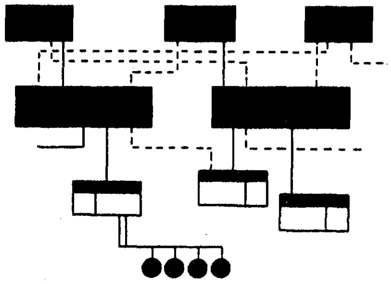
GENERAL ELECTRIC
COMPUTERS

Introduction to LP/600



GENERAL ELECTRIC

CPB-1141B

INTRODUCTION TO LP/600 LINEAR PROGRAMMING SYSTEM

REFERENCE MANUAL

October 1965

Rev. April 1966

Rev. June 1967

GENERAL ELECTRIC

INFORMATION SYSTEMS DIVISION

PREFACE

The GE-625/635 Linear Programming System language, sublanguages, and features are described in this manual. The descriptions do not go into the detail needed for a full understanding of the entire system, but are adequate for the purpose intended, that is, to present an introduction to the LP/600 linear programming system. This edition contains all the information included in previous editions, but has considerably more information about the various LP/600 linear program features. In this edition, changes in technical content from the previous edition are not identified with the customary bar in the margin opposite the change.

Suggestions and criticisms relative to form, content, purpose, or use of this manual are invited. Comments may be sent on the Document Review Sheet in the back of this manual, or may be addressed directly to Documentation Standards and Publications, B-90, Computer Equipment Department, General Electric Company, 13430 North Black Canyon Highway, Phoenix, Arizona 85029.

(2M 12-67)

CONTENTS

APPENDIXES

ILLUSTRATIONS

1. INTRODUCTION

The LP/600 Linear Programming System offers the latest advances in mathematical techniques, and variety of approaches to the solution of linear problems. LP/600 has the most comprehensive set of solution and post-optimal algorithms ever assembled in one linear programming system. It operates in a compile-and-go environment -- truly a data processing system and not merely an optimization technique as most linear programming codes have been in the past. Although the functions performed by LP/600 are very complex, the system is easy to use, flexible, and comprehensive. It embodies the best of yesterday's linear programming techniques and solution algorithms, in addition to recent advances in computational techniques, new solution algorithms, complete problem control and recycling methods, and integrated systems design. Also, many long-desired capabilities such as matrix and report generator languages are included to increase the applicability of the system.

The programming language of the LP/600 Linear Programming System is a problem-oriented, English-like control language containing over 65 system control verbs exclusive of those in the matrix and format generator languages. Its logic, similar to FORTRAN, includes looping, arithmetic and logic operations, subroutine and macro definition capability, conditional and unconditional transfers, data definition and movement, and operator communication. These operations combine with an extensive repertoire of linear program verbs to give the LP/600 user all the programming capabilities required to solve linear program models.

The Agenda Control Language is the basic language of the LP/600 system. This language is extended to contain matrix and format generator sublanguages. The extent of sublanguage usage depends on the level of matrix generation complexity and report sophistication. Each of the sublanguages are complete within the linear programming functional activities they serve.

The Agenda Control Language program is a source deck processed to form an internal agenda control file which is compiled and interpretatively executed by LP/600. This execution phase, together with pre-established actions, eliminates the necessity for operator intervention, which is a must in a multiprogramming environment. This operation is more flexible and comprehensive than "control card" methods.

LP/600 has separate invert, primal, dual, transportation, block decomposition, and group algorithms, with user-controlled multiple or partial pricing. It can solve up to 4,095-row and 262,000-column standard problems; 1000-order matrices can be solved in core. A group problem may contain up to 4000 rows in the master problem representing over 20,000 constraints. A block decomposition problem may have up to 4000 rows in the master problem and the largest subproblem; any number of additional subproblems may be defined containing over 16,000 additional rows. A transportation matrix may have over 20,000 rows and columns, consisting of up to 4000 sources and 16,000 destinations, or vice versa.

Although models of this size have previously been outside the physical capacity of both memory size and computational speed, they are well within the capacity of LP/600. Previously, two constraints existed on a linear programming problem solution--problem size and running time. In this system, the problem size restriction has effectively been removed, and the running time problem has been minimized.

Included in the solution techniques is the ability to quickly reach a nontrivial starting basis -- sometimes called crashing. Separable programming techniques are incorporated for handling nonlinear constraints. Slack variables are generated automatically. Double-precision arithmetic is used for all algorithms. Automatic controls are used to prevent cycling and digital errors.

In addition to the solution algrithms, LP/600 provides the capability to do both dynamic and static post-optimal analysis. Specifically, parametric programming with fully automatic parameter stepping can be done on right-hand-sides, objective functions, matrix columns and rows, and simultaneously the right-hand-side and objective function. Ranging can be performed on right-hand-sides, objective functions, matrix elements and columns and solution values. Updated matrix rows and columns can be obtained, and basic variables can be removed from the basic and nonbasic variables can be forced to enter.

Output capabilities consist of automatically generated restart information, standard reports, special reports by means of the format generator language, matrix picturing by range symbols or coefficient values, range and tableau reports, and intermediate solution values for plotting. The output volume can be limited by specifying particular subsets of rows and columns or by controlling the output frequency with internal control parameters.

Agenda control programming, matrix generation and programmed data formatting open many new approaches to problem formulation and solution control. Most notable, however, are the opportunities intrinsic in LP/600 for generating and recycling control and problem data during execution. Instructions in the control language permit computation and data movement at the control language level during problem execution. This means that the user can monitor in-progress computations and interrupt the solution cycle to generate permanent or temporary revisions to the problem; these revisions can then be incorporated and the solution restarted. Equally important are the facilities provided by the format generator for recycling problem data. Format generator language subroutines may be included in the agenda control program for producing new data in BCD format. Thus--based upon the results of earlier processing--data revisions, modifications, or complete new problem files may be generated and used during the execution run.

The LP/600 Matrix Generator Language is based on the input/process/output concept. Its entities consist of tables, arrays, lists, compound lists, and submatrices, all definable at execution time. In particular, commonly used specification tables, transportation cost tables, quality tables and other raw problem data can be stored in a "linear programming data bank." The data can then be retrieved randomly, set up as the nucleus of a linear program matrix and augmented by automatically generated cost rows, right-hand-sides, and structual vectors and rows. Extensive manipulations can be performed on any of the entities, including character manipulation. Furthermore, matrix generation is not a separate process; it is a routine occurrence.

The LP/600 Format Generator Language specifies report formats which are definable during problem execution. In particular, the LP/600 user may define BCD data, process it, and produce it as output in almost any format. Verbs of the language allow a number of arithmetic and other operations to be performed on current work file or problem solution data. In addition, there are extensive capabilities within this language to perform all those control operations available in the Agenda Control Language, such as looping, arithmetic and logic operations, etc. Although intended primarily to facilitate output report formatting and the recycling of problem data, the format generator has an unlimited number of uses.

Careful attention has been paid to the necessity for interfaces between the LP/600 system and user-generated FORTRAN programs; consequently, several intermediate files of LP/600 are maintained in formats which may be read and/or written by the FORTRAN IV input/output statements. The system files may be read by the read decimal input facility, and/or produced as output via the write decimal output facility, both having appropriate format statements.

LP/600 operates under the GE-625/635 General Comprehensive Operating Supervisor (GECOS). The system thus takes full advantage of the multiprogramming and input/output file control features of GECOS. Moreover, linear programming problems may be freely intermixed with others which make up the normal computing work load.

Appendix A is a diagram of data flow within the system. It shows the facilities that exist for data output and recycle during problem execution.

Each of the LP/600 programming languages are covered in a separate publication. These publications, along with the reference manuals for LP/600 Input File Preparation and LP/600 Output Descriptions are listed below.

GE-625/635 LP/600 Input File Preparation, CPB-1222

GE-625/635 LP/600 Agenda Control Language, CPB-1262

GE-625/635 LP/600 Matrix Generator Language, CPB-1263

GE-625/635 LP/600 Format Generator Language, CPB-1264

GE-625/635 LP/600 Output Descriptions, CPB-1267

LP/600 INPUT FILE PREPARATION

Input File Preparation (CPB-1222) procedures describe:

- Matrix problem input file which defines the matrix file structure, row data, column data, matrix elements and right-hand-side elements.
- Revisions to a matrix problem file which defines the permissible revisions along with the revision classifications of insertions, deletions, and replacements of matrix problem file elements.
- Modifications to a work file which defines permissible alterations through element replacements.
- Basis list which defines the logical and structural vectors for the basis file.
- List file containing a list of column and/or row names/masks.

- Edit file which defines a problem matrix in the SHARE standard format to be translated to the LP/600 format.
- Edit Specification file containing specifications for naming the input file generated as a result of edit file processing. This file also specifies the number of rows to be edited as objective functions.

LP/600 AGENDA CONTROL LANGUAGE

The Agenda Control Language (CPB-1262) consists of statements that, when executed, provide the procedures for linear programming problem solutions. The broad range of functions of the Agenda Control Language also provides procedures for:

- Conditional and unconditional transfers of control
- Arithmetic and logical operations
- Data definition and movement
- Subroutine linkage
- Looping
- Macro definition
- Operator comments
- Matrix snapshots

A summary of the functions of the Agenda Control Language is essentially a summary of the major features of the LP/600 Linear Programming system. These features are discussed below according to the classification of major functions within an Agenda Control Language program.

1. Problem Setup

    - Matrix generation and conversion: statements are available to convert an input file to a problem file, generate a problem file from a Matrix Generator Language subprogram, and translate an input file in SHARE standard format to the LP/600 input file format.
    - Matrix setup and identification: statements are available for creating a work file in core storage from a problem file, allocating memory for buffers and data regions, initializing the communication region and establishing a problem title to be printed on each output page.
    - Solution save - restore: statements are available that cause the current solution information to be written as a save file on an input/output device for subsequent use, the current basis to be punched on cards for subsequent input, and the save file and starting basis file information to be restored when needed.

2. Problem Revision

   - Problem file changes: a statement is available to revise a problem file and to write the revised file on an input/output device for subsequent use.

   - Work file changes: statements are available to modify a problem file, flag unbounded vectors to prevent their entry in the basis, generate a pseudo-right-hand-side to cause an infeasible problem to become feasible, flag specified vectors to keep them from being used in the solution, and to remove the flags from specified vectors.

3. Solution Controls

   - Control setting: statements are provided for setting LP/600 controls to nonstandard values, and user-defined arguments for naming the objective function, right-hand-side, and other argument values. Parameters may be set to values that delimit the scope of operation of certain verbs. Solution control frequencies such as those to define the frequency of printed log lines, inversion and other frequencies may be set. Switches may be turned on or off to control special operating modes. Tolerances may be changed to special values. Demands may be set for program interrupts caused by formulation, computational and other errors.

   - Control resetting: a statement is available to reset specified solution controls back to standard values.

   - Status-display: statements are provided to produce a listing of all solution controls and their current values.

4. Problem Solution

   - Standard linear program formulation: statements may be used to obtain an optimal solution of the standard linear programming problem including separable programming formulations, an optimum dual solution of the standard problem, and a nontrivial starting basis preparatory to obtaining an optimal solution.

   - Decomposition formulation: statements are available for obtaining an optimal solution of the transportation problem, special group problem using a form of decomposition, and the standard problem expressed in decomposition format.

   - Inversion: a statement is available to compute a new set of product form transformations.

5. Post-optimal

   - Parametric programming: statements are provided to parameterize the right-hand-side, objective function, matrix column, matrix row, and simultaneously the right-hand-side and objective function.

   - Ranging: statements are available to determine the range of the right-hand-side, objective function, matrix element, and column, and the levels of specified variables while minimizing the rates of change in the functional.

   - Tableau: statements may be used to update nonbasic structural vectors, a specified row or part of a row, and the explicit inverse of the current basis by updating nonbasic logical vectors.

6. Program Control

- Arithmetic and logic operations: statements may be used to compute the value of arithmetic and logic expressions.

- Data definition and manipulation: statements are available to define numeric constants and logical switches, move the contents of storage locations, control the printing of certain program comments as operator instructions, and control the printing of the contents of locations containing defined constants, switches, and communication region values.

- Sequence control: statements are available which cause branches to a specified Agenda Control Language statement or program subroutine, returns from a subroutine via a demand condition to the statement originating the demand, or to the next sequential statement following the demand. A statement of this type may cause a jump from an embedded subroutine to the statement following the one that executed the demand. Another statement may be used to test the value of a program counter and execute the next sequential instruction, or decrement the counter and branch to a specified location. The Agenda Control Language program is terminated by a statement which processes the remaining problem output, and returns control to GECOS.

7. Output

- Standard solution print: statements are available to process problem answers produced by the solution and parametric verbs.

- Output control: a statement may be used to define a subset of the work file rows and/or columns for which output is produced by the standard solution print, picture operations, and certain post-optimal statements.

- Solution plotting: statements are available to write solution values for up to 40 primal and/or dual variables and later formatting the data in a form suitable for plotting the change in solution values.

- Picture operations: statements may be used to produce a picture in the usual matrix format or a subset of the matrix as defined by an output control statement. A statement of this type can produce output for vector subsets plus those vectors with first-order interactions with the subset; or for all vector subsets in the basis with infeasible values plus all vectors with first-order interactions with the infeasible subset.

- Format generation: statements may be used to generate a format skeleton from a Format Generator Language subprogram, and later produce output in the format defined by the skeleton.

8. Input/Output Device Control

- Input/output device control: statements are available to rewind and unload specified files or erase designated files.

LP/600 FORMAT GENERATOR LANGUAGE

The Format Generator Language (CPB-1264) defines report format and content. This LP/600 sublanguage contains most all the control verbs within the Agenda Control Language in addition to those peculiar to format generation. The Format Generator Language is used for the following:

- Printing regular reports
- Printing special management reports
- Generating input data to be converted as a problem file
- Generating data to revise an existing problem file
- Generating data to modify an existing work file
- Generating a specific set of delimit rows and columns to be output
- Generating Matrix Generator Language subprograms
- Generating additional Format Generator Language subprograms

Operations are provided for obtaining solution data for a report, formatting the data, and producing the report on the printer, card punch, or magnetic tape. Solution data includes the following:

- Values of structural variables (x-values)
- Marginal costs (pi-values)
- Costs of nonbasic vectors (dj-values)
- Slack values
- Matrix, Rhs, and cost row elements

LP/600 OUTPUT DESCRIPTIONS

Examples of output reports with detailed descriptions are provided in the GE-625/635 LP/600 Output Descriptions manual, CPB-1267.

LP/600 MATRIX GENERATOR LANGUAGE

The Matrix Generator Language (CPB-1263) provides a means of defining the operations to:

- Generate and store problem data for specification tables, transportation cost tables, quality tables, lists, compound lists, arrays, and submatrices in an LP/600 data bank.
- Retrieve the stored data randomly, and set it up as the nucleus of a matrix, augmented by automatically generated cost rows, right-hand-sides, or structural vectors or rows.

ABOUT THIS MANUAL

This manual is intended for use as an aid to the understanding of the principles involved and the methods employed in the GE-625/635 Linear Programming System. It includes an explanation of the fundamental characteristics of the system, and is not a reference manual. Discussions cover the Agenda Control Language, the Matrix Generator Language, the Format Generator Language, and input file formats.

This introductory chapter describes the use and usefulness of several prominent LP/600 capabilities. These and new features are outlined in the following LP/600 Features Summary.

LP/600 FEATURES SUMMARY

(See pg 10 in CPB 1401 Brochure)

System Control

- Compiled program
- Problem-oriented, English-like control language containing over 65 system control verbs.
- Complete programming logic including looping, subroutine linkage, arithmetic, and macro definition capability.

Matrix Generation

- Complete matrix generator language.
- Lists, strings, data tables, matrices, and submatrices definable at execute time.
- Special verbs for generating process-flow submatrices.

Format Generation

- Complete format generator language.
- Any BCD data format definable during problem execution.
- Definable arithmetic, dot products, and logical operations.

Problem Definition (pg 11 in Brochure)

- Up to 18-character, 3-part row and column names.
- Free or restricted variables (positive, negative, or zero level).
- Row and/or column scales.
- Upper and/or lower variable bounds.
- Matrix defination in variable-field format.
- Multiple Rhs and objective functions

- Separable, group, and decomposition variables defined explicitly without special name conventions.
- Bounded slack variables.
- Five row types.
- Seven column types.
- Element counts by row or column.

Solution Algorithms and Techniques

- Separate primal, dual, transportation, block decomposition, and group algorithms.
- User controlled multiple and/or partial pricing.
- Crashing capability to quickly reach feasibility.
- Slack variables generated automatically.
- Double-precision (72 bits) arithmetic.
- Automatic controls to prevent looping.

Dynamic Data Recycling

- Intermediate solution results.
- Work file modifications.
- Complete problem file revision capability, including linear forms, and the deletion or insertion of any part of the matrix.
- I/O compatibility with FORTRAN programs.

Post-optimal Operations

- Ranging of the Rhs, objective function, solution values or an individual matrix element.
- Parameterization of the Rhs, objective function, a matrix column, or a matrix row.
- Simultaneous parameterization of the Rhs and objective function.
- Row, column, and inverse tableau.
- Analysis of removing basic variables and introducing nonbasic variables.
- Fully automatic parameter stepping.
- User-controlled output frequency.
- User-controlled row and column selection for output.

Output

- Automatic output of restart information.
- Special tabulation format for graphing the progress of the solution-algorithms.
- Matrix picturing and tracing.
- User-controlled delimiting of rows and/or columns for which output is required.
- User-controlled frequency of output of intermediate solution values.

LP/600 MINIMUM CONFIGURATION

LP/600 requires the following equipment configuration:

- GE-625 or GE-635 Processor with 64k storage and console
- Magnetic tape controller with one tape unit
- Disc or drum storage unit
- Card reader and printer
- Card punch (if punched output is called for).

2. AGENDA CONTROL LANGUAGE

INTRODUCTION

Past linear programming systems were controlled mainly by a string of agendum call cards which were read into the system, interpreted, and executed sequentially and individually. Because only one call card was available to the system at any given time, little or no facility existed for modifying programming procedural changes based on internal events.

A distinctive aspect of LP/600 is that, through compilation, the entire solution procedure--in the form of an agenda control program--is available throughout the execution run. Thus, the LP/600 Agenda Control Language is a problem programming language that incorporates instruction labeling as well as FORTRAN-like control and arithmetic statements. (The fundamental language elements are shown in Figure 1.) The compiler produces, from control language statements, an object program that is executed by LP/600 executive control.

The system operates in a compile-and-go mode. A control language program always begins with a PREPRO statement, which initiates compilation (preprocessing), and ends with an EXECUTE statement which terminates preprocessing and immediately executes the assembled Agenda Control Language program.

Strings of matrix generator and/or format language statements may be included in the source agenda. These, however, are not assembled at preprocess time but are placed on the executive input device by the preprocessor for processing at execute time.

STATEMENT FORMAT

Control statement card format is as follows:

Card Columns	Field Description
1-6	Label
8-15	Verb
16-72	Parameter (variable)
73-80	Identification or sequence

Labels are required with certain verbs but may be used to identify any control statement. The parameter field may be blank or may contain a symbol, phrase, constant, operand, or expression, depending upon the verb used. Continuation of the parameter field to the next statement card is indicated if the rightmost nonblank character is a comma or left parenthesis. Comment cards, containing an asterisk (*) in column 1 and any information in columns 2-72, may be freely commingled with control statements.

CHARACTER SETS

Character Set Name	Character Set
Numeric	0 to 9
Alphabetic	A to Z
Alphanumeric	0 to 9 and A to Z
Punctuation	/, = ( ) :
Arithmetic	+ - * / ( )

CHARACTER STRINGS

String Name	Structure
Identifier	1 to 18 alphanumerics, divided into one to three parts, each part consisting of 0 to 6 alphanumerics and separated from the following part by a colon.
Symbol	Any number of characters from the GE character set.
Verb	1 alphabetic and 1 to 7 alphanumerics.

RELATIONAL OPERATORS

Operator	Meaning	Equivalent Mathematical Notation	Example Expression
.GT.	Greater than	>	SWTCH=ALPHA .GT. BETA
.LT.	Less than	<	SWTCH=ALPHA .LT. BETA
.EQ.	Equals	=	SWTCH=ALPHA .EQ. BETA
.LE.	Less than or equal	≤	SWTCH=ALPHA .LE. BETA
.GE.	Greater than or equal	≥	SWTCH=ALPHA .GE. BETA
.NE.	Not equal	≠	SWTCH=ALPHA .NE. BETA

Figure 1. Fundamental Control Language Elements

BOOLEAN OPERATORS

Operator	Meaning	Equivalent Logical Notation	Example Expression
.AND.	And	$A \cap B$	A .AND. B
.OR.	Or	$A \cup B$	A .OR. B
.XOR.	Exclusive Or	$(A \cap \bar{B}) \cup (\bar{A} \cap B)$	A .OR. B .XOR. A .AND. B
.ANOT.	And Not	$A \cap \bar{B}$	A .ANOT. B
.ONOT.	Or Not	$A \cup \bar{B}$	A .ONOT. B
.XNOT.	Exclusive Or Not	$(\bar{A} \cup B) \cap (A \cup \bar{B})$	A .OR. B .XNOT. A .AND. B

TERMINOLOGY

Term	Character String
Symbol	Name or label (1-6 characters)
Constant	Number or identifier
Operand	Symbol or constant
Expression	Alternating list of operands and operators

ARITHMETIC OPERATORS

Operator	Meaning	Example Expression
+	Add	SUM=ALPHA+BETA
-	Subtract	DIFF=ALPHA-BETA
/	Divide	DIV=ALPHA/BETA
*	Multiply	PROD=ALPHA*BETA

Figure 1. (cont'd)

Expressions

An extensive set of relational and Boolean operators, as well as the necessary verbs, is provided for generating arithmetic expressions such as those shown below.

```
                    .
                    .
RATIO COMPUTE           Q=(THETA- OTH)/(ITERNO- OIT)
      LOGIC             SW1= Q .LE. .01.AND. .GE. 50, OUT
                    .
                    .
```

FORTRAN hierarchy applies to all expressions that do not contain parentheses. [illegible]

Phase Structure

Phrases appearing in the parameter field are of the following general form:

left side = right side

They specify that the specification on the left side is to be set equal to the value of the right side. For example,

```
        .
        .
        .
CONVERT         SOURCE=SFILE, IDENT=SPROB
SETUP           SOURCE=SPROB
SET             RHS=AVAIL, OBJ=PROFIT
        .
        .
        .
```

The source of the convert data is a file named SFILE. Once the data is converted, it is assigned the name SPROB which then becomes the source of the SETUP verb. The current Right-hand-side is assigned the name AVAIL, and the current objective function is assigned the name PROFIT.

Permissible entries--symbols, labels, constants, etc.--in either phrase-part vary according to the verb used.

Separation

In addition to an equal sign (=), which separates the right and left sides of a phrase, two additional separator symbols may appear in the parameter field: a comma (,) and a slant mark (/).

The comma (,) means "and" where a sequence of phrases or parameters is defined. For example:

```
          .
          .
        RESET           ACTIVE, THETA, FIV, TCH, VERBSW
          .
          .
        SET             DIR1=50, DIR2=.1, PARMAX=100
          .
          .
```

The ACTIVE and THETA and FIV and TCH and VERBSW controls are all reset to their standard setting. In the second statement, DIR1 is set to the value 50 and DIR2 to .1 and PARMAX to 100.

The separator slant mark (/) means "on" or "through." For example:

```
          .
          .
        RNGRHS          RLIST=NAMES/EI
          .
          .
        RNGRHS          RLIMIT=ROW1/ROW5
          .
          .
```

The list of row names, whose right-hand-sides are to be ranged, is file NAMES and is located on file code EI. In the second statement, the right-hand-sides for rows ROW1 through ROW5, inclusive, will be ranged.

Verbs

Verbs of the language define the required processing action. The more than 65 verbs can be categorized functionally as:

Problem initialization verbs--I/O device control, data generation and conversion, setup and identification, problem saving and restoring, work file revision and modification.

Solution andpost-optimal verbs--Solution, inversion, crashing, parametric programming, ranging, tableau, forcing.

Solution control verbs--Arguments, parameters, frequencies, tolerances, programmed demands, and toggles (switches).

Output and problem display verbs--Standard solution prints, special solution prints, solution plotting, matrix picturing.

Agenda program control verbs--Sequence control, data definition, macro definition and call, expression generation.

PROBLEM INITIALIZATION

The control language includes verbs for generating and converting problem input data, setting up a problem work file in core memory, and performing other operations that are preliminary to problem solution. (See Figure 2.) Verbs which accomplish each step in the initialization process are designed to give the user comprehensive, flexible, and convenient means for coping with any problem situation.

I/O DEVICE CONTROL AND CORE DUMP

UNLOAD	Rewinds and unloads a specified tape or tapes.
BLANK	Erases the designated file.
DUMP	Dumps all core storage occupied by the LP/600 system.
LPDUMP	Dumps all LP/600 data regions.

MATRIX GENERATION AND CONVERSION

MATGEN	Creates a problem file from an input file or Matrix Generator Language statements.
CONVERT	Converts an input file to a problem file.
EDIT	Edits a SHARE standard linear program input file to an LP/600 matrix input file format for processing by CONVERT.

MATRIX SETUP AND IDENTIFICATION

SETUP	Creates a work file in core storage from a problem file, allocates devices and core storage for buffers and data regions, and initializes a communication region.
TITLE	Establishes the problem title to be printed on each output page.

Figure 2. Problem Initialization Verbs

SAVING AND RESTORING

SAVE	Writes a save file composed of the current solution information onto problem file for subsequent input by RESTORE.
PUNCH	Punches the current basis for subsequent input to LDBASIS.
RESTORE	Restores a designated save file.
LDBASIS	Restores a starting basis produced by PUNCH or prepared by the user.
DESAVE	Writes a file composed of the current solution information for decomposition problems, only, for subsequent input by DELOAD.
DELOAD	Restores a designated DESAVE file produced from DECOMP problems.

PROBLEM FILE CHANGES

REVISE	Revises a specified problem file according to revisions defined in the revise input file. The revised problem file remains written unchanged; the current work file is destroyed.

WORK FILE CHANGES

MODIFY	Modifies the work file according to changes defined in a modify file.
ABOUND	Flags an unbounded vector to prevent it from entering the basis.
AFEAS	Changes the Rhs to allow an infeasible problem to become feasible.
FLAGOUT	Flags specified vectors to prevent them from being used in the solution.
UNFLAG	Removes the FLAGOUT flags from all or specified vectors.

Figure 2. (cont'd)

I/O Device Handling

Input/output devices are defined by the user, via the appropriate GECOS control cards. A small set of reserved file names are available, and may be used as a guide for file definition. (See Figure 3.) Any two-character GECOS file designator is acceptable; the system will interrogate GECOS to determine the total list of files and establish its own requirements.

File Designator	Name	Contents
IN	Input File	BCD input files for processing by CONVERT, REVISE, MATGEN, CALC, MODIFY and RLIST type verbs.
AI	Alternate Input File	Same as IN.
PT or VP	Problem File	Binary results of input file processing by CONVERT, intermediate problem results produced by SAVE or DESAVE, lists and tables produced by MATGEN, and format generator language subroutines produced by DEFINE.
XP	Extra Problem File	Same as PT, but used for input only.
SO	GECOS Output	GECOS system output tape.
XO	Alternate Output File	Problem output
TB	Tabulate File	Output produced by RECORD for processing by TABULATE.
EI	Alternate Input File	Same as IN.
ED	Edit File	SHARE Standard Linear Program files for processing by EDIT.

Figure 3. LP/600 Files

Data Generation

The MATGEN verb compiles matrix generator language statements into a problem file for setup by SETUP. The compilation occurs during problem execution, not during the preprocessing phase; when the MATGEN verb is executed the file is generated.

Data Conversion and Setup

In LP/600, the process of converting BCD problem data to computational format is separated from the process of setting up the converted matrix in core memory for solution. This approach results in greater computational efficiency and programming flexibility, particularly in runs where several problems are to be converted. For example, any number of input files can be batched for conversion to problem files by consecutive CONVERT statements; the problem files are simply stored on PT. Subsequently, they may be retrieved randomly, set up in storage and solved during the same run or in subsequent runs.

Input Compatibility

The EDIT verb provides direct input compatibility with any linear programming system that adheres to the SHARE standard format. Moreover, the conversion programs activated by EDIT are specially designed for easy modification. This allows the LP/600 user to change the EDIT program so that it will accept other input formats similar to that of the SHARE standard.

Data Checking

CONVERT and other LP/600 routines contain extensive tests for illegal row, column, and element types, as well as other flaws in the problem data. The routines are designed to process the input file completely before initiating terminal action because of data error. Minor errors are documented and the problem is continued; major errors cause the run to be aborted if no alternative action is programmed.

Restart-Restore Mechanisms

The fundamental restart mechanism is provided by SETUP, since it establishes in core storage any data file processed previously by CONVERT or MATGEN. Additionally, several verbs are available for saving and restoring advanced problem information. Included is the usual SAVE-RESTORE or DESAVE-DELOAD linkage, as well as PUNCH-LDBASIS, which provides a facility for restoring a system-produced or used-prepared starting basis in punched-card format. LP/600 contains a number of built-in sequences for executing SAVE, DESAVE, and PUNCH whenever a problem must be aborted.

Problem File Changes

The utility of a linear programming system is based primarily on facilities available for carrying out dynamic changes to the problem matrix. Such operational sequences as setup-solution-output-change-solution-output, or simply setup-change-solution-output, often mean sizable savings in formulation and solution time.

REVISE makes virtually any change in the current problem file, including simple element changes, the addition or deletion of entire rows or columns, and the creation of linear combinations of rows and columns. It is useful in problem situations such as the correction of formulation errors and systematic expansion of a basic matrix to reflect several related problems, each of which must be solved independently.

In LP/600, however, REVISE has a more profound importance for the following reason: using the format generator language, REVISE data can be generated during the execution run. This means that work file data, current solution data, and new data can be analyzed, combined, processed arithmetically, and produced in REVISE file format for immediate use. This ability to "program" the REVISE file--particularly the possibilities for analyzing the recycling solution-date--opens an unlimited number of new approaches to matrix formulation and agenda preparation.

Work File Changes

Several verbs are provided for making temporary or exploratory changes to the work file. For instance, FLAGOUT and UNFLAG are particularly useful for solving the problem with and without certain specified vectors. These verbs do not change the content of the work file, but instead simply delineate vectors which can or cannot become active in the solution.

ABOUND, AFEAS, and MODIFY are normally used to overcome difficulties arising from formulation errors. ABOUND locates unbounded vectors and removes them from consideration. It thus allows a run to continue when an unbounded solution arises. AFEAS alters the Rhs to allow an infeasible problem to become feasible. It might, for example, be used in a subroutine to be executed in the event of infeasibility when the objective is to form a feasible right-hand-side for the sole purpose of reaching optimality. Once optimality is attained for the pseudo problem, PARRHS can be used to obtain a solution for the original problem.

Similarly, MODIFY is useful in providing a new Rhs when a no-feasible-solution condition arises. MODIFY makes minor revisions to the work file, such as changing the value of non-null element or the addition of a new Rhs and the creation of a new Rhs as a linear combination of vectors. Note also that data to be used by MODIFY can be generated during the execution run.

SOLUTION ALGORITHMS

The LP/600 solution algorithms described in Figure 4 are implementations of the latest linear program techniques, including recycle capabilities for handling many nonlinear conditions. Capable of solving problems of up to 4095 rows and specially designed to exploit the computational superiority of the GE-600 Series computers, the algorithms offer the LP/600 user unmatched solution power spanning a variety of problem types.

PRIMAL	Obtains an optimal solution of the standard Linear Program including separable programming formulations.
DUAL	Obtains an optimum dual solution of the standard Linear Program problem, using the dual algorithm.
CRASH	Generates a nontrivial starting basis for the PRIMAL algorithm.
TRANSP	Obtains an optimal solution of the transportation problem.
GROUP	Obtains an optimal solution of a special group problem, using a form of decomposition.
DECOMP	Obtains an optimal solution to the standard Linear Program expressed in decomposition format.
INVERT	Computes a new set of products from transformations for the current basis for all solution algorithms except DECOMP.
DCINV	Computes a new set of products from transformations for the current basis for DECOMP.

Figure 4. Solution Verbs

Multiple and Partial Pricing

User-controlled multiple and partial pricing are among many solution tecniques used in LP/600 to achieve utmost iterating efficiency. By means of parameters which are established with the SET verb, the user may specify the following:

1. The number of vectors, from one to five, to be selected simultaneously for pricing.
2. The number of improvements in the selection criterion per pricing operation. After the specified number of improvements and when the best vectors have been selected as candidates for entering the basis, the pricing cycle is terminated. The next pricing cycle resumes at this point.

Composite Modes

Although the PRIMAL algorithm ordinarily solves for a specified Rhs and objective function, it may also solve for a composite of either or both. In the composite modes, the effective Rhs or objective row is of the following form:

$$A_1 + PA_2 \quad (A_1 + P \times A_2)$$

A_1 and A_2 are columns or rows, and P is an established value of THETA (for composite Rhs) or PHI (for composite objective function).

The composite modes are highly useful in many problem situations, either directly with PRIMAL or indirectly with the parametric programming algorithms.

Primal Problems

The primal solution algorithm is based on the product form of the revised simplex method. PRIMAL is expressly constructed to resolve digital problems automatically and to obtain a valid optimal solution from any starting point and under any conditions of degeneracy, linear dependence, and initial infeasibility. Solution checking and inversion are performed automatically but may be controlled by the user with special solution controls.

Separable Programming

An advanced feature of the primal algorithm is its ability to solve formulations employing the separable programming technique. In formulations of this kind, linear approximations to nonconvex problems are represented by vector pairs within specified vector packets. The solution discipline is such that only one vector or two adjacent vectors in a packet may be in the basis at any one time.

Crashing Mode

The crashing mode, implemented by the CRASH verb, permits the user to impart an added degree of efficiency in the operation of the primal solution algorithm. The effect of crashing is to produce quickly a nontrivial starting basis and ultimately, to allow PRIMAL to reach a feasible solution in a much shorter period of time. The use of crashing techniques has been a prominent factor in the increased speed of many recent linear programming systems. In the LP/600 crashing algorithm, emphasis is primarily on sparseness and secondarily on improving feasibility and/or functional value--the approach that has proved best in reducing the number of matrix passes required to reach optimality.

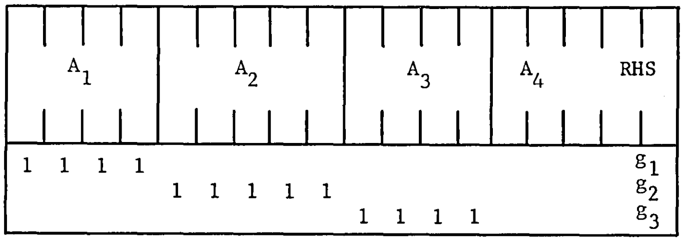
Transportation Problems

The classical transportation problem and related formulations are solved by a specially designed transportation algorithm activated by the TRANSP verb.

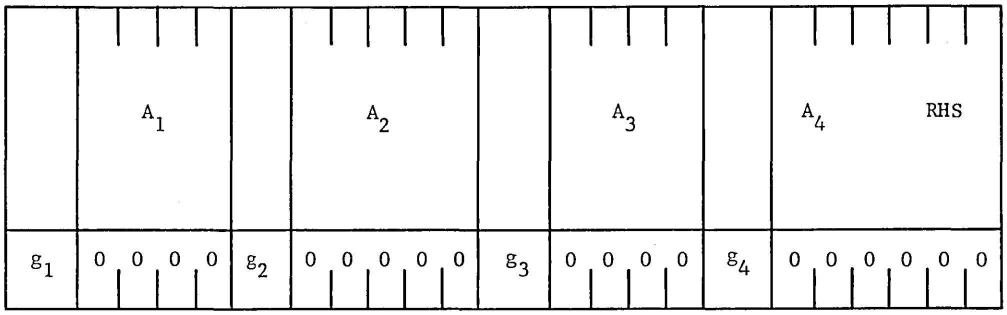
Group Problems

The group algorithm ranks with the primal algorithm in terms of solution power and efficiency. It is a recently developed form of decomposition applied to problems expressed as a general master problem and a special subproblem consisting of rows of 1's below groups of master problem vectors.

Problems requiring such formulations arise in many industrial environments and usually involve distribution and scheduling. The general form of the problem is as shown below.



The vector groups in the upper box constitute the master problem, and the rows in the lower box form the subproblem. In GROUP format, the formulation is as shown below, where g1, g2, and g3 are group values.



Notice that the subproblem is implied by group vectors having nulls in all rows except the group row, which contains the Rhs value for the sum of each group. A more realistic example of how the GROUP approach simplifies and condenses the matrix is given by the gasoline blending formulation in Figure 5.

Decomposition Algorithm

The concept of decomposition applied to linear program models is valid and well established, but it has not been implemented in general form in recent major linear program systems. With LP/600, however, DECOMP may be used to solve models formulated in general decomposition forms such as the one illustrated on the following page.

$$\begin{array}{cccccccc}
 & (C_0) & (C_1) & & (C_p) & & = & (Z) \\
(\pi_0) & A_0 & A_1 & - - - - - - & A_p & X^0 & = & B^0 \\
 & & & & & X^1 & = & B^1 \\
(\pi_1) & & D_1 & & & & & \vdots \\
\vdots & & & \ddots & & \vdots & & \\
 & & & & D_p & & = & B^p \\
(\pi_p) & & & & & & & \\
 & & & & & X^p & &
\end{array}$$

$$\sum_{p=0}^{p} C_p X^p = Z, \text{ minimize}$$

ST

$$\sum_{p=0}^{p} A_p X^p = B^0$$

$$D_p X^p = B^p \qquad p = 1, 2, \ldots, p$$

$$X^p \geq 0 \qquad p = 0, 1, 2, \ldots, p$$

$$\pi_0 A_0 + C_0 \geq 0 \qquad (\pi_p) \text{ is a row}$$

$$\pi_0 A_p + \pi_p D_p + C_p \geq 0 \qquad p = 1, 2, \ldots, p$$

DECOMP is a newly developed "block product form" algorithm which overcomes difficulties arising with other techniques in achieving efficient parametric computations and in interpreting problem answers. These improvements, plus the over-all potential of the decomposition technique, mean that problems not previously amenable to solution because of their size or nature can be solved routinely by LP/600.

need column

	GR1	CR1F	CR1D	CR2F	CR2D	PSLACK	GR2	CATVD	CATLC	CATHC	CSLACK	GR3	RSFUL	HCFUL	LCFUL	VDFUL	FSLACK	GR4	VN1R	VN2R	CNPR	RSLACK	ENDGRP	VNAPD	VDISD	LCCYD	VN1P	VN2P	CNPP		RHS
PROFT		3.0	3.0	3.1	3.1	0		.1	.15	.16	0		-2.5	-2.5	-2.5	-2.5	0		-4.5	-4.5	-4.5	0		-4	-4	-4	-4	-5	-5	=	-70
VNAP1		-.1																	1								1			=	0
VNAP2				-.2	-.25															1				1				1		=	0
VDIS		-.25	-.4	-.2	-.35		1									1									1					=	11
RESID		-.6	-.4	-.5	-.3			.1					.9																	=	0
CNTN								-.7	-.6	-.3					1						1								1	=	0
LCCY								-.3	1	-.7																1				=	-11
HCCY								-.5	-.5	1			.1	1																=	0
CRBAL		1	1	-.5	-.5																									=	0
CR2AV				1	1																									≦	75
DREQ																								1	1	1				≧	30
DSPEC																								-5	-1	10				≦	0
RSPEC																			0	1	-7									≦	0
PREQ																											1	1	1	≧	25
PSPEC																											4	5	-3	≦	0
GROUP	100						46					50						10					0								
SCALE								.625	.833	.909																					

Figure 5. ~~Refining~~ Refinery Example: Condensation

Inversion Algorithm

The inversion algorithm is vital to the efficiency of the entire LP/600 system and, indeed, to any linear programming system. Used by all solution algorithms for obtaining a new set of product form transformations, INVERT is designed for maximum computational speed and reliability. It contains procedures for automatically recovering from several types of computational errors, including detectable machine malfunctions.

INVERT is called and executed automatically by the solution algorithms but may also be initiated by a control statement. More often, however, the user controls inversion frequency by setting one or more of the applicable solution controls.

POST-OPTIMAL CAPABILITIES

Much of the real power of a linear program system and many benefits of its use originate from problem analyses made after a basic optimal solution is reached. Indeed, the answers needed most in many situations can come only by exploring side-cases of a solution.

By providing new ranging, parametric, and other algorithms in additon to those used previously, LP/600 offers post-optimal capabilities exceeding those of any other existing linear program system. (See Figure 6.) A major objective of these new developements is to provide tools for studying the influence on the solution of any part of the problem matrix, from a single element to several columns or rows.

Parametric Programming

PARRHS and PAROBJ provide the parametric tools that have been most used in the past. The other parametric verbs represent two extremes in computational power, both unique to LP/600. At one extreme, PARRIM parameterizes the right-hand-side and the objective function simultaneously. At the other, PARCOL or PARROW parameterizes only one column or row of the problem matrix; these algorithms, not previously available, have vast potential. Since the graph of the values of the parameterized functional is itself nonlinear, the maximum and/or minimum values may be obtained by means of an agenda control program subroutine, activated by a settable iteration frequency parameter.

Ranging

LP/600 combines the conventional ranging algorithms, RNGOBJ and RNGRHS, with two new algorithms of significant impact. RNGAIJ determines the range over which a specific matrix coefficient can be varied without requiring a basis change. It can be used to answer important questions about the sensitivity of technological coefficients. "What if" questions directed at a set of optimal variables are answered with the RNGSOL verb. This algorithm first computes the minimum rate of change in the functional as the level of a specified variable is changed, and then it computes the range of variation of the level without a change of basis.

PARAMETRIC PROGRAMMING

PARRHS	Parameterizes the right-hand-side.
PAROBJ	Parameterizes the objective function.
PARRIM	Simultaneously parameterizes the right-hand side and the objective function.
PARCOL	Parameterizes a matrix column.
PARROW	Parameterizes a matrix row.

RANGING

RNGRHS	Ranges the right-hand-side.
RNGOBJ	Ranges the objective function.
RNGAIJ	Ranges a matrix element.
RNGSOL	Ranges the levels of specified variables, minimizing the rates of change in the functional.

TABLEAU

COLOUT	Updates nonbasic structural vectors.
ROWOUT	Updates a specified row or part of a row.
INVOUT	Produces the explicit inverse of the current basis (updates nonbasic logical vectors).

FORCING

FORCE	Introduces into the basic nonbasic variables having a relative cost less than a specified tolerance.
REMOVE	Computes the effect of removing optimal variables having a value less than a specified tolerance.

Figure 6. Post-optimal Verbs

Updating Algorithms

The updating algorithms produce information for analyzing the effect on each basic variable of introducing a nonbasic variable into the solution. COLOUT is perhaps the most frequently used of the three verbs, since it provides this information for all nonbasic structural vectors. INVOUT produces the same information for nonbasic logical vectors. ROWOUT produces the COLOUT and INVOUT information arranged in row order. These three verbs, plus the DELIMIT verb, offer the user a unique capability for not only selecting vectors to be updated, but also for defining the format and amount of output produced.

Forcing Algorithms

The forcing algorithms are useful in exploring alternate optimal or near-optimal solutions. FORCE introduces into the solution each nonbasic variable whose relative cost is less than a specified tolerance. Feasibility is maintained, but optimality is not. Conversely, REMOVE determines the effect of removing basic variables whose activity levels are less than a specified tolerance. Optimality, or dual feasibility, is maintained; but primal feasibility is not.

SOLUTION CONTROLS

Through solution controls, the LP/600 user exercises personal control of computations performed by individual verbs. These controls fit the computation to the individual requirements of the problems to be solved. Verbs which pertain to the solution controls are explained in Figure 7. The controls themselves are explained in Figure 8. They are divided into the following classes:

Arguments
Parameters
Frequencies
Tolerances
Demands
Toggles

LP/600 contains standard, preset values for all controls in each class. Thus, to execise special control, the user must define the controls to be exercised with the required values. This is done by one or more phrases in a SET statement; each such phrase contains the name of the control followed by the special value as illustrated below.

```
SET          OBJ=PROFT, RHS=SPEC, VERBSW=ON, FCHK=75
```

Once set, the special values remain effective until they are changed by a subsequent SET statement or rescinded by a RESET statement. For example, the following RESET statement resets the verb print switch to OFF and resets all frequencies to their standard values:

```
RESET        VERBSW, FRQS
```

VERB	DESCRIPTION
SET	Sets required and optional parameters, tolerances, frequencies, toggles and demands.
RESET	Resets settable quantities to standard values.
STATUS	Lists all values settable by the SET verb and the values for each that are currently effective.

Figure 7. Solution Control Verbs

PARAMETERS

Name	Standard Value	Description
PARS	(group name)	All parameters.
THETA	0.0	Multiplier of CRHS, for PRIMAL, PARRHS, PAROBJ, etc.
PHI	0.0	Multiplier of COBJ, for PRIMAL, PARRHS, PAROBJ, etc.
PSI	0.0	Multiplier for PARRIM, PARCOL, PARROW.
PARMAX	∞	Maximum value for THETA, PHI, or PSI in parametric algorithms.
SCALE	1.0	Scale factor to control sense (and degree) of optimization.
SSCALE	0.0	Scale factor for composite reduction of primal and dual infeasibilities.
EPSLN	.00001	Value used to perturb RHS in the event of digital cycling during infeasibility.
MVDI	5	Maximum number of vector drops in INVERT.
MSHSZ	1	Mesh size for separable programming packets.
NVMP	5	Maximum number of vectors for multiple pricing.
NIPP	5	Number of improvements required for partial pricing.

FREQUENCIES

Name	Standard Value	Description
FRQS	(group name)	All frequencies.
FIV	75	Frequency of inversion.
FTY	∞	Frequency of typed log when ANPSW=ON.
FIR1	∞	Frequency of iteration interrupt 1.
DIR1	∞	Delta value for delta, interrupt 1.
FIR2	∞	Frequency of iteration interrupt 2.
DIR2	∞	Delta value for delta, interrupt 2.
FREC	∞	Frequency of RECORD.
DREC	∞	Delta parameter for RECORD.
FCHK	75	Frequency of solution check.

[illegible]: FCIR ∞ [illegible] (cycle [illegible])

TOLERANCES

Name	Standard Value	Description
TOLS	(group name)	All tolerances.
TMC	$1. \times 10^{-5}$	Small pivot modified choice in INVERT.
TRJ	$1. \times 10^{-5}$	Small pivot vector reject in PRIMAL.
TZE	$1. \times 10^{-8}$	Primal or dual infeasibility criterion.
TPV	$1. \times 10^{-10}$	Smallest denomination for ratios.
TCH	$1. \times 10^{+10}$	Maximum absolute characteristic difference after linear form (round-off noise).
TABS	$1. \times 10^{-12}$	Absolute zero for packing.
TERR	$1. \times 10^{-5}$	Solution check tolerance; larger error gives CHK demand.
TFLT	$1. \times 10^{-8}$	Smaller values are output in floating point if greater than TZE.
TFR	$1. \times 10^{-8}$	Relative cost tolerance (dj) for FORCE and the value tolerance for REMOVE.
TIV	$1. \times 10^{-8}$	Smallest pivot iterating verbs.
TCK	$1. \times 10^{-7}$	Primal or dual check error report criterion.
TMX	$1. \times 10^{+5}$	Largest magnitude acceptable as input.
TMN	$1. \times 10^{-5}$	Smallest nonzero magnitude acceptable as input.

Figure 8. Solution Controls

ARGUMENTS

Name	Standard Value	Description
ARGS	(group name)	All arguments.
OBJ	blank	Name of the current objective function (OBJ).
COBJ	blank	Name of the change cost row to be used for PAROBJ or the composite OBJ mode.
RHS	blank	Name of current right-hand-side (RHS).
CRHS	blank	Name of the change RHS column for PARRHS or the composite RHS mode.
BCOL	blank	Name of the base column for PARCOL.
CCOL	blank	Name of the change column for PARCOL.
BROW	blank	Name of the base row for PARROW.
CROW	blank	Name of the change row for PARROW.
ACTIVE	blank/blank	Names of the first and last vectors in in the set to be used in iterating verb.

Figure 8. (cont'd)

TOGGLES

Name	Standard Value	Description
TOGS	(Toggle group name)	All toggles
CRHSW	OFF	Composite RHS switch
new → CCLSW	OFF	Composite column switch for PARROW
COBSW	OFF	Composite objective function switch
[illegible] → EPSSW	OFF	Epsilon switch for RHS loading
PKTSW	OFF	Packet switch for separable programming mode
new → CRWSW	OFF	Composite row switch for PARCOL
ANPSW	OFF	Analyst present switch
VERBSW	OFF	Verb print switch
RIMSW	OFF	Composite RHS and OBJ switch for PARRIM
DUMPSW	OFF	Dump on abort switch
DETAIL	OFF	Eliminate detailed printouts (infeasible solution dumps, etc.) switch

Figure 8. (cont'd)

DEMANDS

Name	Standard Value	Description
DEMS	(group name)	All demands
NFS	TERM	No feasible solution
UNB	TERM	Unbounded solution
PREMAX	TERM	Premature maximum on parameter
NOMAX	TERM	No maximum for parameter
NARGM	ABJOB	Necessary argument missing
SUNK	ENDLP	Source file unknown
MIER	ENDLP	Major input error
PIER	ENDLP	Possible input error
SING	TERM	Number of vectors dropped during INVERT exceeds MVDI parameter value
CHK	TERM	Error greater than tolerance during check
RF1	NEXT	Reached frequency of interrupt 1
RF2	NEXT	Reached frequency of interrupt 2
RD1	NEXT	Reached delta interrupt 1
RD2	NEXT	Reached delta interrupt 2
NOFILE	NEDLP	Work file not SETUP
PROER	ENDLP	Agenda program procedure error
DOINV	DINV	Demand for inversion
DOPRI	DPRI	Demand for PRIMAL-CURRENT
DOPRN	DPRN	Demand for PRIMAL-NEXT (initiated automatically by DUAL)
NOMO	ENDLP	Insufficient disc storage assigned
DCINV	DDINV	Demand for DECOMP inversion

Figure 8. (cont'd)

Required Controls

Generally, solution controls have two functions. First, they define values that are required for the execution of certain verbs. Second, they define values that are not required by the system but may be set at the option of the user to meet particular needs.

The most prominent of the required controls are the arguments OBJ and Rhs, which define the objective function and Rhs to be used by the solution algorithms, post-optimal algorithms, and certain other verbs. In problems involving parametric algorithms, other controls must be set to define the change row or column, and, optionally, the maximum parameter value and a starting parameter value. For example, if a PRIMAL-PAROBJ-PARRHS sequence is programmed, the SET statement must contain minimally the following phrases:

```
SET        OBJ=row name,RHS=column name,CRHS=column name,
           COBJ=row name
```

Optional Controls

Any number of optional control phrases may be added to SET phrases which establish required controls. Among the optional controls are frequencies, demands, tolerances, and toggles. Unlike the required controls, these contain standard values, that is, the related verbs would operate normally without a special value. They would, therefore, be set only to meet a specific processing requirement.

Demand Actions

The LP/600 demands are one of the powerful features of the control language. They allow the user to define the procedure to correct, compensate for, or bypass any of a number of undesirable conditions which may arise during execution. They are, in fact, conditional solution controls that become effective only under certain circumstances.

As is shown by Figure 8, each demand has a standard demand action that is taken if the demand condition arises. For example, in the event of an unbounded solution (UNB demand), LP/600 would normally execute the action TERM. The significance of TERM and all other demand actions is defined in Figure 9.

For each demand, the user may define special actions to be taken in preference to the standard actions. This is normally done by phrases in the SET statement. For example, to change the action for UNB to NEXT, the following SET phrase would be used:

```
SET     UNB=NEXT
```

Using this approach, any demand action shown in Figure 9 may be defined for any of the demands. However, when the demand actions are not suitable, the user may program subroutines for execution, if any demand condition arises.

Sequence Name	Verb Sequence
TERM	OUTPUT
	SAVE
	PUNCH
	ENDLP
ABJOB	SAVE
	PUNCH
	ENDLP
ENDLP	ENDLP
DUMPN	DUMP
	NEXT
NEXT	NEXT
LPDUMP	LPDUMP
	JUMP
DINV	INVERT
	CURRENT
CURNT	CURRENT
DPRI	PRIMAL
	CURRENT
DPRN	PRIMAL
JUMP	JUMP
DDINV	DCINV
	CURRENT

[illegible] solution [illegible] [illegible])

"[illegible]"

[illegible] (solution [illegible]) Core dump [illegible] next [illegible] level [illegible]

[illegible] main program [illegible]

Set next statement in next higher level [illegible]

Set next statement in main program

Kill [illegible] Dump/[illegible]

Figure 9. Demand Sequences

The name of each demand is actually an LP/600 communication region location containing the associated demand action. Thus, by simply setting the contents of the communication region location to an instruction label in the agenda program, the user may program any special computation to be carried out if the demand condition arises.

An instruction can also be set by the SET verb. For example, if a subroutine beginning with a statement labeled CHNG is to be executed in the event of a no-feasible-solution (NFS) situation, the SET statement would read:

```
          .
          .
          SET           NFS=CHNG
          .
          .
     CHNG TRACE
          OUTPUT
          ENDLP
```

By either of these two methods, or combinations of both, the user is given almost unlimited capabilities for coping with untoward situations which frequently occur during problem execution.

PROBLEM OUTPUT

The LP/600 user may obtain problem output of several types in an almost unlimited number of formats for meeting a variety of needs. Gross problem answers may be obtained in standard formats or in any format specified by the user in the format generator language. In addition, verbs for limiting output quantity, producing solution plotting information, and displaying the original problem matrix provide outstanding analytical tools, several of which are unique to LP/600. The problem output verbs are explained in Figure 10.

Standard Solution Prints

Standard solution prints produced by OUTPUT show problem answers in compact and easily intelligible formats. The output formats for TRANSP,GROUP, and other solution algorithms are designed to emphasize the most significant results of the problem type. In all solution prints, structural and logical variables are clearly distinguished; symbols show whether a bounded variable is in the solution at upper bound, lower bound, or an intermediate level. By including the PDSOL phrase in the OUTPUT statement, the user may suppress output for all nonbasic variables except those at bound.

Special Report Formats

Facilities for generating special output reports are indispensable in any modern linear programming system. In LP/600 they are provided by DEFINE and CALC. The format generator language provides a means for processing and formatting solution results and other data to meet any reporting need. The format generator language statements are compiled by DEFINE into a report skeleton, which is stored on a mediary device. The skeleton may then be accessed and used repeatedly by CALC in formatting and printing the report.

SOLUTION PRINTS

OUTPUT	Processes problem answers produced by the solution and parametric verbs.
DEFINE	Compiles a format skeleton from Format Generator Language statements for later use by the CALC verbs.
CALC	Produces output in the format defined by a format skeleton compiled by DEFINE.
DELIMIT	Defines a subset of the work file rows and/or columns for which output is produced by OUTPUT, PICTURE, TRACE, and certain postoptimal verbs.

SOLUTION PLOTTING

RECORD	Writes solution values for up to 40 primal and/or dual variables for formatting by TABULATE. The frequency of output for the RECORD operation is controlled by the solution controls DREC and FREC.
TABULATE	Reformats the RECORD data in a format suitable for plotting changes in solution values over a number of solutions.

PICTURE OPERATIONS

PICTURE	Produces a picture of the original work file matrix, or a subset of the matrix defined by DELIMIT. The output appears in usual matrix format including row and column names.
TRACE	Produces in TABULATE format one of the following: (1) all vectors in the DELIMIT subset plus vectors with first-order interaction with the subset, (2) if no DELIMIT has been specified, all vectors in the basis with infeasible values, plus vectors with first-order interaction with the infeasible subset.

Figure 10. Output and Problem Display Verbs

Solution Plotting Data

The RECORD and TABULATE verbs give the LP/600 user a unique facility for obtaining information for plotting a series of solutions, such as those produced during parametric programming operations. At each basis change, RECORD writes the solution values for up to 40 primal and/or dual variables into the LP/600 communication region locations. By exercising the DREC and FREC controls, the user may cause these values to be produced periodically as output to a mediary device for subsequent processing by TABULATE. At the end of the problem, TABULATE lists the names of the variables on the left side of the print page and prepares each recorded solution as a column, with up to eight solutions per page. Thus, the activity of any variable can be plotted over any number of solutions. And, if solution plotting is not an objective, the TABULATE verb is often a highly useful means of obtaining a concise display of the solutions.

Solution Data Recycling

In addition to providing output for TABULATE, RECORD serves another highly significant function. Since RECORD stores the current solution in the CR (Communication Region), solution values are constantly available for use in MOVE, COMPUTE, and LOGIC statements in agenda program subroutines. This means that the user may control the course of the agenda according to solution values obtained at any point in the run.

Work File Displays

Displays in matrix format of the current work file are produced by TRACE and PICTURE. Each output page from PICTURE contains the names of 20 to 40 rows and 50 columns, with symbols denoting the sign and magnitude of all nonzero coefficients. The output from TRACE is in TABULATE format, giving actual values.

PICTURE is often an invaluable aid in checking the over-all accuracy of new formulations. TRACE, on the other hand, is designed to aid in determining the sources of known errors, particularly infeasibilities. Depending on the use of the DELIMIT verb, TRACE will produce as output (1) all vectors in the current problem basis with infeasible values or (2) all vectors in a subset defined by DELIMIT plus vectors with first-order interaction with the subset. In both cases, additional vectors having nonzero elements in a row in which a vector in the subset also have a nonzero element are produced as output.

Delimiting Problem Output

DELIMIT may be used to restrict output produced by OUTPUT, FORCE, TRACE, and the tableau algorithms to a defined subset of vectors. The user may select any one of the methods below for delimiting the subset:

1. The names of the beginning and ending vectors may be specified when the subset comprises contiguous vectors.
2. Each vector may be named individually in a vector names list.
3. Vector name masks may be supplied when all vectors in a class or classes are to be selected.

AGENDA CONTROL

The extensive repertoire of agenda control verbs injects true programming logic into agenda preparation and is an outstanding and unique feature of LP/600. Included are verbs for sequence control, data definition and manipulation, macro-definition, and arithmetic and logical operations. (See Figure 11.) All are based upon the best known problem-oriented programming languages and have been expressly designed to fit the needs of the linear programming user.

ARITHMETIC AND LOGICAL OPERATIONS

COMPUTE	Computes the value of an arithmetic expression.
LOGIC	Computes the value of a logical expression.

DATA DEFINITION AND MANIPULATION

DC	Defines one or more numeric constants.
DSW	Defines one or more logical switches.
MOVE	Moves the contents of one or more locations in the ACL program to other locations.
NOTE	Contains program comments, which are printed when the statement is executed.
DISPLAY	Prints the contents of specified Agenda Control Language (ACL) defined constants/switches, communication region locations.

SEQUENCE CONTROL

GOTO	Branches to a specified Agenda Control Language (ACL) statement.
PERFORM	Branches to an ACL program subroutine.
CURRENT	Returns from a subroutine entered via a settable demand to the statement that originated the demand.
NEXT	Returns from a subroutine entered via PERFORM or a settable demand, to the next sequential statement following PERFORM or the statement that originated the demand.
TALLY	Tests the values of a program counter and either executes the next sequential instruction, or decrements the counter and branches to a specified instruction.
ENDLP	Processes the remaining problem output, pauses for tape dismounting, and returns control to GECOS.
JUMP	Jumps from an embedded subroutine to the statement following the first PERFORM or any other verb which executed a settable demand.

Figure 11. Agenda Control Language Control Verbs

MACRO DEFINITION AND CALL

MACRO	Initiates the definition of a macro skeleton which may be called upon later by using the name in the variable field as a verb.
ENDM	Terminates the macro skeleton. All macros must be defined prior to their use.
Macro name	The name in the verb field is the name of a previously defined macro. All macros defined in the same program must be defined before being used--that is, earlier in the card sequence. Labels may be used as required.

Figure 11. (Cont'd)

Macro-Definitions

The facility for defining macro-operations has many applications in the linear programming environment. An example is shown in Figure 12, where a problem is to be converted and solved by the statement sequence CONVERT, SETUP, SET, PRIMAL, and OUTPUT.

The Agenda Control Language program is considerably shortened because of the macro-operations and is, therefore, simpler and less time-consuming to prepare. These are, of course, the major benefits of macro-operations; in large and complex Agenda Control Language programs, such benefits become increasingly important.

LOCATION 1	2 6	E/O 7	OPERATION 8 14	15	ADDRESS, MODIFIER 16	COMMENTS 32
			PREPRO			
			MACRO		SOLVE	
			CONVERT		SOURCE#1/IN,IDENT=#1	
			SETUP		SOURCE=#1	
			SET		RHS=#2,OBJ=#3	
			PRIMAL			
			OUTPUT			
			ENDM			
			TITLE		SAMPLE 1	
			SOLVE		SFILE,AVAIL,PROFIT	
			TITLE		SAMPLE 2	
			SOLVE		SFILE 2,AVAIL,PROFIT	
			ENDLP			
			EXECUTE			

Figure 12. Agenda Control Language Program - Sample 1

Logical Operations

Other verbs in the program-control set provide for subroutine linkage, looping, conditional and unconditional transfers, Boolean functions, and other logical operations. The sample Agenda Control Language program in Figure 13 includes examples of how most of the logical verbs are used.

As defined in the program shown in Figure 13.

1. To convert and solve the problem.
2. To output results from a parametric Rhs operation each time THETA reaches 0.1.

```
LOCATION E OPERATION    ADDRESS, MODIFIER               COMMENTS
         O
1 2    6 7 8       14 15 16              32
           PREPRO
           TITLE         SAMPLE PROBLEM 6
           CONVERT       SOURCE=SAM6/IN,IDENT=SAM6
           SETUP         SOURCE=SAM6
           SET           RHS=AVAIL,OBJ=PROFIT
           PRIMAL
           COMPUTE       ITERNO=0
           PERFORM       INIT
           OUTPUT
           SET           CRHS=CHANGE,DIR1=.1,RD1=R,PARMAX=1.0
           PARRHS
           SET           THETA=1.1,CRHS=CNG2
           COMPUTE       ITERNO=0
           PERFORM       INIT
           PARRHS
           ENDLP
  R        COMPUTE       Q=(THETA-OTH)/(ITERNO-OIT)
           LOGIC         SW1=Q.LE..01.AND.ITERNO.GE.50,OUT
           MOVE          OTH,OTH1,1
           MOVE          OIT,OIT1,1
           MOVE          OTH2,THETA,1
           MOVE          OIT2,ITERNO,1
```

Figure 13. Agenda Control Language Program - Sample 2

LOCATION	E/O	OPERATION		ADDRESS, MODIFIER	COMMENTS
1 2 6	7	8 14	15	16	32
		OUTPUT		PDSOL	
		CURRENT			
OUT		OUTPUT			
		NEXT			
INIT		MOVE		OTH1, THETA, 1	
		MOVE		OTH2, THETA, 1	
		MOVE		OTH, THETA, 1	
		MOVE		OIT, ITERNO, 1	
		MOVE		OIT1, ITERNO, 1	
		MOVE		OIT2, ITERNO, 1	
		NEXT			
OTH		DC		0	
OTH1		DC		0	
OTH2		DC		0	
OIT		DC		0	
OIT1		DC		0	
OIT2		DC		0	
Q		DC		0	
SW1		DSW		OFF	
		EXECUTE			

Figure 13. (cont'd)

3. MATRIX GENERATOR LANGUAGE

INTRODUCTION

The Matrix Generator is an integral part of the LP/600 system for generating linear program matrices from other types of arrays and data which are selectively combined via a program of Matrix Generator Language statements (MGL program). This program exists initially as an input file that is read and processed by a MATGEN statement in the Agenda Control Language (ACL) program. The other arrays from which matrices are constructed are called lists, strings, and tables. An MGL program can also produce compound lists for use by a subsequent Format Generator Language (FGL) program.

The Matrix Generator Language contains more than forty verbs, flexible variable field syntax, and a number of modifiers and pre-empted identifying words. Words which are common to MGL, FGL, and ACL have the same meaning in all three languages.

MATRIX GENERATOR LANGUAGE CONCEPTS AND DEFINITIONS

The Matrix Generator Language combines a number of computing and data processing facilities. Although its major purpose is to produce linear program matrices from data arrays, it is quite possible to use it for computing tables which can be processed for output by DEFINE and CALC. The concept of a rectangular table is central to MGL.

Arrays

An array is a general term used to describe a table, list, or string. Tables are defined as rectangular arrays of numbers (elements) arranged in rows and columns. All entries are assumed to be non-null: the absence of an entry is equivalent to a value of zero. The columns are named by a zero-th line of symbols called a "heading." The rows are named by a zero-th column of symbols called a "stub." The "zero, zero" entry is the name of the table. For example, a multiplication table up to 4 x 4 and having the name MULT would be defined as follows:

```
TABLE    MULT =     COL1,     COL2,     COL3,     COL4
         LINE1 =       1,        2,        3,        4
         LINE2 =       2,        4,        6,        8
         LINE3 =       3,        6,        9,       12
         LINE4 =       4,        8,       12,       16
```

Either the stub or the heading of a table constitutes a (simple) list. Lists may exist separately and are the simplest form of array handled by MGL. The elements of a list are one-part identifiers, up to 6 characters in length.

The third type of array used with MGL is called a string. A string is used for more efficient specification and recording of a sparse array. It is a list of triplets of the form:

row ID: column ID = value.

Only triplets whose value is non-null need be specified. All identifiers are single part.

A fourth type of array, called a compound list, may be generated and output for later use in DEFINE and CALC. A compound list is a list of three-part identifiers such as those used throughout LP/600 for matrix row and column names. No other use is made of compound lists in MGL.

Arrays may be input directly as part of the MGL program, or remotely, or they may be constructed by the MGL program from other arrays of the same or different types.

Scalar Quantities

In MGL, scalar quantities are defined as non-array data. These are of three types:

1. constants, either fixed- or floating-point numbers
2. switches, Boolean quantities (0 or 1 values)
3. alphanumeric character strings

They may be input directly as part of the MGL program or computed by the MGL program. There are no remote scalars in MGL.

Linear Program Matrix

The final linear program matrix that is to be solved consists of rows, columns, and right-hand-sides. Each of these constituents is identified by a three-part identifier (of which one or two parts may be void).

Much of the real power of MGL is derived from its ability to produce entire submatrices from single statements which operate on tables or strings (and lists, indirectly). The language is also designed to provide simplicity in matrix definition. In particular, individual matrix elements may be defined on a one-to-one basis, and the type and order of logical and structural variables may be specified. The general scheme in producing a matrix is as follows:

- Input and/or compute tables and strings from other tables and strings, lists, and scalars.
- Form submatrices from tables and strings.
- Fill in special matrix constituents with direct matrix definition statements.

STATEMENT FORMAT AND VERB USAGE

The format of MGL statements is the same as the Agenda Control Language statement format. Each verb and its usage is described briefly in Figure 14 and in the following paragraphs.

Classification of Verbs

It is convenient and useful to separate the MGL verbs into seven classes, as follows:

1. Control Verbs - these eight verbs provide for program control, macro definition, and final exit back to the ACL program.

2. Direct Data - these six verbs introduce the definition of scalars and arrays as a part of the MGL program. These verbs are executable, not mere static pseudo-operations.

3. Scalar Operations - these five verbs produce scalar results either from other scalars or from arrays. Powerful indexing arrangements are provided for accessing table elements. One of these verbs (LOGIC) is also a control verb, taking the role of IF.

4. Array Transformations - these seven verbs modify arrays and transform one type of array to another.

5. Micro-operations - these two verbs, COMPOSE and FUNCT, provide microprogramming capability for dealing with lists and strings. Together with their numerous modifiers, they virtually amount to a sublanguage of MGL. By defining macros involving these verbs, the user may create special operations of the most diverse kinds.

6. Matrix Construction - these ten verbs form three subclasses:

    - Five verbs for direct specification of matrix constituents.

    - Four verbs for constructing submatrices from tables according to pre-defined expansions of general applicability.

    - One verb for direct insertion of a table or string as a submatrix.

7. Remote Data - two of these three verbs output and read arrays. The third is for constructing and outputting a compound list.

PROGRAM CONTROL

PERFORM	Branches to an MGL program subroutine.
NEXT	Returns from an MGL program subroutine to the statement following the last PERFORM executed.
GOTO	Branches to a specified MGL statement.
TALLY	Controls repetitive execution of a series of MGL statements (looping control).
MACRO	Introduces and identifies a macro prototype definition.
ENDM	Terminates a macro prototype definition.
EXIT	Exits from the MGL program to the statement following MATGEN in the ACL program.
DUMP	Outputs lists, strings, tables, and SETCON, SETSW, and NAME arrays as a debugging aid.

DIRECT DATA

SETCON	Defines the value of one or more numeric constants.
SETSW	Defines the value of one or more logical variables (switches).
NAME	Defines an n-character string of alphanumeric data.
LIST	Introduces the first card of a list array.
STRING	Introduces the first card of a string array.
TABLE	Introduces the first card of a table array.
NOTE:	Array names are given on the first card. The array is terminated by the next verb.

SCALAR OPERATIONS

DIMEN	Obtains the extent (number of entries) of a column or row of a table, a list, or a string.
INDEX	Places the index of a table row or column in a specified MGL program location.
MOVE	Moves data between MGL program locations.
COMPUTE	Computes the valne of an arithmetic expression. (See section on indexing Modes for Array Elements.)
LOGIC	Computes the value of a logical or relational expression, and either branches to a specified statement or executes the next sequential statement. (See section on Indexing Modes for Array Elements.)

Figure 14. Matrix Generator Language Statement Classes

ARRAY TRANSFORMATIONS

ALTER	Replaces an element of a list, or appends to it a specified (indexed) element from another list.
DECLARE	Uses two lists to form the stub and heading of a new table.
RENAME	Replaces the heading or stub of a table with a list.
EXPAND	Reformats a string (or its transpose) into a table.
EXTRACT	Creates a list from the heading or stub of a table or from the row or column names in a string.
ENPAC	Reformats a table (or its transpose) into a string.
ERASE	Releases storage occupied by an array. (Does not affect an array which has been ENFILE'd.)

MICRO-OPERATIONS

COMPOSE	The defined list operation is performed on two argument lists with corresponding character masks, and a result list with character mask. Character matching and manipulation is permitted with detailed control of the resulting specification.
FUNCT	The defined string operation is executed with two argument strings and corresponding character masks, and result string with character mask. Character matching and manipulation is permitted, together with an arithmetic operation and detailed control of the resulting specification including a second arithmetic operation.

MATRIX CONSTRUCTION

Row and Column Generation	
ROW	Defines coefficients for a specifiec row.
LGL	Defines a logical variable, and therefore a row; also, its type and scale, if any.
STR	Defines a structural variable, and thus a column; also, its type, scale, and translation, if any.
VECT	Defines coefficients for a specific column.
RHS	Defines coefficients for a right side.

Figure 14. (cont'd)

MATRIX CONSTRUCTION (continued)

Submatrix Generation	
YIELD	Builds a submatrix from a table whose heading specifies input streams and whose stub indicates output streams. The table elements are the proportions of output per unit of input.
QUALS	Builds a submatrix from a table of quality coefficients. The heading specifies the input components, and the stub indicates qualities.
SYNTH	Builds a submatrix from a table whose heading specifies output streams and whose stub indicates input streams. The elements are the proportions of input per units of output.
SUBMAT	Builds a submatrix from a table or string.
TRANSP	Builds a submatrix from a table of transportation costs. The heading specifies the sources; the stub specifies the destination.

REMOTE DATA

READ	1. Reads a card image file of one or more lists, strings, or tables from its associated unit (EI, AI, IN). 2. Reads a packed file of string or table data from its associated unit (PT, XP).
ENFILE	Outputs a packed file of string or table data onto the PT for later access by READ.
JOIN	Creates a compound list from three lists and enfiles it onto the PT for later use by DEFINE

Figure 14. (cont'd.)

SUBMATRIX GENERATION

Submatrices are generated from tables by the verbs QUALS, YIELD, SYNTH, SUBMAT, and TRANSP. The verbs produce a submatrix containing the following:

1. Row and column names that are concatenated from names in the tables.
2. The table values as submatrix elements.
3. Unit elements as required.

The functions of each verb are best shown by defining a table and then showing a statement using each verb and the resultant submatrix. A sample table is shown below and the submatrices generated by various statements are described in the following discussion.

```
TABLE      A =    C1,         C2,        C3
          R1 =    V11,        V12,       V13
          R2 =    V21,        V22,       V23
```

YIELD Statement: YIELD :N2:N3=A

YIELD Submatrix:

	C1:N2:N3	C2:N2:N3	C3:N2:N3
C1::	1		
C2::		1	
C3::			1
R1::	-V11	-V12	-V13
R2::	-V21	-V22	-V23

QUALS Statement: QUALS :N2:N3=A

QUALS Submatrix:

	C1:N2:N3	C2:N2:N3	C3:N2:N3
C1::	1		
C2::		1	
C3::			1
R1:N2:N3	V11	V12	V13
R2:N2:N3	V21	V22	V23

SYNTH Statement: SYNTH N1:N2:=A

SYNTH Submatrix:

	N1:N2:C1	N1:N2:C2	N1:N2:C3
C1::	-1		
C2::		-1	
C3::			-1
R1::	V11	V12	V13
R2::	V21	V22	V23

SUBMAT Statement: SUBMAT NR1:NR2:$1,NC1:$2:NC2 = <A,$1,$2>

SUBMAT Submatrix:

	NC1:C1:NC2	NC1:C2:NC2	NC1:C3:NC2
NR1:NR2:R1	V11	V12	V13
NR1:NR2:R2	V21	V22	V23

TRANSP Statement: TRANSP ::N3=A(COST)

TRANSP Submatrix:

	C1:R1:N3	C1:R2:N3	C2:R1:N3	C2:R2:N3	C3:R1:N3	C3:R2:N3
C1::	1	1				
C2::			1	1		
C3::					1	1
R1::	-1		-1		-1	
R2::		-1		-1		-1
COST	V11	V21	V12	V22	V13	V23

MATRIX GENERATOR LANGUAGE INDEXING

Reference to elements of a table is made by COMPUTE and LOGIC statements. Very extensive indexing capability is provided for this purpose.

Single Element References

Individual table element reference makes use of the following general syntax:

<T,R,C>

where T is the table name,

R is the row identifier (if non-numeric) or the row index,

and C is the column identifier (if non-numeric) or the column index.

The option for specifying an index in lieu of an identifier carries through nearly all modes and is referred to as the index option.

The above references are direct. References may also be indirect, with or without the index option. Indirect reference means that a storage cell (SETCON or NAME) in the MGL program is specified in place of R or C. This is indicated by an asterisk prefixing the cell label. Contents of the cell can be either:

1. A non-numeric row or column identifier, or
2. A numeric index.

For example, the reference

<T,*X1,C>

refers to the element in column C of table T whose row is specified in cell X1.

References may also be remote and remote indirect. A list is used for this purpose and the syntax is nested. For example:

<T1, <L1,3>,C>

refers to the element in column C of table T1 whose row identifier is the third entry in list L1. Note that the index option is mandatory here. The construction

<T1, R, <L1,*A>>

is remote indirect and refers to the element in row R of table T1 whose column identifier (not index) is found as the n-th entry of list L1 where the index n is stored in cell A.

Multiple Element References

For COMPUTE (but not for LOGIC), a single reference construction may imply use of several or even all elements of a table. There are two forms of multiple element references: name-matching and automatic indexing. Either may be direct or remote (but not indirect). The index option does not apply.

In the <T,RC> syntax, the R and C symbols are replaced with dummy symbols of the form "=n" for name matching and "$n" for automatic indexing, where n is an integer. The symbolic integer n is used to indicate parallel substitution for equal values of n; unequal n-values give nested substitution with the smallest n-value being the outermost loop. For example, the statement

COMPUTE <T1,R1, =1> = <T2,R2, =1>

means replace all those elements in row R1 of table T1 with elements from row R2 of table T2 wherever column identifiers match in both tables. If it is desired to move all elements, regardless of identifiers, then the statement would be

COMPUTE <T1,R1,$1> = <T2,R,$1>

If T1 and T2 do not have the same number of columns, then the number of elements moved will be the smaller column extent of the two.

Remote multiple references are achieved by substituting "=n" or "$n" for the index of a list. Both forms have the same effect insofar as the list is concerned but one or the other is used depending on the intent of the rest of the statement. For example,

COMPUTE <T1,R1,<L1, =1>>= <T2,R2, =1>

Means to replace all those elements of row R1 of table T1 with elements from row R2 of table T2 where column identifiers of T1 occur in both T2 and in list L1. The list acts as a selection mechanism.

References to values of a string entry can utilize the single element subset of the table reference syntax with the following changes:

- The index option is meaningless and hence not permitted.
- Row and column identifiers are separated by a colon instead of a comma.

Both table and string references may occur in the same statement. For example,

```
COMPUTE <T1,$1,$2> = <S1,R1:C1>
```

replaces every element of table T1 with the value for the entry in string S1 which has identifiers R1:C1. A table is distinguished from a string with the same name by means of the colon used to separate the identifiers of the string.

MATRIX GENERATOR LANGUAGE MICROPROGRAMMING

Detailed manipulation capability is provided in MGL for lists and strings. This is in addition to the ability of the previously discussed general verbs to access or store an individual array entry - which can be incorporated in user defined loops, subroutines, etc. The verbs COMPOSE and FUNCT provide microprogramming on lists and strings respectively; the result of such operations is an entire list or string. The operations are symmetrical in the sense that the result is always the same kind of entity as the operands.

Both character string manipulation and arithmetic are involved in these operations. The terms "composition" (COMPOSE) and "function" (FUNCT) are arbitrarily associated with lists and string arrays, respectively, simply for purposes of discrimination. Both operations are quite similar in philosophy.

SAMPLE MATRIX GENERATOR LANGUAGE PROGRAM

Refer to Figure 15 following the program listing. The following information describes the matrix produced by the coding shown in the listing.

The macro-instruction BLEND is defined following creation of the tables CRUDE, CATPLT, GASOQ, DISTQ, FUELQ, and COST, and the string AVAIL. The first COMPUTE generates ones in the first four columns of the table CRUDE, row PSTILL. The second COMPUTE increases by .25 the entries on the PROFIT row of CRUDE. SUBMAT then generates from CRUDE the first four columns of the matrix.

The YIELD statement generates the next three columns from the table CATPLT, and then BLEND is called. In BLEND, a row type is assigned; then ones are filled in the BAL row of the current table; then three or four columns of the matrix are generated; finally a :SPEC column is defined along with its range or translation. BLEND is repeated four times, generating a total of 17 contiguous columns (from VNAP1:PREM to FUEL:SPEC).

Finally the Right-hand-side is generated from the AVAIL string by a SUBMAT, and several rows are typed by LGLs.

Note the use of a translation for PREM:SPEC and DIST:SPEC eliminates the need for requirements rows for these variables. Otherwise, for example, one would have a constraint of the form:

	PREM:SPEC	AVAIL
PREM:REQ	1	≥ 75

and a similar one for DIST:SPEC.

```
Location       Verb           Variable

FILE                          SAMPRO:2
               TABLE          CRUDE=CR1F,CR2F,CR1D,CR2D
                              CR1=1,-.5,1,-.5
                              CR2=,1, ,1
                              VNAP1=-.1, ,-.15,
                              VNAP2= ,-.2, ,-.2
                              VDIST=-.25,-.2,-.4,-.35
                              RESID=-.6,-.5,-.4,-.3
                              PSTILL=  ,   ,   ,
                              PROFIT=  ,   ,   ,
               TABLE          CATPLT=VDIST,LTYCL,HVYCYL
                              CATNAP=.7,.6,.3
                              LTCYL=.3, ,.7
                              HUYCYL=.5,.5
                              CATCAP=-1.6,-1.2,-1.1
                              PROFIT=-.1,-.15,-.16
                              RESID=-.1 ,  ,
               TABLE          GASOQ=VNAP1,VNAP2,CATNAP
                              OCTANE=85,84,92
                              BAL=  ,  ,
               TABLE          DISTQ=VDIST,VNAP2,LTCYL
                              CONTAM=54,50,65
                              BAL=  ,  ,
               TABLE          FUELQ=HVYCYL,LTCYL,VDIST,RESID
                              RATIO=-9, , ,1
                              BAL=  ,  ,  ,
               TABLE          COST=CR1F,CR2F,CR1D,CR2D
                              PROFIT-2.75,2.85,2.75,2.85
               STRING         AVAIL=PSTILL:AVAIL=100,CATCAP;AVAIL=46,
                              VDIST:AVAIL=11,LTYCL:AVAIL=-11
                              CR2:AVAIL=75,PROFIT:AVAIL=70
               MACRO          BLEND
               LGL            #4:#2(#6)
               COMPUTE        <#1,BAL,$1>=1.0
               QUALS          :#2:=#1
               VECT           #2:SPEC,PROFIT=#3,#4:#2=#5,BAL:#2=-1
               STR            #2:SPEC#7
               ENDM

               COMPUTE        <CRUDE,PSTILL,$1>=1.
               COMPUTE        <CRUDE,PROFIT,=1>=<COST,PROFIT,=1>+.25
               SUBMAT         $1::,$2:PSTILL:=<CRUDE,$1,$2>
               YIELD          :CATPLT:=CATPLT
               BLEND          GASOQ;PREM;-5;OCTANE;-89;MINUS;,TRANSL=25
               BLEND          GASOQ;REGLR;-4.5;OCTANE;-85;MINUS;(RANGE=10)
               BLEND          DISTQ;DIST;-4;CONTAM;-55;PLUS;,TRANSL=30
               BLEND          FUELQ;FUEL;-2.5;RATIO;0;PLUS;(RANGE=50)
               SUBMAT         $1.$2(RHS)=<AVAIL,$1:$2>
               LGL            PSTILL(PLUS)
               LGL            CATCAP(PLUS)
               LGL            CR1(PLUS)
               LGL            CR2(PLUS)
               LGL            PROFIT(FREE)
               EXIT
END***
```

	CR1F PSTILL	CR2F PSTILL	CR1D PSTILL	CR2D PSTILL		VDIST CATPLT	LTCYL CATPLT	HUYCYL CATPLT		VNAP1 PREM	VNAP2 PREM	CATNAP PREM	PREM SPEC		VNAP1 REGLR	VNAP2 REGLR	CATNAP REGLR	REGLR SPEC		VDIST DIST	VNAP2 DIST	LTCYL DIST	DIST SPEC		HUYCYL FUEL	LTCYL FUEL	VDIST FUEL	RESID FUEL	FUEL SPEC	AVAIL
CR1	1	-.5	1	-.5									(T=25)					(R=10)					(T=30)						(R=50)	≤
CR2		1		1																										≤ 75
VNAP1	-1		-.15							1					1															
VNAP2		-.2		-.2							1					1					1									
VDIST	-.25	-.2	-.4	-.35		1														1							1			11
RESID	-.6	-.5	-.4	-.3		.1																						1		
PSTILL	1	1	1	1																										≤ 100
PROFIT	3.0	3.1	3.0	3.1		.1	.15	.16					-5					-4.5					-4						-2.5	≥ -70
LTCYL						-.3	1	-.7														1				1				-11
HVYCYL						-.5	-.5	1																	1					
CATNAP						-.7	-.6	-.3				1					1													
CATCAP						1.6	1.2	1.1																						≤ 46
OCTAVE:PREM										85	84	92	-89																	≥
BAL:PREM										1	1	1	-1																	
OCTANE:REGLR															85	84	92	-85												≥
BAL:REGLR															1	1	1	-1												
CONTAM:DIST																				54	50	65	-55							≤
BAL:DIST																				1	1	1	-1							
RATIO:FUEL																									-9	0	0	1	0	≤
BAL:FUEL																									1	1	1	1	-1	

Figure 15. Matrix Produced by Sample Matrix Generator Language Program

4. FORMAT GENERATOR LANGUAGE

INTRODUCTION

Format generator language statements define the format and content of BCD output. Within the context of LP/600, this output may be any of the following:

1. The problem solution in a special report format.
2. Problem input data for processing by CONVERT.
3. A REVISE file containing revisions to a problem file.
4. A MODIFY file containing revisions to a work file.
5. A set of DELIMIT rows and/or columns.
6. Matrix generator language or format generator language statements.

Data used in producing the output may be:

1. Data taken from an existing problem solution.
2. Data taken from the current work file.
3. New data provided within the context of the format generator language.

From format generator language statements, the DEFINE verb produces a format skeleton on the problem device (PT). Subsequently, the CALC verb may be used to execute the formatting operations specified in the skeleton. Both the DEFINE and CALC verbs are executed at problem execution time. They may occur at any point in the agenda program and may be used repeatedly.

STATEMENT FORMAT

The card format of format generator language statements is the same as that for the Agenda Control Language statements.

Verbs

Verbs in the format generator language are described in Figure 16.

COMPUTE newvar/position.d=arithmetic expression

"Newvar" is the user's symbol for the quantity to be computed. "position" is the print or punch card column number for the units position of the results, and ".d" denotes a decimal number with d digits in the fraction. The ".d" is omitted for integers and "/position.d" is omitted for an intermediate result not to be produced as output. If the computational result is to be produced as output but not retained, "newvar" may be omitted. "Arithmetic expression" follows the rules for the control language COMPUTE verb, subject to the requirements for prefix characters on operands.

LOGIC switch=Boolean expression, label

"Switch" is the user's symbol in which the value (0 for true, 1 for false) of the Boolean expression is stored,and "Boolean expression" follows the rules for the control language verb LOGIC, except that all operands must be prefixed. If the value of "switch" is set to 1, a jump to the statement "label" is executed.

GOTO label

Unconditional jump to the statement identified by label.

PERFORM label

Similar to GOTO, but a subsequent NEXT causes return to the statement following PERFORM; used to enter subroutines.

NEXT

Return to the statement following the most recently executed PERFORM; used for subroutine return.

ALPHA n, any character string

Defines BCD data; "n" is the length of the character string.

INSERT label, position.n

Makes up output lines; "label" designates an ALPHA statement, "position" is the left-most print or punch position of the insertion, and ".n" (optional) is the number of characters to be inserted from the ALPHA statement, if the output is less than the full length of ALPHA.

LINOUT cc, label

Produces a print or punch line as output. The "cc" symbol is the required carriage control character. "Label" is optional; if used, it refers to an ALPHA statement which will be printed or punched as is, without clearing. If no label is given, the standard print line (built with previous statements) will be printed and cleared to blanks.

SCALE prefix=constant

"Prefix" is one of the set-defining prefix characters, causing all quantities of the type specified to be multiplied by the constant before use.

EXIT Terminates the FGL Program and causes return to the statement following the CALC statement in the ACL Program.

MACRO Introduces and identifies a MACRO prototype.

MACRO CALL Calls and executes the prototype named in the verb field using the arguments specified in the variable field.

ENDM Signals the end of a MACRO prototype definition.

Figure 16. Format Generator Language Verbs

Arithmetic Expression and Operators

The COMPUTE and LOGIC verbs are functionally identical with the corresponding Agenda Control Language verbs. The same arithmetic, relational, and Boolean operators may be used.

Labels

Statement labels may be any string of from 1 to 6 legal characters, the first of which must be non-numeric.

Operands

Operands may be decimal numbers, in standard format as shown by the examples below, identifiers, or names.

```
 3.1416
10
 1
-13.62
```

Non-numeric operands are limited to intermediate results or switch variable names, legitimate row and column identifiers, or communication region names. Furthermore, a prefix character must be attached to each such identifier or name to identify its type and usage properly. The set of prefix characters is shown in Figure 17 below.

Prefix	Resulting Identification of Attached Identifier or Name
X	A structural vector identifier denoting a variable whose value is to be used.
P	A row identifier denoting a pi-value.
D	A structural vector identifier denoting a relative cost factor (d_j).
L	A row identifier denoting a logical variable whose value is to be used.
A	A structural vector identifier followed by a comma and a row identifier, indicating an element of the A-matrix.
B	A row identifier indicating an element of the current right side.
C	A structural vector identifier indicating an element of the current cost row.
N	An intermediate result name previously computed or to be computed.
K	A communication region location name.

Figure 17. Prefix Characters

Row and Column Name Summation Convention

Row and column identifiers have the following form:

$$\alpha \quad r_1:r_2:r_3 \qquad (\alpha = P,L,B)$$

and

$$\beta \quad C_1:C_2:C_3 \qquad (\beta = X,D,C)$$

α and β are reference types, and the r_i and C_i are subnames of 0 to 6 characters: an r_i or C_i may be blank; it is then interpreted as being null. If a subname is given as asterisk (*), then summation over the full subname is implied. Thus, the usage:

COMPUTE Value = $Pr_1:r_2:*$

has the meaning:

$$\text{Value} = \sum_i \Pi_i,$$

for all rows whose first and middle names are r_1 and r_2, respectively, regardless of the last name. Similarly the usage:

COMPUTE Value = $DC_1:*:C_3$

means:

$$\text{Value} = \sum_i d_j,$$

for all columns whose first and last names are C_1 and C_3, respectively, regardless of the middle name.

Special Combination Operands

To facilitate the output of special products, two combination operands are included. The first of these is the scalar (dot) product operand:

$$\langle \alpha \; r_1:r_2:r_3,C_1:C_2:C_3 \rangle$$

or

$$\langle \beta \; r_1:r_2:r_3,C_1:C_2:C_3 \rangle$$

where α, β, r_i, and C_i are as defined above, including use of *. For example the usage:

COMPUTE Value = $Dr_1:r_2:r_3,C_1:C_2:C_3$

means:

$$\text{Value} = \sum_i \sum_j a_j^i \, d_j,$$

where the i and j are row and column indices over all those subnames with value of *. Similarly, if α= P,

$$\text{Value} = \sum_i \sum_j a^i_j \pi_i$$

The second special operand is the element sum operand:

$$A \quad r_1{:}r_2{:}r_3{,}C_1{:}C_2{:}C_3{,}$$

denoting

$$\text{Value} = \sum_i \sum_j a^i_{j'}$$

for all i and j with subname values of *.

SAMPLE FORMAT GENERATOR LANGUAGE PROGRAMMING

The sample programming in Figure 18 illustrates how the format generator language is used to produce an output report from the results of a linear program run. Note particularly the structural names in the COMPUTE and LOGIC statements, along with their identifying prefix characters.

LOCATION	E/O	OPERATION		ADDRESS, MODIFIER	COMMENTS
1 2 FILE 6	7	8 NAME 14	15	16	32
		GOTO		START	
A1		NAME		24, PIPESTILL	OPERATIONS
A2		NAME		12, PER CENT OF	
A3		NAME		42, TYPE MB/	D TOTAL CRUDE
A4		NAME		30, CRUDE 1	FUEL
A5		NAME		1, 2	
A6		NAME		10, DISTILLATE	
START		INSERT		A1, 10	
		INSERT		A2, 60	
		LINOUT			
		INSERT		A3, 37	
		LINOUT			
		COMPUTE		TCR=X*:PSTILL:*/100.	
		LINOUT			
C1F		COMPUTE		Z/49. 3=X CR1:PSTILL:	FUEL
		LOGIC		NZ, LE, 0, C2F	
		PERFORM		P	
		LINOUT			
C2F		COMPUTE		Z/49. 3=X CR2:PSTILL:	FUEL
		LOGIC		NZ, LE, 0, C1D	
		PERFORM		P	
		INSERT		A5, 17	
		LINOUT			

Figure 18. Format Generator Language Programming Example

LOCATION 1 2 6	E/O 7	OPERATION 8 14	15	ADDRESS, MODIFIER 16	COMMENTS 32
C1D		COMPUTE		Z/49. 3=XCR1:PSTILL:DIST	
		LOGIC		NZ.LE. 0, C 2D	
		PERFORM		P	
		INSERT		A6, 30	
		LINOUT			
C2D		COMPUTE		Z/49. 3=XCR2:PSTILL:DIST	
		LOGIC		NZ.LE. 0, CAT	
		PERFORM		P	
		INSERT		A6, 30	
		INSERT		A5, 17	
		LINOUT			
		GOTO		CAT	
		INSERT		A4, 11	
		COMPUTE		64. 3=NZ/NTCR	
		NEXT			
CAT					
END***					

Figure 18. (cont'd)

5. INPUT FILE FORMATS

INTRODUCTION

LP/600 input file structure introduces many new approaches to matrix definition. All are intended to give the user the utmost flexibility and convenience in defining his particular problem matrix, or in defining revisions to a problem file or work file.

MATRIX INPUT FILE

A matrix input file defines a problem for processing by CONVERT. The card types used are described in Figure 19. Each card type is identified by a designator, beginning in column 1, and contains problem data in variable-field format beginning in column 8. With certain exceptions, cards within the file may appear in any desired sequence. The variable field format and the absence of stringent card sequencing requirements are outstanding features of LP/600.

Type Indicator	Description
FILE	Contains the file name and is the first card in file.
LGL or L	Specifies a logical variable, and hence, a row. Optionally, a row element count, a row scale, and a row type may be specified.
STR or S	Specifies a structural variable, and hence, a column. Optionally, a column element count, a column translation, a column scale, and a column type may be specified.
MATRIX or A	Defines a matrix element.
RHS or B	Defines an RHS element.
BLOCK	Introduces a series of cards defining a matrix "block" for the DECOMP algorithm. Blocks must be in ascending order. A block is a complete subfile within a matrix input file.
ENDGRP	Ends the last GROUP. Not required if another GROUP-type column definition card follows.
END***	The last card in every file deck.
*	Comment card. May be used anywhere.

Figure 19. Matrix Input File Card Types

Row and Column Name Definition

Row and column names may be defined in one to three parts, each separated by a colon and consisting of a maximum of six characters. Names may be constructed from the characters A-Z, and the numbers 0-9. The use and utility of three-part, 18-character names, particularly for depicting row and column significance in process flow matrices, was shown in earlier discussions of the matrix generator language.

In the input file, row names are defined by their appearance in an LGL card or a MATRIX card. Column names are defined by their appearance in an STR card or a MATRIX card. Thus, except for certain row and column types which require LGL and STR cards, matrix rows, columns, and coefficients may be defined entirely in MATRIX cards. In most cases, LGL and STR will be used only to name specific rows and columns for inclusion in the work file, or to define the required order of rows and columns in the file.

Row and Column Type Definition

LP/600 row and column type designators are described in Figure 20. They permit any row relation and row or column bound to be defined explicitly without increasing matrix size. Except for RANGE, GROUP and PACKET, which require user-defined values and must appear in a particular card type, types are defined at the point of name definition. The examples below show name and type definitions in LGL and STR cards.

```
  .
  .
  .
LGL         PROFIT(FREE)
LGL         GASO:REF1:(ZERO)
  .
  .
  .
STR         GASO:REF1:AREA1(PLUS)
STR         DIST:REF1:AREA1(PLUS)
STR         MFG:PROFIT (ZERO)
  .
  .
  .
```

If the rows and columns were defined in MATRIX cards, the format would be:

```
  .
  .
  .
MATRIX      PROFIT(FREE),GASO:REF1:AREA1(PLUS)=4.78
  .
  .
  .
MATRIX      GASO:REF1:(ZERO),GASO:REF1:AREA1(PLUS)=1
  .
  .
  .
```

To provide an added convenience in matrix definition, all rows are assumed to be type ZERO unless a type is specifically given.

ROW TYPES

Type Indicator	Description
(ZERO) or (Z)	Equality row; logical variable must be zero.
(PLUS) or (P)	Less-than-or-equal row; logical variable must be nonnegative.
(MINUS) or (M)	Greater-than-or-equal row; logical variable must be nonpositive.
(FREE) or (F)	The logical variable may take on any value; used for functionals.
(RANGE) or (R)	Bound row; requires a positive numerical value. If v^i is the specified value and b^i is the RHS element for this row, then the logical variable must lie in the range $[0,v^i]$; that is, the row must satisfy the following restraints: $b^i-v^i \le \sum_j a^i_j x^j \le b^i$.

COLUMN TYPES

Type Indicator	Description
(ZERO) or (Z)	Variable must be zero; column never enters the basis.
(PLUS) or (P)	Variable must be nonnegative.
(MINUS) or (M)	Variable must be nonpositive. The column is multiplied by -1 and given a scale of -1. The variable is then treated as PLUS type until output time.
(FREE) or (F)	Variable can take on any value. Once in the basis the column is never removed.
(RANGE) or (R)	This requires either one or two specified numerical values. If only one is given, the other is assumed to be zero.
(GROUP) or (G)	Pseudo column which precedes a group; the stated value applies to the right-hand-side element.
(PACKET) or (T)	Pseudo column which precedes a packet; the stated value is the extent of the packet.

TYPE MODIFIERS

Modifier	Description
SCALE	Scaling factor for a row or a column. A scale may not be used when the column type is RANGE or when the row type is RANGE or FREE. Scales affect internal computations but are compensated for at output. A negative scale in effect changes a MINUS type to PLUS or vice-versa.
COUNT	Presumed count of the number of elements used as input for the row, column, or right-hand-side.
TRANSL	Translation factor for the column. A translation may not be used for rows, or when the column type is RANGE. Translations affect internal computations but are compensated for at output.

Figure 20. Row and Column Types

Similarly, a column is assumed to be type PLUS. Further, repetition of card type indicators is not necessary, since a card without an indicator is assumed to be the last named type. (NOTE: In examples herein, all row type and card type indicators are defined explicitly.)

Matrix and Rhs Element Definition

Matrix elements are defined by phrases in MATRIX cards. The structure of the phrases is intended to relieve the user of the need to repeat row and/or column names. For instance, the right side of the phrase always contains the value of a coefficient; the left side must contain one name, but may contain both a row and column name, and type designators. If either name is omitted, the coefficient is assumed to pertain to the last named row or column.

For example, in the card below, the comma preceding the name indicates the absence of a row name.

```
MATRIX        ,GASO:REF1:AREA1=1
```

The element is presumed to pertain to the last named row. In the example above, if the leading comma were omitted, the program would assume that the name given was a row name; in this case, the coefficient would pertain to the last named column.

RHS elements are defined in the same manner. For example,

```
  .
  .
RHS           GASO:REF1,RHS01=1.5
              ,RHS02=2.0
              DIST:REF1,RHS01=2.4
```

Element Counts

A count of the number of elements in a row, column or RHS may be included in LGL, STR, or RHS cards at the option of the user for checking during CONVERT. The count is usually defined at the time the related row or column is defined; that is, at the first appearance of the row or column name. For example,

```
  .
  .
LGL           PROFIT(FREE),COUNT=67
  .
  .
  .
STR           GASO:REF1:AREA1(PLUS),COUNT=49
  .
  .
RHS           GASO:REF1,RHS01=2.4,COUNT=35
```

Row Bounds

The "value" of a less-than-or-equal row may be constrained within the range of two values by placing a bound on the logical vector. This is done by defining the row as type RANGE in an LGL card and providing a range value. This value is the difference between the lower value and the upper value (which is the RHS value) in the desired range. For example, to assign a lower bound of 25 and an upper bound of 100, the row would be defined as follows:

```
      .
      .
      .
     LGL                 rowname(RANGE=75)
      .
      .
      .
     RHS                 rowname,RHS name = 100
      .
      .
      .
```

The effect is to bind the value of the logical within the range 0-75 and, hence, the row value between 25-100.

Bounded Variables

Upper and/or lower bounds for a variable are easily definable within the STR card. If both upper and lower bounds are required, the column is defined as type RANGE and the bounds are defined. For example,

```
     STR                 GASO:REF1:AREA1(RANGE=10,20)
```

If only one bound is required, the variable is defined as either PLUS or MINUS and a translation value is specified. For example,

```
     STR                 GASO:REF1:AREA1(PLUS),TRANSL=20
```

Scaling

Automatic scaling is used internally for RANGE and MINUS row or column types. An AUTO-SCALE option is also available during problem SETUP to improve digital accuracy. In addition, LP/600 enables the user to specify row and/or column scales of his choice. Any scaling, whether automatically generated or user supplied, is compensated for in all output. For example,

```
     STR                 GASO (PLUS), SCALE = 17
```

MATRIX REVISION FILE

The matrix revision file contains changes to a problem file to be made by the REVISE verb. Control cards which define the required revisions are described in Figure 21. Cards containing the revision data are identical with those used to define a matrix input file.

The comprehensive matrix revision capability of LP/600 is one of the most useful features of the system. Any desired revision may be made to a problem file, including the linear combination of existing rows or columns.

Indicator	Description
INSERT	Specifies that trailing LGL, STR, MATRIX, RHS, CFORM, or RFORM data is to be inserted in input order into the problem file immediately following the row, column, or RHS specified in the variable field of the INSERT card. To make an insert at the beginning of the section of logical (rows), structural (columns), or right-hand-side elements, the programmer must place '000000' in the variable field of the INSERT card. If the name is '******', insertion takes place at the end of the designated section.
DELETE	Specifies that the row, column, or RHS named in the variable field is to be deleted from the problem. Any cards following the DELETE card are treated as though they followed an INSERT card; that is, the rows or columns named are inserted at the point at which the deletion occurred.
RPLACE	Specifies that the trailing LGL, STR, MATRIX, and Rhs data is to replace existing problem file data containing corresponding row or column names.
RFORM	Specifies that a row is to be computed as a linear combination of a set of one or more existing rows and an additive constant, which are defined in the variable field.
CFORM	Specifies that a column is to be computed as a linear combination of a set of one or more existing columns and an additive constant, which are defined in the variable field.

Figure 21. Revise File Control Cards

MATRIX MODIFICATIONS FILE

In addition to the capability for creating an updated problem file by means of the REVISE verb, temporary modifications can be specified in a matrix modifications file for execution by the MODIFY verb. The modifications file is similar to the revisions file, utilizing all of the revisions file control cards except RFORM and DELETE. The following modifications can be made:

1. The type of any row or column may be changed.
2. Any non-null structural element value, scale, or translation may be changed.
3. Any RHS element may be changed. A new RHS may be added to the existing matrix.
4. A new RHS may be added as CFORM columns.

In addition, all permissible changes may be cascaded by means of multiple, repeated use of MODIFY verbs in a single problem run.

PUNCH/LDBASIS FILE

The basis from a particular solution to a problem may be punched and subsequently reloaded to provide an advanced start for a modified or revised problem. The format consists of a symbolic file; that is, the first card is FILE and the last is END***. The data consists of a list of logical and/or structural vector names separated by commas. Each data card has a type indicator which identifies all the vectors on that card as being the basis or at bound and shows whether the names refer to logical or structural vectors (see Figure 22).

Type	Description
LGL	Logical vectors in the basis.
LAB	Logical vectors not in the basis, but at bound.
STR	Structural vectors in the basis.
SAB	Structural vectors not in the basis but at bound.

Figure 22. Basis File Elements

LIST FILES

Several of the ranging, tableau, solution print, and work file change verbs can call for a list of row (or column) names or masks to denote the set to be used. Such lists are treated as a symbolic file in the format shown in Figure 23.

Indicator	Description
FILE	Contains the file name and is the first card of the file.
MASK	Gives the names or masks (separated by commas) used to specify the required set. Masks are denoted by an asterisk (*) for any character whose value is immaterial. Multiple trailing asterisks are not required within a name part.
END***	The last card in every file deck.

Figure 23. List Files

Several masks and their meanings are given below:

Mask	Meaning
ABC	The vector whose name is ABC::.
A*BC	Any vector whose second and third names are blank and whose first character is A, third character is B, fourth character is C, and fifth and sixth characters are blank. The second character of the first name may be any character due to the *.
ABC:*:*	Any vector whose first name is ABC.

EDIT SPECIFICATION FILE AND EDIT FILE

Data in the SHARE standard format is acceptable as input to the LP/600 system, adding new dimensions to the total versatility of the system. The input file containing this data is translated to an LP/600 input file (convert file) as an initial step to the processing of the data.

Controls for the translation and identification of the SHARE standard data file are provided through the combined use of the Agenda Control Language EDIT verb, an edit specification file, and GECOS control cards. The translation process requires the user to define three files: the edit file, the edit specification file, and the output (convert) file. The interrelationship of these files is illustrated in Figure 24.

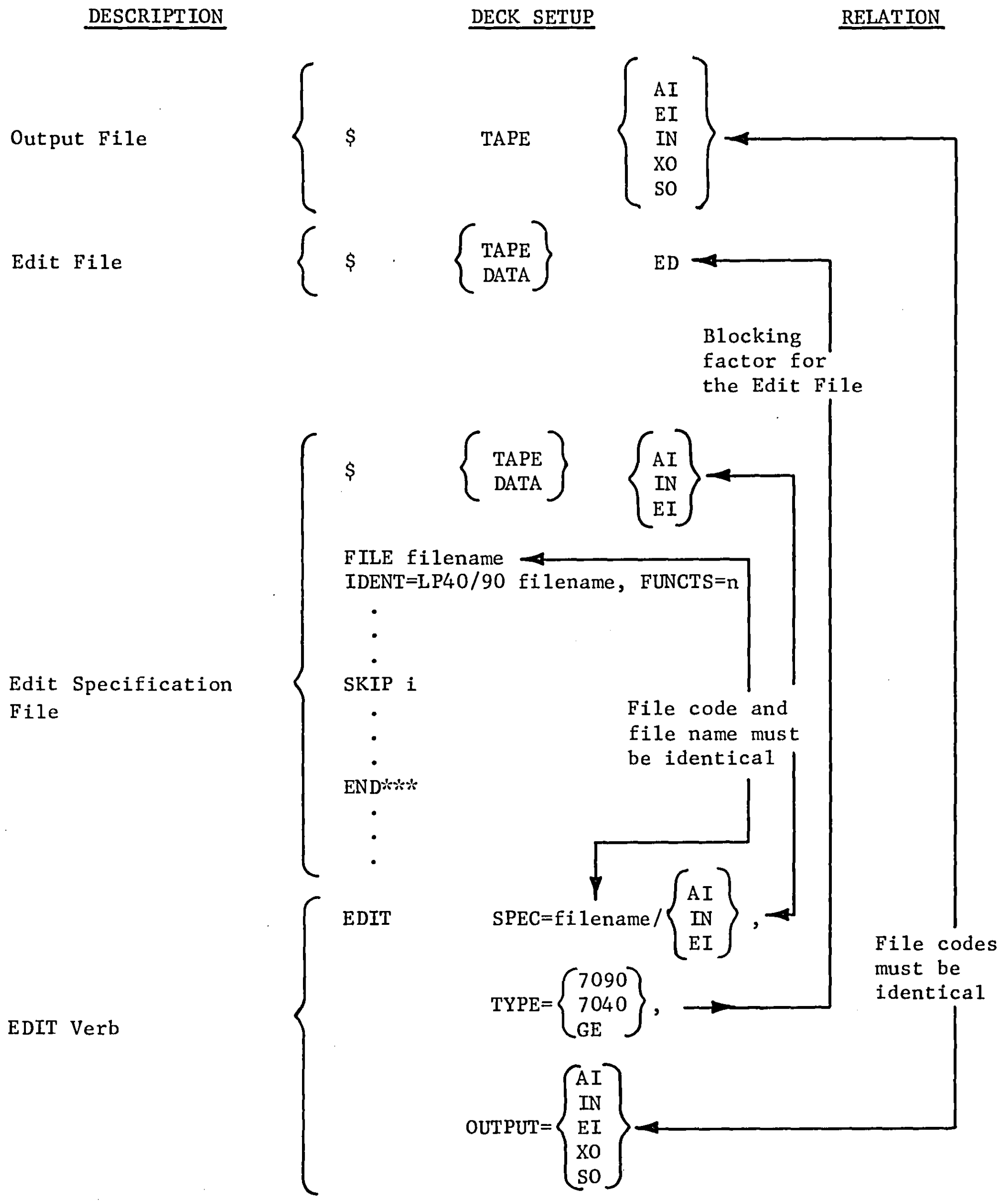


Figure 24. Files Used for Translation of SHARE Standard Data to LP/600 Input File Format

Edit File

The edit file which contains the SHARE standard input data must be in the IBM character set and formatted in one of the record blocking factors discussed below. This file will normally be on magnetic tape, but may be on cards. The input device is designated by the GECOS file code ED. The $ TAPE or $ DATA card is the interface with LP/600 and provides the sole link between the source data (SHARE standard data files) and the LP/600 system. If the SHARE standard data files are on cards, they must follow the $ DATA card within the LP/600 input deck setup.

A linear programming data file prepared on tape for LP90 (7090) has a blocking factor of one card per record. A file for LP40 (7040) has a blocking factor of three cards per record. Either file contains BCD card images at 84 characters per card. For those SHARE standard data files that are input from the GE-625/635 card reader via a $ DATA card or from a tape produced by the GE-625/635 Bulk Media Conversion program, the blocking factor is 320 words per record. The TYPE phrase given on the EDIT verb informs the LP/600 system which of the .three blocking factors above apply to the Edit file (see Figure 24). It should be noted that SHARE standard data files do not conform to the rules followed by all other LP/600 files. That is, they do not begin with a FILE card and file name, and terminate with an END*** card.

The input device may contain one or more edit files to be translated for output and subsequent input to the Agenda Control Language CONVERT verb. Each edit file is translated with individual file indentification via multiple IDENT phrases in the Edit Specification File (see Figure 25).

Edit Specification File

The edit specification file contains the name assigned by the user to each translated SHARE data file along with the number of objective functions in each problem. This file may also specify the number of files to be skipped; it is actually an extension of the EDIT verb variable field.

The edit specification file bears the name assigned in the EDIT verb SPEC phrase and must be located on the GECOS file code which is also given in that phrase (see Figure 24). Since the edit specification file is normally on cards, a $ DATA card must precede the file. However, the file may be on tape, in which case the $ TAPE card is used.

The Edit Specification file must follow LP/600 rules for file structure. It must begin with a FILE card and end with an END*** card. One or more IDENT cards may follow the FILE card and may be interspersed with SKIP cards. The IDENT card specifies the name to be assigned to the corresponding file (problem) in the SHARE standard input data file (ED), and the number of objective functions in that problem. The SKIP card specifies the number of files (problems) to skip on device ED. The naming and skipping of these SHARE files (problems) must be in a strict one-to-one correspondence with the problems in the order they appear on ED.

FILE	Identifies the edit specification file and provides the file name which is identical to the name used in the EDIT verb SPEC phase.
IDENT	Contains the name to be assigned to a translated SHARE data file, and the number (n) of functionals (FUNCTS) in the problem. An IDENT card must be provided for each edit file on the ED input device to be translated.
SKIP	Causes EDIT to skip a specified number of files (problems) on the input tape.
END***	Terminates the edit specification file.

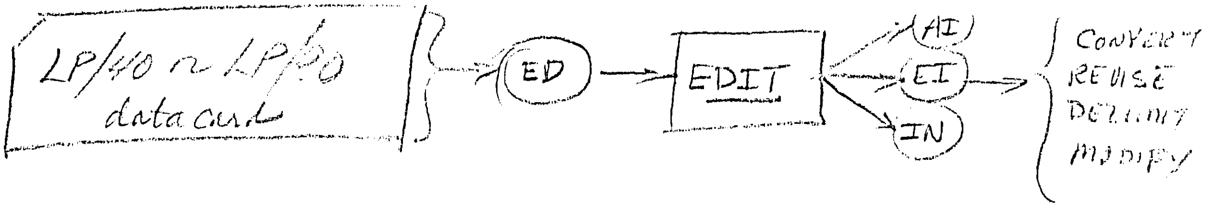
Figure 25. Edit Specification File Card Type Definition

EDIT Verb

The function of the Agenda Control Language (ACL) EDIT verb is to translate SHARE standard data files on the ED device to the convert file format for processing by the ACL CONVERT verb. The EDIT verb does not check the SHARE data files for proper structure and content. It is assumed that the files have been used and checked by the linear programming system for which they were prepared.

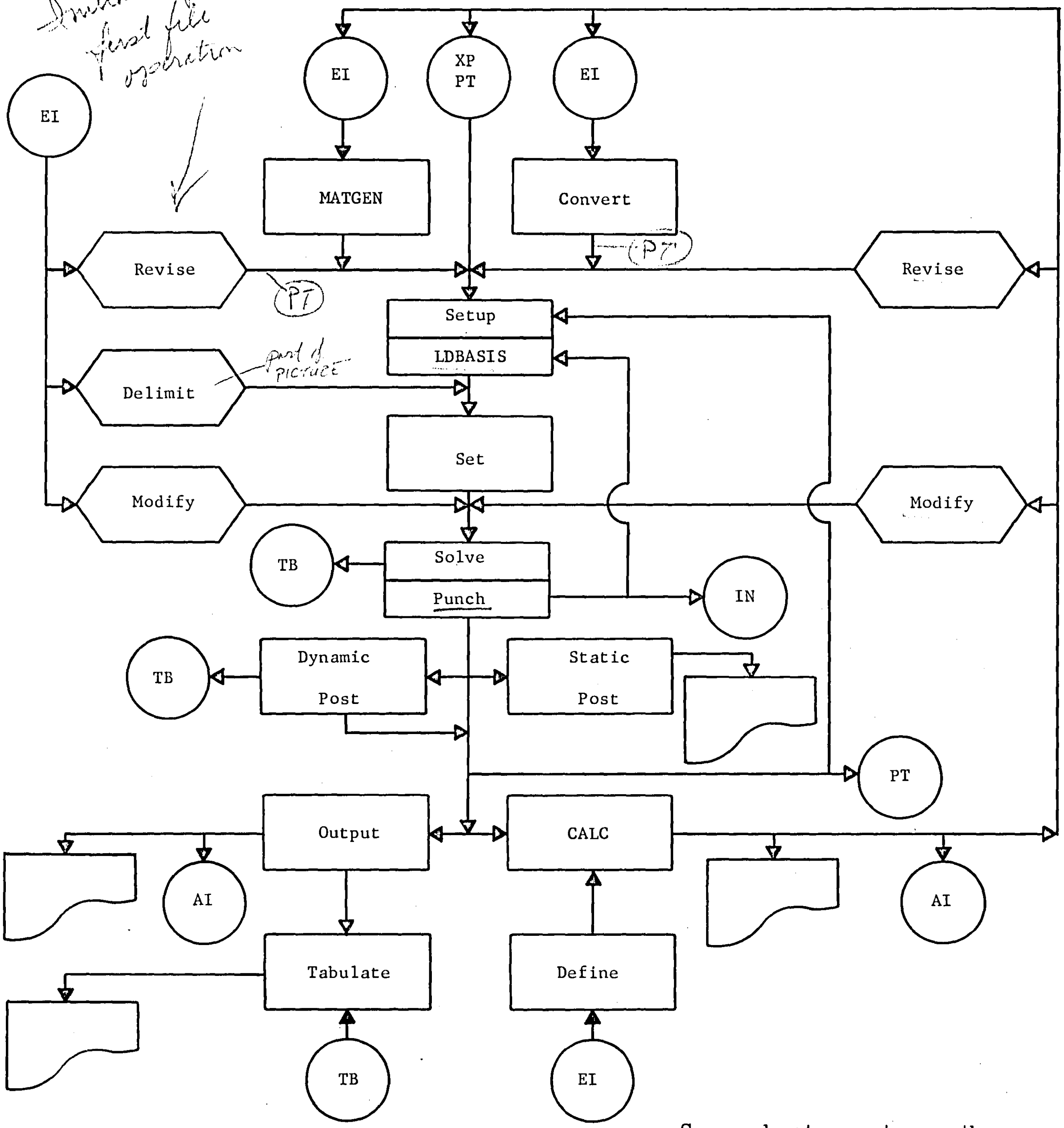
Output File

The output file contains the SHARE standard data translated to an LP/600 format ready for processing by the ACL CONVERT verb. The GECOS file code given on the $ TAPE card must be identical to the file code given in the OUTPUT phrase of the EDIT verb.



APPENDIX A. SYSTEM DATA FLOW

Initial input to first file operation



See explanatory notes on the following page for AI, EI, IN, XP&PT and TB definitions.

Trace Read
Save Picture
Restore Title
Cons Unload

NOTES

Files EI, AI, or IN may be interchanged. However, for purposes of depicting system flow, the following definitions apply:

AI = symbolic data which can be read by FORTRAN

EI = symbolic data which can be written by FORTRAN

IN = symbolic data which can be written by FORTRAN

XP&PT = binary data for LP/600 input

TB = binary data which may be read by FORTRAN

APPENDIX B.
INPUT DECK SETUP

The LP/600 basic system is executed as a one-activity job under control of the GE-625/635 Comprehensive Operating Supervisor (GECOS). As shown in Figure 26, the deck setup consists of GECOS control cards, LP/600 input files which may be input from tape or from the card reader, and the Agenda Control Language program cards.

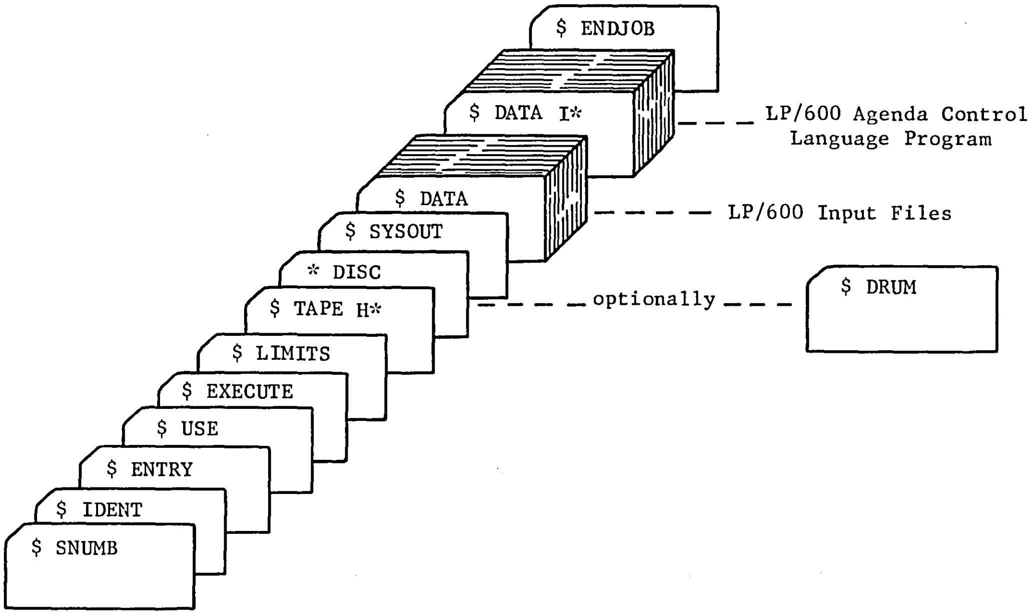


Figure 26. LP/600 Input Deck Setup

SUMMARY OF GECOS CONTROL CARDS

The following control cards are used by GECOS:

Beginning columns

Col.	1	8	16
	$	SNUMB	Job identifier, urgency
	$	IDENT	Account number, identification
	$	USE	Program Name
	$	ENTRY	Program Name
	$	EXECUTE	
	$	LIMITS	Time, storage requirements, number of print lines
	$	SYSOUT	SO
	$	TAPE	File code, logical unit designator, second logical unit designator, file serial number, reel sequence number, file name
	$	DRUM DISC	File code, logical unit designator, storage requirements
	$	DATA	File code, options
	$	ENDJOB	blank
	***EOF		(GECOS end-of-file)

INPUT FILES (ED, IN, AI OR EI)

The LP/600 files (edit, convert, revise, modify, basis, or list) may be input as ED, IN, EI, or AI from the card reader, tape, disc, or drum that was written with the GE Bulk Media Conversion program. (See CPB-1096.) The user must define file code assignments to be consistent with file code specifications used in the SOURCE phrases in the Agenda Control Language program. Input of the file is initiated by execution of the corresponding Agenda Control Language program statement.

Card Reader Input

When a file(s) is input through the card reader, the file(s) must be preceded (in the job control deck) by a $ DATA card containing ED, IN, EI, or AI in the file code field. The file(s) may appear anywhere in the deck following $ EXECUTE, as shown in Figure 27.

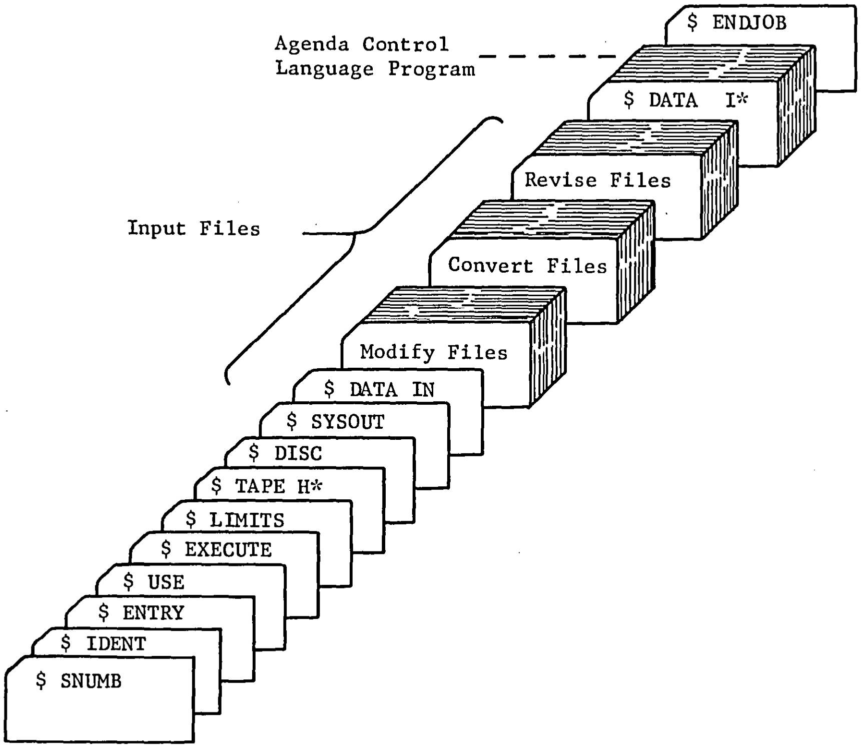


Figure 27. LP/600 Card Reader Input

Magnetic Tape Input

The GE-625/635 Bulk Media Conversion program may be used to put card images onto magnetic tape for input to LP/600. (See CPB-1096, GE-625/635 Series Bulk Media Conversion manual.) SHARE Standard LP data may also be input from magnetic tape (IBM tape in BCD mode). The logical assignment of the tape at execution time must be defined in a $ TAPE card having IN, ED, EI, or AI in the file code field and a logical unit designator and disposition code. Other $ TAPE card information may be included at the option of the user:

```
$ TAPE IN,A3D,,reel #,,INPUT
```

When files are to be input on tape from ED, EI, IN and AI, a $ TAPE card is required for each file, and may appear anywhere in the deck setup following the $ EXECUTE card. For example:

```
$ TAPE EI, A5D,reel #,,INPUT1

$ TAPE ED,A4D,,reel #,,EDIT INPUT

$ TAPE IN,A3D,,reel #,,INPUT2

$ TAPE AI,A2D,,reel #,,AUXINP
```

Disc or Drum Input

The GE-625/635 Bulk Media Conversion program may be used to put card images onto disc or drum for input to LP/600. (See CPB-1096, GE-625/635 Bulk Media Conversion.) The logical assignment of the disc or drum at execution time must be defined in a $DISC or $DRUM card having IN, EI, ED, or AI in the file code file, and a logical unit designator, disposition code, and the number of links:

```
$ DISC  IN,A3,10L
```

Note that disc and drum files must be specified as linked (L) files. When files are to be input on disc or drum from ED, EI, IN, and AI, a $DISC or $DRUM card is required for each file, and may appear anywhere in the deck setup following the $EXECUTE card. For example:

```
$ DISC   IN,A3,10L

$ DRUM AI,A4, 5L

$ DRUM EI,A5,15L

$ DISC   ED,A6, 7L
```

Card Reader, Magnetic Tape, Disc and Drum Input

When files are input from cards, tape, disc and/or drum, procedures given above for each input type must be followed. A typical arrangement is shown in Figure 28.

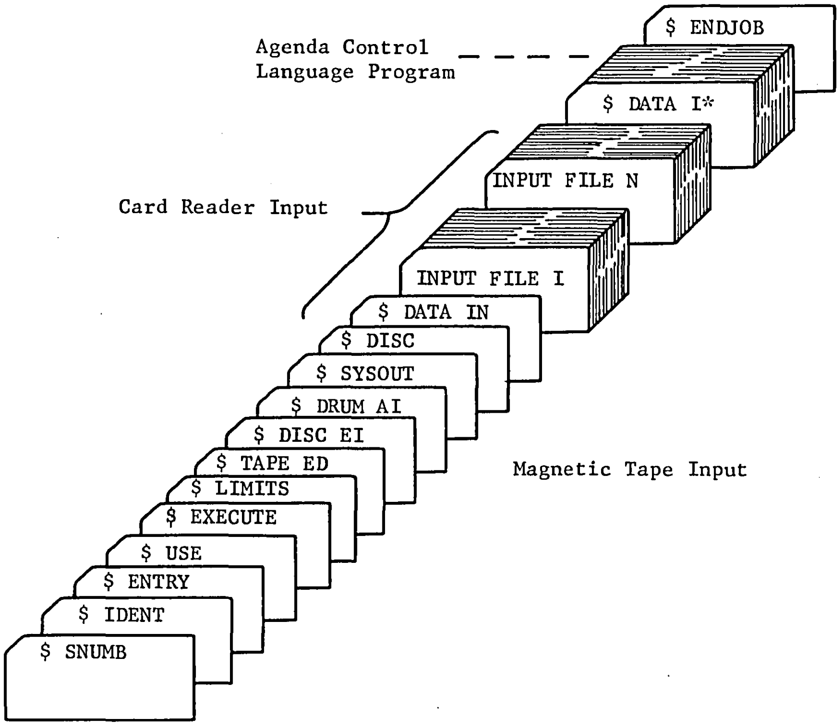


Figure 28. LP/600 Input from Card Reader, Magnetic Tape, Disc and Drum

PROBLEM FILES (PT, VP, AND XP)

Problem file data is produced and automatically written onto PT by the AFEAS, CONVERT, DEFINE, DESAVE, REVISE and SAVE verbs (for further details, see CPB-1262, LP/600 Agenda Control Language). Unless assigned to tape, the problem file data is written automatically to the disc or drum areas allocated by $DISC and $DRUM cards in the control deck. If the problem file is to be saved, it must be assigned to tape and specified as either "valid" or "invalid."

Tape Valid

When new problem file data is to be added to a problem tape which contains valid information produced during another run, and that information is not to be overwritten, it must be assigned to file code VP:

```
$ TAPE      VP,A4D,,REEL NO.,,OLD PT VALID
```

This assignment is made only for control reading and writing: a reference to PT in the Agenda Control Language program will be equated automatically to VP. The VP file code is external to LP/600 and cannot be used in an Agenda Control Language program statement.

When the problem tape contains valid information produced during another run, and new problem file data is to be written on a new PT, it must be assigned to XP:

```
$ TAPE      XP,A5D,,REEL NO.,,OLD PT
```

If the new problem file data is to be saved, then PT must be assigned to tape with another $TAPE card.

Tape Invalid

When the problem tape does not contain valid problem files (that is, a new PT is being written), it must be assigned as PT:

```
$ TAPE PT,A4D,,TAPE NO.,,NEW PT
```

Note that VP and PT refer to the same physical device; assignment of a PT and a VP is illegal. The distinction is made only to control reading and writing; a reference to PT in an Agenda Control Language program statement is equated automatically to the file code used in the $TAPE card.

OUTPUT FILES (XO, SO, AND TB)

Standard Output

LP/600 automatically writes all output on file code SO which normally is assigned as $ SYSOUT SO or, optionally, as tape, disc, or drum if it is desirable to use the GE-625/635 Series Bulk Media Conversion program. (See CPB-1096, GE-625/635 Bulk Media Conversion.) For example, SO may be assigned one of the following:

```
$ TAPE SO,A2S,,,,SO-OUTPUT
$ DISC  SO,A2S,90L
$ DRUM SO,A2S,90L
```

and printed by means of BMC as the second activity:

```
$ CONVER
$ TAPE IN,A2R,,,,SO-OUTPUT
$ INPUT NLABEL
$ PRINT OT,A2R
```

If SO is assigned to disc or drum, the $TAPE card above would be replaced by:

$$\$\left\{\begin{matrix}\text{DISC}\\\text{DRUM}\end{matrix}\right\}\quad\text{IN,A2R,90L}$$

The PUNCH verb causes output to be in card image format. IF SO is not assigned to SYSOUT, the XO option should be specified as the output file for PUNCH.

Auxiliary Output

The XO file code is an auxiliary output file and is used in conjunction with the large report-producing verbs such as PICTURE, TABULATE, TRACE, and others, if the GECOS output-line limit is to be exceeded. XO may be assigned to tape, disc, or drum, and printed by means of BMC as the second activity (see SO above for sample assignments). XO should also be used as the output file designator for the PUNCH verb only if SO is not assigned to SYSOUT, in order that BMC may be used to punch the card images.

Output of Intermediate Results

The TB file code is used only by the RECORD and TABULATE verbs. RECORD writes intermediate results on TB, and TABULATE reads (from TB), formats, and outputs those results.

APPENDIX C.
SAMPLE PROBLEM

The sample problem which follows is described in fourteen sections:

1. GECOS control cards
2. Convert file row descriptions
3. Convert file column descriptions
4. Convert file matrix element descriptions
5. Convert file Rhs definitions
6. LDBASIS file (punched by a previous run)
7. Revise file
8. Modify file
9. Delimit list file for rows
10. Delimit list file for columns
11. Agenda Control Language program to solve first part of problem
12. Agenda Control Language program to solve second part of problem
13. Agenda Control Language program to solve third part of problem
14. Second activity -- Bulk Media Conversion control cards to print TRACE reports from XO.

```
$       IDENT   XXX,JOHN DOE,LP/600
$       ENTRY   .LHSF
$       USE     .LHSF
$       EXECUTE
$       LIMITS  20,45000,,10000
$       TAPE    H*,A1D,,TAPE NO.,,LP/600 SYSTEM TAPE
$       TAPE    XO,A2S,,,,XO-OUTPUT
$       SYSOUT  SO
$       DISC    KF,A1,10R                                              ①
$       DISC    KG,A2,10R
$       DISC    KH,A3,10R
$       DISC    KI,A4,10R
$       DISC    KK,A5,10R
$       DISC    KL,A6,10R
$       DISC    ZR,A7,10R
$       DISC    ZS,A8,10R
$       DISC    ZT,A9,10R
$       DATA    IN
FILE    LP600:SAMPLE:PROBLEM
*               ROW ID    SEGMENT
LGL     CR1     :       :            (PLUS )
LGL     CR2     :       :            (PLUS )
LGL     VNAP1   :       :            (ZERO )
LGL     VNAP2   :       :            (ZERO )
LGL     VDIST   :       :            (ZERO )
LGL     RESID   :       :            (ZERO )
LGL     PSTILL  :CAP    :            (PLUS )
LGL     CATNAP  :       :            (ZERO )
LGL     LTCYL   :       :            (ZERO )
LGL     HVYCYL  :       :            (ZERO )
LGL     TOTAL   :CAT    :FEED        (PLUS )                           ②
LGL     RATIO   :FUEL   :            (PLUS )
LGL     FUEL    :DEMAND :            (PLUS )
LGL     CONTAM  :DIST   :            (PLUS )
LGL     DIST    :BAL    :            (ZERO )
LGL     OCTANE  :REGLR  :            (MINUS)
LGL     REGLR   :BAL    :            (ZERO )
LGL     OCTANE  :PREM   :            (MINUS)
LGL     PREM    :BAL    :            (ZERO )
LGL     DIST    :DEMAND :            (MINUS)
LGL     REGLR   :DEMAND :            (PLUS )
LGL     PREM    :DEMAND :            (MINUS)
LGL     PROFIT  :       :            (FREE )
*               COL ID    SEGMENT
STR     CR1     :PSTILL :FUEL        (PLUS )
STR     CR2     :PSTILL :FUEL        (PLUS )
STR     CR1     :PSTILL :DIST        (PLUS )
STR     CR2     :PSTILL :DIST        (PLUS )
STR     VDIST   :CATPLT :            (PLUS )
STR     LTCYL   :CATPLT :            (PLUS )
STR     HVYCYL  :CATPLT :            (PLUS )
STR     HVYCYL  :FUEL   :            (PLUS )
STR     LTCYL   :FUEL   :            (PLUS )
STR     VDIST   :FUEL   :            (PLUS )
STR     RESID   :FUEL   :            (PLUS )
STR     VDIST   :DIST   :            (PLUS )                           ③
STR     VNAP2   :DIST   :            (PLUS )
STR     LTCYL   :DIST   :            (PLUS )
STR     VNAP1   :REGLR  :            (PLUS )
STR     VNAP2   :REGLR  :            (PLUS )
STR     CATNAP  :REGLR  :            (PLUS )
STR     VNAP1   :PREM   :            (PLUS )
STR     VNAP2   :PREM   :            (PLUS )
STR     CATNAP  :PREM   :            (PLUS )
STR     DIST    :SPEC   :            (PLUS )
STR     REGLR   :SPEC   :            (PLUS )
STR     PREM    :SPEC   :            (PLUS )
```

```
*             MATRIX  SEGMENT
MATRIX CR1    :       :     ,    CR1    :PSTILL :FUEL  =1.0
MATRIX VNAP1  :       :     ,    CR1    :PSTILL :FUEL  =-.1
MATRIX VDIST  :       :     ,    CR1    :PSTILL :FUEL  =-.25
MATRIX RESID  :       :     ,    CR1    :PSTILL :FUEL  =-.6
MATRIX PSTILL :CAP    :     ,    CR1    :PSTILL :FUEL  =1
MATRIX PROFIT :       :     ,    CR1    :PSTILL :FUEL  =3.00
MATRIX CR1    :       :     ,    CR2    :PSTILL :FUEL  =-.5
MATRIX CR2    :       :     ,    CR2    :PSTILL :FUEL  =1.
MATRIX VNAP2  :       :     ,    CR2    :PSTILL :FUEL  =-.2
MATRIX VDIST  :       :     ,    CR2    :PSTILL :FUEL  =-.2
MATRIX RESID  :       :     ,    CR2    :PSTILL :FUEL  =-.5
MATRIX PSTILL :CAP    :     ,    CR2    :PSTILL :FUEL  =1
MATRIX PROFIT :       :     ,    CR2    :PSTILL :FUEL  =3.10
MATRIX CR1    :       :     ,    CR1    :PSTILL :DIST  =1.
MATRIX VNAP1  :       :     ,    CR1    :PSTILL :DIST  =-.15
MATRIX VDIST  :       :     ,    CR1    :PSTILL :DIST  =-.4
MATRIX RESID  :       :     ,    CR1    :PSTILL :DIST  =-.4
MATRIX PSTILL :CAP    :     ,    CR1    :PSTILL :DIST  =1.
MATRIX PROFIT :       :     ,    CR1    :PSTILL :DIST  =3.0
MATRIX CR1    :       :     ,    CR2    :PSTILL :DIST  =-.5
MATRIX CR2    :       :     ,    CR2    :PSTILL :DIST  =1.
MATRIX VNAP2  :       :     ,    CR2    :PSTILL :DIST  =-.25
MATRIX VDIST  :       :     ,    CR2    :PSTILL :DIST  =-.35
MATRIX RESID  :       :     ,    CR2    :PSTILL :DIST  =-.3
MATRIX PSTILL :CAP    :     ,    CR2    :PSTILL :DIST  =1.
MATRIX PROFIT :       :     ,    CR2    :PSTILL :DIST  =3.1
MATRIX VDIST  :       :     ,    VDIST  :CATPLT :      =1.
MATRIX RESID  :       :     ,    VDIST  :CATPLT :      =.1
MATRIX CATNAP :       :     ,    VDIST  :CATPLT :      =-.7
MATRIX LTCYL  :       :     ,    VDIST  :CATPLT :      =-.3
MATRIX HVYCYL :       :     ,    VDIST  :CATPLT :      =-.5
MATRIX TOTAL  :CAT    :FEED ,    VDIST  :CATPLT :      =1.6
MATRIX PROFIT :       :     ,    VDIST  :CATPLT :      =.1
MATRIX CATNAP :       :     ,    LTCYL  :CATPLT :      =-.6
MATRIX LTCYL  :       :     ,    LTCYL  :CATPLT :      =1.
MATRIX HVYCYL :       :     ,    LTCYL  :CATPLT :      =-.5
MATRIX TOTAL  :CAT    :FEED ,    LTCYL  :CATPLT :      =1.2
MATRIX PROFIT :       :     ,    LTCYL  :CATPLT :      =.15
MATRIX CATNAP :       :     ,    HVYCYL :CATPLT :      =-.3
MATRIX LTCYL  :       :     ,    HVYCYL :CATPLT :      =-.7
MATRIX HVYCYL :       :     ,    HVYCYL :CATPLT :      =1.
MATRIX TOTAL  :CAT    :FEED ,    HVYCYL :CATPLT :      =1.1
MATRIX PROFIT :       :     ,    HVYCYL :CATPLT :      =.16
MATRIX HVYCLY :       :     ,    HVYCYL :FUEL   :      =1.
MATRIX RATIO  :FUEL   :     ,    HVYCYL :FUEL   :      =-9
MATRIX FUEL   :DEMAND :     ,    HVYCYL :FUEL   :      =1.
MATRIX PROFIT :       :     ,    HVYCYL :FUEL   :      =-2.5
MATRIX LTCYL  :       :     ,    LTCYL  :FUEL   :      =1.
MATRIX FUEL   :DEMAND :     ,    LTCYL  :FUEL   :      =1.
MATRIX PROFIT :       :     ,    LTCYL  :FUEL   :      =-2.5
MATRIX VDIST  :       :     ,    VDIST  :FUEL   :      =1.
MATRIX FUEL   :DEMAND :     ,    VDIST  :FUEL   :      =1.
MATRIX PROFIT :       :     ,    VDIST  :FUEL   :      =-2.5
MATRIX RESID  :       :     ,    RESID  :FUEL   :      =1.
MATRIX RATIO  :FUEL   :     ,    RESID  :FUEL   :      =1.
MATRIX FUEL   :DEMAND :     ,    RESID  :FUEL   :      =1.
MATRIX PROFIT :       :     ,    RESID  :FUEL   :      =-2.5
MATRIX VDIST  :       :     ,    VDIST  :DIST   :      =1.
MATRIX CONTAM :DIST   :     ,    VDIST  :DIST   :      =54
MATRIX DIST   :BAL    :     ,    VDIST  :DIST   :      =1.
MATRIX VNAP2  :       :     ,    VNAP2  :DIST   :      =1.
MATRIX CONTAM :DIST   :     ,    VNAP2  :DIST   :      =50
```

4

```
MATRIX  DIST    :BAL    :       ,    VNAP2   :DIST    :          =1.
MATRIX  LTCYL   :       :       ,    LTCYL   :DIST    :          =1.
MATRIX  CONTAM  :DIST   :       ,    LTCYL   :DIST    :          =65
MATRIX  DIST    :BAL    :       ,    LTCYL   :DIST    :          =1.
MATRIX  VNAP1   :       :       ,    VNAP1   :REGLR   :          =1.
MATRIX  OCTANE  :REGLR  :       ,    VNAP1   :REGLR   :          =85.
MATRIX  REGLR   :BAL    :       ,    VNAP1   :REGLR   :          =1.
MATRIX  VNAP2   :       :       ,    VNAP2   :REGLR   :          =1.
MATRIX  OCTANE  :REGLR  :       ,    VNAP2   :REGLR   :          =84.
MATRIX  REGLR   :BAL    :       ,    VNAP2   :REGLR   :          =1.
MATRIX  CATNAP  :       :       ,    CATNAP  :REGLR   :          =1.
MATRIX  OCTANE  :REGLR  :       ,    CATNAP  :REGLR   :          =92.
MATRIX  REGLR   :BAL    :       ,    CATNAP  :REGLR   :          =1.
MATRIX  VNAP1   :       :       ,    VNAP1   :PREM    :          =1.
MATRIX  OCTANE  :PREM   :       ,    VNAP1   :PREM    :          =85.
MATRIX  PREM    :BAL    :       ,    VNAP1   :PREM    :          =1.          4
MATRIX  VNAP2   :       :       ,    VNAP2   :PREM    :          =1.
MATRIX  OCTANE  :PREM   :       ,    VNAP2   :PREM    :          =84.
MATRIX  PREM    :BAL    :       ,    VNAP2   :PREM    :          =1.
MATRIX  CATNAP  :       :       ,    CATNAP  :PREM    :          =1.
MATRIX  OCTANE  :PREM   :       ,    CATNAP  :PREM    :          =92.
MATRIX  PREM    :BAL    :       ,    CATNAP  :PREM    :          =1.
MATRIX  CONTAM  :DIST   :       ,    DIST    :SPEC    :          =-55.
MATRIX  DIST    :BAL    :       ,    DIST    :SPEC    :          =-1.
MATRIX  DIST    :DEMAND :       ,    DIST    :SPEC    :          =1.
MATRIX  PROFIT  :       :       ,    DIST    :SPEC    :          =-4.0
MATRIX  OCTANE  :REGLR  :       ,    REGLR   :SPEC    :          =-85.
MATRIX  REGLR   :BAL    :       ,    REGLR   :SPEC    :          =-1.
MATRIX  REGLR   :DEMAND :       ,    REGLR   :SPEC    :          =1.
MATRIX  PROFIT  :       :       ,    REGLR   :SPEC    :          =-4.5
MATRIX  OCTANE  :PREM   :       ,    PREM    :SPEC    :          =-89.
MATRIX  PREM    :BAL    :       ,    PREM    :SPEC    :          =-1.
MATRIX  PREM    :DEMAND :       ,    PREM    :SPEC    :          =1.
MATRIX  PROFIT  :       :       ,    PREM    :SPEC    :          =-5.0
*                RHS        SEGMENT
RHS     CR2     :       :       ,    STIPUL  :ATIONS  :          =75.
RHS     VDIST   :       :       ,    STIPUL  :ATIONS  :          =11.
RHS     PSTILL  :CAP    :       ,    STIPUL  :ATIONS  :          =100.
RHS     LTCYL   :       :       ,    STIPUL  :ATIONS  :          =-11.
RHS     TOTAL   :CAT    :FEED   ,    STIPUL  :ATIONS  :          =46.         5
RHS     FUEL    :DEMAND :       ,    STIPUL  :ATIONS  :          =50.
RHS     DIST    :DEMAND :       ,    STIPUL  :ATIONS  :          =50.
RHS     REGLR   :DEMAND :       ,    STIPUL  :ATIONS  :          =10.
RHS     PREM    :DEMAND :       ,    STIPUL  :ATIONS  :          =25.
RHS     PROFIT  :       :       ,    STIPUL  :ATIONS  :          =-70.
END***
FILE    OLDBAS:SAMPLE:PROBLM
LGL     CR2,FUEL:DEMAND,CONTAM:DIST,REGLR:DEMAND,PREM:DEMAND,PROFIT
STR     CR1:PSTILL:DIST,CR2:PSTILL:DIST,VDIST:CATPLT,HVYCYL:CATPLT     6
STR     HVYCYL:FUEL,RESID:FUEL,VDIST:DIST,VNAP2:DIST,LTCYL:DIST
STR     VNAP2:REGLR,CATNAP:REGLR,VNAP1:PREM,VNAP2:PREM,CATNAP:PREM
STR     DIST:SPEC,REGLR:SPEC,PREM:SPEC
END***
$       DATA    AI                                                    1
FILE    REVISE:PT:FILE
RPLACE                                                                7
RHS     FUEL:DEMAND,STIPUL:ATIONS=75.0
END***
FILE    MODIFY:WORK:FILE
RPLACE                                                                8
RHS     PREM:DEMAND,STIPUL:ATIONS=125.0
END***
```

```
$       DATA    EI                                                        ①
FILE    DELIMT:FILE:ROWS
MASK    *:DEMAND:*,*:BAL:*,OCTANE:*,CONTAM:DIST                           ⑨
END***
FILE    DELIMT:FILE:COLS
MASK    CR*:*:*,*:FUEL,*:DIST,*:SPEC                                      ⑩
END***
$       DATA    I*                                                        ①
        PREPROCESS
        TITLE   INTRODUCTION TO LP/600 SAMPLE PROBLEM
        CONVERT SOURCE=LP600:SAMPLE:PROBLM/IN,IDENT=PTF
        SETUP   SOURCE=PTF
        LDBASIS SOURCE=OLDBAS:SAMPLE:PROBLM/IN                            ⑪
        SET     OBJ=PROFIT,RHS=STIPUL:ATIONS
        DELIMIT ALL
        TRACE   DELIMIT,XO
        PRIMAL
        OUTPUT
* REVISE PROBLEM AND RESOLVE
        REVISE  SOURCE=REVISE:PT:FILE/AI,IDENT=NEWPTF,PFILE=PTF
        SETUP   SOURCE=NEWPTF
        SET     OBJ=PROFIT,RHS=STIPULATIONS                               ⑫
        DELIMIT ALL,RLIST=DELIMT:FILE:ROWS/EI
        TRACE   DELIMIT,XD
        PRIMAL
        OUTPUT
* MODIFY PROBLEM AND RESOLVE
        MODIFY  SOURCE=MODIFY:WORK:FILE/AI
        DELIMIT CLIST=DELIMT:FILE:COLS/EI,
                RLIST=DELIMT:FILE:ROWS/EI                                 ⑬
        TRACE   DELIMIT,XO
        PRIMAL
        OUTPUT
        ENDLP
        EXECUTE
$       CONVER
$       TAPE    XO,A2R,,,XO-OUTPUT                                        ⑭
$       INPUT   NLABEL
$       PRINT   OT,A2R
$       ENDJOB
***EOF                                                                    ①
```

Minimum GE-600 System
Configuration: 72K 6 Tapes
64K Disc System

APPENDIX D.
PROBLEM SIZE AND STORAGE REQUIREMENTS

does not include any operating system requirements

CORE REQUIREMENTS FOR NORMAL LINEAR PROGRAM SOLUTION

Problem Rows (M)	Minimum Storage (1)	Minimum Storage (2)	Approximate Storage for In-Core Solution					
			N=2M			N=4M		
			Z=3	Z=10	Z=15	Z=3	Z=10	Z=15
100	20,000	20,000	21,000	31,000	36,000	24,000	36,000	45,000
200	14,000	15,000	31,000	49,000	61,000	36,000	60,000	78,000
300	15,000	16,000	40,000	68,000	85,000	48,000	84,000	111,000
500	18,000	18,000	59,000	105,000	134,000	72,000	132,000	176,000
700	20,000	21,000	77,000	142,000	183,000	96,000	180,000	242,000
1,000	23,000	25,000	104,000	197,000	256,000	132,000	252,000	-
1,500	29,000	32,000	144,000	-	-	192,000	-	-
2,000	35,000	39,000	196,000	-	-	252,000	-	-
2,500	40,000	45,000	242,000	-	-	-	-	-
3,000	46,000	52,000	-	-	-	-	-	-
3,500	51,000	58,000	-	-	-	-	-	-
4,000	57,000	65,000	-	-	-	-	-	-

(1) Without PARRIM, PARCOL, PARROW, RNGSOL, (N=2M)

(2) With PARRIM, PARCOL, PARROW, RNGSOL, (N=2M)

M=Problem Size (Rows)
N=No. Structural Columns
Z=Avg. no. nonzeros per column

CORE REQUIREMENTS FOR DECOMP

Total Rows (M)	Rows In Master + Largest Submatrix				
	200	500	1000	2000	4000
500	14,400	-	-	-	-
1,000	14,400	17,000	-	-	-
2,000	16,400	17,000	22,000	-	-
5,000	22,400	23,000	24,000	32,000	52,000
10,000	32,400	33,000	34,000	36,000	52,000
15,000	42,400	43,000	44,000	46,000	52,000
20,000	52,400	53,000	54,000	56,000	60,000
25,000	62,400	63,000	64,000	66,000	70,000
50,000	112,400	113,000	114,000	116,000	120,000

APPENDIX E.
AGENDA CONTROL LANGUAGE VERB FORMATS

```
[label]   ABOUND   [blank or comments]

[label]   AFEAS   RHS=colname [,CRHS=colname]

                  ⎧AE⎫
[label]   BLANK   ⎨IN⎬
                  ⎨EI⎬
                  ⎩PT⎭

                                                ⎧SO
                                    ⎧PT⎫        ⎨XO                    ⎧IN⎫ ⎫
[label]   CALC    SOURCE=filename [/⎨  ⎬ ]  [,  ⎨IDENT=filename [/     ⎨AI⎬ ⎬ ]
                                    ⎩XP⎭        ⎨                      ⎩EI⎭ ⎭
                                                ⎩

                    ⎧                                   ⎫
                    ⎪ALL                                ⎪
                    ⎪CLIMIT=colname/colname             ⎪
[label]   COLOUT   [⎨CMASK=mask [/mask..]               ⎬]
                    ⎪                     ⎧EI⎫          ⎪
                    ⎪CLIST=filename [/    ⎨AI⎬ ]        ⎪
                    ⎩                     ⎩IN⎭          ⎭

[label]   COMPUTE   symbol = arithmetic expression

                                       ⎧EI⎫
[label]   CONVERT   SOURCE=filename [/ ⎨IN⎬ ] ,IDENT=filename [,EDIT]
                                       ⎩AI⎭

                    [,TRANSP] [, ⎧SELROW⎫ ] [, ⎧SELCOL⎫ ]
                                 ⎩CHKROW⎭      ⎩CHKCOL⎭
                   ⎧ALL                     ⎫
[label]   CRASH   [⎨CLIMIT=colname/colname  ⎬]
                   ⎩DELIMIT                 ⎭

[label]   CURRENT   [comments]

 label    DC   value [,value...]

[label]   DCINV   [comments]

[label]   DECOMP   [comments]

                                       ⎧EI⎫
[label]   DEFINE   SOURCE=filename [/  ⎨IN⎬ ] ,IDENT=filename
                                       ⎩AI⎭
```

```
                         ALL                                ALL
                         CLIMIT=colname/colname             RLIMIT=rowname/rowname
                         CMASK=mask [/mask...]              RMASK=mask [/mask...]
[label]  DELIMIT [     {                     EI    } ] [  {                     EI    } ]
                         CLIST=filename [/ { IN } ]         RLIST=filename [ { IN } ]
                                             AI                                AI
                         NONE                               NONE

                                           PT
[label]  DELOAD  SOURCE=filename [ / {        } ]
                                           XP

[label]  DESAVE  IDENT=filename

[label]   DISPLAY   symbol [,symbol...]

                   ON       ON
 label    DSW    {     } [,{     }] [ ...]
                   OFF      OFF

[label]  DUAL  [blank or comments]

                                                                   EI
                                  EI             7090              IN
[label]  EDIT  SPEC=filename [/ { AI } ] ,TYPE= { 7040 } ,OUTPUT= { AI }
                                  IN             GE                SO
                                                                   XO

 blank   ENDM  [comments]

[label]  ENDLP  [comments]

         EXECUTE

                        ALL                                ALL
                        CLIMIT=colname/colname             RLIMIT=rowname/rowname
[label]  FLAGOUT      { CMASK=mask [/mask...]          } ,[{ RMASK=mask [/mask...]          } ]
                                             EI                                 EI
                        CLIST=filename [/ {  IN } ]        RLIST=filename [/ {  IN } ]
                                             AI                                 AI
                        NONE                               NONE

                        ALL                              ALL
[label]  FORCE    [   { CLIMIT=colname/colname } ] [   { RLIMIT=rowname/rowname } ]
                        CMASK=mask [/mask...]            RMASK=mask [/mask...]
                        NONE                             NONE

[label]  GOTO   label

[label]  GROUP  [comments]
```

CPB-1141B

```
[label]   INVERT   comments

                     ┌ ALL                              ┐
                     │ RLIMIT=rowname/rowname           │
[label]   INVOUT   [ ┤ RMASK=mask [/mask...]            ├ ]
                     │                     ┌EI┐         │
                     │ RLIST=filename [/  ┤AI├ ]        │
                     └                     └IN┘         ┘

[label]   JUMP   [comments]

                                       ┌EI┐
[label]   LDBASIS   SOURCE=filename [/ ┤IN├ ]
                                       └AI┘

[label]   LOGIC   [symbol =] logical expression [, label]

[label]   MACRO   name

                                      ┌EI┐
[label]   MATGEN   SOURCE=filename [/ ┤IN├ ] ,IDENT=filename
                                      └AI┘

                                      ┌EI┐
[label]   MODIFY   SOURCE=filename [/ ┤IN├ ]
                                      └AI┘

[label]   MOVE   Symbol 1 [+ offset] = symbol 2 [+ offset], n

[label]   NEXT   [comments]

[label]   NOTE   comments

                   ┌DELIMIT┐     ┌ SO                            ┐
[label]   OUTPUT [ ┤PDSOL  ├ ] [,┤ XO                  ┌IN┐      ├ ]
                   └ALL    ┘     │ IDENT=filename [/  ┤AI├ ]    │
                                 └                     └EI┘      ┘

[label]   PARCOL   [comments]

[label]   PAROBJ   [comments]

[label]   PARRHS   [comments]

[label]   PARRIM   [comments]

[label]   PARROW   [comments]
```

```
  [label]   PERFORM   label

                       ⎧ALL                                            ⎫
                       ⎪DELIMIT                                        ⎪
  [label]   PICTURE  [ ⎨BASIS [,RLIMIT=rowname/rowname]                ⎬ ] [, {SO} ]
                       ⎪ACTIVE [,RLIMIT=rowname/rowname]               ⎪      {XO}
                       ⎪RLIMIT=rowname/rowname                         ⎪
                       ⎩CLIMIT=colname/colname [,RLIMIT=rowname/rowname]⎭

   blank    PREPRO    [comments]

✓ [label]   PRIMAL    [comments]

                       ⎧                          ⎧IN⎫    ⎫
✓ [label]   PUNCH    [ ⎨SO                        ⎨AI⎬  ] ⎬ ]
                       ⎪IDENT=filename  [ /       ⎩EI⎭    ⎪
                       ⎩                                  ⎭

                   ⎧P=rowname⎫
                   ⎪L=rowname⎪
✓ [label]   RECORD ⎨X=colname⎬ [,...]
                   ⎪D=colname⎪
                   ⎩C=colname⎭

                      ⎧ALL                   ⎫     ⎧ALL                   ⎫
✓ [label]   REMOVE  [ ⎨CLIMIT=colname/colname⎬ ] [ ⎨RLIMIT=rowname/rowname⎬ ]
                      ⎪CMASK=mask [/mask...] ⎪     ⎪RMASK=mask [/mask...] ⎪
                      ⎩NONE                  ⎭     ⎩NONE                  ⎭

                     ⎧ALL                  ⎫
                     ⎪TOLS                 ⎪
                     ⎪ARGS                 ⎪
  [label]   RESET  [ ⎨PARS                 ⎬ ] [,...]
                     ⎪FRQS                 ⎪
                     ⎪TOGS                 ⎪
                     ⎪DEMS                 ⎪
                     ⎩solution control name⎭

✓ [label]   RESTORE   SOURCE=filename [ / {PT} ]
                                          {XP}
                                          ( EI )
[label]   REVISE   SOURCE=filename [/ {  AI  } ] ,IDENT=filename
                                          ( IN )

                   PFILE=filename [/ {  PT  } ]
                                        XP

[label]   RNGAIJ   rowname,colname

                     ( ALL                              )
                     ( CLIMIT=colname/colname           )
[label]   RNGOBJ   [ { CMASK=mask [/mask...]            } ]
                     (                      ( EI )      )
                     ( CLIST=filename [/ {  AI  } ]     )
                     (                      ( IN )      )

                     ( ALL                              )
                     ( RLIMIT=rowname/rowname           )
[label]   RNGRHS   [ { RMASK=mask [/mask...]            } ]
                     (                      ( EI )      )
                     ( RLIST=filename [/ {  AI  } ]     )
                     (                      ( IN )      )

                     ( ALL                              )
                     ( CLIMIT=colname/colname           )
[label]   RNGSOL   [ { CMASK=mask [/mask...]            } ]
                     (                      ( EI )      )
                     ( CLIST=filename [/ {  AI  } ]     )
                     (                      ( IN )      )

                     ( ALL                              )
                     ( RLIMIT=rowname/rowname           )
[label]   ROWOUT   [ { RMASK=mask [/mask...]            } ]
                     (                      ( EI )      )
                     ( RLIST=filename [/ {  AI  } ]     )
                     (                      ( IN )      )

[label]   SAVE     [ { BASIS                        } ]
                     { IDENT=filename [,BASIS]      }
```

```
                  ┌parameter=value                      ┐
                  │frequency=value                      │
                  │tolerance=value                      │
                  │            ┌rowname        ┐        │
[label]    SET   <  argument = < colname        >      > [,...]
                  │            └colname/colname┘        │
                  │toggle = {ON }                       │
                  │         {OFF}                       │
                  └demand = label                       ┘

[label]    SETUP    SOURCE=filename [/{PT}] [,REGNS=value]
                                      {XP}
                    [,AUTOSCALE] [,KJLIST] [,TROBJ=rowname]

[label]    STATUS    [comments]

[label]    TABULATE    [{SO}]
                        {XO}

[label]    TALLY    label, xloc, value, increment

[label]    TITLE    [alphameric run title]

[label]    TRACE    [{INFEAS }] [,{SO}]
                     {DELIMIT}     {XO}

[label]    TRANSP    [comments]

                   ┌ALL                           ┐   ┌ALL                           ┐
                   │CLIMIT=colname/colname        │   │RLIMIT=rowname/rowname        │
                   │CMASK=mask [mask...]          │   │RMASK=mask [/mask...]         │
[label]    UNFLAG [<                   {EI}       >] [<                   {EI}       >]
                   │CLIST=filename [/ {AI}]       │   │RLIST=filename [/ {AI}]       │
                   │                   {IN}       │   │                   {IN}       │
                   └NONE                          ┘   └NONE                          ┘

                   ┌TB┐
                   │PT│
                   │XP│
[label]    UNLOAD  <XO >
                   │AI│
                   │IN│
                   └EI┘
```

VERBS	VARIABLE FIELDS
EDIT	SOURCE=name/f, IDENT=name/u
$\left\{\begin{array}{l}\text{DELIMIT}\\\text{FLAGOUT}\\\text{UNFLAG}\end{array}\right\}$	$\left[\left\{\begin{array}{l}\text{CLIMIT=name/name}\\\text{CMASK=mask/}\ldots\\\text{CLIST=file [/f]}\\\underline{\text{ALL}}\end{array}\right\}\right]\left[,\left\{\begin{array}{l}\text{RLIMIT=name/name}\\\text{RMASK=mask/}\ldots\\\text{RLIST=file [/f]}\end{array}\right\}\right]$
TITLE	Any string of legal BCD characters, columns 16-69.
UNLOAD	$\left\{\begin{array}{l}\text{TB}\\\text{PT}\\\text{XP}\\\text{XO}\\\text{AI}\\\text{IN}\\\text{EI}\end{array}\right\}$ (only at end of problem)

The following verbs require no variable field:

STATUS
PRIMAL
INVERT
PARRHS
PAROBJ
PARRIM
PARCOL
PARROW
DUAL
TRANSP
GROUP
DECOMP
ABOUND

For syntactic purposes, there are several subgroupings of the reserved files indicated by f and u. The f files are linked input files, and the u files are linked output files, as follows:

Phase	Meaning
AFEAS	Construct a pseudo RHS for which the current basis is feasible
ALL	Use all vectors
AUTOSCALE	Scale all unscaled rows and columns
BASIS	Use only the basis
BLANKS	Do not suppress blank lines
CLIMIT	Column limits
CLIST	Same column list source
CMASK	Column masks
DELIMIT	Use the delimit set only
IDENT	Identify output
INFEAS	Use only the infeasible set
KJLIST	Produce the list of names versus vector numbers as output
NONE	Use nothing
PDSOL	Use only nonzero prime or dual number values
PRHS	Pseudo right-hand side
REGNS	The maximum number of regions to be used
RLIMIT	Row limits
RLIST	Row list source
RMASK	Row masks
SELCOL	Select only those columns with STR cards
SELROW	Select only those rows with LGL cards
SOURCE	The source of the input
TRANSP	The matrix is in the special transportation problem format giving only cost matrix by demands and supplies.

Figure 24. Variable Field Vocabulary

APPENDIX F.
FORMAT GENERATOR LANGUAGE VERB STATEMENTS

```
 label     ALPHA     n,data

                                 ⎧position,n⎫
[label]    COMPUTE   [name] [/   ⎨          ⎬] = arithmetic expression
                                 ⎩position  ⎭

 blank     ENDM      [comments]

[label]    EXIT      [comments]

[label]    GOTO      label

[label]    INSERT    label,position [.n]

[label]    LINOUT    [cc] [,label]

[label]    LOGIC     [switch =] logical expression [,label]

[label]    MACRO     name

 label     NAME      argument #1 [;argument #2;...argument #9]
   (The above format is for a MACRO call statement.)

[label]    NEXT      [comments]

[label]    PERFORM   label

[label]    SCALE     prefix=constant [,...]
```

APPENDIX G.
MATRIX GENERATOR LANGUAGE VERB FORMATS

```
Cols.  1-6       8-15               16-72

     [label]   ALTER    list [,index1] =list2, index2

     [label]   COMPOSE   list3 [/cc1' [,cc2']] =list1 [/cc1]

                             ⎧MATCH  ⎫
                          [( ⎨NOMATCH⎬ )list2 [/cc2] ,DUP =
                             ⎨UNION  ⎬
                             ⎩DIFFER ⎭

                             ⎧ERROR  ⎫
                             ⎨DELETE ⎬
                             ⎩SURVIVE⎭

                          ⎧ symbol           ⎫
     [label]   COMPUTE    ⎨ table reference  ⎬ = arithmetic expression
                          ⎩ string reference ⎭

     [label]   DECLARE   table3=list1,list2

                          ⎧LEXT (list name)  ⎫
     [label]   DIMEN   LOC=⎨CEXT (table name) ⎬
                          ⎨REXT (table name) ⎬
                          ⎩SEXT (string name)⎭

                         ⎧LIST   ⎫
                         ⎨STRING ⎬
     [label]   DUMP    [ ⎨TABLE  ⎬ ] [= name]
                         ⎨SETCON ⎬
                         ⎨SETSW  ⎬
                         ⎩NAME   ⎭

      blank    ENDM    [comments]
```

```
Cols.   1-6       8-15                    16-72

        [label]   ENFILE    { TABLE  } = name [, { TABLE  } = name,...]
                            { STRING }           { STRING }

        [label]   ENPAC     stringname = tablename [/T]

                            { LIST   }          { LIST   }
        [label]   ERASE     { STRING } =name [, { STRING } = name,...]
                            { TABLE  }          { TABLE  }

        [label]   EXIT    [comments]

        [label]   EXPAND   tablename = stringname [/T]

                                       { tablename/  { STUB    } }
        [label]   EXTRACT   listname = {             { HEADING } }
                                       { stringname/ { PART1   } }
                                       {             { PART2   } }

                                                              {         { +       } }
                                                              {         { -       } }
                                                              {         { *       } }
        [label]   FUNCT   str3 [/r:c1'] = str1 [/r:c1[ { (MATCH { /       } ) }
                                                              {         { MIN     } }
                                                              {         { MAX     } }
                                                              {         { REPLACE } }
                                                              { NOMATCH             }

                                                          { ERROR   }
                                                          { DELETE  }
                                                          { SURVIVE }
                                      str2 [/r:c2] [, { REPLACE }]]
                                                          { SUM     }
                                                          { PROD    }
                                                          { MIN     }
                                                          { MAX     }

        [label]   GOTO      label

        [label]   INDEX     location= <tablename, { rowname,0 } >
                                                  { 0,colname }

        [label]   JOIN      ident=list1:list2:list3

                                      { (ZERO)        }
                                      { (PLUS)        }
        [label]   LGL       rowname [ { (MINUS)       } ] [,SCALE=value]
                                      { (FREE)        }
                                      { (RANGE=value) }

        [label]   LIST      name = entry1 [,entry2,...]

        [label]   LOGIC     [switch=] logical expression [,label]

        [label]   MACRO     name

        [label]   name      argument#1 [;argument#2;...argument#n]
            (The above format is for a macro call statement.)
```

Cols.	1-6	8-15	16-72
	[label]	MOVE	label1 [+offset1] = label2 [+offset2] ,n
	[label]	NAME	location = n,data
	[label]	NEXT	[comments]
	[label]	PERFORM	label
	[label]	QUALS	modifier = tablename
	[label]	READ	SOURCE = filename/$\left\{\begin{matrix}\text{PT}\\\text{XP}\\\text{EI}\\\text{AI}\\\text{IN}\end{matrix}\right\}$
	[label]	RENAME	tablename [/$\left\{\begin{matrix}\underline{\text{STUB}}\\\text{HEADING}\end{matrix}\right\}$] =listname
	[label]	RHS	rhs,row1=value,row2=value,...
	[label]	ROW	row,col1=value,col2=value [,...]
	label	SETCON	symbol=value [,value,...]
	[label]	SETSW	symbol =$\left\{\begin{matrix}\underline{\text{ON}}\\\text{OFF}\end{matrix}\right\}$ [,$\left\{\begin{matrix}\underline{\text{ON}}\\\text{OFF}\end{matrix}\right\}$,...]
	[label]	STR	colname [$\left\{\begin{matrix}\text{ZERO}\\\underline{\text{PLUS}}\\\text{MINUS}\\\text{(FREE)}\\\text{RANGE = value,value}\end{matrix}\right\}$] [SCALE = value]
			[,TRANSL = value]
	[label]	STRING	name = part11:part21-value1, [part12:part22=value2,...]
	[label]	SUBMAT	row modifier, col modifier [($\left\{\begin{matrix}\underline{\text{MATRIX}}\\\text{RHS}\end{matrix}\right\}$)] = $\left\{\begin{matrix}\text{< T,R,C >}\\\text{< S,R,C >}\end{matrix}\right\}$
	[label]	SYNTH	modifier = tablename
	[label]	TABLE	name = head1 ,head2 ,...
	blank	blank	stub1 = value(1,1) [,value (1,2),...]
	blank	blank	stub2 = value(2,1) ,value (2,2),...]
	[label]	TALLY	label,index,constant [,increment]
	[label]	TRANSP	modifier = tablename (costrow)
	[label]	VECT	column,row1 = value,row2 = value,...
	[label]	YIELD	modifier = tablename

APPENDIX H.
CARD TYPES, FORMATS, AND USAGE

```
Col.  1-6      8-72                                                                 Usage

      BLOCK    block name                                                           Convert File
                                             ( (S) )
      CFORM    colname= [constant] ,colname [{ (L) }] *value [,...]                 Revise File
                                             ( (B) )

                                             ( (S) )
      CFORM    rhsname= [constant] ,colname [{ (L) }] *value [,...]                 Modify File
                                             ( (B) )

              (L [GL] =rowname [,rowname]   )
              (S [TR] =colname [,colname]   )
      DELETE {   (  B )                      }                                      Revise File
              ( {      } =colname [,colname] )
              (  ( RHS)                     )

      END***   blank or comments                                                    All Files

      ENDGRP   blank or comments                                                    Convert File

      FILE     filename                                                             All Files

              (L [GL] =rowname )
              (S [TR] =colname )
      INSERT {   (  B )         }                                                   Revise File
              ( {      } =colname)
              (  ( RHS)         )

      INSERT                                                                        Modify File

      LAB      rowname [,...]                                                       Basis File

                          ((Z [ERO ])         )
       ( RHS )            ((P [LUS ])         )
      {      }  rowname [{(M [INUS])          }],colname=value [,COUNT=value]       Convert File
       (  B  )            ((F [REE])          )
                          ((R [ANGE] =value)  )

                          ((Z [ERO])          )
       ( RHS )            ((P [LUS])          )
      {      }  rowname [{(M [INUS])          }] =value [,COUNT=value]              Convert File
       (  B  )            ((F [REE])          )
                          ((R [ANGE] =value)  )

       ( RHS )
      {      }  ,colname=value [,COUNT=value]                                       Convert File
       (  B  )

      RFORM    rowname [(XRHS)] = [constant],rowname*value [,...]                   Revise File

      RPLACE                                                                Revise & Modify File

      SAB      colname [,...]                                                       Basis File

      STR      colname [,...]                                                       Basis File
```

```
Col.    1-6       8-72                                                                              Usage

                         ⎧(P [LUS])   [,SCALE=value]  [,COUNT=value]  [,TRANSL=value]⎫
                         ⎪(Z [ERO])   [,SCALE=value]  [,COUNT=value]  [,TRANSL=value]⎪
        ⎧STR⎫            ⎪(M [INUS])  [,SCALE=value]  [,COUNT=value]  [,TRANSL=value]⎪
        ⎩ S ⎭   colname  ⎨(F [REE])   [,SCALE=value]  [,COUNT=value]                 ⎬]          Convert File
                         ⎪(R [ANGE] =value [,value])  [,COUNT=value]                 ⎪
                         ⎪(G [ROUP] =value)                                          ⎪
                         ⎩( [PACKE]T =value)                                         ⎭

                           ⎧(Z [ERO])  [,SCALE=value] ⎫
        ⎧LGL⎫              ⎪(P [LUS])  [,SCALE=value] ⎪
        ⎩ L ⎭   rowname [  ⎨(M [INUS]) [,SCALE=value] ⎬] [,COUNT=value]                              Convert File
                           ⎪(F [REE])                 ⎪
                           ⎩(R [ANGE] =value [,value])⎭

        LGL     rowname [,...]                                                                      Basis File

                ⎧rowname  ⎫
        MASK    ⎨colname  ⎬[,...]                                                                   List File
                ⎩name-mask⎭

                          ⎧(Z [ERO])         ⎫             ⎧(P [LUS])                 ⎫
        ⎧MATRIX⎫          ⎪(P [LUS])         ⎪             ⎪(Z [ERO])                 ⎪
        ⎩  A   ⎭ rowname [⎨(M [INUS])        ⎬],colname [  ⎨(M [INUS])                ⎬]=value      Convert File
                          ⎪(F [REE])         ⎪             ⎪(F [REE])                 ⎪
                          ⎩(R [ANGE] =value) ⎭             ⎩(R [ANGE] =value [,value])⎭

                          ⎧(Z [ERO])         ⎫
        ⎧MATRIX⎫          ⎪(P [LUS])         ⎪
        ⎩  A   ⎭ rowname [⎨(M [INUS])        ⎬] =value                                              Convert File
                          ⎪(F [REE])         ⎪
                          ⎩(R [ANGE] =value) ⎭

                           ⎧(P [LUS])                 ⎫
        ⎧MATRIX⎫           ⎪(Z [ERO])                 ⎪
        ⎩  A   ⎭ ,colname [⎨(M [INUS])                ⎬] =value                                     Convert File
                           ⎪(F [REE])                 ⎪
                           ⎩(R [ANGE] =value [,value])⎭
```

DOCUMENT REVIEW SHEET

TITLE: Introduction to LP/600

CPB #: 1141B

FROM:

Name: ____________________

Position: ____________________

Address: ____________________

____

Comments concerning this publication are solicited for use in improving future editions. Please provide any recommended additions, deletions, corrections, or other information you deem necessary for improving this manual. The following space is provided for your comments.

COMMENTS: ____________________

Please cut along this line

NO POSTAGE NECESSARY IF MAILED IN U.S.A.
Fold on two lines shown on reverse
side, staple, and mail.

STAPLE STAPLE

FOLD

FIRST CLASS
PERMIT, No. 4332
PHOENIX, ARIZONA

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY

GENERAL ELECTRIC COMPANY
COMPUTER EQUIPMENT DEPARTMENT
13430 NORTH BLACK CANYON HIGHWAY
PHOENIX, ARIZONA - 85029

ATTENTION: DOCUMENTATION STANDARDS AND PUBLICATIONS B-90

FOLD

Progress Is Our Most Important Product

GENERAL ELECTRIC

INFORMATION SYSTEMS DIVISION

LITHO U.S.A.