STATEMENT	SYNTAX/FUNCTION	EXAMPLE
	should then type in CONT to continue executing your program where it left off, or type a direct GOTO statement to resume execution of the program at a different line. You could also use assignment statements to set some of your variables to different values. Remember, if you interrupt a program with the F1 key and expect to continue it later, you must not get any errors or type in any new program lines. If you do, you won't be able to continue and will get a "CN" (continue not) error. It is impossible to continue a direct command. CONT always resumes execution at the next statement to be executed in your program when F1 was typed.	
FRE	FRE (expression) Gives the number of memory bytes currently unused by BASIC. A dummy operand—0 or 1—must be used.	270 PRINT FRE(0)
LIST	LIST [[start line] [-[end line]]] Lists current program optionally starting at specified line. List can be interrupted with the F1 key. (BASIC will finish listing the current line.)	
	Lists entire program	LIST
	Lists just line 100.	LIST 100
	Lists lines 100 to 1000.	LIST 100-1000
	Lists from current line to line 1000.	LIST -1000
	Lists from line 100 to end of program.	LIST 100-
LOAD	LOAD Loads a BASIC program from the cassette tape. When done, the LOAD will display the cursor. See Appendix G for more information.	LOAD

STATEMENT	SYNTAX/FUNCTION	EXAMPLE
NEW	NEW Deletes current program and all variables.	NEW
PEEK	PEEK (address) The PEEK function returns the contents of memory address I in decimal. The value returned will be =>0 and <=255. If I is >65535 or <0, an FC error will occur. An attempt to read a non-existent memory address will return an unknown value.	356 PRINT PEEK (I)
POKE	POKE location, byte The POKE statement stores the byte specified by its second argument (J) into the location given by its first argument (I). The byte to be stored must be =>0 and <=255, or an FC error will occur. The address (I) must be =>0 and <=65535, or an FC error result. **Caution:** *Careless use of the POKE statement may cause your program, BASIC, or the Monitor functions to operate incorrectly, to hang up, and/or cause loss of your program. Note that Pages 0 and 1 in memory are reserved for use by BASIC and should not be used for user program variable storage.* A POKE to a non-existent memory location is harmless. One of the main uses of POKE is to pass arguments to machine language subroutines. (See Appendix F.) You could also use PEEK and POKE to write a memory diagnostic or an assembler in BASIC.	357 POKE I, J

STATEMENT	SYNTAX/FUNCTION	EXAMPLE
RUN	RUN line number Starts execution of the program currently in memory at the specified line number. RUN deletes all variables (does a CLEAR) and restores DATA. If you have stopped your program and wish to continue execution at some point in the program, use a direct GOTO statement to start execution of your program at the desired line, or CONT to continue after a break.	RUN 200
	Start program execution at the lowest numbered statement.	RUN
SAVE	SAVE Saves the current program in the AIM 65 memory on cassette tape. The program in memory is left unchanged. More than one program may be stored on cassette using this command. See Appendix G for more information.	SAVE

In the following description of statements, an argument of B, C, V or W denotes a numeric variable, X denotes a numeric expression, X$ denotes a string expression and an I or J denotes an expression that is truncated to an integer before the statement is executed. Truncation means that any fractional part of the number is lost, e.g., 3.9 becomes 3, 4.01 becomes 4.

An expression is a series of variables, operators, function calls and constants which after the operations and function calls are performed using the precedence rules, evaluates to a numeric or string value.

A constant is either a number (3.14) or a string literal ("F00").

STATEMENT	SYNTAX/FUNCTION	EXAMPLE
DEF	DEF FNx [(argument list)] = expression The user can define functions like the built-in functions (SQR, SGN, ABS, etc.) through the use of the DEF statement. The name of the function is "FN" followed by any legal variable name, for example: FNX, FNJ7, FNK0, FNR2. User defined functions are restricted to one line. A function may be defined to be any expression, but may only have one argument. In the example, B and C are variables that are used in the program. Executing the DEF statement defines the function. User defined functions can be redefined by executing another DEF statement for the same function. "V" is called the dummy variable.	100 DEF FNA(V)=V/B+C
	Execution of this statement following the above would cause Z to be set to 3/B+C, but the value of V would be unchanged.	100 Z=FNA(3)
DIM	DIM variable (size 1, [size 2 . . .]) Allocates space for matrices. All matrix elements are set to zero by the DIM statement.	113 DIM A(3), B(10)
	Matrices can have from one to 255 dimensions.	114 DIM R3(5,5), D$(2,2,2)

STATEMENT	SYNTAX/FUNCTION	EXAMPLE
	Matrices can be dimensioned dynamically during program execution. If a matrix is not explicitly dimensioned with a DIM statement, it is assumed to be a single dimensioned matrix of whose single subscript may range 0 to 10 (eleven elements).	115 DIM Q1(N), Z(2*I)
	If this statement was encountered before a DIM statement for A was found in the program, it would be as if a DIM A(10) had been executed previous to the execution of line 117. All subscripts start at zero (0), which means that DIM X(100) really allocates 101 matrix elements.	117 A(8)=4
END	END Terminates program execution without printing a BREAK message. (See STOP.) CONT after an END statement causes execution to resume at the statement after the END statement. END can be used anywhere in the program, and is optional.	999 END
FOR	FOR variable = expression to expression [STEP expression] (See NEXT statement) V is set equal to the value of the expression following the equal sign, in this case 1. This value is called the initial value. Then the statements between FOR and NEXT are executed. The final value is the value of the expression following the TO. The step is the value of the expression following STEP. When the NEXT statement is encountered, the step is added to the variable.	300 FOR V=1 TO 9.3 STEP .6

STATEMENT	SYNTAX/FUNCTION	EXAMPLE
	If no STEP was specified, it is assumed to be one. If the step is positive and the new value of the variable is <= the final value (9.3 in this example), or the step value is negative and the new value of the variable is => the final value, then the first statement following the FOR statement is executed. Otherwise, the statement following the NEXT statement is executed. All FOR loops execute the statements between the FOR and the NEXT at least once, even in cases like FOR V=1 TO 0.	310 FOR V=1 TO 9.3
	Note that expressions (formulas) may be used for the initial, final and step values in a FOR loop. The values of the expressions are computed only once, before the body of the FOR . . . NEXT loop is executed.	315 FOR V=10*N TO 3.4/Q STEP SQR(R)
	When the statement after the NEXT is executed, the loop variable is never equal to the final value, but is equal to whatever value caused the FOR . . . NEXT loop to terminate. The statements between the FOR and its corresponding NEXT in both examples above (310 and 320) would be executed nine times.	320 FOR V=9 TO 1 STEP -1
	Error: do not use nested FOR . . . NEXT loops with the same index variable. FOR loop nesting is limited only by the available memory. (See Appendix C.)	330 FOR W=1 TO 10: FOR W=1 TO 5:NEXT W:NEXT W
GOSUB	GOSUB line number Branches to the specified statement (910) until a RETURN is encountered; when a branch is then made to the statement after the GOSUB. GOSUB nesting is limited only by the available memory.	10 GOSUB 910

STATEMENT	SYNTAX/FUNCTION	EXAMPLE
GOTO	GOTO line number Branches to the statement specified.	50 GOTO 100
IF . . . GOTO	IF expression GOTO line number . . . Equivalent to IF . . . THEN, except that IF . . . GOTO must be followed by a line number, while IF . . . THEN can be followed by either a line number or another statement.	32 IF X<=Y+23.4 GOTO 92
IF . . . THEN	IF expression THEN line number . . . Branches to specified statement if the relation is True.	IF X<10 THEN 5
	t Executes all of the statements on the remainder of the THEN if the relation is True.	20 IF X<0 THEN PRINT "X LESS THAN 0"
	WARNING: *The "Z=A" will never be executed because if the relation is true, BASIC will branch to line 50. If the relation is false BASIC will proceed to to the line following line 25.*	*25 IF X=5 THEN 50:Z=A*
	In this example, if X is less than 0, the PRINT statement will be executed and then the GOTO statement will branch to line 350. If the X was 0 or positive, BASIC will proceed to execute the lines after line 26.	26 IF X<0 THEN PRINT "ERROR, X NEGATIVE": GOTO 350
LET	[LET] variable = expression Assigns a value to a variable.	300 LET W=X
	"LET" is optional.	310 V=5.1
NEXT	NEXT [variable] [, variable] . . . Marks the end of a FOR loop.	340 NEXT V

STATEMENT	SYNTAX/FUNCTION	EXAMPLE
	If no variable is given, matches the most recent FOR loop.	345 NEXT
	A single NEXT may be used to match multiple FOR statements. Equivalent to NEXT V:NEXT W.	350 NEXT V, W
ON . . . GOSUB	ON expression GOSUB line [, line] . . . Identical to "ON . . . GOTO," except that a subroutine call (GOSUB) is executed instead of a GOTO. RETURN from the GOSUB branches to the statement after the ON . . . GOSUB.	110 ON I GOSUB 50, 60
ON . . . GOTO	ON expression GOTO line [, line] . . . Branches to the line indicated by the I'th number after the GOTO. That is: IF I=1, THEN GOTO LINE 10 IF I=2, THEN GOTO LINE 20 IF I=3, THEN GOTO LINE 30 IF I=4, THEN GOTO LINE 40. If I=0, or I attempts to select a nonexistent line (>=5 in this case), the statement after the ON statement is executed. However, if I is > 255 or <0, an FC error message will result. As many line numbers as will fit on a line can follow an ON . . . GOTO.	100 ON I GOTO 10, 20, 30, 40
	This statement will branch to line 40 if the expression X is less than zero, to line 50 if it equals zero, and to line 60 if it is greater than zero.	105 ON SGN(X)+2 GOTO 40, 50, 60

STATEMENT	SYNTAX/FUNCTION	EXAMPLE
REM	REM any text Allows the programmer to put comments in his program. REM statements are not executed, but can be branched to. A REM statement is terminated by end of line, but not by a ":".	500 REM NOW SET V=0
	In this case the V=0 will never be executed by BASIC.	505 REM SET V=0: V=0
	In this case V=0 will be executed.	505 V=0: REM SET V=0
RESTORE	RESTORE Allows the re-reading of DATA statements. After a RESTORE, the next piece of data read will be the first piece listed in the first DATA statement of the program. The second piece of data read will be the second piece listed in the first DATA statement, and so on as in a normal READ operation.	510 RESTORE
RETURN	RETURN Causes a subroutine to return to the statement after the most recently executed GOSUB.	50 RETURN
STOP	STOP Causes a program to stop execution and to enter command mode.	900 STOP
	Prints BREAK IN LINE 900. (As per this example.) CONT after a STOP branches to the statement following the STOP.	
USR	USR (argument) Calls the user's machine language subroutine with the argument. See PEEK and POKE in Subject 303, and Appendix F.	200 V=USR(W)

SYMBOL	SYNTAX/FUNCTION	EXAMPLE
WAIT	WAIT (address, mask [, select]) This statement reads the contents of the addressed location, does an Exclusive–OR with the select value, and then ANDs the result with the mask. This sequence is repeated until a non-zero result is obtained, at which time execution continues at the statement that follows WAIT. If the WAIT statement has no select argument, the select value is assumed to be zero. If you are waiting for a bit to become zero, there should be a "one" in the corresponding bit position of the select value. The select value (K) and the mask value (J) can range from 0 to 255. The address (I) can range from 0 to 65535.	805 WAIT I, J, K 806 WAIT I, J

STATEMENT	SYNTAX/FUNCTION	EXAMPLE
DATA	DATA item [, item . . .] Specifies data, read from left to right. Information appears in data statements in the same order as it will be read in the program.	10 DATA 1, 3, -1E3, .04
	Strings may be read from DATA statements. If you want the string to contain leading spaces (blanks), colons (:) or commas (,), you must enclose the string in double quotes. It is illegal to have a double quote within string data or a string literal. (""BASIC"" is illegal.)	20 DATA "F00", Z1
INPUT	INPUT [!] ["prompt string literal";] variable [, variable] . . . Requests data from the keyboard (to be typed in). Each value must be separated from the preceeding value by a comma (,). The last value typed should be followed by a carriage return. A "?" is displayed as a prompt character. Only constants may be typed in as a response to an INPUT statement, such as 4.5E-3 or "CAT." If more data was requested in an INPUT statement than was typed in, a "??" is printed and the rest of the data should be typed in. If more data was typed in than was requested, the warning "EXTRA IGNORED" will be displayed. Strings must be input in the same format as they are specified in DATA statements.	3 INPUT V, W, W2
	Optionally displays a prompt string ("VALUE") before requesting data from the keyboard. If RETURN is typed to an input statement, BASIC returns to com-command mode. Typing CONT after an INPUT command has been interrupted will cause execution to resume at the INPUT statement.	5 INPUT "VALUE"; V

STATEMENT	SYNTAX/FUNCTION	EXAMPLE
	If the optional character ! is included following INPUT, then the prompts from the INPUT statement and the user's entries will be printed (even if the printer is turned off) and displayed.	15 INPUT! "VALUE"; V
POS	POS (expression) Gives the current position of the cursor on the display. The leftmost character position on the display is position zero. A dummy operand—0 or 1—must be used.	260 PRINT POS(I)
PRINT	PRINT [!] expression [, expression] Prints the value of expressions on the display/printer. If the list of values to be printed out does not end with a comma (,) or a semicolon (;), then a carriage return/line feed is executed after all the values have been printed. Strings enclosed in quotes (") may also be printed. If a semicolon separates two expressions in the list, their values are printed next to each other. If a comma appears after an expression in the list, and the print head is at print position 11 or more, then a carriage return/line feed is executed. If the print head is before print position 11, then spaces are printed until the carriage is at the beginning of the next 10 column field. If there is a blank string enclosed in quotes, as in line 370 of the examples, then a carriage return/line feed is executed.	360 PRINT X, Y; Z 370 PRINT " " 380 PRINT X, Y; 390 PRINT "VALUE IS"; A 400 PRINT A2, B,
	"VALUE IS" will be displayed and printed.	410 PRINT ! "VALUE IS"; A
	String expressions may be printed.	420 PRINT MID$(A$, 2);

STATEMENT	SYNTAX/FUNCTION	EXAMPLE
READ	READ variable [, variable] Read data into specified variables from a DATA statement. The first piece of data read will be the first piece of data listed in the first DATA statement of the program. The second piece of data read will be the second piece listed in the first DATA statement, and so on. When all of the data have been read from the first DATA statement, the next piece of data to be read will be the first piece listed in the second DATA statement of the program. Attempting to read more data than there is in all the DATA statements in a program will cause an OD (out of data) error.	490 READ V, W
SPC	SPC (expression) Prints I space (or blank) characters on the terminal. May be used only in a PRINT statement. I must be => 0 and <=255 or an FC error will result.	250 PRINT SPC(I)
TAB	TAB (expression) Spaces to the specified print position (column) on the printer. May be used only in PRINT statements. Zero is the leftmost column on the termainl, 19 the rightmost. If the carriage is beyond pos position I, then no printing is done. I must be => 0 and <=255. If I is greater than 19, the printer will skip the required number of lines to arrive at the specified position.	240 PRINT TAB(I)

STATEMENT	SYNTAX/FUNCTION	EXAMPLE
ASC	ASC (string expression) Returns the ASCII numeric value of the first character of the string expression X$. See Appendix E for an ASCII/number conversion table. An FC error will occur if X$ is the null string.	300 PRINT ASC(X$)
CHR$	CHR$ (expression) Returns one character, the ASCII equivalent of the argument (I) which must be a number between 0 and 255. See Appendix E.	275 PRINT CHR$(I)
GET	GET string variable Inputs a single character from the keyboard. If data is at the keyboard, it is put in the variable specified in the GET statement. If no data is available, the BASIC program will continue execution. GET can only be used as an indirect command.	10 GET A$
LEFT$	LEFT$ (string expression, length) Gives the leftmost I characters of the string expression X$. If I<=0 or >255 an FC error occurs.	310 PRINT LEFT$(X$, I)
LEN	LEN (string expression) Gives the length of the string expression X$ in characters (bytes). Non-printing characters and blanks are counted as part of the length.	220 PRINT LEN(X$)

STATEMENT	SYNTAX/FUNCTION	EXAMPLE
MID$	MID$ (string expression, start [, length]) MID$ called with two arguments returns characters from the string expression X$ starting at character position I. If I>LEN(I$), then MID$ returns a null (zero length) string. If I<=0 or >255, an FC error occurs.	330 PRINT MID$(X$, I)
	MID$ called with three arguments returns a string expression composed of the characters of the string expression X$ starting at the Ith character for J characters. If I>LEN(X$), MID$ returns a null string. If I or J<=0 or >255, an FC error occurs. If J specifies more characters than are left in the string, all characters from the Ith on are returned.	340 PRINT MID$(X$, I, J)
RIGHT$	RIGHT$ (string expression, length) Gives the rightmost I characters of the string expression X$. When I<=0 or >255 an FC error will occur. If I>=LEN(X$) then RIGHT$ returns all of X$.	320 PRINT RIGHT$ (X$, I)
STR$	STR$ (expression) Gives a string which is the character representation of the numeric expression X. For instance, STR$(3.1)="3.1."	290 PRINT STR$(X)
VAL	VAL (string expression) Returns the string expression X$ converted to a number. For instance, VAL("3.1")=3.1. If the first non-space character of the string is not a plus (+) or minus (-) sign; a digit or a decimal point (.) then zero will be returned.	280 PRINT VAL(X$)

STATEMENT	SYNTAX/FUNCTION	EXAMPLE
ABS	ABS (expression) Gives the absolute value of the expression X. ABS returns X if X>=0, -X otherwise.	120 PRINT ABS(X)
ATN	ATN (expression) Gives the arctangent of the expression X. The result is returned in radians and ranges from -PI/2 to PI/2 (PI/2=1.5708). **If you want to use this function, you must provide the code in memory. See Appendix H for implementation details.**	210 PRINT ATN(X)
COS	COS (expression) Gives the cosine of the expression X. X is interpreted as being in radians.	200 PRINT COS(X)
EXP	EXP (expression) Gives the constant "E" (2.71828) raised to the power X (E∧X). The maximum argument that can be passed to EXP without overflow occurring is 88.0296.	150 PRINT EXP(X)
INT	INT (expression) Returns the largest integer less than or equal to its expression X. For example: INT(.23)=0, INT(7)=7, INT(-.1)=-1, INT(-2)=-2, INT(1.1)=1. The following would round X to D decimal places: INT(X*10∧D+.5)/10∧D	140 PRINT INT(X)
LOG	LOG (expression) Gives the natural (Base E) logarithm of its expression X. To obtain the Base Y logarithm of X use the formula LOG(X)/LOG(Y). Example: The base 10 (common) log of 7 = LOG(7)/LOG(10).	160 PRINT LOG(X)

STATEMENT	SYNTAX/FUNCTION	EXAMPLE
RND	RND (parameter) Generates a random number between 0 and 1. The parameter X controls the generation of random numbers as follows: X<0 starts a new sequence of random numbers using X. Calling RND with the same X starts the same random number sequence. X=0 gives the last random number generated. Repeated calls to RND(0) will always return the same random number. X>0 generates a new random number between 0 and 1. Note that (B-A)*RND(1)+A will generate a random number between A and B.	170 PRINT RND(X)
SGN	SGN (expression) Gives 1 if X>0, 0 if X=0, and -1 if X<0.	230 PRINT SGN(X)
SIN	SIN (expression) Gives the sine of the expression X. X is interpreted as being in radians. **Note:** *COS(X)=SIN(X+3.14159/2) and that 1 Radian=180/PI degrees=57.2958 degrees; so that the sine of X degrees=SIN(X/57.2958).*	190 PRINT SIN(X)
SQR	SQR (expression) Gives the square root of the expression X. An FC error will occur if X is less than zero.	180 PRINT SQR(X)
TAN	TAN (expression) Gives the tangent of the expression X. X is interpreted as being in radians.	200 PRINT TAN(X)

DERIVED FUNCTIONS

The following functions, while not intrinsic to BASIC, can be calculated using the existing BASIC functions:

FUNCTION	FUNCTION EXPRESSED IN TERMS OF BASIC FUNCTIONS
SECANT	SEC(X) = 1/COS(X)
COSECANT	CSC(X) = 1/SIN(X)
COTANGENT	COT(X) = 1/TAN(X)
INVERSE SINE*	ARCSIN(X) = ATN(X/SQR(-X*X+1))
INVERSE COSINE*	ARCCOS(X) = -ATN(X/SQR(-X*X+1))+1.5708
INVERSE SECANT*	ARCSEC(X) = ATN(SQR(X*X-1))+(SGN(X)-1)*1.5708
INVERSE COSECANT*	ARCCSC(X) = ATN(1/SQR(X*X-1))+(SGN(X)-1)*1.5708
INVERSE COTANGENT*	ARCCOT(X) = -ATN(X)+1.5708
HYPERBOLIC SINE	SINH(X) = (EXP(X)-EXP(-X))/2
HYPERBOLIC COSINE	COSH(X) = (EXP(X)+EXP(-X))/2
HYPERBOLIC TANGENT	TANH(X) = -EXP(-X)/(EXP(X)+EXP(-X))*2+1
HYPERBOLIC SECANT	SECH(X) = 2/(EXP(X)+EXP(-X))
HYPERBOLIC COSECANT	CSCH(X) = 2/(EXP(X)-EXP(-X))
HYPERBOLIC COTANGENT	COTH(X) = EXP(-X)/(EXP(X)-EXP(-X))*2+1

*These functions require the user-defined ATN function. See Appendix H for details.

FUNCTION	FUNCTION EXPRESSED IN TERMS OF BASIC FUNCTIONS
INVERSE HYPERBOLIC SINE	ARGSINH(X) = LOG(X+SQR(X*X+1))
INVERSE HYPERBOLIC COSINE	ARGCOSH(X) = LOG(X+SQR(X*X-1))
INVERSE HYPERBOLIC TANGENT	ARGTANH(X) = LOG((1+X)/(1-X))/2
INVERSE HYPERBOLIC SECANT	ARGSECH(X) = LOG((XQR(-X*X+1)+1)/X
INVERSE HYPERBOLIC COSECANT	ARGCSCH(X) = LOG((SGN(X)*SQR(X*X+1)+1)/X)
INVERSE HYPERBOLIC COTANGENT	ARGCOTH(X) = LOG((X+1)/(X-1))/2

If an error occurs, BASIC outputs an error message, returns to command level and displays the cursor. Variable values and the program text remain intact, but the program can not be continued and all GOSUB and FOR context is lost.

When an error occurs in a direct statement, no line number is printed.

Format of error messages:

Direct Statement	?XX ERROR
Indirect Statement	?XX ERROR IN YYYYY

In both of the above examples, "XX" will be the error code. The "YYYYY" will be the line number where the error occured for the indirect statement.

The following are the possible error codes and their meanings:

ERROR CODE	MEANING
BS	Bad Subscript. An attempt was made to reference a matrix element which is outside the dimensions of the matrix. This error can occur if the wrong number of dimensions are used in a matrix reference; for instance, LET A(1,1,1)=Z when A has been dimensioned DIM A(2,2).
CN	Continue error. Attempt to continue a program when none exists, an error occured, or after a new line was typed into the program.
DD	Double Dimension. After a matrix was dimensioned, another DIM statement for the same matrix was encountered. This error often occurs if a matrix has been given the default dimension 10 because a statement like A(I)=3 is encountered and then later in the program a DIM A(100) is found.
FC	Function Call error. The parameter passed to a math or string function was out of range. FC errors can occur due to: 1. A negative matrix subscript (LET A(-1)=0) 2. An unreasonably large matrix subscript (>32767)

ERROR CODE	MEANING
	3. LOG-negative or zero argument
	4. SQR-negative argument
	5. A^B with A negative and B not an integer
	6. A call to USR before the address of the machine language subroutine has been patched in
	7. Calls to MID$, LEFT$, RIGHT$, WAIT, PEEK, POKE, TAB, SPC or ON. . .GOTO with an improper argument.
ID	Illegal Direct. You cannot use an INPUT, DEF or GET statement as a direct command.
LS	Long String. Attempt was made by use of the concantenation operator to create a string more than 255 characters long.
NF	NEXT without FOR. The variable in a NEXT statement corresponds to no previously executed FOR statement.
OD	Out of Data. A READ statement was executed but all of the DATA statements in the program have already been read. The program tried to read too much data or insufficient data was included in the program.
OM	Out of Memory. Program too large, too many variables, too many FOR loops, too many GOSUB's, too complicated an expression, or any combination of the above. (see Appendix B)
OV	Overflow. The result of a calculation was too large to be represented in BASIC's number format. If an underflow (too small result) occurs, zero is given as the result and execution continues without any error message being printed.
RG	RETURN without GOSUB. A RETURN statement was encountered without a previous GOSUB statement being executed.
SN	Syntax error. Missing parenthesis in an expression, illegal character in a line, incorrect punctuation, etc.

ERROR CODE	MEANING
ST	String Temporaries. A string expression was too complex. Break it into two or more shorter expressions.
TM	Type Mismatch. The left side of an assignment statement was a numeric variable and the right side was a string, or vice versa; or, a function which expected a string argument was given a numeric one or vice versa.
UF	Undefined Function. Reference was made to a user function which has never been defined.
US	Undefined Statement. An attempt was made to GOTO, GOSUB or THEN to a statement which does not exist.
/0	Division by Zero

In order to make your program smaller and save space, the following hints may be useful.

1. Use multiple statements per line. There is a five-byte of overhead associated with each line in the program. Two of these five bytes contain the line number of the line in binary. This means that no matter how many digits you have in your line number (minimum line number is 0, maximum is 63999), it takes the same number of bytes. Putting as many statements as possible on a line will cut down on the number of bytes used by your program.

2. Delete all unnecessary spaces from your program. For instance:

   ```
   10 PRINT X, Y, Z
   ```

   uses three more bytes than

   ```
   10 PRINTX,Y,Z
   ```

   Note: All spaces between the line number and the first non-blank character are ignored.

3. Delete all REM statements. Each REM statement uses at least one byte plus the number in the comment text. For instance, the statement 130 REM THIS IS A COMMENT uses 24 bytes of memory.

   In the statement 140 X=X+Y: REM UPDATE SUM, the REM uses 14 bytes of memory including the colon before the REM.

4. Use variables instead of constants. Suppose you use the constant 3.14159 ten times in your program. If you insert a statement

   ```
   10 P=3.14159
   ```

   in the program, and use P instead of 3.14159 each time it is needed, you will save 40 bytes. This will also result in a speed improvement.

5. A program need not end with an END, so an END statement at the end of a program may be deleted.

6. Reuse variables. If you have a variable T which is used to hold a temporary result in one part of the program and you need a temporary variable later in your program, use it again. Or, if you are asking the terminal user to give a YES or NO answer to two different questions at two different times during the execution of the program, use the same temporary variable A$ to store the reply.

7. Use GOSUB's to execute sections of program statements that perform identical actions.

8. Use the zero elements of matrices; for instance, A(0), B(0,X).

STORAGE ALLOCATION INFORMATION

Simple (non-matrix) numeric and strong variables like V use 7 bytes; 2 for the variable name, and 5 for the value. Simple non-matrix string variables also use 7 bytes; 2 for the variable name, 1 for the length, 2 for a pointer, and 2 are unused.

Matrix variables require 7 bytes to hold the header, plus additional bytes to hold each matrix element. Each element that is an integer variable requires 2 bytes. Elements that are string variables or floating point variables require 3 bytes or 5 bytes, respectively.

String variables also use one byte of string space for each character in the string. This is true whether the string variable is a simple string variable like A$, or an element of a string matrix such as Q1$(5,2).

When a new function is defined by a DEF statement, 7 bytes are used to store the definition.

Reserved words such as FOR, GOTO or NOT, and the names of the intrinsic functions such as COS, INT and STR$ take up only one byte of program storage. All other characters in programs use one byte of program storage each.

When a program is being executed, space is dynamically allocated on the stack as follows:

1. Each active FOR. . .NEXT loop uses 22 bytes.

2. Each active GOSUB (one that has not returned yet) uses 6 bytes.

3. Each parenthesis encountered in an expression uses 4 bytes and each temporary result calculated in an expression uses 12 bytes.

The hints below should improve the execution time of your BASIC program. Note that some of these hints are the same as those used to decrease the space used by your programs. This means that in many cases you can increase the efficiency of both the speed and size of your programs at the same time.

1. Delete all unnecessary spaces and REM's from the program. This may cause a small decrease in execution time because BASIC would otherwise have to ignore or skip over spaces and REM statements.

2. *THIS IS PROBABLY THE MOST IMPORTANT SPEED HINT.*

   Use variables instead of constants. It takes more time to convert a constant to its floating point representation than it does to fetch the value of a simple or matrix variable. This is especially important within FOR. . .NEXT loops or other code that is executed repeatedly.

3. Variables which are encountered first during the execution of a BASIC program are allocated at the start of the variable table. This means that a statement such as 5 A=0:B=A:C=A, will place A first, B second, and C third in the symbol table (assuming line 5 is the first statement executed in the program). Later in the program, when BASIC finds a reference to the variable A, it will search only one entry in the symbol table to find A, two entries to find B and three entries to find C, etc.

4. Use NEXT statements without the index variable. NEXT is somewhat faster than NEXT I because no check is made to see whether the variable specified in the NEXT is the same as the variable in the most recent FOR statement.

Though implementations of BASIC on different computers are in many ways similar, there are some incompatibilities which you should watch for if you are planning to convert some BASIC programs that were not written in AIM 65 BASIC.

1. Matrix subscripts. Some BASICs use "[" and "]" to denote matrix subscripts. AIM 65 BASIC uses "(" and ")".

2. Strings. A number of BASICs force you to dimension (declare) the length of strings before you use them. You should remove all dimension statements of this type from the program. In some of these BASICs, a declaration of the form DIM A$(I,J) declares a string matrix of J elements each of which has a length I. Convert DIM statements of this type to equivalent ones in AIM 65 BASIC: DIM A$(J).

   AIM 65 BASIC uses "+" for string concatenation, not "," or "&".

   AIM 65 BASIC uses LEFT$, RIGHT$ and MID$ to take substrings of strings. Other BASICs uses A$(I) to access the Ith character of the string A$, and A$(I,J) to take a substring of A$ from character position I to character position J. Convert as follows:

OLD	AIM 65
A$(I)	MID$(A$,I,1)
A$(I,J)	MID$(A$,I,J-I+1)

   This assumes that the reference to a substring of A$ is in an expression or is on the right side of an assignment. If the reference to A$ is on the left hand side of an assignment, and X$ is the string expression used to replace characters in A$, convert as follows:

OLD	AIM 65
A$(I)=X$	A$=LEFT$(A$,I-1)+X$+MID$(A$,I+1)
A$(I,J)=X$	A$=LEFT$(A$,I-1)+X$+MID$(A$,J+1)

3. Multiple assignments. Some BASICs allow statements of the form: 500 LET B=C=0. This statement would set the variables B & C to zero.

In AIM 65 BASIC this has an entirely different effect. All the "="s" to the right of the first one would be interpreted as logical comparison operators. This would set the variable B to -1 if C equaled 0. If C did not equal 0, B would be set to 0. The easiest way to convert statements like this one is to rewrite them as follows:

```
500 C=0:B=C.
```

4. Some BASICs use "/" instead of ":" to delimit multiple statements per line. Change all occurrences of "/" to ":" in the program.

5. Programs which use the MAT functions available in some BASICs will have to be re-written using FOR. . .NEXT loops to perform the appropriate operations.

6. A PRINT statement with no arguments will not cause a paper feed on the printer. To generate a paper feed (blank line), use PRINT "space"

DECIMAL	CHAR.	DECIMAL	CHAR.	DECIMAL	CHAR.
000	NUL	043	+	086	V
001	SOH	044	,	087	W
002	STX	045	-	088	X
003	ETX	046	.	089	Y
004	EOT	047	/	090	Z
005	ENQ	048	0	091	[
006	ACK	049	1	092	/
007	BEL	050	2	093	]
008	BS	051	3	094	↑
009	HT	052	4	095	←
010	LF	053	5	096	`
011	VT	054	6	097	a
012	FF	055	7	098	b
013	CR	056	8	099	c
014	SO	057	9	100	d
015	SI	058	:	101	e
016	DLE	059	;	102	f
017	DC1	060	<	103	g
018	DC2	061	=	104	h
019	DC3	062	>	105	i
020	DC4	063	?	106	j
021	NAK	064	@	107	k
022	SYN	065	A	108	l
023	ETB	066	B	109	m
024	CAN	067	C	110	n
025	EM	068	D	111	o
026	SUB	069	E	112	p
027	ESCAPE	070	F	113	q
028	FS	071	G	114	r
029	GS	072	H	115	s
030	RS	073	I	116	t
031	US	074	J	117	u
032	SPACE	075	K	118	v
033	!	076	L	119	w
034	"	077	M	120	x
035	#	078	N	121	y
036	$	079	O	122	z
037	%	080	P	123	{
038	&	081	Q	124	|
039	'	082	R	125	}

DECIMAL	CHAR.	DECIMAL	CHAR.	DECIMAL	CHAR.
040	(	083	S	126	~
041	)	084	T	127	DEL
042	*	085	U		

LF=Line Feed FF=Form Feed CR=Carriage Return DEL=Rubout on TTY

AIM 65 BASIC allows a user to link to assembly language subroutines, via the USR(W) function. This function allows one parameter to be passed between BASIC and a subroutine.

The first step is to allocate sufficient memory for the subroutine. AIM 65 BASIC always uses all RAM memory locations, beginning at decimal location 530 (hex location 212), unless limited by the user. You can limit BASIC's memory useage by answering the prompt MEMORY SIZE? (see Subject 100) with some number less than 4096, assuming a 4K system. This will leave sufficient space for the subroutine at the top of RAM.

For example, if your response to MEMORY SIZE? is "2048", 1518 bytes at the top of RAM will be free for assembly language subroutines.

Parameter (W), passed to a subroutine by USR(W), will be converted to floating-point accumulator located at $A9. The floating-point accumulator has the following format:

ADDRESS	CONTENT
$A9	Exponent + $81 ($80 if mantissa = 00)
$AA - $AD	Mantissa, normalized so that Bit 7 of MSB is set. $AA is MSB, $AD is LSB.
$AE	Sign of mantissa

A parameter passed to an assembly language subroutine from BASIC can be truncated by the subroutine to a 2-byte integer and deposited in $AC (MSB) and $AD (LSB). If the parameter is greater than 32767 or less than -32768, an FC error will result. The address of the subroutine that converts a floating-point number to an integer is located in $B006, $B007.

A parameter passed to BASIC from an assembly language subroutine will be converted to floating-point. The address of the subroutine that performs this conversion is in $B008, $B009. The integer MSB ($AC) must be in the accumulator; the integer LSB ($AD) must be in the Y register.

Prior to executing USR, the starting address of the assembly language subroutine must be stored in locations $04 (LSB) and $05 (MSB). This is generally performed using the POKE command. Note that more than one assembly language subroutine may be called from a BASIC program, by changing the starting address in $04 and $05.

Figure F-1 is the listing for a BASIC program that calls an assembly language subroutine located at $A00. Here's what the BASIC program does:

- Line 10 — Stores the starting address of the assembly language subroutine ($A00) into locations $04 and $05, using POKE.
- Line 20 — Asks for a number "N".
- Line 30 — Calls the subroutine, with N as the parameter.
- Line 40 — Upon return from the subroutine, the BASIC program prints X, the parameter passed from the subroutine to the BASIC program.
- Line 50 — Loops back to get a new "N".

```
  ROCKWELL AIM 65

<5>
MEMORY SIZE? 2048
WIDTH?
 1518 BYTES FREE
AIM 65 BASIC V1.1
OK
10 POKE 04,0:POKE 05
,10
20 INPUT"NUMBER";N
30 X=USR(N)
40 PRINTX
50 GOTO 20
```

Figure F-1. BASIC Program That Calls Assembly Language Subroutine

The assembly language subroutine (Figure F-2) performs these operations:

- Prints the floating-point accumulator ($A9 - $AE), using Monitor subroutines NUMA ($EA46), BLANK ($E83E) and CRLF ($E9F0).
- Converts the floating-point accumulator to an integer, using the subroutine at $BF00. The address $BF00 was found in locations $B006, $B007. (Address $BF00 may vary with different versions of BASIC. Be sure to check locations $B006 and $B007 for the correct address.)
- After conversion, the program again prints the floating-point accumulator.
- The program then swaps the bytes of the integer.
- Finally, the program converts the result to floating-point and returns to BASIC (JMP C0D3). Address $C0D3 was found in locations $B008, $B009. (Address $C0D3 may vary with different versions of BASIC. Be sure to check locations $B008 and $B009 for the correct address.

```
<1>
0A26          *=A00
0A00 A0 LDY #00
0A02 A2 LDX #00
0A04 B5 LDA A9,X
0A06 20 JSR EA46
0A09 20 JSR E83E
0A0C E8 INX
0A0D E0 CPX #05
0A0F D0 BNE 0A04
0A11 20 JSR E9F0
0A14 C0 CPY #00
0A16 F0 BE0 0A1F
0A18 A5 LDA AD
0A1A A4 LDY AC
0A1C 4C JMP C0D3
0A1F 20 JSR BF00
0A22 C8 INY
0A23 D0 BNE 0A02
0A25 00 BRK
0A26
```

Figure F-2. Assembly Language Subroutine

Figure F-3 shows the print-out for various values of "N".

```
<6>
OK
RUN
NUMBER? 128
89 80 00 00 00 00
89 00 00 00 80 00
-32768

NUMBER? 1
81 80 00 00 00 00
81 00 00 00 01 00
 256

NUMBER? 4097
8D 80 00 00 00 00
8D 00 00 10 01 00
 272

NUMBER? 256
89 80 00 00 00 00
89 00 00 01 00 00
 1
```

Figure F-3. Output for Example

AIM 65 BASIC programs can be stored on cassette tape by using BASIC's SAVE and LOAD commands, or by using the AIM 65 Editor. Before employing either procedure, be sure to carefully observe the recorder installation and operation procedures given in Section 9 of the AIM 65 User's Guide.

RECORDING ON CASSETTE USING THE BASIC SAVE COMMAND

The procedure to store a BASIC program is:

1. Install a cassette in the recorder, and manually position the tape to the program record position. Be sure to initialize the counter at the start of the tape.

   Note: Since remote control must be used to retrieve a BASIC program, observe the tape gap CAUTION in Section 9.1.5 (Step 1) of the AIM 65 User's Guide.

2. While in BASIC, type in SAVE. BASIC will respond with:

   OUT=

3. Enter a T (for "Tape"). BASIC will display:

   OUT=T F=

4. Enter the file name (up to five characters). If the file name is FNAME, BASIC will display:

   OUT=T F=FNAME T=

5. Put the recorder into Record mode.

6. Enter the recorder number (1 or 2) and type RETURN.

7. If remote control is being used, observe the procedures outlined in Section 9.1.5 of the AIM 65 User's Guide.

8. When recording has been completed, BASIC will display the cursor.

9. Switch the recorder out of record mode.

RETRIEVING A PROGRAM FROM CASSETTE USING THE BASIC LOAD COMMAND

The procedure to retrieve a BASIC program is:

1. Install the cassette in the recorder, and manually position the tape to about five counts before the beginning of the desired file.

   Note: Remote control must be used when retrieving a file via BASIC.

2. While in BASIC, type in LOAD. BASIC will respond with:

   IN=

3. Enter a 1 (for "Tape"). BASIC will display:

   IN=T F=

4. Enter the file name. If the file name is FNAME, BASIC will display:

   IN=T F=FNAME T=

5. Enter the recorder number (1 or 2) and type RETURN.

6. Put the recorder into play mode. Be sure to observe the procedures outlined in Section 9.1.6 of the AIM 65 User's Guide.

   While the file is being read, each line will be displayed (and printed, if the printer is on). If the printer is on, the tape gap ($A409) will probably have to be increased.

   The file being loaded will not overlay any BASIC statements already entered unless the statement numbers are the same.

7. When loading has been completed, BASIC will display the cursor.

8. Switch the recorder out of play mode.

CASSETTE OPERATIONS USING THE AIM 65 EDITOR

AIM 65 BASIC programs can also be stored and retrieved from cassette using the AIM 65 Editor. However, if the program is to be retrieved by BASIC at some future time, one rule must be observed:

> When BASIC stores a program on cassette, it inserts a CTRL/Z after the last line. The AIM 65 Editor will strip off the CTRL/Z when it retrieves the program. Therefore, before storing a BASIC program from the Editor, the user must insert a CTRL/Z following the last line of the program.

SECTION

APPENDICES

SUBJECT

ATN IMPLEMENTATION

The ATN function (see Subject 307) can be programmed in RAM using the AIM 65 Mnemonic Entry (I) and Alter Memory Locations (/) commands, as shown below. The program is written for the AIM 65 with 4K bytes of RAM. The ATN function can be relocated elsewhere in memory by changing the starting addresses of the instructions and constants, the conditional branch addresses, the vector to the constants start address and the vector to the ATN function start address.

ATN FUNCTION CONSTANTS ENTERED BY ALTER MEMORY <M>

```
<M> = 0F80   XX  XX  XX  XX
</> = 0F80   0B  76  B3  83
</>   0F84   BD  D3  79  1E
</>   0F88   F4  A6  F5  7B
</>   0F8C   83  FC  B0  10
</>   0F90   7C  0C  1F  67
</>   0F94   CA  7C  DE  53
</>   0F98   CB  C1  7D  14
</>   0F9C   64  70  4C  7D
</>   0FA0   B7  EA  51  7A
</>   0FA4   7D  63  30  88
</>   0FA8   7E  7E  92  44
</>   0FAC   99  3A  7E  4C
</>   0FB0   CC  91  C7  7F
</>   0FB4   AA  AA  AA  13
</>   0FB8   81  00  00  00
</>   0FBC   00
```

Constants Starting Address = $0F80_8$

ATN FUNCTION INSTRUCTIONS STORED BY MNEMONIC ENTRY (I)

```
<I>
XXXX *=0FBD
0FBD  A5  LDA  AE
0FBF  48  PHA
0FC0  10  BPL  0FC5
0FC2  20  JSR  CCB8
0FC5  A5  LDA  A9
0FC7  48  PHA
0FC8  C9  CMP  #81
0FCA  90  BCC  0FD3
0FCC  A9  LDA  #FB
0FCE  A0  LDY  #C6
0FD0  20  JSR  C84E
```

Instructions Starting Address = 0FBD

```
0FD3  A9  LDA  #80 }
0FD5  A0  LDY  #0F }     Starting Address of Constants = 0F80
0FD7  20  JSR  CD44
0FDA  68  PLA
0FDB  C9  CMP  #81
0FDD  90  BCC  0FE6
0FDF  A9  LDA  #4E
0FE1  A0  LDY  #CE
0FE3  20  JSR  C58F
0FE6  68  PLA
0FE7  10  BPL  0FEC
0FE9  4C  JMP  CCB8
0FEC  60  RTS
0FEC
```

BASIC INITIALIZATION FOR ATN FUNCTION

BASIC memory must be initialized below the memory allocated to the ATN function. The ATN vector in RAM must also be changed from the address of the FC error message to the starting address of the ATN function instructions. This can be done using BASIC initialization, as follows:

```
<5>
MEMORY SIZE? 3968          Limit BASIC to F80(16)
WIDTH?
 3438 BYTES FREE
   AIM 65 BASIC V1.1
POKE 188, 189              Change ATN function vector low to BD(16)
POKE 189, 15               Change ATN function vector high to 0F(16)
?ATN (TAN(.5))             Test case to verify proper ATN function program
 .5                        Expected answer = .5
```

SAVING ATN OBJECT CODE ON CASSETTE

The object code for the ATN function can be saved on cassette by dumping addresses $00BB through $00BD (Jump instruction to ATN) and $0F80 through $0FEC (constants and instructions) after the function is initially loaded and verified.

The ATN function can then be loaded from cassette by executing the Monitor L command after BASIC has been initialized via the 5 command. After the ATN function has been loaded, reenter BASIC with the 6 command.

NOTES