=0 : B=1
          IF A=0 OR B=0 THEN PRINT "YES" ELSE
            PRINT "NO"
          YES
          IF A AND B THEN PRINT "YES" ELSE
            PRINT "NO"
          NO
           (Note: This is a pure boolean operation.
           See Boolean Operators.)
          IF A>2 THEN PRINT "YES" ELSE IF B=1
            THEN PRINT "B IS ONE " ELSE PRINT
            "NO"
          B IS ONE
```

ON ERROR

Format: 10 ON ERROR (str variable1, str variable2)
any statement

The ON ERROR statement is used to give the user's program first access to any error detected by Business BASIC. Previous versions of Maxi-BASIC would terminate execution upon detection of any error. If Business BASIC detects any type of error a check will be made to see if an ON ERROR statement has been encountered. If so, the statement number at which the error occurred will be placed in str variable1, an error message will be placed in str variable2, and control will be passed to the statement contained in the ON ERROR. Control is always passed to the statement specified in the last ON ERROR statement executed. If no ON ERROR statement is executed and Business BASIC detects an error, it will terminate execution of the program.

```
Example: 5  DIM E$ (11)
         10  ON ERROR (L$,E$) GOTO 2000
         20  LET X=50/0
         30  PRINT X
         40  STOP
         2000  PRINT "A ";E$;" error occurred
               in line # ";L$
         2010  END
         RUN
         A DIVIDE ZERO error occurred in line # 20
```

ON . . . GOSUB

Format: 10 ON expr GOSUB statement number1,
statement number2, . . . statement number n

The ON . . . GOSUB statement is used to transfer control to one of several statements in a program dependent upon the value of expr. The linkage is also established to allow subsequent transfer back to the statement following the ON . . . GOSUB (using a RETURN). The value of expr must be an integer greater than zero. If the value of expr is one, control will be passed to the first statement number named. If the value of expr is two, control will be passed to the second statement number. Likewise, if the value of expr is n, control will be passed to the nth statement number. If the value of expr is greater than n, an error will be generated.

```
Example: 10  ON X GOSUB 30,40,50,60

         Control will be passed to:
             statement 30 if X is one
             statement 40 if X is two
             statement 50 if X is three
             statement 60 if X is four
```

ON . . . GOTO

Format: 10 ON expr GOTO statement number1,
statement number2, . . . statement number n

The ON . . . GOTO functions exactly the same as ON . . . GOSUB, except that ON . . . GOTO does not establish the linkage necessary for a RETURN. Otherwise the two statements are equivalent. The value of expr is checked and control is transferred to the appropriate statement number.

Example: ON X GOTO 30, 40, 50, 60

RETURN

Format: 10 RETURN [variable]

The RETURN statement is used to transfer control back to the statement following the last executed GOSUB or ON . . . GOSUB. It is also used to transfer control back to the program after execution of a user function call. The optional variable name is used only with the latter type, and indicates the value to be sent back to the program (see "User Defined Functions"). Control may be passed to the same subroutine (via a GOSUB or ON . . . GOSUB) from several places in a program. The RETURN statement will be able to correctly identify the point from which control was transferred, and return control to that point.

```
Example: 10  GOSUB 500
         20  PRINT "LINE 20"
         30  GOSUB 500
         40  PRINT "LINE 40"
         50  STOP
         500  PRINT "LINE 500"
         510  RETURN
         RUN
         LINE 500
         LINE 20
         LINE 500
         LINE 40
         STOP IN LINE 500
```

STOP

Format: 10 STOP

The STOP statement terminates execution of a program. The message "STOP IN LINE xxx" is then displayed (if the STOP is not the last line of the program) where xxx is the line number of the statement following the STOP. The program may be continued after execution of a STOP by typing CONT.

```
Example: 1090  STOP
```

TRACE START

Format: 10 TRACE START [LINE]

The TRACE START statement is used to instruct Business BASIC to list each statement on the terminal device before it is executed. This statement facilitates debugging of the user's programs. If the option LINE is specified, only line numbers are printed.

TRACE STOP

Format: 10 TRACE STOP

The TRACE STOP statement instructs Business BASIC to terminate listing of statements initiated by a TRACE START.

General Input/Output Statements

CURSOR

Format: 10 CURSOR expr1 [,expr2]

The CURSOR statement is used to position the cursor of the TV-64 controller at a specified location. If only expr1 is present it may have a value of zero through 1023. The TV-64 display is then treated as a single dimensioned array and the cursor placed at the indicated position. If expr1 and expr2 are present, expr1 may have a value of zero through fifteen and expr2 may have a value of zero through sixty-three. The TV-64 display is then treated as a two dimensional array and the cursor placed at the indicated position. The value of expr1 indicates the desired row and the value of expr2 indicates the column desired.

```
Example:  10  CURSOR 3,15
          20  CURSOR 521
```

ENTER

Format: 10 ENTER expr,str variable,variable

The ENTER statement is used for timed input of data from the keyboard. Expr indicates the number of tenths of seconds to await input. Str variable name indicates the string variable in which to place the incoming data. Numeric variable name indicates the variable in which to place the number of tenths of seconds actually used in completing the input. ENTER will transfer control to the statement following the ENTER after one of three events occurs. First, control is transferred if the input is not completed within the allotted time. In this case, the numeric variable will be set to zero. Second, control is transferred if a carriage return is entered. In this case the numeric variable will contain the time used. Finally, control will be transferred if the number of characters entered is equal to the dimensioned size of the str variable name. In this case the numeric variable will contain the time used. Note that in the final method, no carriage return is required. It is useful for controlling maximum input length.

```
Example:  10  ENTER 50,A$,N
```

IMAGE

Format: 10 IMAGE mask1 mask2 mask3 . . . maskn

The IMAGE statement provides the output format to be used for variables and constants in a PRINT USING statement. The first value in the PRINT USING uses mask1, the second value uses mask2 and so on with the nth value using maskn. The masks are made up of combinations of the following characters:

- `#` is a replacement character, i.e., in the resulting output, a character of a variable will replace the #.
- `.` is used to indicate placement of the decimal point in the resulting output.
- `$` is used to indicate that a dollar sign is to be placed to the immediate left of the field.
- `+` is used to indicate desired output of a sign, whether positive or negative, with a numeric variable.
- `-` is used to indicate output of a minus sign for negative numeric variables and suppression of plus signs.
- `\` is used to indicate output of a carriage return and line feed.
- `↑` is the replacement character for exponents in scientific notation.
- `;` is used to separate masks if you do not want space(s) printed between values. It has no effect on the resulting output.

String constants may also occur within the IMAGE statement. Spaces in the IMAGE are treated as literal spaces and need not be quoted.

```
Example:  10  X=15: A$="Barrels"
          20  PRINT USING 30;X,A$
          30  IMAGE "Qty on hand is "####.##
              #########
          RUN
          Qty on hand is   15.00 Barrels
```

INPUT and INPUT1

Format: 10 INPUT[1] [str constant,] variable1, variable2, . . . variable n

The INPUT statement is used to assign values, obtained from the keyboard, to the named variables. The variables may be string or numeric. If present the str constant will be printed on the terminal as a prompt. If no str constant is present, a "?" will be used. The INPUT1 statement is identical to the INPUT except that INPUT1 will suppress echoing of the carriage return/line feed at the end of the user's input.

```
Example:  10  INPUT A,X$
          20  INPUT1 B,Z$
```

During execution of an INPUT statement, the user may provide multiple values on the same line by separating them by commas. Note, however, that the value assigned to a string variable is not terminated by a comma, but a carriage return. If the user gives fewer values than required to satisfy the INPUT statement, the prompt "??" will appear.

```
Example:  10  DIM A$(15)
          20  INPUT X,A$,Y
          30  PRINT X,A$,Y
          RUN
          ?23,test,45  (typed by user)
          ??93  (typed by user)
          23    test,45    93
```

KEYIN

Format: 10 KEYIN str variable

The KEYIN statement is used for entry of single key data entries. The value of any key pressed will be placed in the string variable named. This statement differs from INPUT in that data from any key will be placed in the variable, and no echo to the current output device is provided. INPUT does not allow entry of certain keys (for example, carriage return).

OUT

Format: 10 OUT expr1, expr2

The OUT statement is used to output a specific value to a particular port. Expr1 indicates the port number and expr2 indicates the value. Expr1 and expr2 may have a value of zero through 255.

Example: 10 OUT 5,200

PRINT

Format: 10 PRINT [format string;] expr1, expr2, . . . exprn
or 10 # [format string;] expr1, expr2, . . . exprn

The PRINT statement is used to output values to the terminal. The # sign may be used in place of the word PRINT. The expressions may be string or numeric. If the expressions are separated by commas, the values will be printed in eight fields of eight characters. For expressions separated by a semicolon, a space precedes a numeric expression; no space precedes a string expression. If a value will not fit on the current output line, it will be placed on the next line. Output of the values is in the default format, unless formatting is specified. The default format is initially free format. A format string may appear anywhere in the PRINT statement and begins with a % character. A format string is made up of optional format characters followed by format specifications. The format characters are:

C	which places commas to the left of the decimal as needed.
$	which places a dollar sign to the left of the value.
Z	which suppresses trailing zeros.
?	which makes this format string the default format.

Valid format specifications are:

nFm	Floating point format. The value is printed in an n character field with m digits to the right of the decimal point.
nI	Integer format. The value is printed in an n character field. (An error will occur if a non-integer value is used.)
nEm	Scientific format. The value is printed in an n character field with m digits to the right of the decimal point in scientific notation (i.e., a mantissa and exponent).

(Note: In each format specification the n character field must include any commas and/or any dollar sign.)

Values will be rounded if necessary to fit the format specification.

```
Example: X=1234.5609
         Y=5678.9035
         PRINT X,Y
         1234.5609        5678.9035
         # X,Y
         1234.5609        5678.9035
         PRINT X;Y
         1234.5609  5678.9035
         PRINT %7F2;X
         1234.56
         PRINT %C8F2;X
         1,234.56
         PRINT %11E5;X
         1.23456E+03
         X=1234
         PRINT %C$6I;X
         $1,234
```

PRINT HEX

Format: 10 PRINT HEX str expr

The PRINT HEX statement is used to output the value of the string expression in hexadecimal format. Each character of the string expression is converted to its corresponding two-character hexadecimal code and sent to the terminal device.

```
Example: A$="10AZ"
         PRINT HEX A$
         3130415A
```

HEX may be intermixed with other data to be printed, for example when examining packed data fields.

```
Example: X=314.159
         PACK (###.##) B$ FROM X
         PRINT "PACKED DATA: ": HEX B$
         PACKED DATA: 314150
```

PRINT USING

Format: 10 PRINT USING statement number; expr1, expr2, . . . exprn
or 10 PRINT USING str variable; expr1, expr2, . . . exprn

The PRINT USING statement is used to provide formatted output of data. The statement provides a list of expressions, numeric or string, the value of which is to be PRINTed using the masks provided by the IMAGE statement referred to by the statement number. Optionally, the masks may be contained in the string variable referred to by str variable. A ";" may be used as the last character of a PRINT USING statement to suppress the carriage return at the end of the output line.

```
Example: 10 DIM B$(50)
         20 X=15 : A$="barrels"
         30 B$="###############
             ####.## #########"
         40 PRINT USING B$;
          "QTY ON HAND IS", X,A$
         RUN
         QTY ON HAND IS  15.00 barrels
```

File Statements

CLOSE

Format: 10 CLOSE (expr, variable)

The CLOSE statement is used to disassociate a logical file number from a physical file, thus, allowing the logical file number to be assigned to another physical file. Expr indicates the logical file number to be CLOSEd. If the file did not exist at OPEN time, an entry will be placed in the directory of the tape on which the physical file is located. After execution of the CLOSE, the variable named contains a status code. (See Appendix A.)

Example: CLOSE (4,E)

(Note: Files are NOT automatically CLOSEd upon termination of the program. This allows one program to pass an open file to another.)

GET

Format: 10 GET (expr1, variable, str variable, expr2 [,expr3])

The GET statement is used to retrieve a record from the specified logical file. Expr1 indicates the logical file number. Expr2 indicates the record number within the file to be retrieved. Note that the record number is not the hardware block id number used by PHIMON, but rather a logical record number. Record numbers are specified relative to the beginning of the named file. This frees the user from concern with hardware block id's. The first logical record number in a file is record number zero. The contents of the specified record are placed in the indicated str variable. The length of the string variable is set to the smallest of the following values:

the record size
the dimensioned size of the string
the value of optional expr3

A status code is placed in the named variable after execution of the GET. (See Appendix A.)

```
Example:  10  DIM A$(256),F$(8)
          20  F$="TEST.DA"+CHR$(0)
          30  OPEN (0,E,F$,3,1,1)
          40  IF E>1 THEN STOP
          50  A$="THIS IS A DATA RECORD"
          60  PUT (0,E,A$,0)
          70  IF E THEN STOP
          80  GET (0,E,A$,0)
          90  IF E THEN STOP
          100  PRINT A$
          110  GET (0,E,A$,0,7)
          120  IF E THEN STOP
          130  PRINT A$
          140  UNDIM A$
          150  DIM A$(11)
          160  GET (0,E,A$,0)
          170  IF E THEN STOP
          180  PRINT A$
          190  CLOSE (0,E)
          RUN
          THIS IS A DATA RECORD
          THIS IS
          THIS IS A D
```

OPEN

Format: 10 OPEN (expr1, variable, str variable, expr2, expr3 [,expr4])

The OPEN statement is used to associate a physical file with a logical file number. Expr1 indicates the logical file number to be used. The valid logical file numbers are zero through nine. Up to ten files may be in use at one time; however, only one output or input/output file may be in use on a given physical tape drive at one time. The string variable contains the name of the file including the PHIMON extension (see "File Names and Extensions" in part 2 and "File Name Extensions" in Appendix A of the PHIMON manual) plus an ASCII null or CHR$(0). Expr3 indicates the physical tape drive in which the file will be used. Valid tape drive numbers are zero through three. The type of file is indicated by expr2. The types of files are:

Type: 1 Output file. Records may only be PUT to this file. It must not already exist on the tape mounted in the indicated drive.

Type: 2 Input file. Only GET's may be issued to this file. It must already exist on the tape mounted in the indicated drive.

Type: 3 Input/output file. GET's and PUT's may be issued to this file. No assumption is made as to whether or not the file exists on the tape mounted in the indicated drive.

The optional expr4 indicates the number of records in the file. For input-type files it is ignored. For output-type files a check is made to determine if sufficient space exists to establish the file. If not, a particular status code (see Appendix A) is placed in the named variable. For input/output files that already exist at the time the OPEN is executed, no action is taken. For input/output files that do not already exist, a check is made to verify that the tape has sufficient space. If no file size is specified for output files or input/output files that do not already exist, a default value equal to the maximum available free space on the tape is assumed. (Note that for files OPENed in this manner the file size will be adjusted when the file is CLOSEd to reflect the actual number of records used.) A status code is placed in the named variable after execution of an OPEN statement. If the requested file name is not found in the directory, a value of 1 is placed in the named variable. For input-type files this is an error. For other types of files it serves as a warning to let the user know that the file has not previously been created. (See Appendix A.)

```
Example:  DIM F$(8)
          LET F$="TEST.DA"+CHR$(0)
          OPEN (0,E,F$,1,2)
```

PURGE

Format: 10 PURGE (expr,variable)

The PURGE is used to CLOSE a file and delete the file's name from the directory on its associated tape. The logical file number, indicated by expr, is disassociated from the physical file, just as in a CLOSE. A PHIMON delete is then issued to remove the file name entry from the directory. A status code is then placed in the named variable.

Example: PURGE (0,E)

PUT

Format: 10 PUT (expr1, variable, str variable, expr2 [,expr3])

The PUT statement is used to place a record on the specified file. Expr1 indicates the logical file number. The contents of the string variable are placed into the record number indicated by expr2. Optionally, expr3 indicates the number of characters of the string variable that are to be placed in the file. Initially, records must be put into a file sequentially. As an example, record number five must already exist in the file before a PUT to record six can take place. (Note: The authors have been using random GET's and PUT's to a file after it has initially had blank records written into it. Further, we have also been using an update-in-place scheme, i.e., multiple PUT's to the same record, with no detrimental results. The user is cautioned that write errors might cause subsequent records to be lost. It is therefore advised that suitable backup procedures be instituted for files used in this way.)

```
Example: 10  DIM A$(256),F$(8)
         20  F$="TEST.DA"+CHR$(0)
         30  OPEN (0,E,F$,3,1,2)
         40  IF E>1 THEN STOP
         50  A$="THIS IS A DATA RECORD"
         60  PUT (0,E,A$,0)
         70  IF E THEN STOP
         80  PUT (0,E,A$,1,7)
         90  IF E THEN STOP
         100  GET (0,E,A$,0)
         110  IF E THEN STOP
         120  PRINT A$
         130  GET (0,E,A$,1)
         140  IF E THEN STOP
         150  PRINT A$
         160  CLOSE (0, E)
         RUN
         THIS IS A DATA RECORD
         THIS IS
```

REWIND

Format: 10 REWIND (expr, variable)

The REWIND statement is used to physically position the tape to the beginning of the logical file specified by expr. A status code is then placed in the named variable.

Example: REWIND (0,E)

Functions

The purpose of functions in Business BASIC is to allow the programmer to use implicit subroutine access within most statements. In other words, most references to a variable may be replaced with a reference to a subroutine and the value returned by that subroutine is used in the execution of the statement. Two types of functions are available in Business BASIC: system functions and user defined functions.

System Functions

The system functions are used by replacing the reference to a variable in most statements with the name of the desired function followed by a parameter enclosed in parentheses. Business BASIC will execute the desired function and the result returned by the function will be used in the statement.

```
Example: X=24.56
         PRINT X,INT(X)
         24.56       24
         Y=INT(X)+5
         PRINT Y
         29
```

The system functions available in Business BASIC are:

ABS

Format: ABS(expr)

The ABS function is used to obtain the absolute value of the expr.

```
Example: X=-354
         PRINT ABS(X)
         354
```

ASC

Format: ASC(str variable)

The ASC function is used to obtain the numeric value of the binary bit pattern contained in the first character of the indicated string.

```
Example: A$="3"
         PRINT ASC(A$)
         51
         A$="m"
         PRINT ASC(A$)
         109
```

CALL

Format: CALL(expr1 [,expr2])

The CALL function is used to pass control to a machine language subroutine. Expr1 indicates the address of the machine language subroutine to be CALLed. The value of optional expr2 is converted to an integer and placed in the DE register pair before the machine language routine is executed. The machine language routine should place a value, to be returned to the main program, in the HL register pair. This value is then returned to the statement which referenced the CALL function.

```
Example: PRINT CALL(2051,300)
         600
```
(This would place the hex value 012C in the DE pair and then execute a fictitious machine language routine at hex address 0803. The fictitious routine shifted the number in the DE left one bit and placed the result, hex 0258, in the HL pair. This value was then returned to the PRINT statement.)

CHR$

Format: CHR$(expr)

The CHR$ function is functionally the opposite of the ASC function. It obtains the string character represented by the binary bit pattern equal in value to the expr.

```
Example: PRINT CHR$(51)
         3
         PRINT CHR$(109)
         m
```

COS

Format: COS(expr)

The COS function is used to obtain the cosine of the expr. Expr must be expressed in radians.

```
Example: PRINT COS(.234)
         .9727467
```

EXAM

Format: EXAM(expr)

The EXAM function is used to obtain the value of the contents of a particular memory location. Expr indicates the address of the location desired.

```
Example: PRINT EXAM(2)
         224
```

EXP

Format: EXP(expr)

The EXP function is used to obtain the value of e raised to a specified power. Expr indicates the desired power to which e is to be raised.

```
Example: PRINT EXP(3)
         20.085535
```

FREE

Format: FREE(0)

The FREE function is used to obtain the number of unused bytes remaining in memory. A parameter of zero is always used.

```
Example: PRINT FREE(0)
         5825
```

HEX

Format: HEX str expr

There is one special data conversion function, HEX. HEX will take the specified string expression and assume that the data contained therein is pairs of hexadecimal digits. These digits will be converted into true hexadecimal and placed in the variable named. If the variable is numeric, two hex pairs are converted. If the variable is a string, any number of hex pairs may be converted.

```
Example: A$=HEX "D4C5D3D4C9CEC7"
         PRINT A$
         TESTING
         X=HEX"D875"
         PRINT X
         55413
```

INP

Format: INP(expr)

The INP function is used to obtain the data available from the specified input port. Expr indicates a port number to which a hardware input instruction is to be issued.

```
Example: PRINT INP(0)
         13
```

INT

Format: INT(expr)

The INT function is used to obtain the integer value of the specified expr. Note that the value returned is the next smaller integer number.

```
Example: X=23.45
         PRINT INT(X)
         23
         X=-12.34567
         PRINT INT(X)
         -13
```

LEN

Format: LEN(str variable)

The LEN function is used to obtain the length of the specified string variable. The value returned is a count of the actual number of characters currently in the specified string.

```
Example: DIM A$(15)
         A$="TEST"
         PRINT LEN(A$)
         4
         A$=A$+"ING"
         PRINT LEN(A$)
         7
```

LOG

Format: LOG(expr)

The LOG is used to obtain the natural logarithm of the specified expr.

```
Example: PRINT LOG(23)
         3.1354941
```

RND

Format: RND(0)

The RND is used to obtain a psuedo random number with a value between zero and .99999999. The parameter zero is always used.

```
Example: PRINT RND(0)
         .01234912
```

SGN

Format: SGN(expr)

The SGN function is used to obtain an indication of the sign of the specified expr. If the value of expr is positive, a value of one is returned. If the value of expr is zero, a value of zero is returned. If the value of expr is negative, a value of minus one is returned.

```
Example: X=10 : Y=0 : Z=-15
         PRINT SGN(X),SGN(Y),SGN(Z)
         1      0      -1
```

SIN

Format: SIN(expr)

The SIN function is used to obtain the sine of the specified expr. Expr must be expressed in radians.

```
Example: PRINT SIN(23)
         -.846220Ø7
```

SQRT

Format: SQRT(expr)
or SQR (expr)

The SQRT function is used to obtain the positive square root of the specified expr.

```
Example: X=81
         PRINT SQRT(X)
         9
```

STR$

Format: STR$(expr)

The STR$ is used to obtain a string containing the character representation of expr.

```
Example: X=23.5Ø
         A$=STR$(X)
         A$=A$+" each"
         PRINT A$
         23.5 each
```

TAB

Format: TAB(expr)

The TAB function is used in print statements to specify that output is to begin at a particular location specified by expr.

```
Example: PRINT TAB(5); "ABC"
              ABC
```

VAL

Format: VAL(str expr)

The VAL format is functionally the opposite of the STR$ function. It is used to obtain the numeric value of the characters of the string expr.

```
Example: A$="23.7865"
         X=VAL(A$)+2Ø
         PRINT X
         43.7865
```

User Defined Functions

There are two types of user defined functions: single statement and multiple statement functions. Usage of both types is the same as the usage of system functions. Definition of functions is accomplished by use of the DEF statement.

DEF

```
Format: 1Ø  DEF FNvariable (variable 1, variable 2,
            . . . variable n)
or      1Ø  DEF FNvariable (variable1, variable2,
            . . . variable n)= expr
```

The named variable may be a numeric or string variable. The first format is for multiple statement functions, the second is for single statement functions.The execution of a single statement function call evaluates the expr on the right of the equal sign and returns the value of that expr. The execution of a multiple statement function call executes the statements contained in the function definition and returns the value specified in the RETURN statement. The variables used as parameters are "local" to the function definition. That is, any change in the value of these variables within the function definition is not reflected within the main program (see example below paying particular attention to the variable Y).

```
Example: 1Ø  Y=45
         2Ø  PRINT FNA(Y,5),FNB(Y)
         3Ø  PRINT Y
         4Ø  STOP
         5Ø  DEF FNA(Y,X)
         6Ø  Y=Y+3
         7Ø  X=Y*5
         8Ø  RETURN X
         9Ø  FNEND
         1ØØ  DEF FNB(Z)=Z*2Ø
         RUN
         24Ø       9ØØ
         45
         STOP IN LINE 5Ø
```

FNEND

Format: 1Ø FNEND

The FNEND statement is used to indicate the end of a multiple statement function definition.

Advanced PRINT Usage

Control Characters

Business BASIC has incorporated certain control characters to provide special output functions. Any time these control characters are encountered in a PRINT statement the associated function will be performed. These characters are:

CTRL-H - ASCII Backspace - Move screen cursor left
CTRL-I - ASCII Horizontal Tab - Move screen cursor right
CTRL-K - ASCII Vertical Tab - Home screen cursor
CTRL-L - ASCII Form Feed - Clear screen
CTRL-N - ASCII Shift Out - Carriage return and line feed
CTRL-Q - ASCII Device Control 1 - Move screen cursor down
CTRL-R - ASCII Device control 2 - Move screen cursor up

The functions of these characters are suppressed during a LIST. A CTRL-M will generate a carriage return and line feed during execution and LIST.

Printer Functions

The printer driver provided with Business BASIC offers increased utility of the Digital Group printer. Several options are available with this driver. The options are selected by printing a CTRL-G followed by a one-byte option code.

Valid option codes are:

~	7E Double width off
}	7D Double width on
|	7C Proportional spacing off
{	7B Proportional spacing on
z	7A Linefeed off
y	79 Linefeed on

All other characters are taken to be character size parameters. DO NOT use hex 13, 14, 17, or 18 as character sizes!! When listing a program, any control character is printed as its uppercase equivalent followed by a block.

```
Example:  10 PRINT "NORMAL SIZE PRINT"
          20 PRINT "CTRL-G} DOUBLE WIDTH
             PRINT"
```

```
NORMAL SIZE PRINT
DOUBLE WIDTH PRINT
```

Output Device Selection

Business BASIC has the capability of selecting the device or devices to which output from PRINT statements is to be sent. The devices currently implemented are the TV-64 display and the printer. To cause output to be sent to the TV-64 display, an OPEN (CRT, variable) statement is executed. To cause output to be sent to the printer an OPEN (PRINTER, variable) statement is executed. To suspend output to the TV-64 display a CLOSE (CRT, variable) is executed. Likewise, to suspend output to the printer a CLOSE (PRINTER, variable) is executed. When a program is executed, output is initially assumed to be sent to the TV-64 display. The variable specified in the OPEN or CLOSE for CRT or PRINTER currently has no function; it is implemented for future expansion.

Appendix A

File I/O Error Codes

The following are the status codes returned by the file statements:

00	No errors.
01	File not found. File does not exist on this tape.
02	Not enough free blocks exist on this tape.
03	Duplicate file exists on this tape.
04	File has not been opened.
05	End of file. Or logical record id is greater than file size.
07	File number already OPENed.
08	Invalid file number.
09	System error.
10	IO mismatch. (GET on output-type file or PUT on input-type file.)
11	Invalid file type.
12	Too many output files on one drive.
14	An output-type file was closed, but no records were written to it. The file is closed, but no directory entry is created.

The following codes indicate PHIMON errors (see Appendix C in the PHIMON manual):

257	Equivalent to error 1 - CRC error
258	Equivalent to error 2 - Block not found
259	Equivalent to error 3 - Tape end or jam

Appendix B

Program Installation

The Business BASIC program is distributed on one cassette tape, organized as follows:

Side 1 - Contains the Business BASIC program as a PHIMON file named "BASIC. GO".

Side 2 - Contains the Business BASIC program in audio form (1100 baud - Suding format). This form can be read using the PHIMON "READ" command, and is supplied as an alternate to Side 1. An audio cassette recorder is required to read this side of the tape.

An operational PHIMON system is required to prepare Business BASIC for use. The following procedure is used to install the program, and assumes familiarity with use of the PHIMON system.

1. Start up the PHIMON system, using the PHIMON ROM ("PZB").
2. Apply the recommended patches to PHIMON described in Appendix D.
3. Using the PHIMON "PIP" program, copy the file on Side 1 of the distribution tape to the PHIMON system tape or to another tape used for program storage.
   Note: It is recommended that the distribution tape **not** be used for anything other than a master copy of the program. Working copies should be made as required, using this procedure.
4. Execute Business BASIC and verify the copy performed in Step 3 by entering the command "RUN#n BASIC", where n specifies the tape drive number containing the program.

If difficulties are encountered in reading Side 1 of the distribution tape, the following alternate procedure can be used. (Steps 1 & 2 are the same.)

3. Prepare the audio cassette recorder with Side 2 of the distribution tape.
4. Enter the command "READ 1000-105377" and read in the audio file.
5. Save the program on the PHIMON system tape or on another tape used for program storage by entering the command "SAVE#n BASIC 1-105*5000", where n specifies the appropriate tape drive number.
6. Execute Business BASIC and verify the above steps by entering the command "RUN#n BASIC", where n specifies the tape drive number containing the program.

Note: When RUNning Business BASIC, automatic execution of a source program may be effected by specifying a start address of 1270 (rather than 5000) in the "SAVE" command. The program to be executed must be saved along with BASIC by using a PHIMON "SAVE" command, not a BASIC "SAVE" statement.

If difficulties were encountered in installing the PHIMON file (Side 1), please let us know by responding on the Reader's Comments Form at the back of the manual.

Appendix C

Conversion of MAXI-BASIC Source Programs

The following procedure can be used to convert MAXI-BASIC source programs to Business BASIC. It is assumed that MAXI-BASIC has been previously installed and is operating under PHIMON. (Refer to "Getting Started with PHIMON" in the PHIMON documentation for more information.)

1. LOAD MAXI-BASIC, then the source program to be converted.
2. START execution of MAXI-BASIC (required to set internal pointers).
3. WRITE to an audio cassette, starting at Ø63253 through the end of the source program. The end point can be determined by examining Ø62140 for location and Ø62141 for page.
4. LOAD Business BASIC.
5. READ from the audio cassette generated in step 3, starting at location 1Ø5131.
6. START execution of Business BASIC.
7. Using the SAVE statement in the Business BASIC language (not PHIMON), save the source program.

Appendix D

PHIMON Considerations

It is suggested that the following patches be applied to the PHIMON system being used with Business BASIC. Refer to the PHIMON documentation for detailed information on applying patches using the DTO command.

Location	Data	Explanation
345354	270 346	Compensates for a tape overrun problem which
346270	076037 315150345 311	may cause erroneous 258 error codes.
340034	315361343	Fixes a potential
343361	076200 323003 257 323003 041104 341 311	printer problem which can develop if RESET is pressed while a line is being printed.
340364	373004	Fixes a problem with
340371	372000	use of the rubout key.

READER'S COMMENTS

The Digital Group would like to improve the quality and usefulness of this publication. To do this effectively, we need user feedback — your critical evaluation of this manual.

Please comment on this manual's completeness, accuracy, organization, usability, and readability.

Did you find errors in this manual? If so, specify by page.

How can this manual be improved?

Other comments.

NAME: _____ DATE: _____

STREET: _____

CITY: _____ STATE: _____ ZIP: _____

TELEPHONE NUMBER: _____

Please send this form to:

SOFTWARE DEVELOPMENT/BUS-1
DIGITAL GROUP INC.
P. O. BOX 6528
DENVER, COLORADO 80206