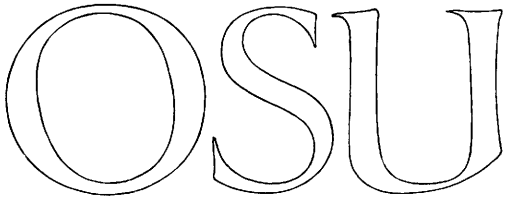
ccm-70-10

BASIC LIST PROCESSOR (BLIP-I)
Preliminary Reference Manual

by
Gil Bachelor

January, 1970



COMPUTER CENTER
Oregon State University
Corvallis, Oregon 97331

BASIC LIST PROCESSOR (BLIP-I)

Preliminary Reference Manual

by
G. Bachelor

January, 1970

ccm 70-10

Computer Center
Oregon State University
Corvallis, Oregon 97331

TABLE OF CONTENTS

	page
Preface	i
References	ii
Introduction	1
Part I BLIP-I Basic Principles	2
Atoms	2
Cells	2
Pointer	2
Variables	3
Free Storage List	3
Reference to a Cell	3
List Structure (example)	3
List Processing (example)	4
Registers and Arithmetic Operations	4
Part II The BLIP-I Language	5
Subprograms	5
Main Subprogram Structure	5
Subroutine Subprogram Structure	5
Language Elements	5
Characters	5
Integers	6
Variables	6
Registers	6
Labels	6
Special Characters and Words	6
Storage	7
Initial Conditions	7
Assignment Statements	7
Label Definition	8
BEGIN, END, and Declarations	8
Transfer of Control	9
PUSH and POP	12
Input and Output	13
Register Operations	14
Part III BLIP-I Program Preparation, Translation and Execution	17
Program Preparation	17
Program Translation	18
Program Execution	20
Program Termination	21
Example of a BLIP Run	24

PREFACE

This manual describes a "basic list processor" called BLIP, which is now available to OS-3 users. BLIP is similar to the WISP language designed by M. V. Wilkes (see references 4 and 5). It has been implemented by using the macro expander SIMCMP (reference 3). The current version (BLIP-I) is considered "experimental". It can be improved considerably by writing an actual BLIP compiler, which would make possible more flexibility in the statement formats. Whether such improvements will be implemented depends on the amount of interest shown by users.

REFERENCES

1. Berkeley, E. C. The programming language LISP: an introduction and appraisal. Computers and Automation, 13, 9 (Sept. 1964), 16-23.

2. McCarthy, J. et al. LISP 1.5 programmer's manual. MIT Press, 1962.

3. Orgass, R. J. and Waite, W. M. A base for a mobile programming system. Communications ACM, 12,9 (Sept. 1969), 507-510.

4. Wilkes, M. V. An experiment with a self-compiling compiler for a simple list-processing language. Annual Review of Automatic Programming, Vol. 4 (1964), 1-48.

5. Wilkes, M. V. Lists and why they are useful. ACM Proceedings 19th National Conference (1964) F1-1 to F1-5.

6. 3100/3200/3300/3500 Computer Systems COMPASS Reference Manual. Control Data Corporation, Pub. No. 60236800A (July, 1969).

BASIC LIST PROCESSOR (BLIP-I)

"BLIP" is an acronym for Basic LIst Processor. The BLIP language is a very simple, relatively "low-level" language. Each BLIP statement generates from 1 to 9 machine instructions. In the field of list processing languages, BLIP is at about the level of an assembly language. Still, BLIP is machine-independent, except for character codes and the permissible ranges of numbers.

Using BLIP, one can construct and manipulate lists and list structures of any desired complexity. The basic data items are integers or characters. (BLIP-I does not provide floating point operations.) Some of the terminology used in BLIP is borrowed from LISP (1,2): ATOM, CAR, CDR. Recursive subroutines are easily written in BLIP. Subprograms can be compiled separately, to be linked at time of loading.

At present, there is no BLIP compiler. The BLIP language is defined by a set of macro definitions, which show how to translate BLIP statements into COMPASS instructions. The macro expander *SIMCMP (3) uses the BLIP macro definitions to translate a BLIP program into a COMPASS (6) program. The COMPASS assembler is then used to translate the COMPASS program into machine form (relocatable binary).

There are three parts in this manual, as listed below:

I. BLIP-I Basic Principles
   (describes the basic data items and storage facilities of BLIP)
II. The BLIP-I Language.
III. BLIP-I program preparation, translation, and execution.

I. BLIP-I Basic Principles

ATOMS

An atom is a positive or negative (or zero) integer in the range -4,000,000 to +4,000,000 (approximately). Small non-negative integers may represent characters for input or output. Such atoms are represented in BLIP language by notations such as:

'1' '7' 'A' 'X' '$' DOT RET

The integers 0 to 9 represent the characters 0,1,...,9. The sequence of integer representations for character is:

0,1,2,...,9,A,B,C,...,Z,:, other special characters, RET.

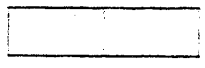
RET is a special character denoting carriage return (end of line). DOT denotes the period (.).

(NOTE: There are gaps in the sequence of integers that represents characters. For example, the integer that represents the letter A is not 10 but 17.)

CELLS

A cell is a storage element that is used in constructing lists and list structures. Each cell has two fields, called the CAR and CDR fields.

CELL



CAR field CDR field

POINTER

Each field of a cell may contain either an atom or a pointer to another cell. Normally, the CDR field contains a pointer to the next cell in a linked list, except that the CDR field of the last cell in a list contains the atom '0'. The CAR field usually contains either an atom or a pointer to a sub-list.

VARIABLES

A variable is the name of a special storage element which can contain either an atom or a pointer to a cell. In BLIP-I, variable names are single letters: A,B,C,...,Z.

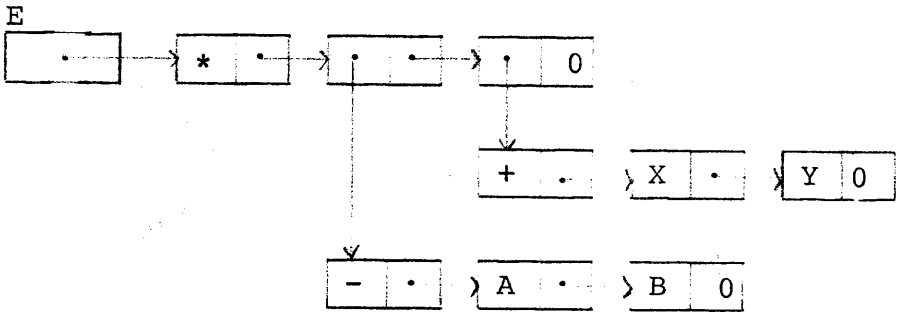
FREE STORAGE LIST

The variable F has a special purpose; F points to the free storage list (FSL), which is a simple linked list of all cells which are not currently in use. Cells are removed from the FSL when needed to construct lists; they are returned to it when no longer needed. In BLIP, this is not automatic (except in the operations PUSH, POP, CALL, and RETURN). The BLIP programmer has the responsibility to return cells to the FSL when he is finished using them.

REFERENCE TO A CELL

In BLIP-I, the only cells which can be referred to are those which are pointed to by a variable. For example, if the variable V contains a pointer to a cell, then CAR V refers to the CAR field of that cell and CDR V refers to its CDR field. By using these references, one can put atoms or pointers into the fields of a cell; one can test the contents of a field; and one can copy the contents of a field to another cell or variable.

LIST STRUCTURE (Example)

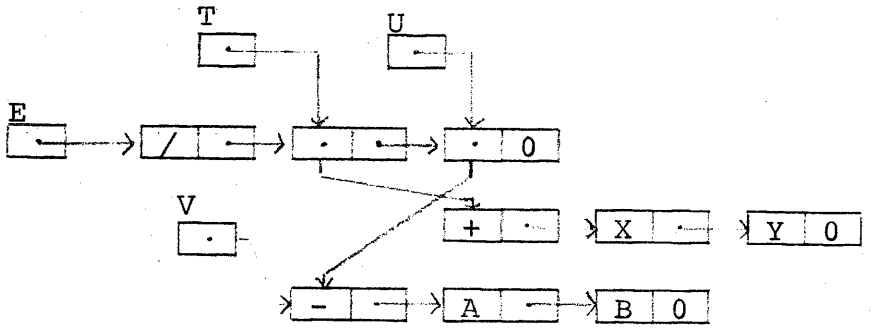


Here is an example of a simple list structure. It is a "tree" representation of the expression (A-B)*(X+Y), and the variable E contains a pointer to its "first" cell.

LIST PROCESSING (example)

Starting with the list structure shown in the example on the preceding page, suppose we do the following operations:

```
CAR E = '/'.
T = CDR E.
U = CDR T.
V = CAR T.
CAR T =
   CAR U.
CAR U = V.
```



The resulting list structure is shown above. Also shown are the cells to which variables T,U,V are pointing. The structure is now a tree representing the expression (X+Y)/(A-B).

REGISTERS and ARITHMETIC OPERATIONS

A register is the name of a special storage element that may contain only an atom. In BLIP-I, the names of registers consist of the letter "R" followed by a digit: R0,R1,...,R9.

Arithmetic operations (+,-,*,/) and arithmetic comparisons can be performed on the contents of registers. Arithmetic operations cannot be performed on atoms contained in cells, and comparisons on cells do not work properly for negative atoms. However, atoms can be copied from the CAR fields of cells to registers, and vice versa.

II. The BLIP-I Language

SUBPROGRAMS

A BLIP program may be divided into a number of subprograms, which can be compiled separately, then loaded together for a run. A group of subprograms to be loaded and run must contain one and only one "main" subprogram; it may contain none or any number of "subroutine" subprograms.

MAIN SUBPROGRAM STRUCTURE

```
BEGIN MP.
CELLS 00400.
[statements]
END MP.
```

MP is the name and entry point of this main subprogram; it is also the starting location for the program. The CELLS declaration causes 400 cells to be placed in the free storage list, before the program is started.

SUBROUTINE SUBPROGRAM STRUCTURE

```
BEGIN SB.
[statements]
END.
```

SB is the name and entry point of this subroutine subprogram.

LANGUAGE ELEMENTS

A subprogram is composed of statements and declarations. Statements and declarations in BLIP are composed of basic elements of the kinds listed below. Each BLIP-I statement or declaration must be on a separate line and must end with a period (.). The elements within each statement or declaration must be separated by single spaces, except that no space should precede the period.

CHARACTERS

Characters are represented in statements by notations such as '3','A',RET. The notation '<u>char</u>' will stand for any character (except period, DOT, or RET) in the descriptions of statements.

INTEGERS

Integers can appear in certain statements and declarations. An integer consists of 5 digits, or a minus sign followed by 4 digits. The permissible range is -9999 to 16383. The notation INTEGER stands for an integer in the descriptions.

VARIABLES

A variable is a single letter (A,B,...,Z). Variables are represented in the descriptions by notations such as $v_1, v_2, \ldots$

REGISTERS

A register is the letter "R" followed by a digit (R0,R1,...,R9). Registers are denoted in the descriptions by $r_1, r_2, r_3, \ldots$

LABELS

A label consists of two letters (for example, PA, MP, SB,...). It identifies a place in the program to which control can be transferred. Labels are denoted by the notation LABEL in the descriptions.

SPECIAL CHARACTERS AND WORDS

Other elements of statements are special characters (such as = or +) and special words (such as CAR, IF, BEGIN). These elements serve to identify the action or meaning of a statement or declaration. They are represented in the descriptions by themselves. Notations such as $\left\{ \begin{matrix} \text{CAR} \\ \text{CDR} \end{matrix} \right\}$ mean that any of the elements shown in the brackets can be used, with various meanings according to which element is chosen.

STORAGE

In BLIP-I, variables are common to all subprograms. For example, a reference to X in one subprogram refers to the same variable as a reference to X in another subprogram.

Registers in BLIP-I are local to each subprogram; each subprogram has its own set of 10 registers, which are not accessible to other subprograms.

INITIAL CONDITIONS

Initially (when the main subprogram is started), all variables except F contain the atom '0'. F contains a pointer to the free storage list (FSL), which is a linked list of all available cells. The CAR fields of all cells contain '0', and the CDR field of each cell points to the next cell. The CDR field of the last cell contains '0'. The number of cells is equal to the number specified in the CELLS declaration.

The initial content of registers is not specified.

ASSIGNMENT STATEMENTS

An assignment statement copies the quantity specified on the right of the equals sign (=) into the variable or field (of a cell) specified on the left. The quantity on the right can be an atom or pointer contained in a variable or cell, or it can be a specified character (an atom). BLIP assignment statements can have the following forms:

FORM	EXAMPLE
$v_1 = v_2$.	P = X.
$v_1 = \begin{matrix}\text{CAR}\\\text{CDR}\end{matrix} v_2$.	A = CAR Q.

FORM	EXAMPLE
{CAR / CDR} v_1 = v_2.	CDR B = T.
{CAR / CDR} v_1 = {CAR / CDR} v_2.	CAR C = CDR U.
{CAR / CDR} v_1 = 'char'.	CAR S = '$'.
CAR v_1 = {DOT / RET}.	CAR T = RET.
v_1 = 'char'.	E = '0'.

LABEL DEFINITION

Labels are defined by the statements shown below, and also by BEGIN statements. A label identifies a location in a program; a transfer of control to a label causes statements following the label to be executed.

FORM	EXAMPLE
label.	XB.

Label refers to the statement that follows it; it is local, unknown outside the subprogram in which it occurs.

ENTRY label.	ENTRY PE.

Label refers to the statement that follows it; it is global and can be referred to form other subprograms.

BEGIN, END, AND DECLARATIONS

BEGIN and END statements mark the beginning and end of subprograms. Declarations provide informa- to the BLIP compiler.

BEGIN label.	BEGIN ZP.

This is the first statement of a main or subroutine subprogram. The label is the name of the subpro- gram, and is also defined as an entry point (see ENTRY) that refers to the first statement of the

subprogram.

FORM	EXAMPLE
END.	END.

This is the last statement of a subroutine subprogram. If execution reaches the END statement, a RETURN is executed (see RETURN statement).

END label.	END ZP.

This is the last statement of a main subprogram. The label specifies the statement at which execution is to begin (normally the same as the label on the BEGIN statement).

If execution reaches the END statement, a RETURN is executed.

CELLS integer.	CELLS 00350.

The CELLS declaration specifies the number of cells to be placed in the free storage list before starting execution of the program. One and only one CELLS declaration must appear among all the subprograms to be loaded for a given run.

EXT label.	EXT PE.

The EXT declaration declares the label to be external to the subprogram. That is, it is an ENTRY point in some other subprogram.

TRANSFER OF CONTROL

To "transfer control" means to cease executing statements at one place in a program and start executing statements at the place to which control is transferred. The statements TO, CALL and RETURN unconditionally transfer control. The IF statements transfer control if a certain condition is satisfied. Most of the IF statements compare two quan-

tities to see if one of the relations listed below is satisfied. If so, the transfer of control takes place.

RELATION	CONDITION IS SATISFIED IF:
x EQ y	x = y (x is equal to y)
x NE y	x ≠ y (x is not equal to y)
x LT y	x < y (x is less than y
x GE y	x ≥ y (x is greater than or equal to y)

NOTE: In the IF statements listed in the group below, all atoms appear to be positive, with negative atoms appearing to be larger than positive atoms. See the register operations for tests that work correctly with negative atoms.

FORM	EXAMPLE
IF v_1 {EQ / NE / LT / GE} v_2 TO label.	IF X NE J TO BD.
IF {CAR / CDR} v_1 {EQ / NE / LT / GE} v_2 TO label.	IF CAR Y EQ B TO QX.
IF {CAR / CDR} v_1 {EQ / NE / LT / GE} {CAR / CDR} v_2 TO label.	IF CDR K GE CAR T TO LB.
IF {CAR / CDR} v_1 {EQ / NE / LT / GE} 'char' TO label.	IF CAR Z LT ':' TO DL.
IF CAR v_1 {EQ / NE / LT / GE} {DOT / RET} TO label.	IF CAR B NE DOT TO MR.

The following IF statements test a quantity to see if it is or is not an atom. (If not an atom, it is a pointer.)

FORM	EXAMPLE
IF v_1 {EQ / NE} ATOM TO label.	IF S EQ ATOM TO SA.
IF {CAR / CDR} v_1 {EQ / NE} ATOM TO label.	IF CDR Q NE ATOM TO QL.

Unconditional transfers of control (and a special IF statement) are listed below.

TO label. TO PQ.

Unconditionally transfers control to specified label.

CALL label. CALL SB.

Saves location of following statement on a push down list (ADSTAK) and transfers control to specified label.

CALLX label. CALLX SX.

Same as CALL, but also declares the label to be external (see EXT).

RETURN. RETURN.

Returns control to the statement following the most recently executed CALL or CALLX statement that has not been "returned to". (Location is obtained from ADSTAK, and ADSTAK is popped.)

ERETURN. ERETURN.

This is the "error return" statement; it is the

same as RETURN, but sets an "error flag" that can be te ted by an IF ERROR statement.

FORM	EXAMPLE
IF ERROR TO label.	IF ERROR TO RR.

This statement (if used) must immediately follow a CALL or CALLX statement. It will transfer control to the specified label if the subroutine that was called returned with an ERETURN statement.

NOTES: The labels specified in transfer of control statements can be either local or external.

The main subprogram is "called" initially to start execution; a RETURN to this call terminates execution.

PUSH AND POP

To "push" a list means to add a new cell at the beginning. To "pop" a list means to remove the first cell. These operations can easily be performed by BLIP assignment statements. However, they are so useful and frequently used that special statements are provided to carry them out.

FORM	EXAMPLE	EQUIVALENT CODE
PUSH v_1.	PUSH M.	T = F. F = CDR F. CDR T = M. M = T.

A PUSH statement removes a cell from the FSL and puts it at the head of the list pointed to by v_1. (If v_1 contained an atom, it will now point to a single-cell list.)

FORM	EXAMPLE	EQUIVALENT CODE
POP v_1.	POP K.	T = K. K = CDR K. CDR T = F. F = T.

A POP statement removes a cell from the head of the list pointed to by v_1 and puts it in the FSL.

NOTE: In the "equivalent code" shown at right above and on the preceeding page, the variable T is used. Of course, PUSH and POP statements do not actually use a variable like this. The only variables affected are F and v_1.

INPUT AND OUTPUT

Input and output for BLIP programs are handled by special subroutines. For input, one line (or card) is read and characters are passed one at a time to the BLIP program, with codes converted to those used in BLIP. When the last character of a line has been "read", the special code RET is supplied next. The next call for a character causes another line to be read. The BLIP program can also cause the rest of the current input line to be discarded, by using the INRET statement.

For output, a BLIP program outputs one character at a time. These characters are stored, to build up a line. When the line is full, or when the code RET is output, the line is actually written. (When a BLIP program terminates, with or without error, the last partially filled line, if any, is written anyway.)

FORM	EXAMPLE
CAR v_1 = IN.	CAR R = IN.

One character (an atom) is read from the input unit and stored in CAR v_1. The character RET is always

supplied at the end of a line.

FORM	EXAMPLE
INRET.	INRET.

Causes characters to be read and discarded until a RET has been received.

ON EOF TO label.	ON EOF TO EF.

If an IN or INRET statement results in reading a file mark (denoting end of input data), control will automatically be transferred to the specified label. Each ON EOF statement overrides the effect of any previously executed ON EOF statement.

OUT = CAR v_1.	OUT = CAR W.

Outputs the (presumably) character contained in CAR v_1.

OUT = 'char'.	OUT = 'E'.

Outputs the specified character.

OUT = DOT.	OUT = DOT.

Outputs a period.

OUT = RET.	OUT = RET.

Outputs a RET, causing a line to be written.

REGISTER OPERATIONS

Due to the manner in which atoms and pointers are represented in variables and in cells, it is not feasible to perform arithmetic operations on these quantities. Atoms can be copied from the CAR fields of cells into registers, and vice versa. The copying process alters the representation of an atom, so that arithmetic operations can be performed on it. (If a pointer is copied to a register, it will be converted into an atom.) The atoms in registers are integers in the (approxi-

mate) range -8,000,000 to +8,000,000. An atom whose magnitude is larger than (about) 4,000,000 cannot be copied correctly into the CAR field of a cell. (No error indication is given.)

FORM	EXAMPLE
r_1 = CAR v_1.	R3 = CAR B.

Copies atom in CAR v_1 into register r_1.

CAR v_1 = r_1.	CAR H = R7.

Copies atom in r_1 into CAR v_1.

r_1 = r_2.	R2 = R9.

Copies atom in r_2 into r_1.

r_1 = integer.	R5 = 00003.

Puts specified integer into r_1.

r_1 = r_2 + r_3.	R0 = R8 + R1.

Adds the atoms in r_2 and r_3, puts result in r_1.

r_1 = r_2 - r_3.	R4 = R4 - R6.

Subtracts r_3 from r_2, puts result in r_1.

r_1 = r_2 * r_3.	R3 = R9 * R2.

Multiplies r_2 by r_3, puts result in r_1.

r_1 = r_2 / r_3.	R5 = R1 / R8.

Divides r_2 by r_3, puts quotient (an integer) in r_1.

r_1 = REM.	R4 = REM.

If previous statement was a divide operation, stores the remainder from the division in r_1.

NOTE: If overflow or division by zero occurs in arithmetic operations, an erroneous result is stored, with no error indication.

The three registers used in an arithmetic operation do not have to be distinct registers.

FORM	EXAMPLE
IF r_1 { EQ / NE / LT / GE } 0 TO label.	IF R7 LT 0 TO RN.
IF r_1 { EQ / NE / LT / GE } r_2 TO label.	IF R3 NE R6 TO JP.

The comparisons above work correctly; negative integers are "less than" positive integers.

III. BLIP-I Program Preparation, Translation, and Execution

PROGRAM PREPARATION

A BLIP program can be punched on cards, one statement per card; or it can be typed in and stored in a file, one statement per line. In either case, there must not be any spaces at the beginning of the line; there must be exactly one space between each element of a statement and the next element; and each statement must end with a period (no space in front of the period). Anything that follows the period on a line or card will be ignored.

A BLIP program may consist of several subprograms. The first line must be a BEGIN. After the END of one subprogram comes the BEGIN of the next one (if any). After the END of the last subprogram, type a line with only a single period. (An end of file card can be used instead.)

If EDIT is used to prepare the program, one should use the OUT command to put the program into a file (either a unit number or a name). Since EDIT writes a file mark at the end of the output, the last line mentioned above (a single period) is not needed.

The following illustration shows the overall form of a BLIP program.

```
BEGIN AB.
CELLS 00200.
(Main subprogram)
END AB.
BEGIN CD.
(A subprogram)
END.
BEGIN EF.
(Another subprogram)
END.
.   (or a file mark, or end of file card)
```

PROGRAM TRANSLATION

At present, there is no BLIP compiler. The BLIP-I language is defined as a set of "macros", in a file called *BLIPM. The macro expander *SIMCMP translates a BLIP program into a COMPASS-language program, using the macro definitions in *BLIPM. Then the COMPASS assembler is used to translate the COMPASS-language program into relocatable binary form.

If the BLIP program to be translated is in a file, one first calls SIMCMP with a control statement of the form:

```
*SIMCMP,M=*BLIPM,I=(lun or name),O=50,D,N
```

The (lun or name) following "I" is the logical unit number or name of the file where the BLIP program is stored. The output unit (50 in the example above) is the unit on which the COMPASS-language program is written. The "D" causes unrecognized lines to be written on logical unit 61, with sequence numbers to indicate where they appeared in the BLIP program. The notation D=(lun or name) can be used to cause the diagnostics to be written on a unit other than 61. The "N" option causes SIMCMP to put sequence numbers in columns 76 to 80 of the output, which correspond to the input lines. (For example, the 23rd line of the BLIP program might generate 5 lines of COMPASS language. All 5 lines will have the sequence number 00023.)

If the BLIP program is on cards, one would use the control statement:

```
7
8*SIMCMP,M=*BLIPM,O=50,D,N
```

followed immediately by the BLIP program deck. (Omitting the "I" parameter causes unit 60 to be used for input.)

If the BLIP program has been successfully translated into COMPASS (no diagnostics), then one calls COMPASS with a control statement of the form:

```
COMPASS,I=50,X,D
```

If COMPASS produces no diagnostics, one can proceed to run the program. (See next section.) However, there are several kinds of errors in BLIP programs that may show up during the COMPASS assembly. If such errors occur, and the user is on-line, one should use the control statement UNEQUIP,56 before the next attempt to translate the program. A typical COMPASS diagnostic is shown below:

```
C=00125  U 00216     UJP XZ
```

The number (if any) following the "C=" is taken from columns 76 to 80 of the line that COMPASS read. The letter (or letters) following this number indicates what kind of error (or errors) was detected. The number following the error "flag" is the relocatable machine address of the instruction that COMPASS generated for this line. The rest of the diagnostic message consists of the first 36 columns of the line.

If the "N" parameter was used on the *SIMCMP call, then the number following "C=" will indicate which line of the BLIP program generated the COMPASS instruction on which the error was found. Some of the COMPASS error flags and possible BLIP programming errors that could cause them are listed below.

Error flag	Meaning	Possible BLIP errors
A	Address error.	1. Improper variable. Example: B = CAR %. 2. Improper register specifier. Example: R5 = R2 + R$.

ERROR FLAG	MEANING	POSSIBLE BLIP ERRORS
		3. Improper label in a TO, IF, or CALL statement. Example: TO $5.
D	Duplicate symbol.	1. Definition of a particular label more than once in a subprogram.
DA	Duplicate symbol in address field.	1. All references to a duplicate symbol are flagged "DA".
L	Label error.	1. Improper label. Example: 5Z.
M	Modifier error.	1. Improper condition in an IF statement. Example: IF CAR B LE CAR W TO DE.
U	Undefined symbol.	1. Label in IF, TO, or CALL statement is not defined. Example: CALL XY. (where XY is not defined.)
		2. Misspelled words in various statements. Examples: CUR X = B. OUT = RAT.

PROGRAM EXECUTION

If there are no errors in either the SIMCMP or COMPASS translations, one can attempt to load and run the program. First, the BLIP subroutine library must be equipped:

```
EQUIP,63=*BLIPLIB
```

Then one uses the control statement

```
LOAD,56
```

followed by the line

```
RUN
```

(Teletype users must type "return", then "line feed" after the word RUN.)

The loader may detect some errors. The most likely ones are DS (duplicate symbol), UD (undefined symbol), and TR (transfer symbol). DS means that the label (following "DS") appeared more than once as the name of a subprogram or as an ENTRY label. UD means that the label (following "UD") appeared in an EXT or CALLX statement, but was not defined as an ENTRY label or as the name of a subprogram. A "TR" error means that there is more than one main subprogram.

If there are no loading errors, the loader prints RUN and starts the user's program. The program may be terminated immediately, if one of two errors occurs. If so, a message is printed:

NO MAIN PROGRAM	means that there is no main subprogram.
TOO MANY CELLS	means that there is not enough storage space for the number of cells that the user declared.

Finally, if no errors have been detected, the program actually starts running, at the first statement of the main subprogram. The program may print outputs (on logical unit 61) and/or read inputs (from logical unit 60). Teletype users must end each input line with "return" and "line feed" (in this order).

PROGRAM TERMINATION

There are several conditions that can terminate the execution of a BLIP program. In each case, a message is printed, as shown below:

END OF BLIP RUN	This message is printed when the main subprogram executes a RETURN to the initial CALL that started execution.

UNCHECKED EOF

This message is printed if a file mark (end of file card) is read, and no ON EOF statement has been executed.

ERROR FOLLOWING XY: V IS EMPTY

A message of this form is printed when a reference to the CAR or CDR of a variable is made, and the variable contains an atom instead of a pointer to a cell. The label (XY in the example above) indicates the section of program where the error occurred. This message can be produced by a PUSH or CALL statement, if the free storage list (F) becomes empty.

ERROR FOLLOWING XY: BAD RTN ADR

This message means that a cell (in ADSTAK) containing a return address has been altered. This situation can occur if some variable contains a pointer to a cell that is in the free storage list, if this cell is used for a return address (on a CALL), and then the cell is changed before a RETURN is executed. The label (XY in this example) may also be destroyed in a situation like this.

ERROR FOLLOWING XY: INTERRUPTED

This message is printed if the user presses "break" or "control A" to stop the program, and then types the control statement MI (manual interrupt). If a program is in a loop, this technique can be used to find out what part of the program is being executed.

Two other conditions can cause OS-3 to stop the program and print a message:

INSUFFICIENT FILE SPACE

This means that the program is attempting to write a line on a file or line printer (or other device), and the file space limit has been reached.

TIME CUT

This means that the time limit has been reached.

In either case, the OS-3 control mode is entered. If this occurs in a batch job, the job is aborted. If the user is on-line, he can increase the file space limit or the time limit (if possible) and type the control statement GO to resume execution. Or, he may decide that something is wrong, and use the statement MI. (see discussion above, under INTERRUPTED.)

EXAMPLE OF A BLIP RUN

Here is an example of an actual BLIP run. The program is very simple. It reads a line, stacking the characters in a push down list (P). Then it prints the line out, with the characters appearing in reverse order.

```
#EDIT

]INPUT
00001:BEGIN EX.
00002:CELLS 00100.
00003:RD.
00004:PUSH P.
00005:CAR P = IN.
00006:IF CAR P NE RET TO RD.
00007:PT.
00008:POP P.
00009:IF P EQ ATOM TO ND.
00010:OUT = CAR P.
00011:TO PT.
00012:ND.
00013:OUT = RET.
00014:TO RD.
00015:END EX.
00016:    (escape)
]OUT,EXAMPLE

]EXIT

#*SIMCMP,M=*BLIPM,I=EXAMPLE,D,N,O=50

#COMPASS,I=50,X,D
   NUMBER OF LINES WITH DIAGNOSTICS        0

#EQUIP,63=*BLIPLIB
#LOAD,56
RUN
RUN



THIS IS A TEST.
.TSET A SI SIHT
12345
54321
A QUICK BROWN FOX JUMPED OVER THE LAZY DOG'S BACK.
.KCAB S'GOD YZAL EHT REVO DEPMUJ XOF NWORB KCIUQ A
(control A)
#
```