IBM

Application Program

GH20-0490-1

Problem Language Analyzer (PLAN)

(1130-CX-25X, 360A-CX-26X, 360A-CX-27X)

Application Description Manual

The IBM Problem Language Analyzer (PLAN) assists IBM and customer development of a variety of problem-solving computer applications. The value of a computer installation to technical, professional, and management personnel can be significantly increased, using PLAN. Programmers and system designers will also find that PLAN offers new tools for building interactive or fast-changing applications.

This is an introductory manual. The scope, advantages, features, and use of PLAN are presented in only enough detail to permit the reader to judge the impact of PLAN in his own environment. Additional detail will be found in formal documentation of the PLAN programs, and in program descriptions of IBM-written, PLAN-based applications.

CONTENTS

Second Edition (January 1969, reprinted June 1970

This edition is a major revision obsoleting H20-0490-0.

This edition applies to Version 1, Modification Level 0 of Problem Language Analyzer (PLAN) (1130-CX-25X, 360A-CX-26X, 360A-CX-27X), and to all subsequent versions and modifications until otherwise indicated in new editions or Technical Newsletters.

Changes are continually made to the specifications herein. Therefore, before using this publication, consult the latest 1130 and System/360 SRL Newsletters (N20-1130, N20-0360) for the editions that are applicable and current.

Copies of this and other IBM publications can be obtained through IBM branch offices. Address comments concerning the contents of this publication to: IBM, Technical Publications Department, 112 East Post Road, White Plains, N. Y. 10601.

INTRODUCTION

The IBM Problem Language Analyzer (PLAN) was originally developed as an internal tool to help IBM produce engineering applications for its 1130 Computing System that would be easy for a variety of customers to use, change, or expand.

This manual describes PLAN for both the IBM 1130 and the IBM System/360 computers. Those who intend to implement problem-solving or interactive applications will find that PLAN offers economy and modularity.

The main external feature of PLAN is its support of a variety of IBM-and customer-written problem-solving languages. The main internal feature of PLAN is a resident program loader, or interaction supervisor, that allows program modules to run together and communicate without completely preplanned linkage editing.

WHAT IS A PROBLEM-SOLVING LANGUAGE?

Problem-solving languages are better known as POL's (problem-oriented languages). They are often confused with programming languages, such as FORTRAN; but the difference is very important.

A problem-solving language is designed to be used by people who are not necessarily familiar with computer programming. Therefore, the terminology of any particular problem-oriented language is specific and familiar to one group of people, one business, or one discipline. In contrast, a programming language uses general terms to accommodate trained programmers working in a variety of different application areas.

Both kinds of language exist and are required. Often, it is difficult to classify a particular language. The best test of a problem-oriented language is its objective: to produce an answer. Programming languages are designed to produce a program, which will later be used to produce an answer.

WHAT IS NEW IN PLAN?

Problem-oriented language processors, in the past, were created to handle one specific vocabulary, which then was used without change by many different groups of people. PLAN allows each group to vary, expand, or even combine its problem-solving languages.

PLAN allows each part of a problem-solving language to be changed independently. Language changes do not require changes to the PLAN processor. The one processor handles many different vocabularies.

In the past, the set of programs supporting a language was also linked within a closed system. One program (the mainline) attempted to define all of the options that were open to the user of the input language. With PLAN, this mainline program is not required; it is replaced by the PLAN loader and input interpreter.

When the total scope of an application was predefined, all details of communication between subprograms could be specified within the programs themselves. Since PLAN does not presume a closed family of subprograms, many details of communication must be resolved during the solution of each specific problem. A large number of FORTRAN subroutines are provided within PLAN to allow production of subprograms that permit communication and linkages to change during execution.

Taken together, the new features of PLAN allow one to evolve and adapt problem-solving languages and their supporting computer subprograms. PLAN encourages the development of more complex interactive applications by lowering the cost of change and growth.

USER RESPONSIBILITY

An individual using a calculating device may select any series of procedures that seems appropriate. The value of the result can be judged only by the person who produced it. Similarly, the PLAN processor will accept many different series of potentially meaningful input statements. The validity of results must be judged by the user, as in any other process.

Substantial input discipline and checking can be imposed within PLAN-based problem-oriented languages; but the user retains an unusual amount of freedom, and is finally responsible for his calculation logic.

Note: In this manual the terms "application", "language", "program", "designer", "programmer", and "user" do not have their usual meanings. To avoid confusion, the intended meanings of these terms and their synonyms are:

Program (module, link, phase, subprogram). A named member of the library of executable code, that follows PLAN conventions.

Language (vocabulary, command set, dictionary). Any collection of predefined entities used within the PLAN grammar. This differs from a natural language by having fixed grammar; and from well-known programming languages or POL's (FORTRAN, APT) by having an open vocabulary.

Application. In PLAN, any useful collection of language definitions and modular programs that are invoked by those definitions. Additions, deletions, and substitutions are expected.

Designer. One who proposes and/or defines new PLAN phrases and identifies programming requirements. This includes data requirements, input validity testing, normal program module sequences, and default options. Knowledge of the available (or feasible) program module library is presumed. An active role in inventing the application input language is usual.

Programmer. One who is trained to produce effective modular codes (usually via FORTRAN). Use of PLAN conventions and subroutines is assumed.

User. One who uses the language (and indirectly uses the programs) provided for him by designers and programmers. Ideally, he could be unaware that his problem description is to be processed by a computer. In practice, the user will have to be kept advised of the available options and grammatical restrictions in his problem-solving languages.

USING PLAN IN APPLICATIONS

From an economic point of view, the most important opportunities for using PLAN are in the area of the half-hour to half-day manual calculation sequences that fill so much of the working day in a technical environment. Good examples are the IBM-written PLAN applications, such as gear design, spring design, linkage motion analysis, structural element design, data plotting, machine tool productivity, and lens system analysis.

Examination will show that each of these contains a family of procedures that are selected and combined in many different ways to satisfy particular problems, as they occur.

Discussions in this section are intended to show how PLAN-based applications look to a user. They are not intended as a guide for choosing applications or designing problem-oriented language. Choosing PLAN applications and designing the problem-oriented languages for those applications is relatively easy, once the particular business environment is known.

To illustrate the use of PLAN in an application, assume that a company has a computer installation and wants its pump engineering department to benefit. One example of the routine work in the department could be a request to quote price and delivery for 50 special sump pumps to lift oil field drilling mud.

Given this problem, the pump designer is not expected to design an entirely new pump body, impeller, gear set, and motor mount. To be competitive and to give short delivery, he would try to select an existing pump design and power supply; then, he would simply design a new mounting. For the purpose of this example, assume the designer's solution process is the following:

1. Search the displacement pump catalog for the lowest-cost body that can satisfy the lift and delivery requirement.

2. Using the intake pipe diameter and rated efficiency of this pump, calculate its power requirement.

3. Search the motor catalog for a moistureproof electric motor that will deliver at least the calculated power. Choose the lowest-cost item.

4. Consider other options, for example:
   Would a more expensive pump have a smaller power requirement, allowing for a cheaper motor?

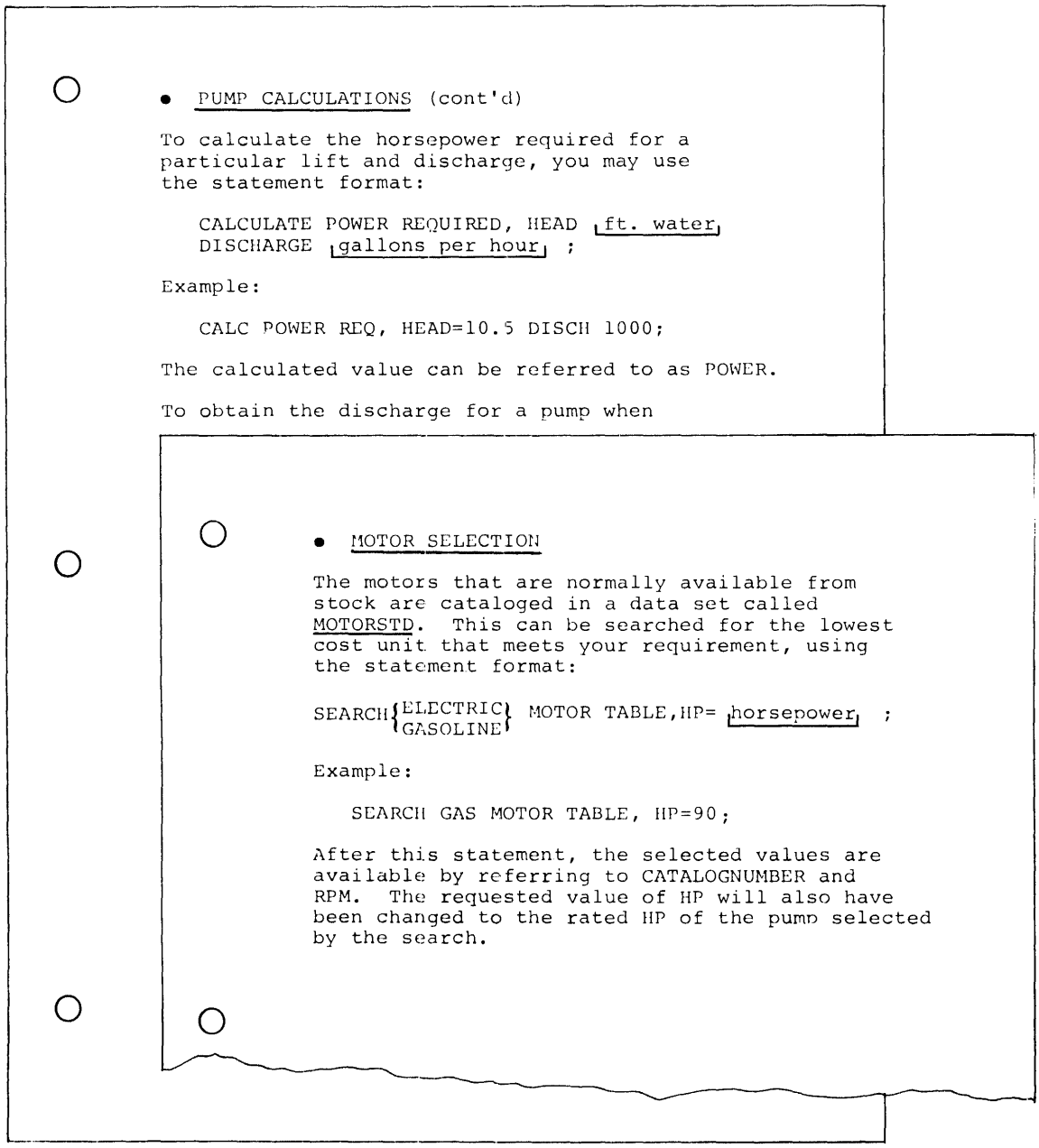
5. Evaluate the designs that have been produced. Select one pump and motor.

6. Design a gear set to mate the motor drive-shaft to the pump crankshaft.

7. Prepare a quotation and technical data sheet for the customer.

This manual procedure is one choice available to the designer. Another possibility is that he has access to a library of computer programs for pump design, motor design, gear design, etc. To use this resource, the designer would fill out a data sheet, for each part of his problem, and submit it for processing. As each part would be processed, the results would be used to prepare input for subsequent runs. Such a series of runs may easily take longer than a manual solution.

If these same computer library programs were arranged to operate under PLAN, the designer would refer to a book of standards, like those in Figure 1.



• PUMP CALCULATIONS (cont'd)

To calculate the horsepower required for a particular lift and discharge, you may use the statement format:

CALCULATE POWER REQUIRED, HEAD ⌊ft. water⌋
DISCHARGE ⌊gallons per hour⌋ ;

Example:

CALC POWER REQ, HEAD=10.5 DISCH 1000;

The calculated value can be referred to as POWER.

To obtain the discharge for a pump when

• MOTOR SELECTION

The motors that are normally available from stock are cataloged in a data set called MOTORSTD. This can be searched for the lowest cost unit that meets your requirement, using the statement format:

SEARCH{ELECTRIC / GASOLINE} MOTOR TABLE,HP= ⌊horsepower⌋ ;

Example:

SEARCH GAS MOTOR TABLE, HP=90;

After this statement, the selected values are available by referring to CATALOGNUMBER and RPM. The requested value of HP will also have been changed to the rated HP of the pump selected by the search.

Figure 1. Sample User's Manual pages

The problem could then be described as follows:

```
SEARCH DISPLACEMENT PUMP TABLE,
DISCHARGE 100 GPH, LIFT 10 FT;
```

Referring symbolically to the efficiency factor for the pump that will be located, continue by specifying a power requirement calculation:

```
CALCULATE POWER REQUIRED,
LIFT = LIFT/EFFICIENCY;
```

Again, referring symbolically to the horsepower value (POWER) that will be calculated, call for a motor selection:

```
SEARCH ELEC MOTOR TABLE,
HP = POWER, MOISTUREPROOF;
```

These tasks can be completed in one machine run. When it is finished, the geometry of the motor and pump is available. The designer can then proceed to design his mounting and gear set, using problem-solving statements from his operations manual.

The effectiveness of this approach is not limited to online computer terminals, although they are very desirable. Using the names of results as values in subsequent steps allows the designer to describe all, or a large part, of his problem at one time, for batch processing.

We can now compare three ways to handle the sample problem:

1. The manual process gives maximum control, but the designer must do all the work.

2. A series of partial solutions, using library programs, requires several separate runs and intermediate data transcriptions.

3. Using a problem-solving language, the designer retains control, avoids the detailed work, and minimizes the number of machine runs. He can even use partial results to decide on each new solution step.

If a problem-solving language can be made easy and natural, and if batch or terminal facilities are operated to give prompt response, everyday problems can be profitably serviced by computers and problem-solvers will use the service.

THE STRUCTURE AND OPERATION OF PLAN

FUNCTIONS

Appendix 1 lists each major PLAN program or subroutine. These allow the system as a whole to operate as a:

1. Language definer
2. Language interpreter
3. Submonitor
4. Programming support package

Language Definer

PLAN establishes and maintains a problem-oriented language dictionary in an online program library. At any time, new phrases and their definitions can be added, so that they are available for subsequent use. The basic elements that can be entered into the dictionary are:

1. The text of a phrase, consisting of from one to five words of one or more alphabetic characters; for example:

   CALCULATE POWER REQUIRED.

2. An arbitrary number of names to identify data values that can be entered when using this phrase; for example:

   LIFT, DISCHARGE, POWER

3. A predefined value for any data name for example:

   STRESS 30000.

4. A list of programs for execution each time the phrase is used, for example:

   PROGRAMS 'CALC, PRINT'

5. Tests, formulas, and conversions to be applied to data values during the input statement scan, for example:

   INCHES=FEET * 12 + INCHES

Language Interpreter

PLAN accepts a user's input statement, then interprets, and completely executes it before moving on to the next statement. It is this technique that allows PLAN to support an interactive mode of operation. A problem-solver at the console of an 1130, for example, can describe his problem, one statement at a time, basing

each new input on the partial results observed. Interpretive operation is limited to the problem-oriented language statements themselves. As soon as each phrase definition has been retrieved and a list of programs set up, the remaining execution takes place at full machine speed, using precompiled program modules. In our example, the input statement CALC POWER REQ, LIFT10 DISCHARGE = 100; would be accepted and analyzed. Data values of 10 and 100 would be set up, and programs CALC and PRINT would then be invoked.

Submonitor

PLAN is implemented to operate as a JOB under the 1130 Disk Monitor, DOS, and OS/360. PLAN acts as a submonitor, retaining control through many series of input statements. This submonitor supplies dynamic loading of unlinked modules and transfers data between them through COMMON and on direct access storage.

Programming Support Package

PLAN provides additional facilities for managing disk files as variable-length lists that are addressable on word boundaries. PLAN also offers additional linkage routines to simplify partitioning and sequencing in large problems. A library of generalized utility functions supplies program services that will simplify the task of any application programmer, by insulating him from specific machine considerations.

OPERATION

Figure 2 shows the functional parts of PLAN during execution. This figure will be used to trace the operation of the system for one user's problem description statement.

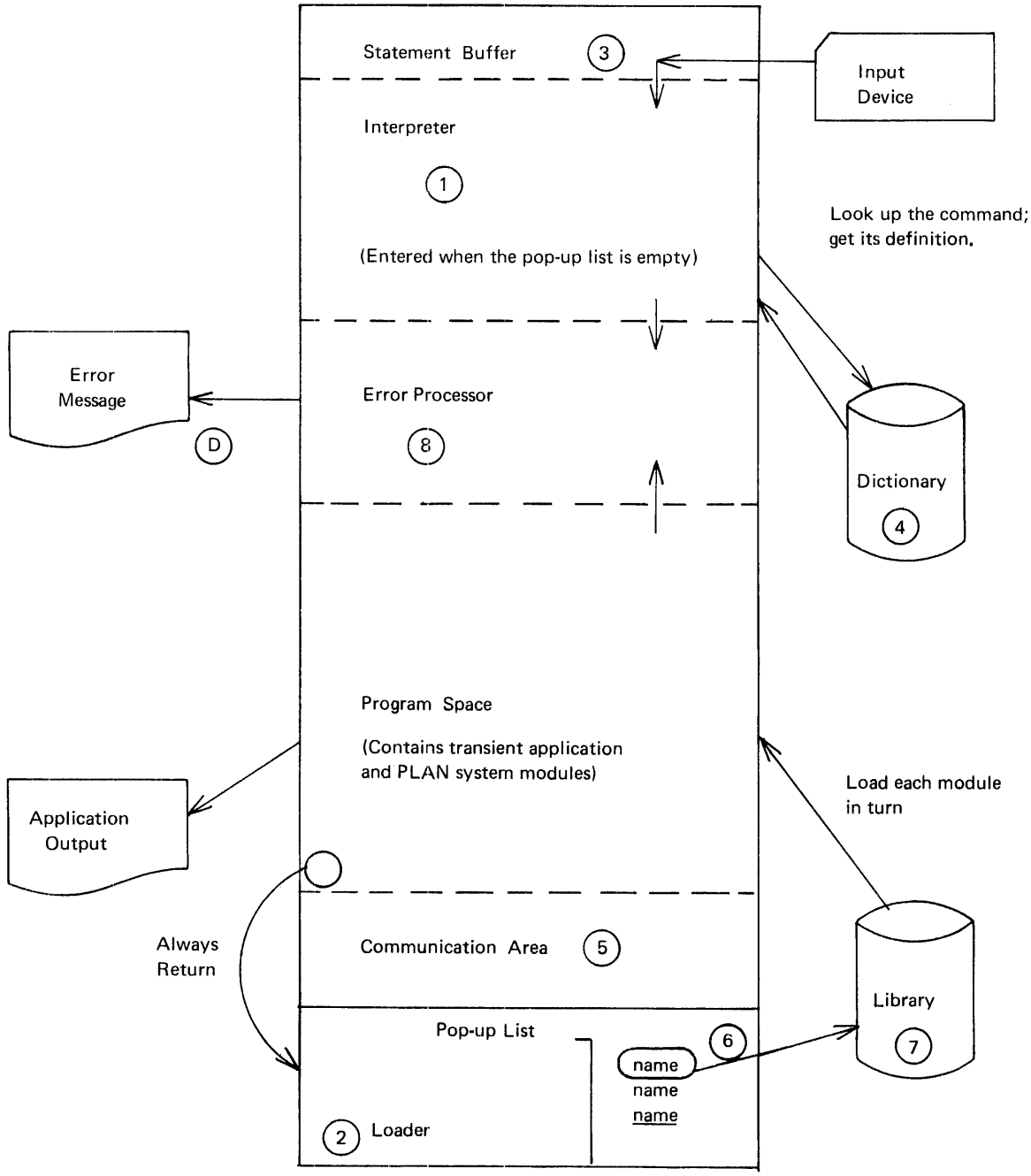


Figure 2. PLAN operating concepts

PLAN Processing

Initial loading of PLAN by the monitor or operating system readies the interpreter (1), the loader (2), and an input buffer (3). The interpreter examines the PLAN statement in the input buffer. First, its command phrase is used to access the dictionary (4), where the definition of each acceptable phrase has been stored. Information read from the dictionary allows the interpreter to scan the balance of the input statement. At the end of statement scanning, data from the user's statement plus the default data from the definition have been stored in the part of COMMON storage called the communication area (5). The names of the program modules that are now to be executed have been taken from the dictionary and placed in the pop-up list (6) (part of the loader). Data conversion, input checking, and formula evaluation will also have been performed, as specified in the dictionary entry.

At this point, use of the interpreter is temporarily ended. Control passes to the loader, which loads the first module named in the pop-up list from the library (7) removes the name of that module from the top of the pop-up list, and transfers control to that module.

When execution of any module is complete, control must return to the loader, which again loads the module whose name is at the top of the pop-up list. Serial loading and execution continue until the pop-up list is empty. Then, control returns to the interpreter, which accesses and interprets a new statement. Normal output is determined and produced by the program modules that are executed.

The efficiency of this procedure increases when more than one module is in main storage at a time. This is particularly true for modules that are used in a looping fashion. Multiple-loading is supported by PLAN. The facilities are different, but the system is upward compatible from 1130 to DOS to OS/360.

One additional block in Figure 2 should be mentioned: The error processor (8) is loaded (instead of the next-expected application module) when an error is found in an input statement. This error processing module can be invoked by any module during execution.

Management of the pop-up list is not confined to the interpreter module. Any program module (through PLAN subroutines) may examine, extend, or alter the pop-up list contents.

TIMING

The total machine time to solve a problem using PLAN can compare quite favorably with its alternatives. The timing estimates given here will be useful in general system design. Specific problem timings should not be inferred.

Problem timing under PLAN involves four major considerations:

1. Job initiation is a function of the monitor or operating system. PLAN initiation can be relatively fast. Its random access data sets do not have to be initialized for every new execution.

2. Handling one input statement in PLAN requires, typically, four direct access reads and writes. The average processing time per statement ranges from 0.8 to 5 seconds per statement, depending on the direct access device used, the amount of main storage assigned to PLAN, and the nature and sequence of the statements themselves.

   Note that language statements can, but should not, be used to enter bulk data. This sort of data can be read directly from within a program module, using formatted input. Thus, bulk data can be interspersed with PLAN language input.

3. PLAN loading time for one module is not very different from the basic monitor or operating system loading time (assuming the module is stored in core image or executable form).

   Unnecessary program loading can be avoided when programmers or analysts arrange for concurrent loading of modules that are used in a loop.

4. Execution time of any one module will be essentially unchanged, when compared with its standard FORTRAN environment.

ADVANTAGES AND FEATURES OF PLAN

The general introduction indicated that the two main features of PLAN are its support of a variety of problem-solving languages, and its resident loader, or problem supervisor, that eliminates much programmer preplanning of linkage and communication. These two features are keys to the advantages listed below:

- Supports many new and different problem-solving languages
- Lowers application development cost
- Lowers maintenance cost
- Operates on both IBM 1130 and System/360
- Allows easy application growth and change
- Operates interactively, as well as in batch mode

Each of these advantages is discussed in more detail; but the first and foremost advantage is simply a new freedom to develop open-ended problem-solving systems. Standard application design techniques do not attempt to provide this flexibility. They assume well-defined, bounded, and repetitious work.

PLAN supports many new and different problem-solving languages. Past practice in the development of problem-solving languages has been to write a special program to recognize one particular syntax. This works well enough for a single discipline for example, optics, numerical control, if there is general agreement on theory and practice. More complex areas, involving several disciplines, haven't even been attempted, because of cost.

Language design and implementation under PLAN breaks this cost barrier because it does not require specially coded language processors. Language is defined to PLAN in small units (phrases), so complete definition of a whole language is not necessary before implementation starts. Revisions and expansions can be tailored to any combination of disciplines at an incremental cost. Changes are additive; they do not force complete system reconstruction.

Because PLAN supports language at the low (phrase) level, there ceases to be any pure "language" concept. Phrase definitions and program libraries from several sources can be mixed and used together with relatively little change.

PLAN lowers application development costs. Since PLAN provides a unified approach to problem-oriented languages, maintenance and other support costs are reduced. PLAN also allows reuse of modules in a variety of contexts, eliminating much duplicate effort. The modular nature of PLAN languages and programs makes unit development and unit testing much easier. Applications can support limited production to reveal design flaws and unexpected user requirements even in their earliest development stage.

PLAN lowers maintenance cost. The cost-saving factors found in development also exist in maintenance; but an important extra saving comes from the independence of modules. Adding (or deleting) a PLAN module is an independent

operation. It is not generally necessary to change other programs that transfer control to the new module. It is also not necessary to repeat linkage editing of all the applications that use the new module.

PLAN operates on both IBM 1130 and System/360. FORTRAN offers a high degree of machine independence for computational logic. However, management of data and main storage is usually changed when shifting an application from one monitor or operating system to another. Subroutines have been incorporated into PLAN to eliminate most of the programmer's concern with these system differences and to improve the degree of system independence.

Applications can operate under the 1130 Monitor, DOS, and OS/360 if their component modules use a common subset of FORTRAN and follow PLAN programming conventions. PLAN language definition is the same on all three systems. PLAN subroutines (see Appendix 2) offer a uniform method of main storage management and data-defined data management in all three systems.

(Note that there is not a required use of subset FORTRAN, or of PLAN subroutines. Use of full FORTRAN IV simply limits the machines on which a module can be compiled.)

PLAN allows easy application growth and change. In the PLAN approach, each program module usually performs a single function. For example, a module might collect input data according to variable format, but it would not process the data or create output. Those are different functions, subject to different combinations. Explicit linkage between functions is not usually included in a PLAN module. It is not prohibited, but it is not necessary. Linkage between successive modules of code is performed at execution time. Complete linkage editing of applications is avoided. Application programming can grow, even in an unplanned fashion, without reworking previous segments every time a new capability is added. Adding new capabilities to any PLAN system simply requires that the command(s) defining the new capability be added to the language dictionary and that logic modules (if new ones are required) be added to the program library. The known, or existing, parts of an application can continue in operation.

PLAN operates interactively, as well as in batch mode. Every user input statement is executed by PLAN immediately after it has been accepted. A problem description can therefore contain conditional statements that depend on intermediate results in the problem. On the 1130 and under System/360 DOS, PLAN statements and data may be entered from either the console keyboard or the monitor input reader. Under DOS and OS/360 the PLAN input interpreter accesses SYSINP and SYSOUT as required for each statement. Convenient support for the 2250 graphic console is furnished by the PLAN Graphic Support (PGS) programs, under the 1130 Disk Monitor and OS/360.

DEFINING PLAN LANGUAGES

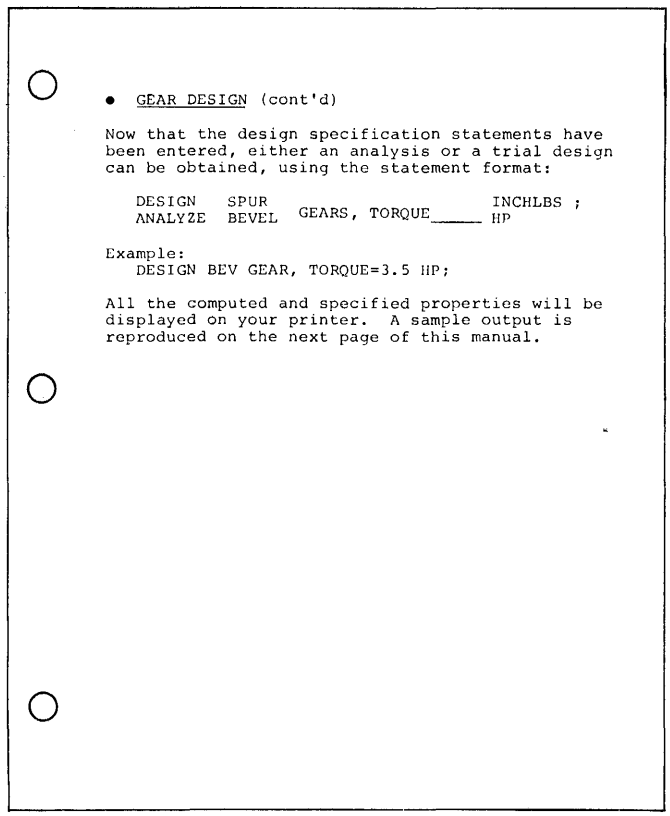
Since language design and definition in the PLAN environment are not delegated to the programmer, active cooperation of the people who will finally use the language is important. Their feedback must be used for incremental improvement of a language.

A language designer must be fairly experienced in programming and must also be familiar with PLAN features. One specialist could serve several departments or disciplines. This section presents more detail on the features of PLAN, showing how a language designer implements a problem-oriented language, by using PLAN statement definitions.

One application that has been written under PLAN allows such user statements as the following:

DESIGN SPUR GEAR, TORQUE = 5 HP;

We will assume that the language designer's external description of this statement was documented in a program description manual as shown in Figure 3.



• GEAR DESIGN (cont'd)

Now that the design specification statements have been entered, either an analysis or a trial design can be obtained, using the statement format:

```
DESIGN    SPUR                          INCHLBS ;
ANALYZE   BEVEL   GEARS, TORQUE_____ HP
```

Example:
DESIGN BEV GEAR, TORQUE=3.5 HP;

All the computed and specified properties will be displayed on your printer. A sample output is reproduced on the next page of this manual.

Figure 3. Example of a problem-oriented language statement to be defined (user's manual entry)

To support this specification, its options will be implemented with four phrases:

```
DESIGN SPUR GEARS,
ANALYZE SPUR GEARS,
DESIGN GEARS,
ANALYZE BEVEL GEARS,
```

The language designer catalogs these new command phrases by using the one predefined command, ADD PHRASE. For example:

```
ADD PHRASE:  DESIGN SPUR GEARS, . . . ;
```

Instructions to the PLAN input interpreter for handling this phrase are established in the data section of its definition statement. (The data section is omitted in the above example.) We will now develop a data section.

Assume that the following program modules are available:

CALC3	(gear stress analysis)
CALC4	(initial link of a design series)
PRT5	(design output)
PRT6	(analysis output)

For the phrase DESIGN SPUR GEARS, the required program series is to be CALC4, followed by PRT5. This is indicated in the definition statement by the keyword PROGRAMS, followed by the program name sequence within single quotes:

```
PROGRAMS'CALC4,PRT5',
```

We will assume that programs CALC4 and CALC3 expect an input torque value (in inch-pounds) to have been stored at location 80 of the communication array. (The communication array is part of COMMON storage - see Appendix 2.)

To specify that the symbol TORQUE will be recognized when used, and to assign the user's input value to location 80 to suit the program, the notation (80) TORQUE must appear in the definition statement. An asterisk following this notation tells the PLAN input processor that unless a value other than FALSE was supplied by the user for TORQUE, no programs are to be executed from the defined list:

```
(80) TORQUE * ,
```

We will assume that the choice between spur and bevel gear calculations in CALC3 and CALC4 depends on whether the integer 1 or 2 is stored at communication array location 81. The language designer can indicate spur gear design by preloading a 1 at location 81. The notation for this is:

```
I (81) 1,
```

Again, assuming that CALC3 expects a value in inch-pounds, while the user may prefer to specify horsepower, the language designer can instruct the PLAN processor to multiply any value given for TORQUE by a constant (396000), if the word HP also occurs in the same input statement. Recognizing that the use of any data name in an input statement without a value will produce the value TRUE, this is accomplished as follows:

1. Specify that some convenient location, 79 for example, be equated to HP.

2. Store a value there that is different from TRUE. One notation for this is:

```
(79)HP-,
```

3. Instruct the PLAN interpreter to check the value of HP after the input statement has been accepted. If the value of HP is TRUE, multiply the value at TORQUE by 396000. Notation for this is:

```
TORQUE:HP?=TORQUE*396000,
```

A complete definition statement for the DESIGN SPUR GEARS phrase has now been developed:

```
ADD PHRASE: DESIGN SPUR GEARS, (80) TORQUE*:HP?=TOR*396000,
1(81)1,(79)HP-,PROGRAMS 'CALC4,PRT5';
```

The PLAN processor builds compressed and coded dictionary entries from each such definition statement to guide its later processing. The entries include symbol definitions, data, program names, conversion equations, and testing rules that the language designer wants to apply when a specific input phrase is submitted in a problem description.

Note that all characters beyond three for each word in PLAN input may be omitted or supplied for readability. The order of data entry is also variable. The definition statement used earlier would have an identical effect if entered as follows:

```
ADD PHR: DES SPU GEA, (80) TOR*:HP?=TORQUE
*396000,PROG'CAL4,PRT5',(79)HP,1(81)1.0;
```

Input conversion of PLAN statements is designed to succeed whenever possible. Legitimate variations of the use, by gear designers, of the phrase we have just defined include:

```
DES SPUR GEA, TOR 5.0 HP;
DES SPU GEAR, 5*396000;
DESIGN SPUR GEAR, TOR=5.0,HP;
DESIGN SPUR GEAR, TOR3,,,TOR5,,,HP,,;
DES SPU GEA,TOR5HP;
```

The best way to design a problem-oriented language under PLAN is to begin quickly with a few very simple commands. Confidence and easy use of advanced features will follow.

PROGRAMMING SYSTEM REQUIREMENTS

Versions of PLAN operate under three IBM monitor or operating systems. PLAN is written, primarily, in Assembly Language. Some utility functions are programmed in the 1130 (BASIC) subset of the FORTRAN IV language.

IBM 1130 PLAN operates under the 1130 Disk Operating System, Version 2.

System/360 DOS PLAN operates under the IBM System/360 Disk Operating System.

OS/360 PLAN operates under the IBM Operating System/360, (any option).

In all cases, the computer system and the operating system must have the options needed for compiling and executing FORTRAN.

MACHINE CONFIGURATIONS

The minimum machine configurations that must be available for execution of application logic modules under PLAN are listed below for each version of PLAN. (Note: The configuration required in each case must meet FORTRAN requirements.)

IBM 1130 System PLAN

Required for PLAN operation:

1131 Central Processing Unit Model 2B, 2C, 2D,
3B, 3C, or 3D

2501 Card Reader Model A1 or A2
#3630
#8042 Attachment
#3854 Expansion Adapter

1442 Card Read Punch Model 5
#4449 Attachment

OR

1442 Card Read Punch Model 6 or 7
#4454 Attachment

Optional (Supported by PLAN)

1132 Printer
#3616 Attachment
#3854 Expansion Adapter

1403 Printer Model 6 or 7
#4424 or #4425 Attachment
#1865 Channel Multiplexer

2310 Disk Storage Model B1 or B2 (one or two)
#3201, #3202, #3203, #3204 Disk Control
on 1133

1133 Multiplex Control Enclosure
(required for 1403 or 2310)
#1865 Channel Multiplexer
#7490 Storage Access Channel

IBM System/360 Disk Operating System Version PLAN

Central Processing Unit

Model 25, 30, 40, 50, 65, or 75 (32K bytes or larger).
(An 8K DOS supervisor is assumed.)
Floating-point arithmetic
One I/O channel (either multiplexer or selector)

One card reader (1442, 2501, 2520, or 2540)
(one 2400 series tape drive may be substituted)
One card read punch (1442, 2520, or 2540)
(one 2400 series tape drive may be substituted)

One printer (1403, 1404, or 1443)
(one 2400 series tape drive may be substituted)

One 1052 Printer-Keyboard
One 2311 Disk Storage Drive
One 2841 Storage Control

IBM System/360 Operating System Version of PLAN

Central Processing Unit Model 30, 40, 50, 65,
67 (in 65 mode) or 75 which provides a partition of 32K
or larger. (44K is required under the OS/360 MVT option.)
Floating-point arithmetic
One console device

Any direct access device supported by OS, in addition to those required for OS/360, with a storage capacity equal to or greater than one 2311 Disk Storage Drive and appropriate control unit.

One or more input devices supported by QSAM
One or more output devices supported by QSAM

All versions of PLAN
The availability of a 29 Card Punch will prove to be an asset in the preparation of PLAN language statements.

APPENDIX 1: GENERAL DESCRIPTION OF MODULES AND SUBROUTINES

Two types of PLAN program components are defined in this appendix. The first portion of the appendix provides a general description of the functions and purposes of the program modules that are required to make PLAN operate. The second portion discusses support subroutines provided for programming to allow communication with PLAN and to provide expanded processing capability.

No PLAN module, other than the one named PLAN, should have execution initiated by Job Control. Any attempt to do so, will probably result in program failure.

PLAN MODULES

PLAN — PLAN is the one "mainline" program for an entire series of logical modules executed to solve a problem or series of problems. PLAN executes all program loading functions, and is therefore referred to as the "loader". It is the phase that handles initialization for a PLAN execution, and it remains in control, either directly or indirectly, until control is returned to the monitor or the operating system for a non-PLAN operation, such as a FORTRAN compilation.

It sets up the PLAN system switch area and collects the information necessary to communicate with other PLAN system modules. If the language dictionary file (PFILE) has not been initialized, it creates the necessary tables and calls in PHRAS to add the command ADD PHRASE.

If the language definition file has been initialized, PLAN execution is started. If the language definition file (PFILE), phrase scan module (PSCAN), or PLAN error module (PERRS) are not in the library, PLAN execution is inhibited, and control returns to the monitor or the operating system.

PSCAN — This module of PLAN is the command interpreter. The command is read from the appropriate input device. PSCAN saves and restores the managed communication array in its save file, as required by phrase level definitions.

PSCAN collects input data values and places them in the communication (COMMON) array. The programs defined by the command are added to the program list (pop-up loader). Command-defined expressions are evaluated. Checking of any required values is performed, and if there are no errors,

execution is transferred to the programs called by the command. If an error is detected, the system error routine is called to generate the appropriate diagnostic.

PSCAN is loaded by PLAN whenever there are no other program names in the pop-up loader.

PHRAS Problem language command definitions are added to, or deleted from, the language definition file by this module.

An index is posted to allow efficient phrase name lookup at execution time. An availability table is maintained to record use of the definition file. Extensive logical and syntax verification is performed on each phrase (command) that is added. The system error module is called to log any required diagnostics. PHRAS is loaded by PLAN in a manner identical to any other module.

PERRS This module is the system error module. It produces diagnostic messages and performs error recovery to continue processing after defective input has been found.

FIOCS This module uses the PLAN subroutine IOCS to change PLAN system parameters through the use of commands. It allows the user to switch command input to a new device in the middle of a job stream.

PSRTA This module performs the sorting of PLAN files as defined in a preceding call to the subroutine PSORT.

PMRGA This module performs the merging of PLAN files as defined in a preceding call to the subroutine PMERG.

PSTSV This module saves PLAN input statements when required. It is called in by PSCAN whenever a statement is to be saved for subsequent execution or a saved statement is to be executed.

PTDMP This module produces a printout in a tabulated format of all the phrases currently defined (added by PHRAS) in a PLAN language definition file.

PFDMP This module provides a dump of a PLAN logical file in hexadecimal. Identical print lines are suppressed.

PCDMP This module provides a dump of the communication array as specified by the systems switch words. Identical print lines are suppressed.

PIDMP This module provides a dump of the phrase currently being executed. This module is executed only if user action causes its name to appear in the pop-up list.

PDIAG This module maintains diagnostics (literal strings) in a disk file. The file is maintained in a format accessible by the subroutine PHIN.

PLITL This module produces a listing of all literals maintained in a PLAN literal file by the module PDIAG.

LOADER SUBROUTINES

LEX This subroutine allows a user to modify (add to or delete from) the PLAN pop-up loader. Transfer to PLAN then occurs to load the first (top) program defined in the pop-up list.

LIST This subroutine allows a user to modify the PLAN pop-up list. Processing continues at the next executable statement in the calling program.

LISTB This subroutine allows a user to add a program name to the bottom of the PLAN pop-up list. Processing continues at the next executable statement in the calling program.

LCHEX This subroutine allows a user to modify the PLAN pop-up list. The current program in execution is saved for future reentry, at the next executable statement, when an asterisk (*) is found in the pop-up load list.

LOCAL This subroutine allows a user to modify the PLAN pop-up list. The top program in the pop-up list is loaded, and control passes to the program. The program must coexist in storage with the calling module, because the calling module will be reentered at a later time. The program is not loaded if already in core. Note that the calling module remains in core with the called module; this is true of a nest of locals as well.

LRET This subroutine is the normal exit from a logic module. It does not modify the PLAN pop-up list. It exits to PLAN to load and transfer to the top program in the pop-up list. If the pop-up list is empty and saved statements are not being executed, a new command is processed. If the program executing a CALL LRET was called by a CALL LOCAL, return is made to the calling program at the next executable statement after the CALL LOCAL.

LNRET This subroutine breaks the chain of returns normally followed within PLAN LOCAL processing.

LREPT This subroutine allows reexecution of the last command processed.

ERROR RECOVERY SUBROUTINES

ERRET This subroutine allows application logic modules to generate diagnostic messages through use of the PLAN system error module (PERRS). The diagnostic literal is user-supplied. Processing continues at the next executable statement following the call to the subroutine.

ERROR This subroutine allows application logic modules to generate diagnostic messages through use of the PLAN error module (PERRS). Processing does not return to the calling module, and PLAN error recovery is initiated.

ERREX This subroutine interface to the PLAN error module PERRS does not return to the calling program. PLAN is entered to load any remaining programs in the pop-up list.

ERRAT This subroutine interface to the PLAN error module PERRS returns to the next executable statement in the calling program. PLAN will load any remaining programs in the pop-up list. However, the next time that PSCAN is entered, PLAN error recovery is initiated.

ERLST This subroutine causes all diagnostics that are in the PLAN error file to be printed on the PLAN system diagnostic device. Processing of the current phrase is terminated. This subroutine provides the capability required for error postlisting.

EWRIT This subroutine allows the user to write messages into the PLAN error file without using PERRS.

DIRECT ACCESS DEVICE SUPPORT

Permanent File Support

GDATA This subroutine performs the file open function for permanent, fixed-size files established outside of PLAN. The call places the file location pointer in the user-defined file control block.

WDATA This subroutine provides for transfer of information from core to a permanent file. The file is addressed by word displacement from the beginning of the file.

RDATA This subroutine provides for transfer of information from a permanent file to core.

Dynamic File Support

FIND This subroutine performs the open function for PLAN files. PLAN files are established when needed by execution time logic. Disk space is assigned to the file in modular segments as required by the use of that file. Transfer is by groups of FORTRAN words to or from any desired displacement within the file. Priority may be assigned to PLAN files to allow orderly release of low-priority files when insufficient file space is available.

READ This subroutine transfers data from a PLAN file to core.

WRITE This subroutine transfers information from core to a PLAN file. Space is automatically allocated if a write requires more than the current allocation.

PFSPC This subroutine provides the facility to verify the availability of a block of storage, before use.

RELES This subroutine releases space held by a PLAN file to the pool of available disk space. RELES performs the opposite function of the FIND subroutine.

PSORT This subroutine initializes the PLAN systems module PSRTA. The module issuing the CALL PSORT is reentered when the sort is completed, at the next statement.

PMERG This subroutine performs initialization for the PLAN systems module PMRGA. The module issuing the CALL PMERG is reentered when the merge is completed.

SEQUENTIAL DEVICE SUPPORT

Buffer Acquisition and Management (X=A through E)

PSBFX These subroutines provide single-buffering assignment (for 1130 PLAN) for devices specified by the PLINP and PLOUT routines.

PDBFX These subroutines provide double-buffering assignment (for 1130 PLAN) for devices specified by the PLINP and PLOUT routines.

PBFTR This subroutine allows direct transfer between PLAN buffers.

Status Testing and Device Control

PIOC — This function allows a user to test the device status for busy.

PBUSY — This subroutine tests all PLAN devices controlled by PLINP and PLOUT. PBUSY returns only when none are found to be busy.

PEOF — This function allows testing of end-of-file conditions generated as a result of CALL PLINP or CALL PLOUT.

PCCTL — This subroutine provides such device control functions as carriage skipping, carriage spacing, and card stacker selection.

Device to Buffer and Buffer to Device Transfer

PLINP — This subroutine provides the input processing for overlapped transfer from a PLAN-supported input device to a buffer.

PLOUT — This subroutine provides the output processing for overlapped transfer from a buffer to a PLAN-supported output device.

Program Array Buffer Transfer and Convert (X=Format Code)

PXIN — These subroutines provide for transfer to PLAN buffers from program arrays with the indicated format control.

PXOUT — These subroutines provide for transfer from PLAN buffers to program arrays with the indicated format control.

PROGRAMMER UTILITIES

PARGO — This subroutine provides transfer of data from a user array to the PLAN communication array.

PARGI — This subroutine provides the ability to move data lists from the PLAN communication array to a user array.

INPUT — This subroutine transfers the EBCDIC representation of the last command processed and the length of the command in characters to core to allow interrogation or printing of the command.

PUSH — This subroutine provides the ability to execute PLAN commands from core.

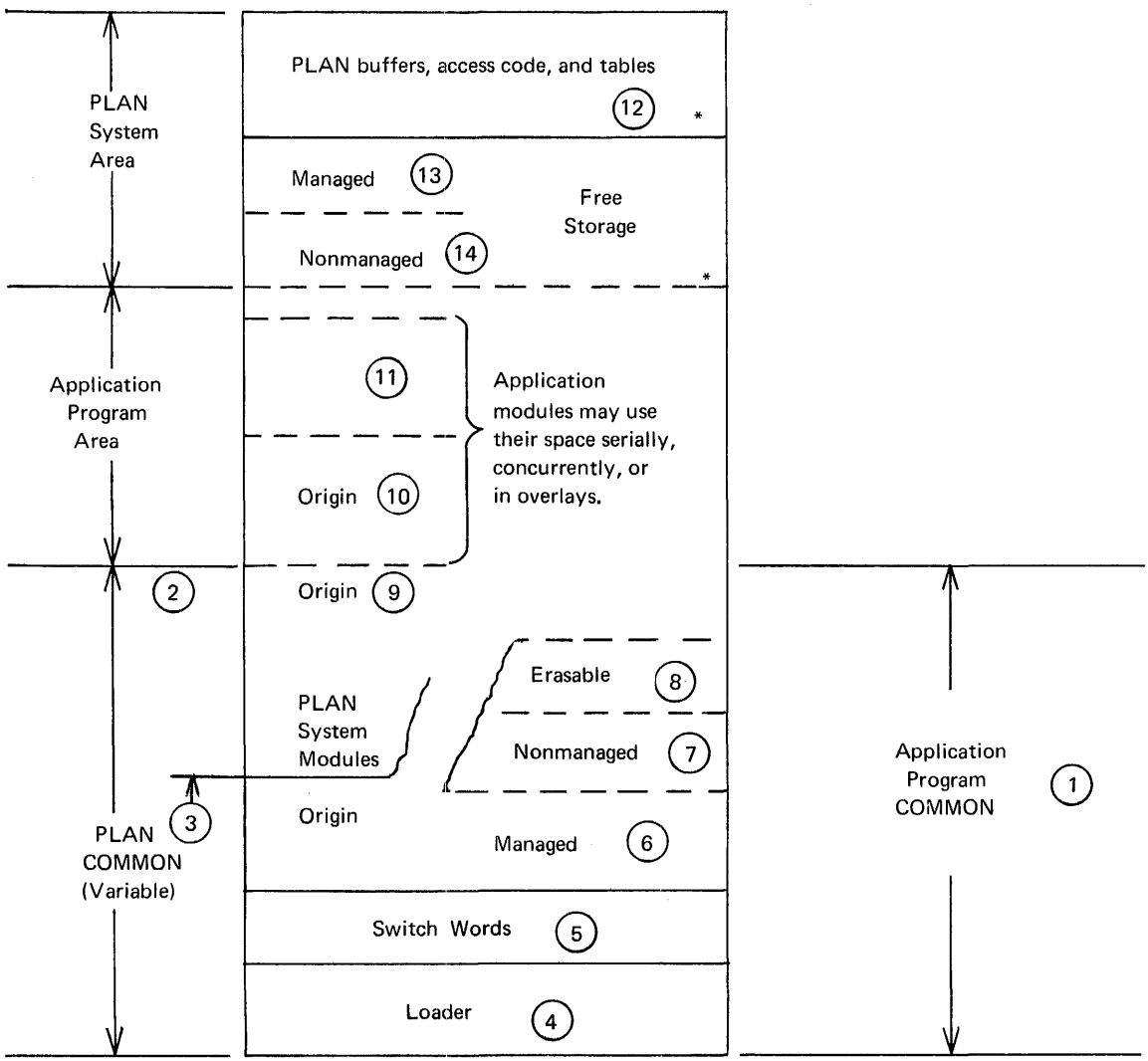
GTVAL STVAL	These subroutines provide for efficient internal transfer of data arrays.
BREAK	This subroutine divides the four bytes of a PLAN word into the low-order position (rightmost eight bits) of four words of an integer array.
PPACK	This subroutine masks the low-order byte of a PLAN integer word into any byte position of a FORTRAN array.
PUNPK	This subroutine extracts any byte position from a FORTRAN array and places it in the rightmost byte position of an integer word.
PCOMP	This subroutine performs a logical comparison of one PLAN (32-bit) array with a second PLAN array and sets a high-low-equal indicator.
PHTOE	This subroutine converts hexadecimal arrays to EBCDIC arrays. The EBCDIC array produced occupies twice as many words as the hexadecimal array.
PBTST	This subroutine allows for testing or setting any bit or bits (0-31) within a PLAN word. It also allows a test or extract under mask.
TRUE	This subroutine sets a word in core to the value of logical TRUE.
FALSE	This subroutine sets a word in core to the value of logical FALSE.
NDEF	This subroutine (function) allows the arithmetic IF to be used as a logical IF with a FALSE, TRUE, NUMERIC rather than -, 0, + branch.

APPENDIX 2: USE OF STORAGE

MAIN STORAGE

Main processor storage contains the monitor or operating system nucleus and one or more problem program regions. A PLAN job can operate in any region of an OS/360 multiprogrammed system, in the background partition only under DOS, or as the one job allowed by the 1130 Monitor.

Figure 4 illustrates the problem program region(s) used by PLAN. COMMON is shown at the origin of the region, because correct FORTRAN programs can be written for PLAN, following that assumption. (It is actually located in reverse order at the end of main storage on the 1130 system.)



*Note:
Solid lines show fixed boundaries; dotted lines show variable boundaries, during PLAN execution.

Figure 4. General arrangement of a PLAN problem program region

The areas and boundaries in Figure 4 are numbered to key them to the following discussion:

(1) (Blank) COMMON, as defined in the programmer's code. Required to be at least 2560 bytes long.

(2) A program loading limit (variable during execution) to allow extended COMMON data protection.

(3) The portion of COMMON that is never overlaid by PLAN system modules during a user's PLAN problem run. (Varies upward from 4600 bytes, depending on the problem region size.)

(Areas (4) through (8) are subdivisions of COMMON.)

(4) The 2500-byte LOADER area contains the PLAN submonitor. It is not available for data.

(5) The 60-byte SWITCH area provides a secure area for communication between PLAN system and application modules.

(Areas (6), (7), and (8) are the COMMUNICATION AREA.)

(6) The MANAGED array is a variable-length area used jointly by the PLAN input interpreter and the application modules. The content of this array is managed according to the rules for PLAN statement hierarchy. Within the processing cycle for a single statement, however, it is simply part of COMMON. The managed array is a primary source of input data for PLAN modules (data that has been collected, saved, or restored by the PLAN interpreter).

(7) The NONMANAGED array is that part of the communication area which is not saved or restored by the input interpreter. Its contents are not affected by the relative "level" of succeeding input statements.

(8) The ERASABLE array is a variable-length scratch space at the end of COMMON, used by system utilities to avoid interfering with application program communication.

(9) The EXECUTION area contains one or more program modules.

(10) Load modules may have storage boundaries that are fixed before PLAN execution (1130 and DOS) or at load time (OS/360).

(11) A PLAN LOCAL module may load above or below its caller (10). There can be multiple modules in main storage, within the capacity of the problem program area.

(Areas (12), (13), and (14) apply only to System/360.)

(12) The PLAN systems area contains basic I/O programs, buffers, and additional code. Its length is variable.

(13) GETMAIN area of specifiable length, to be purged automatically by PLAN.

(14) Unmanaged GETMAIN area of specifiable length.

DIRECT ACCESS STORAGE

PLAN uses direct access storage, in addition to main storage. PLAN DASD files are listed below. Note that the language definition (PFILE) and library files are mandatory on all systems.

Data Files

PLINPXXX — This file is required on OS/360. It contains the PLAN input stream.

PLOUTXXX — This file is required on OS/360. It contains the PLAN output stream.

PLANDRVN — These files on System/360 define storage space for groups of PLAN dynamic files.

PLFSYNNN — Each of these files on System/360 defines storage space for one PLAN permanent file.

System Files

PFILE or DFJPFIL — This file contains the PLAN system tables, including the coded text of the user-oriented command definitions.

PDATA or PLMANFIL — This file is required when the communication array (managed array) is to be managed by levels. It is used to save communication array images.

PCHPT or PLCHKPT — This file must be defined if the LCHEX subroutine is used (see Appendix 1 for a description of LCHEX). The image of the calling program is saved in this file. The required file size depends on the nesting of LCHEX calls and on the size of the problem region. This file is also required if multiple errors are to be detected during command processing.

PLANLIB	This is the 1130 Monitor USER AREA, the DOS Core Image Library, or a named partitioned data set in OS/360. Storage of programs into these areas is accomplished by normal monitor or operating system techniques.

BIBLIOGRAPHY

Problem Language Analyzer (PLAN), Program Description Manual (H20-0594). This manual provides technical specifications and rules for the use of the system. It is a designer's and programmer's reference.

1130 Problem Language Analyzer (PLAN), Operations Manual (H20-0595).
System/360 Problem Language Analyzer (PLAN) (OS), Operations Manual (H20-0596).
System/360 Problem Language Analyzer (PLAN) (DOS), Operations Manual (H20-0597).

These manuals describe the steps necessary to operate PLAN in three machine environments.

Program for Optical System Design/II, (POSD/II) Application Description (H20-0489). This manual describes a POL implemented under PLAN. It provides a very flexible facility for the geometric analysis of image-forming optical systems, with a means for automatically correcting such systems.

Rigid Frame Selection Program, (RFSP) Application Description (H20-0495). This manual describes a POL implemented under PLAN. It provides direct design procedures for rigid frame construction in steel, timber, or concrete.

Mechanism Design System - Kinematics, Application Description (H20-0493). This manual describes a POL implemented under PLAN. It provides a programmed system for the kinematic analysis of linkage mechanisms for the mechanical engineer.

1130 Data Presentation System Application Description (H20-0235). This manual describes a POL implemented under PLAN. It provides a programmed system for generating graphic reports using the 1627 Plotter.

1130 Work Measurement Aids Application Description (H20-0249). This manual describes a POL implemented under PLAN. It provides a series of programs that help industrial engineers. The programs are (1) Machinability, which determines machine tool parameters and calculates process time for machining operations, and (2) Work Measurement Sampling, which determines job standards for long-cycle operations and the distribution of time to job activities.

1130 Mechanism Design System - Gears and Springs Application Description (H20-0268). This manual describes a POL implemented under PLAN. It provides a series of programs for the design and analysis of spur and helical gears and for compression, extension, and torsion springs.

PLAN Graphic Support for the IBM 2250 - Application Description Manual (H20-0535). This manual describes a set of programs analagous to the PLAN language definer and interpreter, for adding a 2250 graphic display interface to PLAN-based applications.

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
(USA Only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)