Ground Operations Aerospace Language

GOAL

Final Report

Volume III

Data Bank

31 July 1973
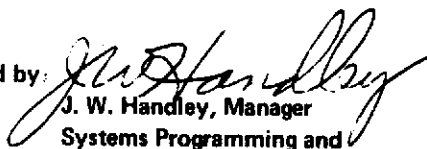
Ground Operations Aerospace Language

GOAL

Final Report

Volume III

Data Bank

Contract NAS 10-6900

Approved by: *J. W. Handley, Manager*
Systems Programming and
Advanced Programs

I

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont)

1.0   INTRODUCTION

The GOAL (Ground Operations Aerospace Language) test programming language was developed for use in ground checkout operations in a space vehicle launch environment. To insure compatibility with a maximum number of applications, a systematic and error-free method of referencing command/response (analog and digital) hardware measurements is a principle feature of the language. Central to the concept of requiring the test language to be independent of launch complex test equipment and terminology is that of addressing measurements via symbolic names that have meaning directly in the hardware units being tested. To form the link from test program through test system interfaces to the units being tested the concept of a data bank has been introduced. The data bank is actually a large cross-reference table that provides pertinent hardware data such as interface unit addresses, data bus routings, or any other system values required to locate and access measurements.

Three aspects of future aerospace operations can be considered key when determining whether single or multiple data bank(s) can be justified. First, the management of a very large number of sensor based measurements at distinct facilities constitutes a large engineering management effort. Secondly, verification and launch checkout of future ground and flight hardwares will require use of this large measurements base in conjunction with the generation of a large body of automation and test program software packages. A third problem area is then created when combining certain aspects of the first two - when hardware modifications are required as a result of normal or abnormal operations these changes must be carried over into the software programs also. The solutions to any or all of these problems are further degraded when confronted with the potential space shuttle environment of tight launch schedules involving multiple vehicles.

Little can be accomplished in alleviating the problems of the large measurements base or the large number of required test programs as these items are controlled by engineering factors not under direct control of ground data processing systems. Data processing equipments can be effectively used in the support of the ground checkout and launch "business" just as they have been in other applications of the scientific and commercial world. The GOAL test programming language has evolved as an attempt at alleviating the problem of generating and managing a large number of test programs. The use of a data bank in conjunction with the GOAL language is vital since its sole purpose is to assist the programmer in dealing with a large and variable set of measurements and to make as easy as possible the introduction of the effects of hardware changes into the test programming system. Accordingly, throughout the remainder of this document, the aspect of meaningful use of measurements and the software impacts of hardware measurements changes will be discussed over and over again. A significant portion of this document will deal with suggestions as to improvement of operations using the GOAL language compiler data bank and extending the use of a measurements data bank into online use in ground operations.

2.0　　　CONVENTIONAL PROGRAM I/O DEVICE ADDRESSING

2.1　　　EARLY BINDING

In general, software communication with any external computer accessible device is considered an input/output operation. The process of associating actual device addresses with input/output instruction sequences is known as "binding" in data processing parlance. The binding of device addresses to I/O instruction can be effectively accomplished at or before program execution.

"Early binding" techniques have been customary in the past with compiled high level languages such as FORTRAN. Thus, at the time that the programmer coded the sequence

> WRITE (6,10) A, B, C

directing output to symbolic device #6, it had already been determined (usually by compiler decree) that device #6 was to be, for instance, a tape drive at device address A. Once compiled, only tape drive A could be used as a target for the output operations directed to device #6. If for unseen reasons tape drive A was not operational, the early binding of this device address A to the symbolic name 6 forces one of the following two alternates:

　　　1.　　If at all possible, the program run is held in abeyance until the required tape drive becomes operational.

　　　2.　　If the program must be run, a workaround for the required hardware address change must be effected. In the traditional early binding situation for FORTRAN, the following steps are required:

　　o　　Assign an alternate tape drive to replace the non-
　　　　　operational drive A. Assume that tape drive C
　　　　　(symbolic address #8) was available.

　　o　　The FORTRAN source program is then modified such
　　　　　that every WRITE statement addressing device #6
　　　　　is redirected to device #8. The statements of
　　　　　concern must be repunched.

　　o　　The revised source program must then be recompiled
　　　　　prior to its use.

2.2　　　MODIFIED EARLY BINDING

A significant improvement to the classic early binding problem can be effected by associating symbolic device addresses with actual hardware addresses via compiler control cards or special source program control cards at compile time. Thus, with "modified early binding" the problem

of the previous FORTRAN example would be resolved by modifying a device
control card and recompilation of the source program. Note that modifi-
cations to the actual source program are no longer required in this
instance - this could amount to a significant number of changes in many
programs. Several programming languages have directly or indirectly
adopted this technique of improved early binding. For instance, the
sole purpose for the ENVIRONMENT DIVISION Section of a COBOL program
is to implement this feature.

2.3   LATE BINDING

"Late binding" became a reality with some of the more advanced Operating
Systems available late in the so-called 2nd Generation era of software.
With late binding, the actual hardware address versus symbolic address
linkage is to be made after compilation. Thus, the salient feature of
this type of binding is that no source program modifications or re-
compilations are required to accommodate device address changes. Late
binding is usually one of the service functions of an Operating System.
The actual binding or device selection is customarily performed imme-
diately before execution of the program involved. In the selection of
specific devices for allocation to a program, the Operating System may
follow control card direction as to exactly which device is desired;
some Operating Systems are capable of selecting any appropriate device
from a pool of available devices.

3.0   MEASUREMENT ADDRESSING IN SENSOR-BASED PROGRAMMING

The data processing environment in which a sensor-base (process control) program must operate is in many areas parallel with the traditional batch scientific or commercial environment. As such, many of the tried and true solutions to traditional batch programming problems can be expected to bear similar fruit in the process control environment. The two environments are not the same, however, and it should not be surprising if some of the solutions to the past problems do not yield equivalent results; in some instances, new solutions to new problems must be sought.

Sensor-base measurements (discrete and analog inputs/outputs) form a set of simple input/output devices to manipulate - the measurements problem in the process control environment is not one of complexity but one of sheer bulk. As the total number of input/output device addresses grows larger, the possibility of an individual address change becomes more significant. Experience has also shown that the negative aspects of the software binding techniques discussed previously become more intractable for the case of managing a system of process control programs. For instance, a single discrete measurement address change may require several hundred changes to a dozen or so programs - all of which must then be recompiled and reintroduced to the online system before run time.

3.1   BINDING IN SATURN/APOLLO SOFTWARE

In the process-control environment of SATURN/APOLLO, the discrete or analog measurement symbolic name and measurement hardware address were synonymous. By the choice of such a symbolic naming scheme, early binding of symbolic name and hardware address was forced on the process-control software - at the assembler language level as well as at the high-order language level. Thus, the following succession of events could be anticipated following a hardware change - possibly as minor as a single address modification:

o   The first problem falls properly in the area of configuration management - that of finding from a system of (n) software modules just which ones access the measurements whose addresses have been changed. In general, a manual search of program listings has been the only method of solving this problem.

o   Having determined which program(s) require modification, the imposing task of source program modification can ensue. Two factors accentuate the difficulty of this endeavor. First, it is common for a single measurement to be mentioned many (possibly, hundreds) of times within the bounds of a single program; the problem of finding and making all of the required changes is a very real problem. Often, "almost all" of the changes are located and made. Secondly, the modification task must in many instances be attended to by personnel not responsible for coding the original programs; this adds a factor of difficulty to any program change.

o   All of the modified software modules must then be re-
      assembled or recompiled prior to use. Some may also
      require revalidation of program operation after changes.
      Once the update process has completed, the updated
      modules will replace the currently active modules in
      the online system.

o   If the measurement address changes were only temporary,
      the above sequence of operations must be reversed to
      return the software system to its original configuration.

3.2   BINDING IN THE GOAL LANGUAGE

The disadvantages of early binding in the existing SATURN/APOLLO computer
languages were well known when specifications were first drafted for GOAL
(Ground Operations Aerospace Language). This high level language, intended
for use in the post SATURN/APOLLO time frame introduced the concept of a
"data bank" as a repository for all sensor-base hardware procedural,
addressing, and routing data. With the initial implementation of the
GOAL language, this data bank took the form of a disk-resident file from
which the GOAL compiler could randomly access data relative to all avail-
able measurements. GOAL also introduced a meaningful symbolic naming
scheme for all hardware measurements and classifies such devices as
"function designators." As an example, the discrete signal that
activates/deactivates 28 volt stage power might have been addressed
in the SATURN/APOLLO software via its hardware address, e.g., DISCRETE
OUTPUT #414 - early binding of the hardware address is thus completed.
The same measurement could be addressed in a GOAL program by the symbolic
function designator name "28V STAGE POWER." Using the function desig-
nator name as a search argument, the GOAL compiler would query the data
bank during compilation and determine that a discrete output measurement
at hardware address 414 was being referenced. This GOAL implementation
of a modified early binding technique eliminates many of the frustrations
of accommodating measurement address changes. All such changes will be
introduced to the GOAL system by an update to the data bank. Recompila-
tion of the appropriate test programs will then effect the required
modifications in the programs.

3.3   EXTENSIONS OF DATA BANK USAGE INTO OTHER AREAS

One of the basic design philosophies of the GOAL language stipulates that
a GOAL test program must have the ability of addressing measurements and
specifying test points at the lowest level possible within the bounds of
a given system configuration. It was recognized that the level of access
or the method of accessing a particular measurement could vary significantly
- dependent upon the physical or geographic site at which the program was
to operate.

As an example, consider a flight component called "Box A". A test program is developed that will issue stimuli to Box A and measure subsequent Box A responses in a fashion that will verify the internal operational status of Box A. As illustrated in Figure 3-3-1, both the type as well as quantity of test hardware which might exist between the test program host computer and Box A could vary significantly at three geographical sites at which testing of Box A is required.

o   At the manufacturing site, the final acceptance test interface is shown as comprising a simple hybrid set-up solely dedicated to the testing of Box A.

o   At the launch site, Box A assumes the role of a single component within a complex vehicular hardware system. The test interface in this instance is quite complex since it addresses the entire vehicle - only a small portion of the overall interface can be dedicated to the control or testing of Box A.

o   During preventive or fault isolation maintenance, a third variant of interface exists between the test program and Box A. In this instance, controllable diagnostic test hardware has been introduced. Although Box A is shown as being the sole object of test, maintenance testing could well have been directed at a larger system within which Box A was a minor component.

In each of the instances in the example, the contents and operation of the test program could be assumed to be constant, i.e., the procedure used for the internal verification testing of Box A is the same irrespective of the location of Box A. The method of sensor-based measurement address binding now becomes most significant. If early binding is chosen, the test program must be programmed in a unique manner in each of the locations of the example. This implies unique program coding as well as unique verification of the proper operation of the program at each site. The function designator naming scheme and data bank concept of GOAL combine to make it possible to fulfill the test program obligations of all three sites with a single source program. Since each site will support its own version of the GOAL data bank, a compilation of the same program at each site will result in an executable module tailored to each of the distinct interface requirements.
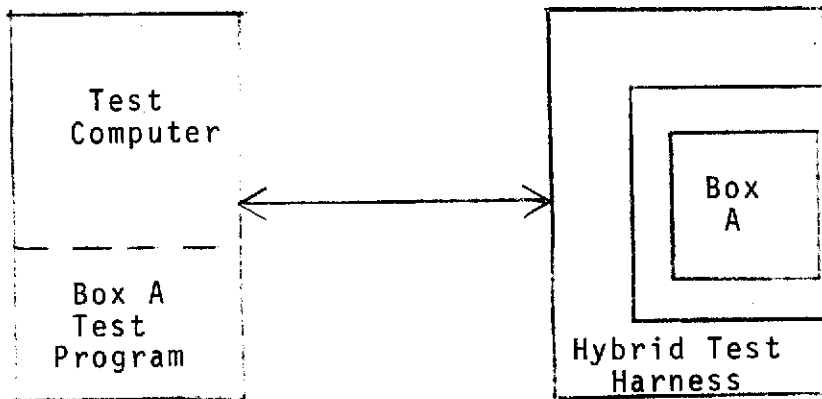
FINAL ACCEPTANCE TEST AT MANUFACTURING SITE
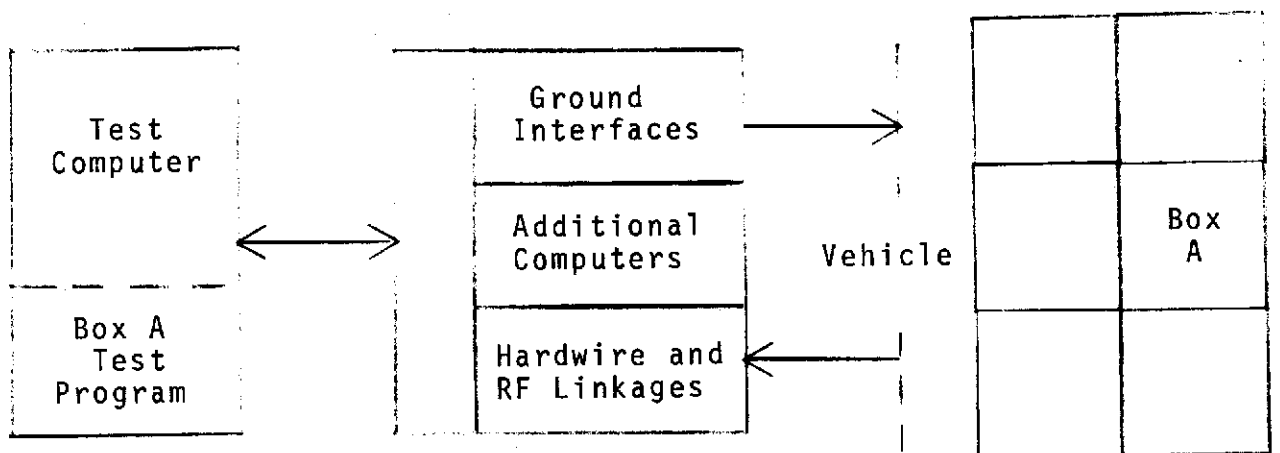
LAUNCH COMPLEX PRE-FLIGHT TESTING
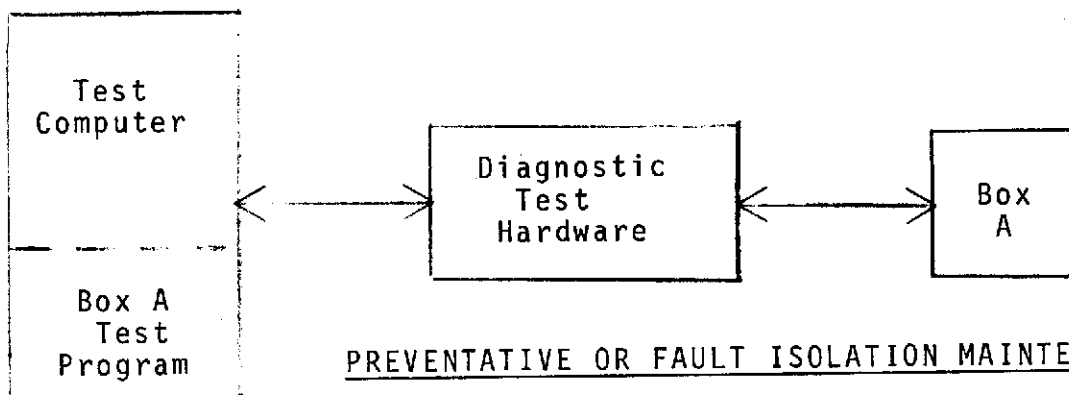
PREVENTATIVE OR FAULT ISOLATION MAINTENANCE

Figure 3-3-1.   SAMPLE HARDWARE CONFIGURATIONS

4.0   IMPLEMENTATION OF LATE BINDING

The single inconvenience encountered in a system which incorporates early
or modified early binding is that of requiring program re-compilation to
adjust for input/output addressing changes. Put in other words, the
actual binding operation takes place during (and only during) compilation.

To effect late binding, the binding process must ensue after compilation;
this essentially divorces the compiler from binding responsibilities but
introduces the requirement that another program accomplish the binding.
In many modern systems this binding responsibility will be delegated as
a utility service task for the Operating System. Generally, lack of late
binding resources within a given data processing system can be attributed
to one or more of the following reasons:

   o   Occasional recompilation of a program to adjust for
       input/output device reassignment is not considered
       by some users to be an inconvenience worth mentioning.

   o   That which could be cleanly designated as an "Operating
       System" is absent in many small or so-called "mini" systems.
       The high-level language compiler (if any) of such a system
       must produce a program module that is completely self-
       supporting - it can be loaded and carried through all
       phases of execution without assistance from other pieces
       of software. Manual assistance, e.g., control-card
       direction after compilation is almost never required,
       nor, in most instances, possible.

       A method will be presented later in this document in
       Section 4.2 to demonstrate that late binding can be
       effectively implemented on a system which hosts only
       a computer of small capability. This would seem in
       contradiction with the statements preceding, however,
       the "small" computer of Section 4.2 will be augmented
       considerably by offline support of a larger system on
       which the burden of compilation and software binding
       will be placed.

   o   In some systems the effort associated with implementing
       a late binding scheme was simply not justified in the
       minds of the system designers. Not only must the high-
       level language compilers and the Operating System software
       be of greater scope, but the use of a late binding system
       is generally more complex for the programmer.

4.1   LATE BINDING IN GOAL

Two features must be present in the GOAL execution-time, i.e., online, system if late binding is to be effected. The first is the software module that will actually accomplish the binding operation. This can be conveniently delegated to either a utility program or a utility service of the online operating system. The second need is that of a large cross-reference table that can be used to relate symbolic measurement names to actual hardware accessing data; in other words, an online** data bank is required.

Two positive factors make the operation of late binding in the GOAL language system distinct from the implementation customarily found in scientific and commercial data processing systems. First, the compile of a GOAL program and the execution of the same program will probably be accomplished on different computer systems. This means, for instance, that the qualifications that must be met by the computer system that hosts compiler operations need not be reflected or duplicated by the execution-time computer systems, or vice versa. Secondly, for the GOAL system, no appreciable user effort is required to effect or control the binding operation whereas traditional late binding techniques customarily involve modest to extensive control-card direction by the programmer.

4.2   LATE BINDING IN A GOAL SYSTEM - EXAMPLES

Figure 4-2-1 illustrates the process flow of the production of a loadable program module via the first implemented GOAL compilation process. The file marked "intermediate text" in this case is the source program in another more readily processable format. Within the intermediate text file is the reformatted source program plus all applicable hardware addressing or usage data as supplied by the compiler data bank. In the instance of Figure 4-2-1, the translator program provides the interface between online SYSTEM-X and the general-purpose intermediate text form of the user program. Each distinct online system will require a unique translator program. The program module shown as output from the translator is complete; function designator hardware binding was accomplished at the GOAL compilation step - in this instance, modified early binding. The final step of the programming sequence is the introduction of the tailored program module to the program library associated with the online computer of SYSTEM-X.

**"Online" is perhaps a poor choice (as will be seen later) to attach to the data bank associated with the binding task. For the present, let it suffice to say that the word "online" is used strictly to establish a distinction between the online data bank and the data bank associated with compiler operations (the compiler data bank).

A possible method of accomplishing the transition from the current GOAL modified early binding system to a late binding variant is demonstrated in Figure 4-2-2. This diagram can be seen to differ significantly from that of the current modified early binding system of Figure 4-2-1 in the flow logic following the translator phase. Preceding the translator, the compiler data bank now contains no hardware binding data. Function designator data consists of symbolic name and associated hardware type (discrete, analog, etc.), description. The translator program of the late binding system accomplishes the same general functional task as before, i.e., translating the intermediate text format of the user program into an acceptable form for use on the online system. The output of the translator is unique in the late binding system - declared data and program text (direct or indirectly executable code) are combined into a distinct output labeled the "program module." Associated with each program module is a second output from the translator - the "measurements table." This table will contain no entries for those programs which make no reference to external input/output devices (including sensor-base devices). For any other program which will require binding of function designator name to hardware address, the measurements table will contain individual table entries - one for each unique function designator mentioned in the program. Each table entry will indicate the function designator symbolic name in conjunction with table space (reserved, but not filled in at translate time) sufficient to hold any associated hardware addressing, usage, accessing, scaling, or routing data. The symbolic function designator name will be used as a search argument when seeking hardware binding data from the online data bank.

Shown in Figure 4-2-2 as a peripheral to the computer online SYSTEM-Y is the data bank (called the "online" data bank) from which the hardware binding data** can be extracted. Introduction of the entire program to the online computer program library takes place in the following two-step operation:

1.   The program module is transferred and edited into the online computer program library in exactly the same fashion as with the modified early binding system.

2.   The measurements table will be entered into the measurements table library via a utility program entitled the "table editor" in Figure 4-2-2. The table editor will locate each function designator in the online data bank by using its name as a search argument. Once a function designator has been found in the online data bank, all pertinent hardware accessing data (binding data) will be filled into the space reserved for it in the measurements table. The completed table will then be written into the measurements table library.

**There is a definite need for an online data bank for use in areas other than late binding support. More on this later, in Section 5.1.

The "program fetch" (locating and fetching a program into computer storage prior to its execution) operation of the online computer Operating System as shown in Figure 4-2-2 will require modification to accommodate location and load of <u>both</u> the program module and its associated measurements table. Even though late binding involves a distinction between program module and measurements table, no more computer memory should be required at execution time to support a late binding operation than that which would have been used in an early or modified early binding system.

Online SYSTEM-Z as depicted in Figure 4-2-3 is offered as an example of extending late binding capability into a hardware system of unusually small capacity. The computer of online SYSTEM-Z is assumed incapable of supporting an online data bank as a normal peripheral device. A practical instance of this situation might be in the case of a so-called "mini" computer system. Even in this instance, ability to accomplish late binding has significant advantages - predominant is the ability to accomplish binding without requiring program recompilation. The late binding operation in this system occurs at translate time on a computer of higher capability than the online mini system. The translator data bank is labeled "online" to distinguish it from the compiler data bank. The program module produced for use on the target online computer would be complete, i.e., all binding accomplished. The program module is not shown as entered to a program library since a computer system of this minimal size suggests that such a program library might also be beyond the support capability of the system. Tape (magnetic or paper) or punched cards could be an acceptable storage media for the program module.
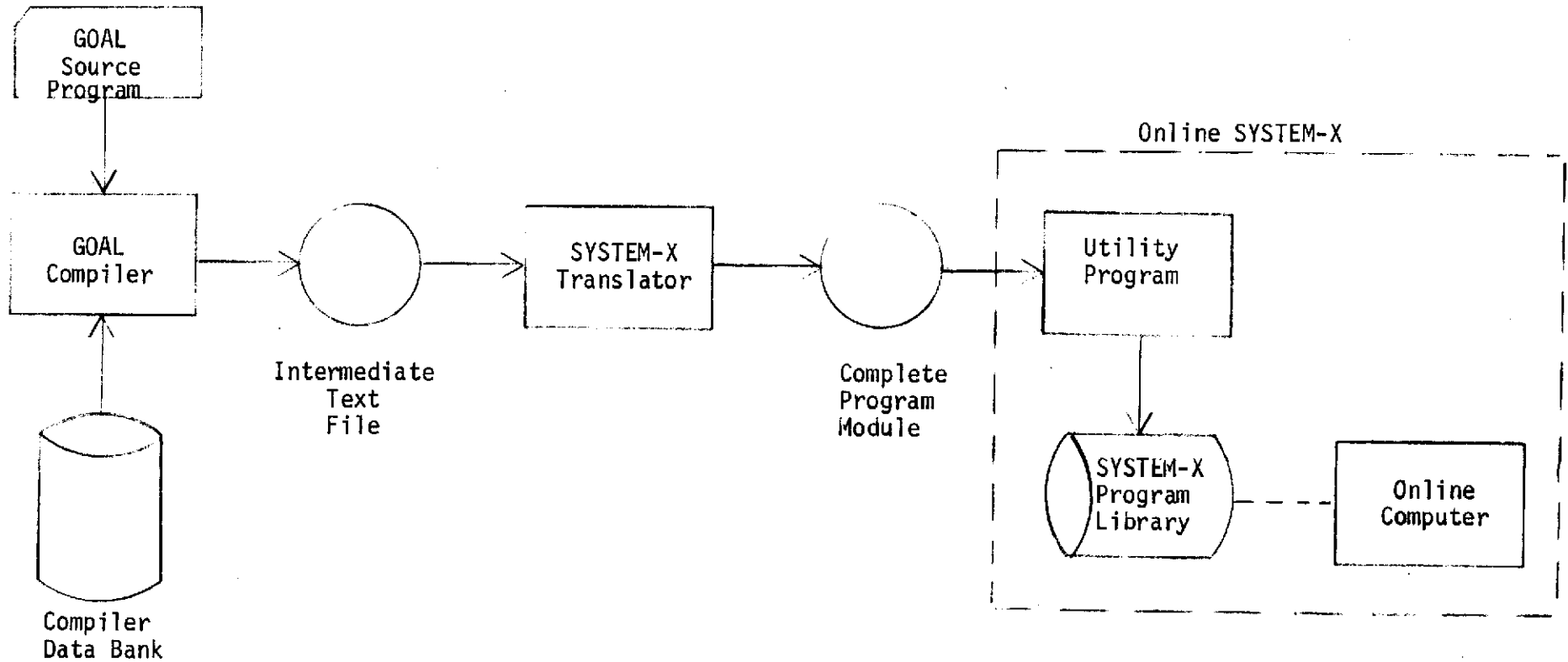
Figure 4-2-1. - GOAL Program Production Flow
As Implemented with Modified Early Binding
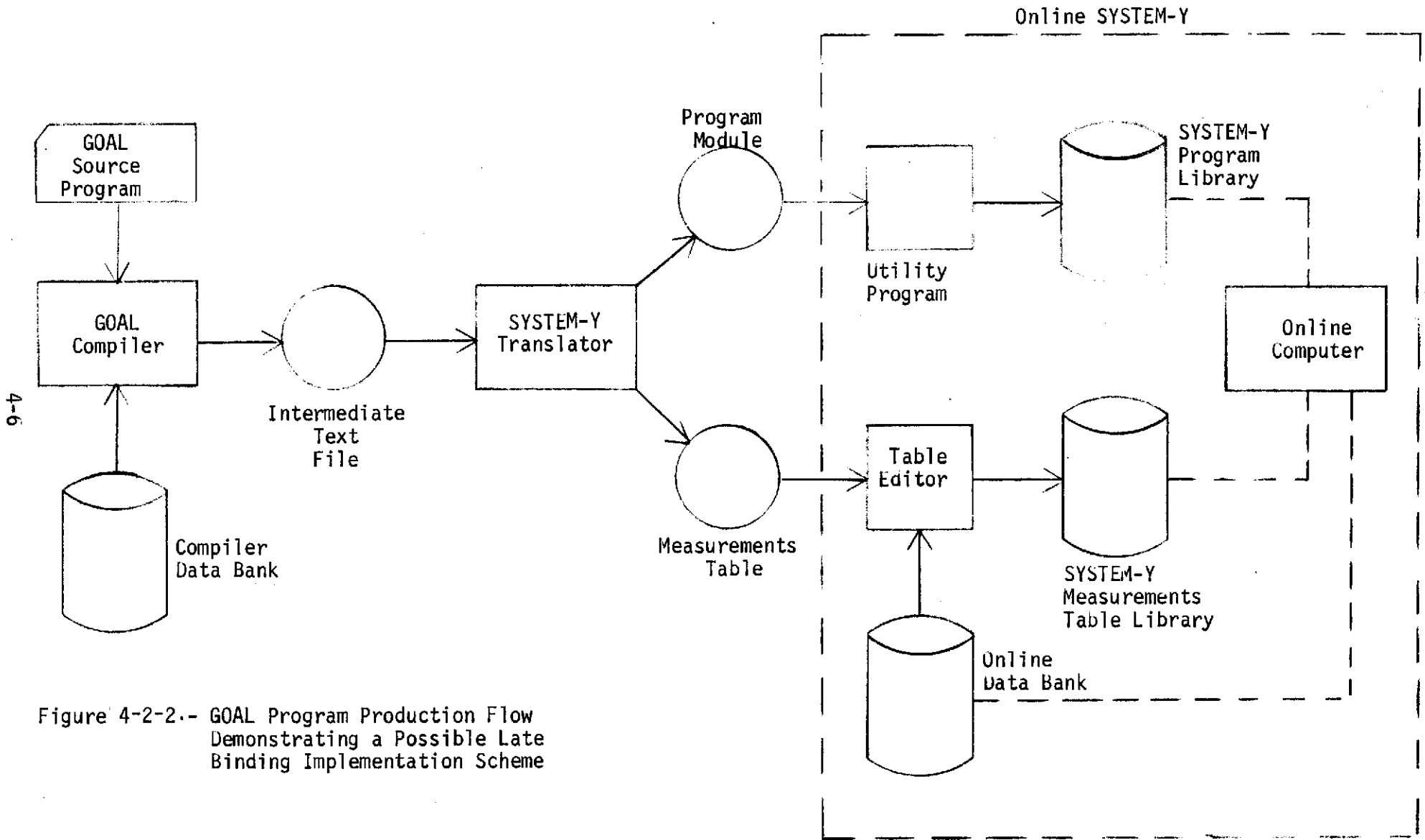
Online SYSTEM-Y

Figure 4-2-2.- GOAL Program Production Flow
Demonstrating a Possible Late
Binding Implementation Scheme

4-6
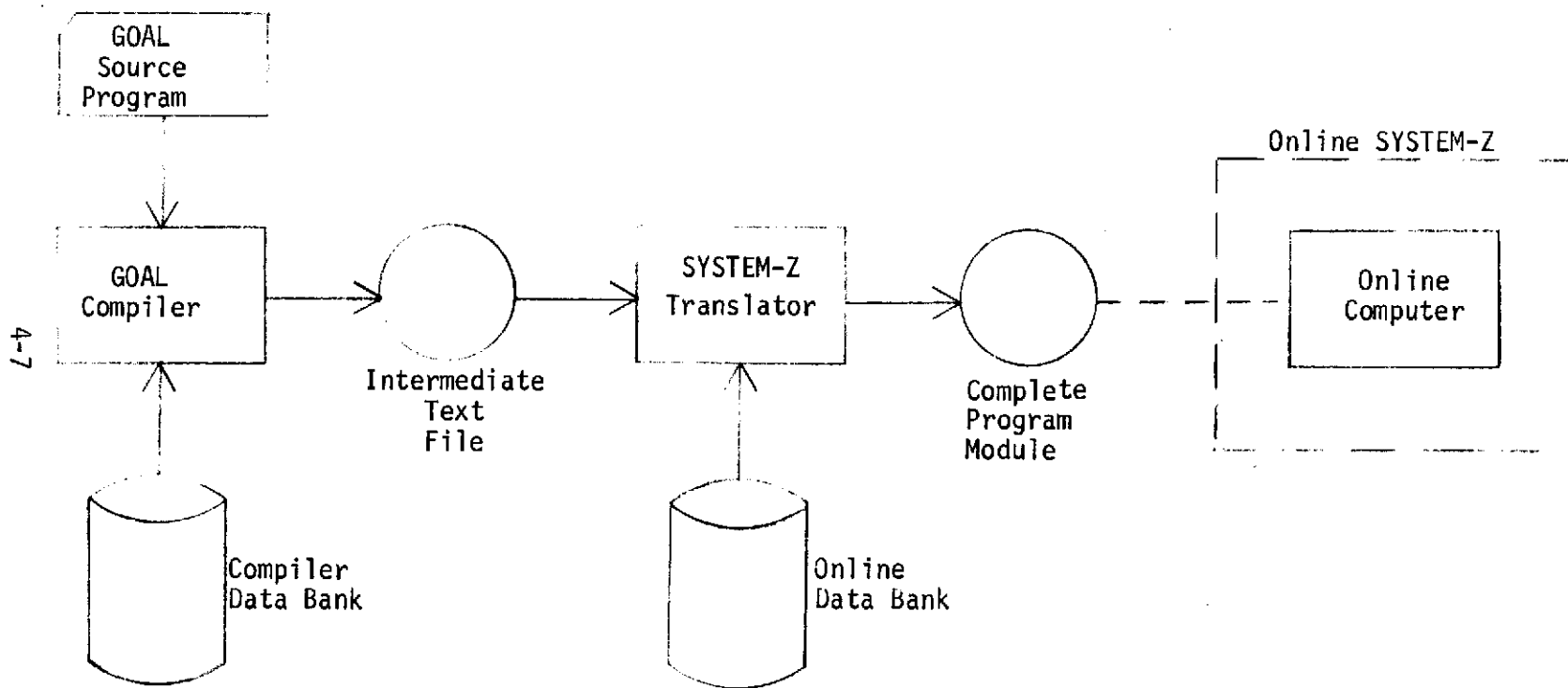
Figure 4-2-3. -   Entending Late Binding Support to an Online
                 GOAL System which Hosts a "Mini" Computer

5.0   MISCELLANEOUS

5.1   ANOTHER REQUIREMENT FOR AN ONLINE DATA BANK

Previous experience with the computer environment of SATURN/APOLLO has indicated that the execution of many process-control programs requires continual man-machine monitoring and communication. In particular, test programs used in fault isolation tasks generally rely on their human counterpart for redirection and specific remedial actions in the detection and reporting of hardware anomalies. Some type of keyboard-driven terminal appears to be the most effective man-machine communication medium. Many actions that originate as a keyboard request or entry deal actively, passively, or both, with the status of sensor-base hardware measurements. If the launch complex terminal operator is to have the capability of formulating queries relative to the status of measurements via his keyboard (either associated with or divorced from the operational requests of any test program which may be running at the time), a measurement addressing or name convention must be established.

As discussed previously, the SATURN/APOLLO era approach to measurement naming centered about hardware dependent terminology, e.g., the discrete measurement controlling "2nd stage LOX vents" would be referred to via its hardware routing "discrete output number 414" (or, simply "DO 414"). The rationale has already been presented as to why the GOAL test programming language addresses measurements by meaningful symbolic function designator names rather than by hardware dependent terms. The same justifications can be extended into the keyboard operations are to enable selection of a meaningful measurement convention.

That the measurement addressing technique associated with keyboard operations should be the same symbolic scheme as that which has been chosen for the GOAL test programming language seems almost indisputable. To do otherwise would make it mandatory that a single unique measurement be addressable by one name within the bounds of a test program but require still another distinct "name" be used in making keyboard referral to the same measurement. The implications of such an incompatibility can be readily projected into the amount of additional training required for keyboard operator and test program writer personnel; a significant amount of additional hard copy support documentation would also be required to keep track of legitimate synonymous measurement names and to draw parallels between the names used in manual keyboard operations procedural documentation and the names to be found in test program listings.

When making keyboard access to measurements by hardware descriptive symbols, e.g., "DO 2344" (discrete output # 2344), the terminal software can easily transform the symbolic address to an actual hardware address. Even an extreme such as the more cryptic 14-character name convention used for Digital Data Acquisition System (DDAS) measurements in the SATURN/APOLLO system lends itself to ready conversion by a strictly software method into its physical routing equivalent. Any type of name convention whereby certain or all of the characters of the name impart hardware relations or processing

information will always appeal to those personnel whose principal tasks involve heavy reliance on circuit diagrams or other systems drawings or prints. To the keyboard operator, such a name scheme severely lacks many human factors attributes. First of all, it is admitted that such schemes yield names that are quite cryptic and therefore easily entered via keyboard - but, such names because of their cryptic nature are difficult to remember exactly. Secondly, with a truly symbolic name such as "28VDC GROUND POWER", a spelling error (even a single character error) almost always results in an illegal name and the keyboard operator can be so notified via an appropriate error response from the online computer operating system. On the other hand, a single character misspelling with a hardware codified name can many times result in a syntactically legal name and the logical error can pass by undetected. Consider the example of the single character misspelling of "DO 4144" in lieu of "DO 4145" - both could be completely legal references to actual measurements. The terminal or operating system software packages have no method whatsoever of making the distinction between what was requested and what was intended. A third difficulty arises when a measurement must be rerouted and its addressing changed. In the case of codified measurement names, physical address modification dictates revision of the symbolic measurement name also. Such changes in turn force "red-line" or permanent editing of references to the rerouted measurements in system drawings and hard-copy documentation.

The introduction of function designator type name standards for keyboard usage implies that some type of data bank be online available to at least the computer that accomplishes keyboard request processing. This requirement can be merged with the desire for late binding facilities in the online system; a single online data bank can fulfill all obligations of both tasks.

5.2   THE EFFECT OF MEASUREMENT NAMES ON DATA BANK USAGE

As a general rule-of-thumb, when a file is used as a repository of variable data, it should be searched using search arguments that do not change, i.e., constant search arguments. Consider the everyday use of a telephone directory; a constant name is used as a search argument to locate variable data (address and phone number). Assume that an individual named "J. Doe" maintains residences in 20 locations. It would be possible to query the appropriate 20 telephone directories with the constant name "J. Doe" and retrieve 20 variable phone numbers. Now, if the directories were restructured to be ordered by telephone number rather than by surname, their usefulness (at least by human beings) becomes severely curtailed. For instance, with the present example, the task of determining the location of 20 occurrences of the constant "J. Doe" is resolved in one of two ways. The first technique requires that the 20 variable phone numbers (which are now the search arguments) be already known - in which case, of what use are the telephone directories? The second method dictates a serial search of the 20 directories; a messy proposition, even with the assistance of data processing equipment.

The above concept of utilizing constant search arguments to retrieve variable file contents is vital from an efficiency standpoint when considering the design of either the compiler or the online data banks. Specifically, measurement (function designator) names used as search arguments into data banks should be constant. This indicates that any name convention which uses characters of the symbolic name as indications of hardware routing or other access information is unacceptable for names used as data bank search arguments. The hardware indication characters of such names are subject to change, thus making the entire name a variable. Violations of the constant search argument principle leads to one or more of the following ramifications:

o   The possibility of the keyboard operator utilizing one measurement name convention and the test program writer another implies that at least two or more symbolic names must exist for each single measurement. The net effect is that all readability between measurement names as they exist on program listings and what appears at the output of operations terminals disappears.

o   If one name convention is used for keyboard operations and another for offline compiler usage, then two data banks of entirely distinct format and usage conventions must be maintained. The very real possibility of each data bank being in a different state of update becomes a constant maintenance headache - more so because such errors of incompatibility are virtually impossible to diagnose or prevent by installing software maintenance safeguards.

o   In the unlikely event that the GOAL language is forced to relinquish the function designator name convention and adopt a variable name scheme, then the present GOAL test programming luxury of addressing the "system under test" rather than the "test system" must be abandoned.

5.3   MULTIPLE ADDRESSING CONVENTIONS FOR ONLINE KEYBOARD USAGE

In the keyboard operations area there is a definite need for an abbreviated method of addressing measurements. The test program writer can afford the time required to code with pencil long (up to 32 characters) function designator names. A keyboard operator will find entry of such names a time consuming process - particularly during the times when he must address several measurements in rapid succession. In an effort to establish a name scheme that will be useful and acceptable to a wide variety of users, it is obvious that more than one distinct conventions are required. The most important consideration to remember is that these distinct methods can coexist and not be in contention with one another (if implemented properly). If, on the other hand, one name scheme must be selected thus excluding all other(s), then the resulting compromises and drawbacks must

also be acceptable since one name convention will simply not fulfill the needs and desires of all users in an equitable manner. In the following paragraphs, a technique will be demonstrated to prove that several name conventions can be used in a compatible manner. The conventions themselves plus their individual methods of implementation must be carefully selected to prevent the negative aspects of one convention/implementation from carrying over into another area. In some aspects of the system illustrated below, the negative aspects of one convention/implementation can be effectively cancelled by the positive features of one or more of the other convention/implementations.

Consider each sensor-based hardware measurement to be addressable as follows:

1.   By symbolic function designator name, e.g.,

<GROUND POWER COOLING PUMP>.

2.   By a unique "measurement number", e.g., 1436.

3.   By a symbolic name in which the characters indicate hardware routing of accessing information, e.g., 10/A14C2/16.

The function designator version will follow the current GOAL language requirements of one to thirty-two characters enclosed within brackets. The measurement number mentioned in 2., above, will be a constant number which has been assigned to this particular measurement; each measurement in the system will have such a number assigned to it, starting the sequence at one, 1. Of prime importance is that this number be considered just as "constant" as the symbolic function designator name - absolutely no hardware implications are present in these numbers. The hardware-dependent symbolic name mentioned in 3., above, can be formed using any rules suitable to the user group concerned with such terminology. The example extended above was generated strictly as an example. Since this name version utilizes hardware-dependent characters in its formation, the resulting name is not a constant. Certain drawbacks associated with using variable names as data bank search arguments were presented hitherto in Section 5.2. When a variable name version is used in conjunction with a fixed convention, most of the negative aspects of the sole use of the variable technique are nullified. The problem remains of assuring that the variable name is updated in a timely and precise manner whenever the inevitable measurement hardware changes occur.

Specific rules in regards the use of the three names presented in the example above are as follows:

o   The function designator and measurement number are constant and will not change throughout the life of the measurement. The hardware dependent version can be expected to change if hardware accessing modifications are required.

o   Only the function designator name convention can be
     used in the GOAL test programming language.

o   All three name versions can be used interchangeably
     for addressing measurements via keyboard terminal.

The use of all versions at the online terminals implies that a variety of
users can be serviced - each using the convention most suitable to his
needs. All three versions will seek hardware accessing and scaling infor-
mation from the appropriate measurements record of the online data bank.
The function designator and hardware dependent versions will require a
data bank directory search prior to locating the required measurements
record. A significant feature associated with the measurements number
name version is that the number can point directly to the measurements
record. In the example, a measurement number of 1436 would lead to access
of the 1436th data bank record; thus, no prior directory search would be
required. The use of measurements number names would be feasible in place
of the function designator types in the measurements table application
mentioned in Section 4.2. This would result in an appreciable time savings
in the update of the measurements table library; elimination of the direc-
tory search decreases the total number of input/output operations required
to locate a given measurements record in the data bank by approximately 75%.

As an example of terminal output, using any one of the three names mentioned
previously will result in the following terminal sequences:

    (keyboard
    input)   <GROUND POWER COOLING PUMP>?

    (terminal
    response)   <GROUND POWER COOLING PUMP> = M1436 = 10/A14C2/16 = ON

Thus, use of any one name will result in appearance at the terminal of the
selected name plus the other two versions in addition to the measurement
status. The output format illustrated above was chosen carefully since
certain drawbacks of using individual name conventions can be eliminated.
Consider the following advantages and features:

o   It was mentioned previously that a short name convention
     (like the measurement number) can be very attractive to
     a keyboard user from the standpoint of brevity and rapid
     entry. Such names are subject to misspelling errors that,
     in general, cannot be detected. For example, consider the
     following input/output sequence:

         (Inquiry)   M1436?

         (Response)   M1436 = ON

     If the user had entered the 1436 measurements number in
     error thinking that it was associated with the "STAGE II
     LOX VENTS", no indication of error can be ascertained in

abbreviated output sequence above. The extended output illustrated in the previous example would have served as an error indication when the terminal operator noted that the <GROUND POWER COOLING PUMP> name was not what he intended.

o   If a user was in doubt as to the exact spelling of one name version of a given measurement, and he knew either of the other two, then a simple keyboard query using the known name will suffice to determine the unknown version(s). This gives the online keyboard operator a convenient method of cross referencing measurement names. Consider the instance of a technician in a remote location who required the hardware dependent name for the "Ground Power Cooling Pump" for use in circuit diagram reference. Intercom correspondence with any available launch complex terminal operator will provide the required name.

5.4   IMPLICATIONS OF A TWO-DATA BANK SYSTEM

Consider an automation system which is supported by a general purpose GOAL compiler. For each active and distinct online system there also exists a separate translator program whose function is to tailor the general-purpose GOAL program module as produced by the compiler to the specific needs of each online system. Within the same system framework there exists a data bank to support compiler operations called the "compiler data bank". Each distinct online system will be supported by a separate "online data bank". The following salient features and considerations attendant to such a system:

o   Maintenance of the compiler data bank with regard to function designators is considerably simplified in a system with two data banks. For each function designator, the compiler data bank need contain only symbolic function designator name plus measurement type (discrete, analog, etc.), data. Specifically, no addressing data is to be present in this data bank; therefore, hardware addressing changes have no maintenance implications in this data bank. Function designator maintenance within the compiler data bank involves only addition and deletion of measurements.

o   Absence of hardware measurement data in the data bank associated with the compiler implies that only one logical data bank need be used at compile time to store measurement information for all programs. All measurement names for all distinct online systems can be present in the same compiler data bank.

o   Presence of an "online data bank" implies that the keyboard language and the offline compiled test program language have consistent name conventions.

o   To inject late binding capabilities into the GOAL test
     programming system, translator production of both a
     program module plus a separate measurements table were
     discussed in Section 4.2. As an indirect advantage of
     having the measurements used within each test program
     isolated as a separate data structure, consider the
     following hypothetical problem: "...the Stage-II AC
     Bus Overload Indicator is faulty, but cannot and need
     not be replaced until tomorrow. What test programs,
     if any, would be effected by unavailability of this
     measurement?..." Such a question could be introduced
     to the online system via keyboard. A relatively simple
     software utility routine could be devised to search the
     tables located in the measurements table library for an
     occurrence of the measurement name(s) of concern. The
     name(s) of the test programs containing such measurements
     would then be output to the terminal that initiated the
     search. The attractive aspect of seeking an online
     solution to such problems is that the answers so derived
     are generated very rapidly and 100% confidence can be
     placed in the accuracy of the results. Such information
     can form the basis for a great many technical and manage-
     rial decisions that would otherwise be either impossible
     to resolve, or so time consuming as to be impractical.

o   Measurement addressing changes are introduced to the
     online system via a two-step process. First, a utility
     program is used to inject addressing modifications to
     the online data bank. This program could be run on a
     computer system distinct from the computer associated
     with the online data bank. This would be quite feasible
     in the event that the data bank was to reside on a remov-
     able disk pack. The utility operation could also be
     accomplished in an interactive manner with commands
     entered via keyboard terminal. The data bank update
     process involves locating the required measurement record
     in the online data bank by using the function designator
     name as the search argument. Once the record has been
     found, the required measurement addressing data can be
     modified and the revised record rewritten back to the
     data bank. Once all data bank update(s) have taken
     place, the second step involves starting a utility
     service program to propagate any measurement addressing
     changes into the measurements table library. The action
     of this service process would be as follows:

     *   Fetch a measurements table from the library.

*   Compare each function designator name in the
    table against the list of modified function
    designators. If present in the table, revise
    the table addressing data accordingly.

*   If any table revision has been necessary, rewrite
    the table back to the measurement table library
    and proceed with the next table. If no changes
    were required in the current table, proceed im-
    mediately with the processing of any remaining
    table(s) in the library.

Modification of a program's measurement table would likely
be logged as a significant data processing event. If the
data bank update was accomplished online, the keyboard
operator would be notified of the results of the update
by an appropriate output message indicating program name
and the function designator(s) that underwent change. Such
changes could be introduced into an online and operational
system in a convenient and timely manner - the concept of
introducing an expedient temporary hardware change or over-
ride that is destined to remain in effect for a short
duration (say, a manner of hours) and then reverting to a
normal system configuration becomes an attractive and
practical possibility. A crude estimate of the time re-
quired to accomplish a data bank update appears in the
following example:

Let: P = 100 = Total number of test programs in
the current online program library

R = 10 = Average number of DASD read operations
required to completely access a mea-
surements table for a single program

M = 50% = Percentage of programs whose measurement
tables will require modification due to
recent measurement addressing change

W = 3 = Average number of rewrite operations
required for those tables to be modified

T = 8 = Average number of read/write operations
that can be accomplished per second on
the DASD used for the measurement table
library

Negligible computer central procesor time required for
all operations involved in the measurement table update

Then:

Total time required for
measurements table $= \dfrac{P\ R}{T} + \dfrac{P\ W}{T} \times \dfrac{M}{100}$
library update (seconds)

$= \dfrac{100(10)}{8} + \dfrac{100(3)}{8} \times \dfrac{50}{100}$

$= 143.75$ Seconds

APPENDIX A

THE DATA BANK FILE DESIGN

A.   INTRODUCTION

Within this Appendix are indicated the criteria which evolved into the
functional design of the data bank file of the first GOAL compiler
implementation. Although the original implementation utilized IBM
Direct Access Storage Devices (DASD) and support software, it is felt
that the file design could be successfully transcribed to the DASD of
other manufacturers. Specifically, the file design relies on no fea-
tures that are found only on IBM DASD or support software. In general,
the following functional DASD capabilities must be present to support
the current data bank design:

o   The DASD must be capable of supporting a data set
     organization of fixed length logical records.

o   Random read and write access to any one of the (n)
     records within a data set of (n) total records must
     be possible by specifying the relative record number
     utilizing the first record of the data set as the
     origin. With the additional requirement of fixed
     length logical records, the absolute address of a
     given record can be easily determined by knowing the
     relative record position.

o   To accomplish data bank maintenance in the same fashion
     as that of the current IBM implementation, a general
     purpose sort utility program is required. If in another
     system such a program is not available or feasible,
     adequate computer storage must be available to accomplish
     the sort operation in a memory resident fashion.

A.1   GENERAL FILE DESIGN

The records of any file must be logically organized such that they can be
inserted or retrieved for processing. In the selection of a certain file
structure, all of the following are to be considered:

o   File Creation - separate program(s) must usually be
     developed to initialize the first copy of a file.
     Thereafter, a revised or new copy of the file is
     created by update of an existing version. The GOAL
     data bank requires a special initialization program
     to prepare the DASD storage before records can be
     inserted.

o   File Use - Usually, a given file application will
     dictate that the file be usable either to read records
     from or to write into. The data bank application
     requires both read and write capability within the
     same program.

o   File Maintenance - Very few files can be envisioned
     that will not require either periodic or sporadic
     maintenance. The term "maintenance" is generally
     construed to mean the addition, replacement, changing,
     or deletion of records. The data set design will in
     some ways dictate and other ways be constrained by the
     maintenance techniques or requirements. In the case
     of the data bank, the existence of a sort utility program
     makes directory maintenance immensely easier.

o   File Backup - This may be as simple a process as copying
     the file on to another medium (magnetic tape is popular
     for this usage) to enable partial or complete recovery
     from an unforeseen disaster to the file prime copy. The
     GOAL data bank system has been structured such that the
     directory data set can be completely destroyed and
     successfully recovered using the contents of the data
     bank set alone. The data bank data set must be backed
     up by copy to another medium such as tape. As this
     time, no specific software modules have been provided
     to effect these recoveries.

A.2   DATA FILE CHARACTERISTICS

The following inherent characteristics form the basis for selecting an
efficient method of file organization:

o   Size - A file so large that it cannot be all online
     (available to the system) at one time dictates very
     specific organization and processing techniques. The
     data bank was designed with the intent of having all
     of the file online, but resident on DASD. Generally,
     only one record at a time will be of interest in data
     bank utilization. These records must be randomly
     accessible.

o   Growth Potential - It was convenient to size the original
     data bank at a capacity (8000 entries) considerably above
     that currently being used (approximately 2000 entries).
     Thus, the data bank may expand significantly without
     requiring an overall size increase of the data set. If
     and when an increase in overall capacity is required,
     this can be accommodated with modest effort within the
     framework of the original data bank design. The original
     data bank support and maintenance modules were written

in FORTRAN - because of FORTRAN language constraints, a data bank resizing will require changes to the declarations section of several modules. Except at very infrequent intervals, it is considered unlikely that resizing will be required; the inconveniences resulting from these software changes were not there- fore, considered prohibitive.

o   Activity - In regards the amount of activity, an inactive file may be referred to so infrequently that the particular file structure or processing technique chosen does not matter. The GOAL data bank is considered a file which typifies the other extreme, i.e., an active file to be heavily used by the GOAL compiler. The file structure was chosen carefully to avoid an inordinate amount of time searching for desired records.

The percentage of activity is also a factor of consideration. The data bank is an application which requires retrieval of only a low percentage of the available records in a file during an average processing run. This use implies a random type organization such that any record can be located con- veniently without having to scan all or a large number of other records in the file. In the GOAL data bank each record may be essentially considered an independent data entity having no relationship (physical or logical) with any other record of the file.

In some instances the activity distribution can be a file design factor. If some group of records are more frequently significant than others, some method of locating these records is in order so that they can be effectively fetched. The Data Bank Reference Records in the GOAL data bank are an example of records of this type; they have been located in a fixed specific area within the file so that they can be efficiently searched and processed.

A.3   FILE PROCESSING CHARACTERISTICS

The usage of a file usually dictates the type of processing in which the file will be involved. Sequential processing of a file implies that the records will be processed in a monotonically increasing or decreasing sequence. A deck of cards or magnetic tape is ideally suited to this application type. To be effective, the records of a sequentially processed file must be sorted according to the sequence to be followed in processing. The records are then written in a physically contiguous fashion in the file. Thus, the next record to be processed is always immediately adjacent to the current record. Although sequential processing can be extremely rapid in many applications, its use demands that all input transactions be batched (collected) and sorted to the same physical key sequence as the file before processing.

Random processing allows (or, rather, requires) that all records of the file be accessible and writeable in a random order. No collection or sorting of input transactions is required before processing. The same could be said of the records within the file itself, i.e., there is no required record sort sequence or rules concerning which records need be physically adjacent. The GOAL compiler requires access to the data bank in a strictly random manner; the actual method chosen for implementing the data bank allows sequential processing also.

A.4   DATA BANK CONTENTS

GOAL compiler support was the prime function of the data bank in the first implementation of the GOAL language. This encompasses data records of the following type:

o   Function Designators - In almost all instances, a function designator record contains sensor-base hardware data such as measurement type (discrete/analog, input/output, etc.) and physical hardware addresses.

        Note - The term "function designator" is used within the GOAL language for purposes other than symbolic names for sensor-based measurements.

o   GOAL Subroutine References - These records provide abbreviated system dependent names for GOAL subroutines.

o   GOAL Macros - These records contain user-written and system macros to be used as writer aides during compilation of a GOAL test program.

The functional design presented in Appendix B indicates a few other record types in the data bank file used for file structuring and searching aids. Most (possibly, 99+%) of the data bank file records will be function designator records. A direct access (random) file structure for relative record retrieval must contain fixed length logical records. Since the file contains records of varying data context, the fixed record length must be large enough to accommodate all of the data of the largest record type.

Records of other types will then utilize only a portion (and, consequently, waste the remainder) of the total record capacity. Fortunately, the record type requiring the largest record capacity was the function designator type; total waste space will therefore be minimized in the file. An individual function designator will require only one data record for its description. A GOAL subroutine will require only one record for its name reference. A macro will require two or more records.

A.5   DATA BANK USAGE

The GOAL language stipulates that each function designator, macro, or
subroutine is identified by a unique 1 - 32 character symbolic name.
The GOAL compiler queries the data bank providing only this 1 - 32
character name. Using this name as a search argument (record key), the
data bank software must either locate and retrieve the required record
or signify that a record associated with the supplied key does not exist
within the data bank.

Features of the GOAL language also require that a number of uniquely named
'data banks' is resident in one data bank file.

APPENDIX B

THE DATA BANK FUNCTIONAL DESIGN

B.   INTRODUCTION

Within this Appendix is presented a functional description of the data
bank design as implemented for use by the initial version of the GOAL
compiler.

B.1   BASIC SPECIFICATIONS

The principal functions of the GOAL data bank are outlined in the following:

    o   The data bank shall provide for the storage and retrieval
         of uniquely named GOAL function designator records, macro
         records, and subroutine reference records.

    o   Function designator, macro, and subroutine records are to
         be individually retrievable by specifying: (1) the name
         of the data bank in which the record is assumed to reside,
         and; (2) the unique record name. Record names are assumed
         to be unique only within the bounds of the specific data
         bank in which they are found, e.g., the record called A
         can be the only record with that name within the data bank
         in which it is found. There can be, for instance, a record
         A within a data bank called X; another record A can exist
         in data bank Y, etc.

    o   The data bank system shall provide a file organization
         capable of supporting and selectively using several
         uniquely named independent data banks. The total number
         of data banks to be supported at any one time shall be
         selectable at file initialization time. Provision has
         been made to allow increase of the number of allowed data
         banks after file initialization without requiring complete
         reinitialization of the system; utility software has not
         been provided to accomplish this extension at this time.

B.2   GENERAL APPROACH

The data bank will be a disk resident file constructed primarily for random
(non-sequential) processing.** The file structure is to support random access
to any uniquely named record (member) by supplying the record's symbolic name
(record key). To support this operation, two disk resident data sets will be
required. One will be referred to as the "data bank," "data bank file," or

---

**The file structure chosen allows for sequential as well as random processing.

B-1

"data file." This data set will contain the GOAL function designator, macro, and subroutine records. The second data set will be deemed the "data bank directory," or "directory." The directory will contain an index to all named records in the data bank. Each index entry will consist of the record name, an indication of the uniquely named data bank in which the record logically resides, and the relative record position of the record within the data bank data set. All of the index entries within the directory shall be sorted in a manner that is conducive to efficient search.

The operational concept of the data bank is analogous in every way except physical record layout to the technique required to use an everyday telephone directory. The telephone number and address are the data records of the telephone directory; the names are the record keys. If one is to conduct an efficient search through so large an assortment of keys, the keys must be placed in a physical structure in such a way that some type of search is strategically possible. For alphanumeric keys (such as human names or data bank record names), sorting the keys into alphanumeric collating sequence is a method which allows effective human or machine search.

Figure B-1 illustrates the functional relationship between the distinct elements of the data bank system. In the data bank design, the data portion of the file is resident in the data bank data set while the sorted grouping of record keys is to be found in a distinct data set called the directory. Accompanying each key in the directory is the record location (in the data bank) where the data associated with that key is to be found. This split arrangement of keys in one data set and data in another has four distinct maintenance advantages over the combined record technique of the telephone book, to wit:

o   The directory must be in sorted key sequence for search
     reasons. When maintenance decrees the addition or deletion
     of records, the directory must be re-sorted to accommodate
     the new keys and to eliminate the old. It is to be noted
     that the keys must be sorted to accomplish this aim - there
     is no requirement that the data be rearranged in any manner.
     By involving only the contents of the data bank directory in
     the maintenance key sort operation, the speed at which both
     the sort and the directory data set reconstruction can be
     accomplished is significantly enhanced.

o   With distinct directory and data bank data sets a larger
     number of keys can be present in a given size of record than
     would be possible if both keys and data were present in the
     same size record. The speed at which the directory search
     can be accomplished will be almost wholly dependent on how
     many input operations are required to fetch records contain-
     ing keys.

If a small number of keys are present in each record, then many records may be required, etc. A crude analogy exists in the example of the telephone directory - if the current directory page size were increased to newspaper size sheets, one would have to turn fewer pages to find a given key. In contrast, if the telephone directory were reprinted on 3 x 5 cards, a great deal of page turning would be required.

o   Although the directory must be maintained in sorted order, there is no reason why the effort must be expended to maintain a sorted data bank file (containing the data). Put in another way, the data records, once entered into the data bank, need never be reshuffled to accommodate new or deleted records. This feature leads to a significant savings in maintenance time. As an illustration to the contrary, consider the maintenance required for a non-random sequentially organized data set, e.g., a magnetic tape. The time spent in reshuffling (recopying) the entire data set to accommodate one record addition/deletion/change could comprise 99.99% of the total maintenance time. This disproportionate time mix degrades even further as the data set size increases.

o   By avoiding a reshuffle operation on record addition/deletion in the case of the data bank file, it was possible to adopt a scheme for effectively reclaiming the space vacated by deleted records. Once a record has been deleted, the record position becomes available for use in adding a new record in a subsequent maintenance run. During data bank initialization, all free records are chained together to form what is referred to as the "free record queue," i.e., all record spaces available for use. When a record is to be added to the file, the first available record on the free queue is removed from the chain and used for the addition. When a record is deleted, the record space is reinserted to the head of the free queue chain, thus making the space reusable.

B.3   GENERAL DIRECTORY STRUCTURE

The illustration of Figure B-1 depicts the directory as a single table containing one entry for each record in the data bank. Such a directory that contains (n) entries for (n) record references is deemed an "inverted" directory. If it were possible for the table to be totally memory resident at use or maintenance time, access to any record of the file would proceed at maximum speed - a simple memory table search followed by one I/O read

would be required. Unfortunately, such a single-level table becomes prohibitively large even for a relatively modest size data bank file. It must therefore be broken into segments of a size such that any one can be conveniently contained in computer memory.

Figure B-2 indicates an example of a directory that has been structured into a series of blocks ("directory blocks" or "directory records") in two levels. In general, two types of blocks are present - one "master" block and several "lower level" blocks. The lower level blocks accumulatively represent the single inverted directory table broken into segments of a more practical size. Each individual entry in a lower-level block consists of a record name and a pointer (record number) to that record in the data bank file. If the directory contained only lower-level blocks, any one of which could be memory-resident at a given time, then a block-by-block serial search would be required to locate a record in the directory. Access of any one directory block would require one I/O read operation. As a method of eliminating the serial search and thus enhancing overall directory search speed, a master directory block has been introduced as shown in Figure B-2. This block indicates the contents of all of the lower-level blocks of the directory. Each master block entry points to a lower block and indicates the highest (in collating sequence) named entry in that block. If the master block can be made memory-resident, a search for a given record proceeds as follows:

1.   Via the master block, determine the lower-level block which must contain the sought name. In the example of Figure B-2, a name "D" must be contained in block 3 since block 2 contains as the highest entry "B", and block 3 as the highest entry "G".

2.   Access the required lower-level block.

3.   Search the lower-level block for the required name.

4.   After finding the directory entry containing the record name in the lower-level block, use the record number portion of the entry to fetch the sought record from the data bank file.

The above process requires only two I/O read operations to find and retrieve any given named record from the data bank file; note that the assumption has been made in this instance that the master block was memory-resident before the search started.

Since the directory itself is a direct-access (randomly organized) data set, it will be significantly easier to manipulate if all records (blocks) are of constant or fixed record length. This results in a "balanced" directory - each block contains a constant number of directory entries. The unbalanced directory of Figure B-2 has been rearranged to a balanced form in Figure B-3.

The "directory blocking factor" (N), i.e., the total and constant number of directory entries possible per directory block, for this illustration is four (4). Note that for a two-level balanced directory there will always be (N) blocks at the lowest level; this then implies the following size relationships:

Given a directory blocking factor (N):

Total number of blocks
in the directory data set $\quad = N + 1$

Directory block size (words
or bytes) $\quad = N \times$ (Directory entry size in words or bytes)

$\quad = $ the amount of computer memory that must be present for one directory block

Total number of named records
that can be referenced in the
directory $\quad = N^2$

Using the above as a basis for determining directory blocksize, assume that a data bank to support a total of 8,000 entries is required:

Let (N) $\quad = 90$

Total entries supported $\quad = N^2 \quad = 8,100$

Directory blocksize $\quad = N \times$ (directory entry size)

$\quad = 90 \times 76$ (current implementation directory entry size)

$\quad = 6,840$ bytes

The directory blocksize in the above instance has grown to a very large entity-requiring a large memory block for support. If the master block is to remain memory-resident, and the lower-level blocks retrieved as required one-at-a-time, a total of 13,680 bytes of computer memory would be required to support directory operations alone. This is considered a severe cost to service a data bank file of 8,000 records. The eventual size of the operational GOAL data bank could run as high as 30,000 to 100,000 entries. A two-level directory to support such a sizeable data bank would require directory blocks of enormous capacity.

The solution selected to obviate the above sizing problem is illustrated in Figure B-4. Here, the two-level directory has been expanded vertically into a three-level balanced version. Figure B-5 represents a tabular comparison

of the size potentials between two-level and three-level directories. The calculations are related to an IBM 2314 disk unit which supports a total track capacity of 7,294 bytes (data plus inter-record gaps, as required). The directory blocking factor (N) was considered in the range of 10 to 50 entries per block - this upper limit will support a data bank of 125,000 entries with a three-level directory.

B.4   GENERAL CONTENTS OF DIRECTORY RECORDS

Figures B-6 and B-7 depict the layout of the master, upper-level, and the lower-level directory blocks. Each block is seen to be composed of a block header and (N) directory entries. The block header indicates:

o   the record (block) number in the directory file

o   the total number of entries possible in the block (N)

o   the total number of directory entries currently in use in the block

Each directory entry is seen to contain:

o   A delete field - used during maintenance to delete reference to an existing record

o   A record name - consisting of a data bank reference number and a 1-32 alphanumeric character symbolic name

o   A pointer to another record (record number) either in the directory data set (master and upper-level blocks) or to the data bank file (lower-level blocks only)

As indicated above, the total record name contains two significant portions - the data bank reference number and the proper alphanumeric name. Each uniquely named data bank is assigned a unique reference number when the data bank is originally created. This reference number becomes the first 32 bits of the names of all records contained within that data bank. The data bank reference number is actually the record number of the applicable Data Bank Reference Record in the data bank file.

Figure B-8 indicates pictorially the relative positioning of all blocks to be found in the directory data set. The actual block numbering scheme and order of loading is indicated in Figure B-9.

B.5   DATA BANK FILE CONTENTS

Figures B-10 and B-11 tabularize the general record types and contents of the records to be found in the data bank file. The relative positioning of the records within the file was indicated in Figure B-8. All data bank records are fixed in length and each contains a header followed by a data segment.

B.5.1   Data Bank Capcity Record

This record is the first record of the file. The data bank initialization program composes this record in accordance with control card input when the data bank file is first established. Figure D-1 indicates the initialization action. Only a portion of the header and none of the data portion is used in this record type. A brief description of the field contents is given below; the exact record format is to be found in Appendix E.

Record Number - This record is located first in the file so that the use and maintenance modules can find it without search.

Total number of Records Available (RAVAIL) - This is the total number of record spaces which were allocated for the data bank file when it was originally created. This value is established during initialization (read from a control card).

Total Number of Records Still Unused (FQTOT) - This field represents the total number of records remaining on the Free Queue for use in adding new records to the data bank. This is initialized to the total number of records reserved for the entire file minus those used for the capacity record and the data bank reference records. Symbolically:

$$FQTOT = RAVAIL - (DBMAX + 1)$$

Pointer to Head of Free Queue (FQPTR) - A field indicating the first record available for allocation in the data bank. The next time that a new record is added, the record at the head of the Free Queue is used. This field is initialized to point to the first record beyond the data bank reference records in this file. Symbolically:

$$FQPTR = DBMAX + 2$$

Maximum Number of Allowed Data Banks (DBMAX) - This is the total number of uniquely named data banks that will be allowed to exist. DBMAX is initialized to the amount read in via control card during initialization plus one more to allot for a mandatory data bank called "SYSTEM." This SYSTEM data bank will become a part of the system at initialization time.

Total number of Data Banks Now in Use (DBCUR) - This field reflects how many data banks are currently being used. The value is initialized to one (1) - to account for the mandatory SYSTEM data bank.

Directory Blocking Factor (DBF) - This value has been hardcoded into the data bank initialization program and is set at 20 for the current implementation. A change of this value will dictate several changes to different program modules - discussion of these changes is covered in this document, Appendix C.

B.5.2   Data Bank Reference Records

These records follow the capacity record in the data bank file. One
reference record exists per allowed data bank - the user specifies via
control card during data bank initialization as to the total desired. A
factor of thirty (30) will be used for the original implementation. The
first data bank reference record (record number 2) will be initialized for
the data bank called "SYSTEM." All other reference records are initialized
to the unused state and will remain so until data bank(s) are established
during normal maintenance. A detailed discussion of the record fields is
given in Appendix E. A brief coverage of the general contents is below:

> Record Number - This set of records appears in the data bank
> starting at record position number 2. The data bank called
> SYSTEM will be initialized at record position 2. Record positions
> will be assigned to subsequently established data banks on a first-
> come, first-serve basis.

> Forward Chain - All data bank reference records will be chained
> together at initialization time. This chain provides an efficient
> method of serial search to determine if a given data bank exists
> or not. Also, when adding a new data bank reference, the chain is
> followed until an unused reference record is located.

> Type Field - All but the type field associated with the reference
> record for the mandatory SYSTEM data bank will be initialized to
> one (1), which implies the record is unused and free for use. When
> maintenance requires the addition of a new data bank, the reference
> record chain is followed until a vacant (type = 1) record is located;
> this record position will then be used for the new data bank.

> Maximum Number of Allowed Data Banks - Established during file
> initialization.

> Data Bank Number - The data bank reference numbers are established
> during initialization to be equal to the record numbers in which
> the fields appear. Each named data bank is referenced by the user
> via a symbolic (alphanumeric) name. Within the data bank software,
> all references to data banks are made by use of data bank reference
> numbers. This number becomes a portion (the leading 32 bits) of
> each record name in the system that "belongs to" or is associated
> with that data bank.

> Data Bank Name - The 1-32 alphanumeric character data bank name.
> The first character must be a letter.

> Revision Label - This field is optional with any given data bank.
> If present, it is a 1-32 character name. If the revision label is
> not used in a given record, this field will be totally blank.

B.5.3   Other Data Bank File Record Types

Table B-12 tabularizes the functional contents of all record types to be found in the data bank file. Data Bank Reference records have already been discussed. The remaining record types consist, in general, of a header and a data area. Function Designator and Subroutine Name records will always occur singularly - accordingly, the header (see Figure B-11) for these types will be consistent in format as indicated in the following:

Record Number   - As required

Forward Chain   - Always null (zero)

Type Field   - See Table B-12; as required

Total Number of Records This Chain - Always one (1)

Pointer to Last Record, This Chain - Always points to this record number (the record containing the field)

Data Bank Number and Record Name Fields - Always used

Print Expansion - One to 32 character name; record name as it is to appear on compiler listings; this may or may not be the same as the record name field

Data Portion - Used as required for record type; Appendix E indicates an exact record description.

Macros will be represented in the data bank file by a Macro Header record followed by one or more Macro Skeleton records. The records pertaining to one macro will be linked via their respective forward chain fields. Figure B-11 indicates that the header of the Macro Skeleton records is abbreviated from the type normally found on other record types. The macro skeleton card image occupies a portion of the header as well as all of the normal data segment. The maintenance processing program will scan each macro skeleton card, locate each occurrence of a formal parameter, and replace each occurrence with the following character sequence:

1st Character   = & (ampersand)

2nd Character   = Single digit, 1-0, representing formal parameters one through 10, inclusive

3rd Character   = & (ampersand)

4th through
Next to Last   = & (ampersand)

Last Character   = Blank

A maximum of ten (10) formal parameters will be allowed for any given macro in the current implementation.
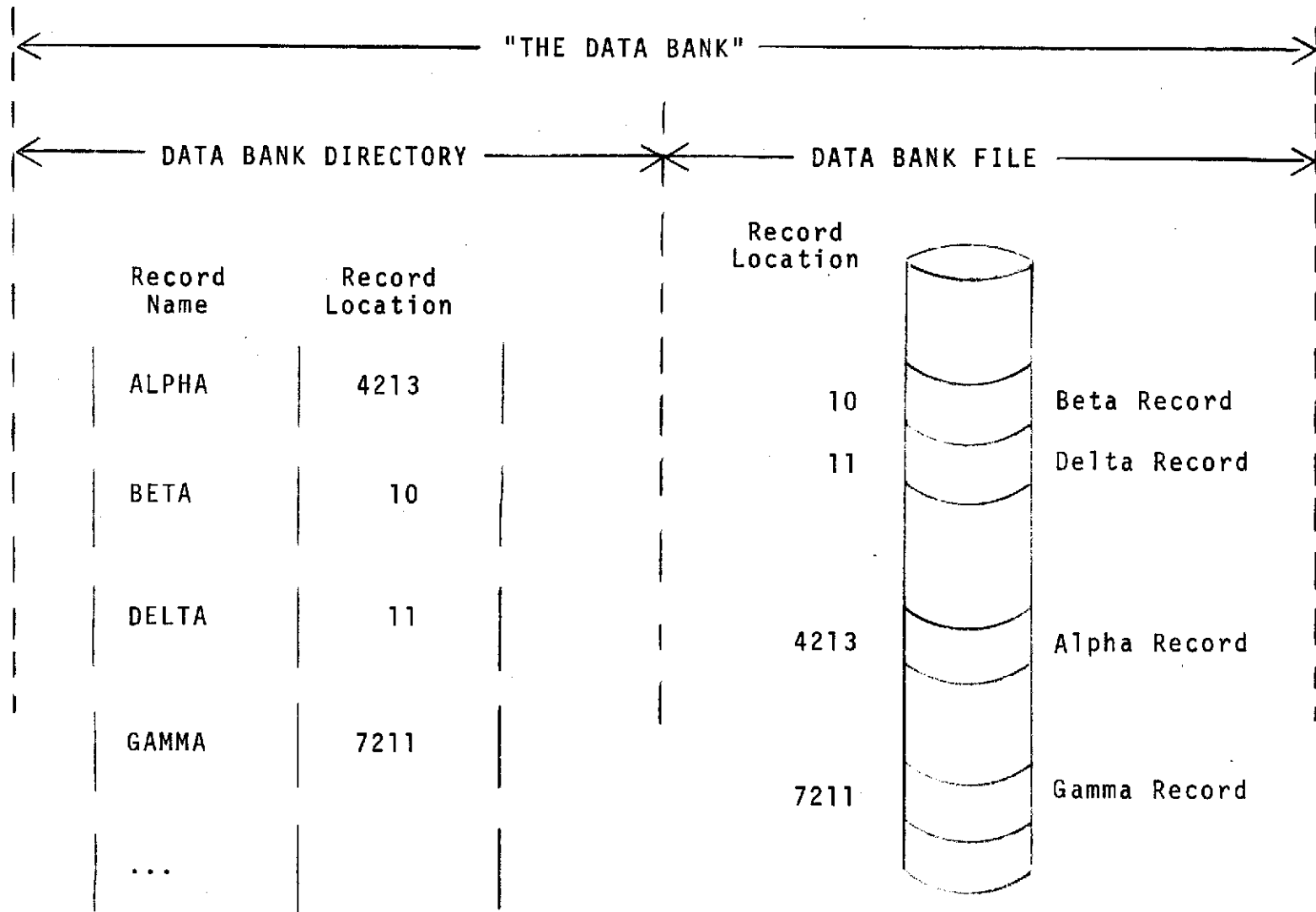
Figure B-1.   FUNCTIONAL RELATIONSHIP OF DATA BANK ELEMENTS

Figure B-2.

Block 1

G	2
M	3
X	4
N/U	N/U

DIRECTORY MASTER
BLOCK

(N) Entries

ILLUSTRATION OF TWO-LEVEL
BALANCED DIRECTORY

(N) = Directory Blocking Factor;
total number of directory
entries per directory block

"N/U" = Not Used At Present;
Available for Future Use

Block 2

A	**
B	
D	
G	

Block 3

J	
K	
L	
M	

(N) Entries
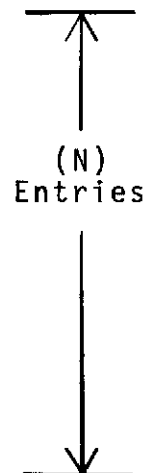
Block 4

Q	
T	
W	
X	

Block 5

N/U	
N/U	
N/U	
N/U	

** Pointers to Databank File Omitted
in this Illustration

Figure B-3.

Figure B-4. SAMPLE DATABANK DIRECTORY SHOWING BLOCK STRUCTURE
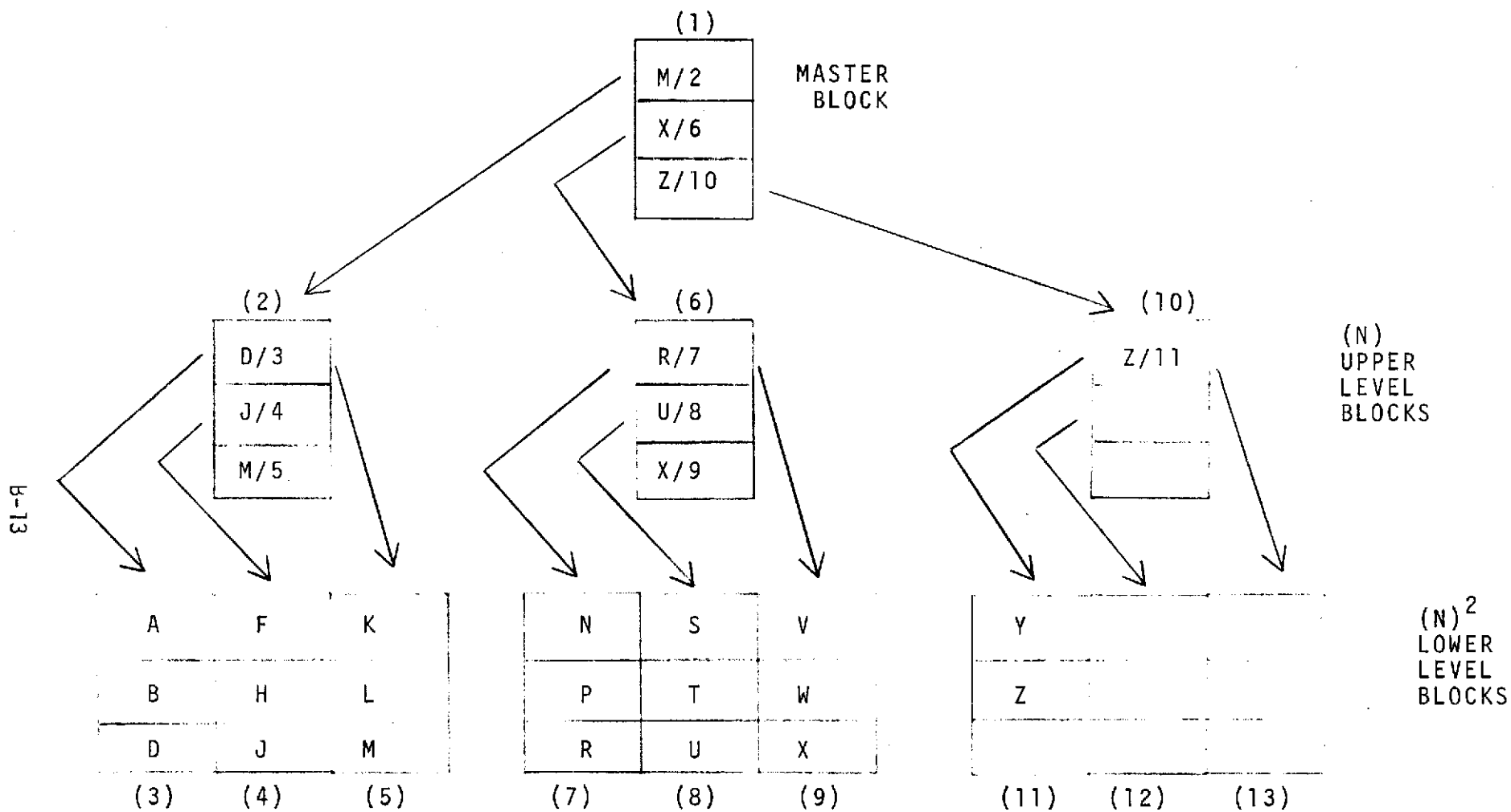
Directory Blocking Factor = N = 3 in this example

Maximum # Databank Member Names = N^3 = 27 (Only 20 used in this example)

Total # Directory Blocks Required = $(N)^2 + (N) + 1$ = 13 in this example

Directory Block (Record) Numbers Shown in Parenthesis Above

***ASSUMING 76-BYTE DIRECTORY ENTRY SIZE*
***BLOCKSIZE INCLUDES 12-BYTE HEADER*

N	BLOCKSIZE	BLOCK PER TRACK	WASTE PER TRACK	*********** 2-LEVEL ***********			*********** 3-LEVEL ***********		
				TOTAL ENTRIES	TOTAL DIRECTORY BLOCKS	TOTAL SPACE ALLOCATION	TOTAL ENTRIES	TOTAL DIRECTORY BLOCKS	TOTAL SPACE ALLOCATION
10	772	8	1118	100	11	2	1000	111	14
11	848	7	1358	121	12	2	1331	133	19
12	924	6	1750	144	13	3	1728	157	27
13	1000	6	1294	169	14	3	2197	183	31
14	1076	6	838	196	15	3	2744	211	36
15	1152	5	1534	225	16	4	3375	241	49
16	1228	5	1154	256	17	4	4096	273	55
17	1304	5	774	289	18	4	4913	307	62
18	1380	4	1774	324	19	5	5832	343	86
19	1456	4	1470	361	20	5	6859	381	96
20	1532	4	1166	400	21	6	8000	421	106
21	1608	4	862	441	22	6	9261	463	116
22	1684	4	558	484	23	6	10648	507	127
23	1760	3	2014	529	24	8	12167	553	185
24	1836	3	1786	576	25	9	13824	601	201
25	1912	3	1558	625	26	9	15625	651	217
26	1988	3	1330	676	27	9	17576	703	235
27	2064	3	1102	729	28	10	19683	757	253
28	2140	3	874	784	29	10	21952	813	271
29	2216	3	646	841	30	10	24389	871	291
30	2292	3	418	900	31	11	27000	931	311
31	2368	2	2558	961	32	16	29791	993	497
32	2444	2	2406	1024	33	17	32768	1057	529
33	2520	2	2254	1089	34	17	35937	1123	562
34	2596	2	2102	1156	35	18	39304	1191	596
35	2672	2	1950	1225	36	18	42875	1261	631
36	2748	2	1798	1296	37	19	46656	1333	667
37	2824	2	1646	1369	38	19	50653	1407	704
38	2900	2	1494	1444	39	20	54872	1483	742
39	2976	2	1342	1521	40	20	59319	1561	781

Figure B-5. (1 of 2)

***ASSUMING 76-BYTE DIRECTORY ENTRY SIZE
***BLOCKSIZE INCLUDES 12-BYTE HEADER

		BLOCK PER TRACK	WASTE PER TRACK	********** 2-LEVEL **********			********** 3-LEVEL **********		
N	BLOCKSIZE			TOTAL ENTRIES	TOTAL DIRECTORY BLOCKS	TOTAL SPACE ALLOCATION	TOTAL ENTRIES	TOTAL DIRECTORY BLOCKS	TOTAL SPACE ALLOCATION
40	3052	2	1190	1600	41	21	64000	1641	821
41	3128	2	1038	1681	42	21	68921	1723	862
42	3204	2	886	1764	43	22	74088	1807	904
43	3280	2	734	1849	44	22	79507	1893	947
44	3356	2	582	1936	45	23	85184	1981	991
45	3432	2	430	2025	46	23	91125	2071	1036
46	3508	2	278	2116	47	24	97336	2163	1082
47	3584	1	3710	2209	48	48	103823	2257	2257
48	3660	1	3634	2304	49	49	110592	2353	2353
49	3736	1	3558	2401	50	50	117649	2451	2451
50	3812	1	3482	2500	51	51	125000	2551	2551

Figure B-5. (2 of 2)

(N) Individual Directory Entries Per Block

BLOCK HEADER	DIRECTORY ENTRY # 1	DIRECTORY ENTRY # 2	...	...	DIRECTORY ENTRY # (N)

TYPICAL BLOCK HEADER FOR MASTER (RECORD #1) AND UPPER-LEVEL BLOCKS

Block (Record) Number	Total # Entries Possible in this Block	Total # Entries now in Use in this Block

← Header Length = 12 Bytes →

(N) = Directory Blocking Factor

A TYPICAL DIRECTORY ENTRY FOR MASTER OR UPPER-LEVEL BLOCKS

Pointer (Block #) to Block at Next Lower Level	Delete Field	Databank Reference Number	Search Key (Member Name) (32-Chars)

← Length of Individual Entry = 76 Bytes →

Figure B-6.   DATABANK DIRECTORY: MASTER AND UPPER-LEVEL BLOCK LAYOUT

(N) Individual Directory Entries Per Block

| BLOCK HEADER | ENTRY # 1 | ENTRY # 2 | ... | ... | ENTRY # (N) |

| Block (Record) Number | Total # Entries Possible in this Block | Total # Entries now in Use in this Block |

← Header Length = 12 Bytes →

(N) = Directory Blocking Factor

| Pointer (Record #) to Member Record in Databank File | Delete Field | Databank Reference Number | Search Key (Member Name) (32-Chars) |

← Length of Individual Entry = 76 Bytes →

Figure B-7. DATABANK DIRECTORY: LOWER-LEVEL BLOCK LAYOUT

Record #1

Master Block	(N) Upper-Level Directory Blocks $(N)^2$ Lower-Level Directory Blocks	DATABANK DIRECTORY
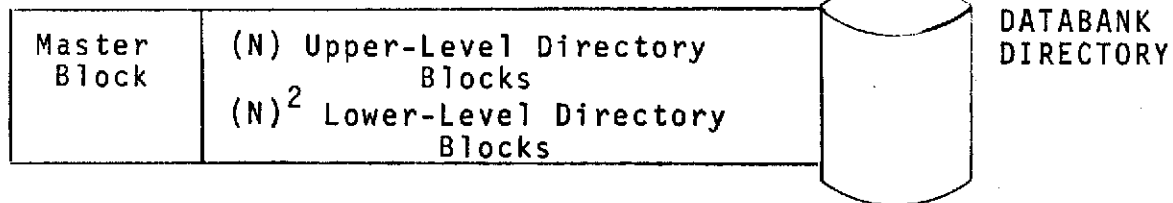
N = Directory Blocking Factor = 20 in Release Version

Record Size = FIXED = (N) x (Directory Entry Size) + 12

$$= (20) \times (76) + 12 = 1,532 \text{ Bytes in Release Version}$$

Record #1

Databank Capacity Record	(M) Databank Name Reference Records	$(N)^3$ - (M+1) Individual Member Records	DATABANK FILE
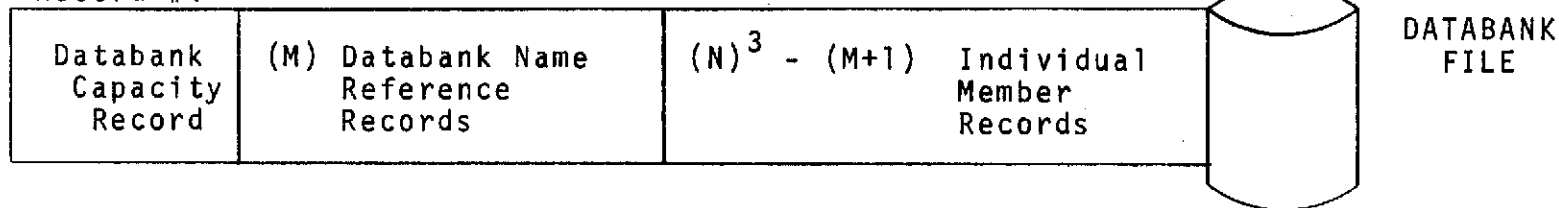
M = Total # Allowed Individual Databanks = 30 in Release Version

Record Size = FIXED = 172 Bytes

Figure B-8. PICTORIAL LAYOUT OF DATABANK FILE AND DIRECTORY

(N) LOWER-LEVEL BLOCKS   (N) LOWER-LEVEL BLOCKS   (N) LOWER-LEVEL BLOCKS

Record #1   Rec #2

Master Block

First Upper Block = UB#1

Second Upper Block = UB#2

DATABANK DIRECTORY

Records #
(UB#1) + 1 to
(UB#1) + (N),
Inclusive

Record # UB#2 = UB#1 + N + 1

Records # (UB#2) + 1 to
(UB#2) + (N),
Inclusive

(N) = Directory Blocking Factor
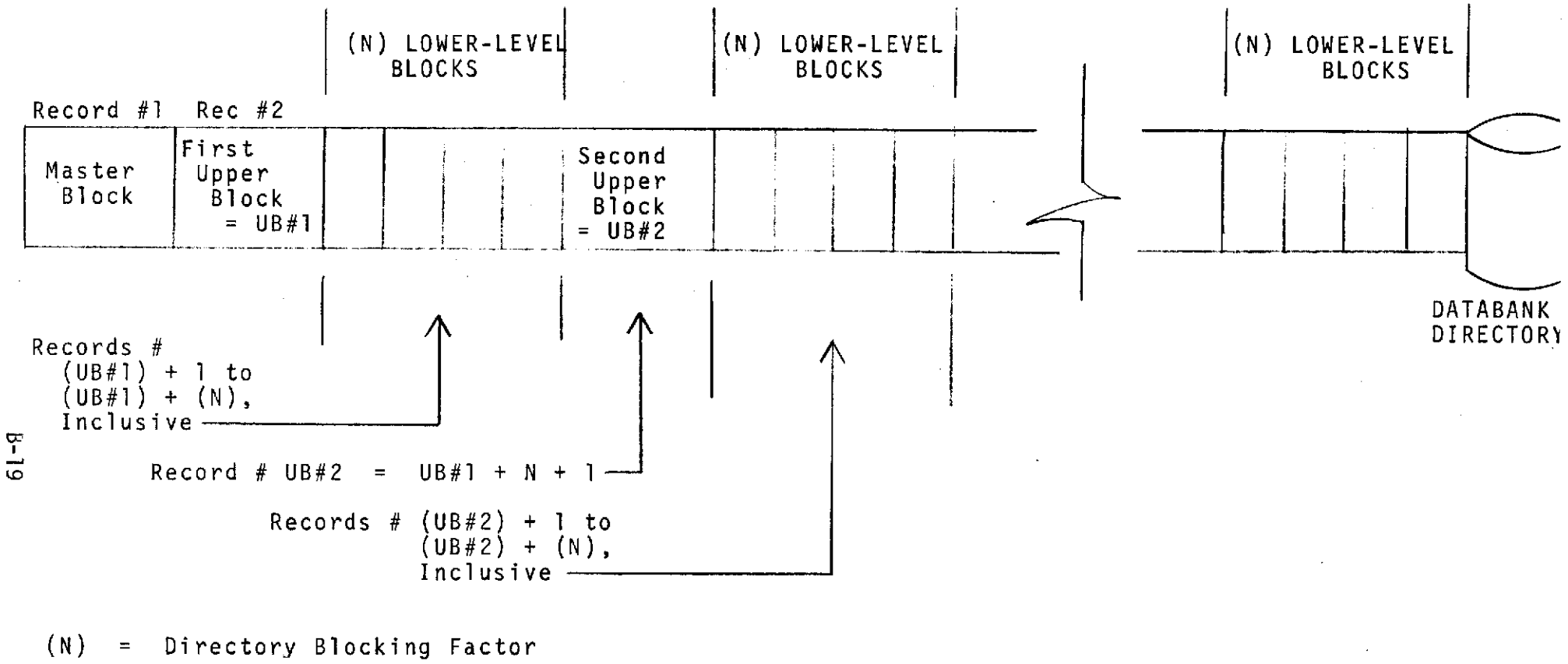
Figure B-9.   <u>DATABANK DIRECTORY: BLOCK NUMBERING ALGORITHM</u>

[MACRO]

Header	MACRO Name	Total # Formal Parameters

Header	Skeleton Card Image (80 Chars)

...(as many Skeleton Card Records as required)...

Header	Skeleton Card Image (80 Chars)

[SUBROUTINE]

Header	GOAL Subroutine Name	FORTRAN Subroutine Name

[FUNCTION-DESIGNATOR]

Header	FD Name	Print Expansion	FD Type	FD Address

[DATABANK REFERENCE RECORD]

Header	Databank Reference Number	Databank Name

[DATABANK CAPACITY RECORD]

Header

Figure B-10.   DATABANK FILE: FUNCTIONAL RECORD CONTENTS

DATABANK CAPACITY RECORD

	RAVAIL	FQTOT	FQPTR	DBMAX	DBCUR	DBF	
Record # 1	Total # Available Records	Total # Records Still Unused	Pointer to First Record on Free Queue	Maximum Number Allowed Databanks	Total # Databanks Currently In Use	Directory Blocking Factor	Unused

DATABANK REFERENCE RECORD

Record #	Forward Chain	Type = 1 (Unused) Type = 2 (In Use)	Maximum Number Allowed Databanks	Pointer to Last Databank Reference Record	Databank #	Databank Name	Revision Label (or blank)	Unused

B-21

HEADER OF ALL OTHER RECORD TYPES - EXCEPT MACRO SKELETON RECORD

Record #	Forward Chain	Type	Total # Records on This Chain	Pointer to Last Record, This Chain	Databank #	Record Name	Print Expansion	Data Portion of Record

MACRO SKELETON RECORD

Record #	Forward Chain	Type = 5	Macro Skeleton Card Image

Figure B-11.   GENERAL CONTENTS OF THE HEADER PORTION OF DATABANK RECORDS

DATA BANK FILE
GENERAL CONTENTS OF RECORDS

Type Field	Record Description	Header	General Contents Data Portion of Record
1	Data Bank Reference (Unused)	Hybrid	Unused
2	Data Bank Reference (In Use)	Hybrid	Unused
3	Function Designator	Standard	FD Type FD Address or Subroutine Name
4	Macro Header	Standard	Total Number of Formal Parameters in Macro
5	Macro Skeleton	Hybrid	Macro Skeleton Card Image
6	Subroutine Name	Standard	Equivalent FORTRAN Subroutine Name

NOTES: (1) Explicit record contents by field are to be found described in APPENDIX E

Figure B-12.

APPENDIX C

DASD SIZE ESTIMATES FOR GOAL DATA BANK IMPLEMENTATION

C.   INTRODUCTION

As indicated in Appendix B, the physical storage space required by the data bank on external DASD is directly proportional to the blocking factor of the data bank directory file. Table C-1 of this Appendix illustrates the total DASD space requirements for both the data bank and the directory files in terms of directory blocking factor (N). Table C-1 also illustrates the implications of using a 2-level versus a 3-level directory structure. The relatively ineffective use of DASD storage in the case of a 2-level system was the deciding factor that led to the use of a 3-level directory in the original GOAL data bank implementation.

Table C-1 is intended for use as follows:

1.   Determine the total number of named entries to be supported in the GOAL data bank.

2.   Find an equal or higher value entry in the "TOTAL ENTRIES" column of the 3-level directory section of the table.

3.   From the entry selected in Step 2, the blocking factor (N) can be determined. The total amount of IBM 2314 Disk unit space (in physical tracks) required to support a data bank of the chosen size is indicated on the same line as the blocking factor.

Use of Table C-1 is demonstrated in the following example:

o   A data bank is required to support 30,000 named entries.

o   The value of closest proximity to 30,000 in the TOTAL ENTRIES column of the 3-level directory section is 32,768. This corresponds to a blocking factor of 32.

o   A data bank and a directory file to support 30,000 (the chosen system will actually accommodate up to 32,768) named entries will require a total of 1,790 disk tracks.

For the initial GOAL data bank implementation, a blocking factor of 20 was selected in sizing the data bank. This will allow support of a total of 8,000 named entries and require a total of 414 disk tracks.

One IBM 2314 disk pack can provide 4,000 data tracks of DASD storage. Table C-1 indicates that data banks selected with blocking factors in excess of 42 will require that more than one physical disk pack be used.

Table C-1. (1 of 2)

** *ALL SIZES IN BYTES*
** *(N) = NUMBER OF DIRECTORY ENTRIES PER DIRECTORY BLOCK*

172: *DATA BANK PHYSICAL RECORD SIZE*
26: *TOTAL DATA BANK RECORDS PER TRACK*
76: *DIRECTORY ENTRY SIZE*

************* 2-LEVEL ************** ************* 3-LEVEL *************

N	TOTAL ENTRIES	DATA TRACKS	DIRECTORY TRACKS	TOTAL TRACKS	TOTAL ENTRIES	DATA TRACKS	DIRECTORY TRACKS	TOTAL TRACKS
10	100	4	2	6	1000	39	14	53
11	121	5	2	7	1331	52	19	71
12	144	6	3	9	1728	67	27	94
13	169	7	3	10	2197	85	31	116
14	196	8	3	11	2744	106	36	142
15	225	9	4	13	3375	130	49	179
16	256	10	4	14	4096	158	55	213
17	289	12	4	16	4913	189	62	251
18	324	13	5	18	5832	225	86	311
19	361	14	5	19	6859	264	96	360
20	400	16	6	22	8000	308	106	414
21	441	17	6	23	9261	357	116	473
22	484	19	6	25	10648	410	127	537
23	529	21	8	29	12167	468	185	653
24	576	23	9	32	13824	532	201	733
25	625	25	9	34	15625	601	217	818
26	676	26	9	35	17576	676	235	911
27	729	29	10	39	19683	758	253	1011
28	784	31	10	41	21952	845	271	1116
29	841	33	10	43	24389	939	291	1230
30	900	35	11	46	27000	1039	311	1350
31	961	37	16	53	29791	1146	497	1643
32	1024	40	17	57	32768	1261	529	1790
33	1089	42	17	59	35937	1383	562	1945
34	1156	45	18	63	39304	1512	596	2108
35	1225	48	18	66	42875	1650	631	2281
36	1296	50	19	69	46656	1795	667	2462
37	1369	53	19	72	50653	1949	704	2653
38	1444	56	20	76	54872	2111	742	2853
39	1521	59	20	79	59319	2282	781	3063

Table C-1. (2 of 2)

** *ALL SIZES IN BYTES*
** *(N) = NUMBER OF DIRECTORY ENTRIES PER DIRECTORY BLOCK*

172: *DATA BANK PHYSICAL RECORD SIZE*
 26: *TOTAL DATA BANK RECORDS PER TRACK*
 76: *DIRECTORY ENTRY SIZE*

| | ************ 2-LEVEL ************* | | | | ************ 3-LEVEL ************* | | | |
N	TOTAL ENTRIES	DATA TRACKS	DIRECTORY TRACKS	TOTAL TRACKS	TOTAL ENTRIES	DATA TRACKS	DIRECTORY TRACKS	TOTAL TRACKS
40	1600	62	21	83	64000	2462	821	3283
41	1681	65	21	86	68921	2651	862	3513
42	1764	68	22	90	74088	2850	904	3754
43	1849	72	22	94	79507	3058	947	4005
44	1936	75	23	98	85184	3277	991	4268
45	2025	78	23	101	91125	3505	1036	4541
46	2116	82	24	106	97336	3744	1082	4826
47	2209	85	48	133	103823	3994	2257	6251
48	2304	89	49	138	110592	4254	2353	6607
49	2401	93	50	143	117649	4525	2451	6976
50	2500	97	51	148	125000	4808	2551	7359

C.1   MODIFICATION OF GOAL DATA BANK MAINTENANCE PROGRAMS

The GOAL data bank maintenance programs were generated using FORTRAN as
the implementation language. Sizing modifications of data bank capacity
by altering the directory blocking factor are effected by changes to
FORTRAN declaration type statements in the maintenance program modules.
Blocking factor modification must then be extended to the Operating System
/360 Job Control Language (JCL) statements that govern the physical size
allocation of the data bank and data bank directory files on disk.

Table C-2 is a list of the FORTRAN declaration statements that will require
modification due to blocking factor change. Variables indicated with an
"N" prefix, e.g., N, N1, N2, etc., will be replaced with the appropriate
numeric constants from Table C-4. In all data bank maintenance program
modules where the declarations appear, they must be so modified and the
programs recompiled.

Table C-3 is a list of the Operating System/360 JCL statements effected by
blocking factor change.

Table C-4 is a tabular list of constants that are to replace the "N" prefix
variables of Tables C-2 and C-3. Substitution constants are provided for
blocking factors ranging from 10 to 50. For the relation of blocking factor
versus total number of supported data bank entries, see Table C-1.

C.2   FORTRAN DECLARATION STATEMENTS AFFECTED BY DIRECTORY BLOCKING
       FACTOR CHANGES

The following FORTRAN declaration statements are directly affected by block-
ing factor modifications within the GOAL data bank directory file. Variables
that appear below with an "N" prefix, e.g., N, N2, etc., are to be replaced
with appropriate values from Table C-4. In any data bank maintenance program
module in which these declarations appear, they must be modified, replaced,
and the programs recompiled when data bank size changes occur.

```
            GIVEN:    N  = Directory Blocking Factor
            INTEGER       DBLOCK ( N1 )
            INTEGER       ENT ( 2, N )
            INTEGER*2     SK ( 34, N )
            EQUIVALENCE   ( ENT(1,1), DBLOCK(4) ),
                          ( SK(1,1), DBLOCK( N2 ) )
            DEFINE FILE   10 ( N3, 172, L, AVDB )
            DEFINE FILE   11 ( N4, N5, L, AVDIR )
            DEFINE FILE   13 ( N4, N5, L, AVUTL )
            DEFINE FILE   18 ( N3, 172, L, AVDB )
            DEFINE FILE   19 ( N4, N5, L, AVDIR )
```

Table C-2.

C.3   OPERATING SYSTEM/360 JOB CONTROL LANGUAGE STATEMENTS FOR GOAL
               DATA BANK CREATION

The following JCL statements are directly effected by changes to the GOAL
data bank directory file blocking factor. Names that appear below with an
"N" prefix, e.g., N, N1, N2, etc., are to be replaced with appropriate
constants from Table C-4. The JCL statements are presented in the format
usually adopted when creating the data bank and the data bank directory
data sets from scratch. It is to be noted that the IBM 2314 disk pack
supports 4,000 physical data tracks. A blocking factor (N) of value 42
(as presented in Table C-1) requires 3,724 tracks total. For blocking
factors in excess of 42, the JCL statements presented below must reflect
allocation of the data bank and the data bank directory data sets to two
(2) or more disk volumes.

Table C-3.

```
//FT10F001    DD   DSN=GOAL.DATAB,
//                 UNIT=SYSDA,BOL=SER=XXXXX,
//                 SPACE=(172,(N3),,CONTIG),
//                 DCB=(RECFM=F,LRECL=172,BLKSIZE=172),
//                 DISP=(NEW,CATLG,DELETE)


//FT11F001    DD   DSN=GOAL.DATAD,
//                 UNIT=SYSDA,VOL=SER=XXXXX,
//                 SPACE=(N5,(N4),,CONTIG),
//                 DCB=(RECFM=F,LRECL=N5,BLKSIZE=N5),
//                 DISP=(NEW,CATLG,DELETE)


//FT18F001    DD   DSN=GOAL.DATAB1,
//                 UNIT=SYSDA,VOL=SER=XXXXX,
//                 SPACE=(172,(N3),,CONTIG),
//                 DCB=(RECFM=F,LRECL=172,BLKSIZE=172),
//                 DISP=(NEW,CATLG,DELETE)


//FT19F001    DD   DSN=GOAL.DATAD1,
//                 UNIT=SYSDA,VOL=SER=XXXXX,
//                 SPACE=(N5,(N4),,CONTIG),
//                 DCB=(RECFM=F,LRECL=N5,BLKSIZE=N5),
//                 DISP=(NEW,CATLG,DELETE)
```

C.4   BLOCKING FACTOR CONSTANTS TO BE APPLIED IN THE DECLARATIVES OF
       TABLE C-2 AND THE JOB CONTROL LANGUAGE STATEMENTS OF TABLE C-3

In the table presented below, (N) is the data bank directory blocking factor.
The table below covers the same range of blocking factor (range of 10 to 50)
as that of Table C-1.

Table C-4.

N	$N1$	$N2$	$N3$	$N4$	$N5$
10	193	24	1000	111	772
11	212	26	1331	133	848
12	231	28	1728	157	924
13	250	30	2197	183	1000
14	269	32	2744	211	1076
15	288	34	3375	241	1152
16	307	36	4096	273	1228
17	326	38	4913	307	1304
18	345	40	5832	343	1380
19	364	42	6859	381	1456
20	383	44	8000	421	1532
21	402	46	9261	463	1608
22	421	48	10648	507	1684
23	440	50	12167	553	1760
24	459	52	13824	601	1836
25	478	54	15625	651	1912
26	497	56	17576	703	1988
27	516	58	19683	757	2064
28	535	60	21952	813	2140
29	554	62	24389	871	2216
30	573	64	27000	931	2292
31	592	66	29791	993	2368
32	611	68	32768	1057	2444
33	630	70	35937	1123	2520
34	649	72	39304	1191	2596
35	668	74	42875	1261	2672
36	687	76	46656	1333	2748
37	706	78	50653	1407	2824
38	725	80	54872	1483	2900
39	744	82	59319	1561	2976
40	763	84	64000	1641	3052
41	782	86	68921	1723	3128
42	801	88	74088	1807	3204
43	820	90	79507	1893	3280
44	839	92	85184	1981	3356
45	858	94	91125	2071	3432
46	877	96	97336	2163	3508
47	896	98	103823	2257	3584
48	915	100	110592	2353	3660
49	934	102	117649	2451	3736
50	953	104	125000	2551	3812

APPENDIX D

PROGRAM MODULE DESCRIPTIONS

D.   INTRODUCTION

Included in this Appendix are short verbal descriptions of all of the
program modules required to generate and support the GOAL data bank
system. In this Appendix:

o   Figure D-1 indicates the functional flow during data bank
     and data bank directory initialization.

o   Figure D-2 is the functional diagram of the data bank
     maintenance process.

o   Figure D-3 represents the elements involved in the data bank
     initialization process. Also indicated on this diagram is
     the operation of the data bank print utility program. The
     operation and use of these two programs is completely inde-
     pendent of one another - they have been included on the same
     diagrams for convenience.

o   Diagrams D-4, D-5, and D-6 depict the three phases of the data
     bank maintenance process.

o   Following the above diagrams are descriptions (one module per
     page) of the programs and subroutines required to support the
     data bank. The program modules and their system names are as
     follows:

Data Bank Print Utility

          Mainline   =   SUPERD
          Subroutines =   SLEW

Data Bank Initialization Program

          Mainline   =   DBI
          Subroutines =   None

Data Bank Maintenance Program

          Mainline   =   MAINT
          Subroutines =   BEGDB   NAMESR   RCRETN
                                      BUST   SPECFY   VNAME
                                      DELDB   DBEND   COMPAR
                                      DELETE   DBSEEK   SPACE
                                      MACRO   FIND   SVSAVE

Data Bank Pre-Processor

This module is actually the GOAL compiler; action as a data bank pre-processor is achieved by providing the proper syntax tables. Documentation covering the GOAL compiler module is in separate documentation. The syntax tables required for data bank maintenance usage are included in this document as Appendix I.

DATABANK
INITILIZATION
CONTROL CARDS

Type(1)
Card

Type(2)
Card

DATABANK
INITILIZE
PROGRAM

Directory Blocking
Factor from Constant
Compiled into the
Initilization Program

* Writes Dummy
  Blocks

* Writes Zeroed
  Block Headers

* Writes Capacity
  Record

* Writes Dummy Databank
  Name Reference
  Records

* Forms Remaining Records
  on to Free Queue

DIREC-
TORY

DATA
BANK

DATABANK
CAPACITY RECORD

Record
Number
= 1

Total Number
of Records
Available
in Databank
File

Total Number
Records Now
Available on
Free Queue

Pointer
(Record #) to
First Record
on Free Queue

Maximum
Number of
Allowed
Databanks

Total Number
of Databanks
Currently in
Use

Directory
Blocking
Factor

DATABANK INITIALIZATION

Figure D-1

D-3

Figure D-2.   DATABANK MAINTENANCE PROCESS

D-4

N

Initialization
Control
Parameters

//FT05F001

Data Bank
Directory
Work Data Set

Data Bank
Initialization
Program

(DBI)

//FT13F001

L

Ø

//FT06F001

Maintenance
Listing

//FT11F001

//FT10F001

Data Bank
Directory

D

E

Data Bank
File

//FT11F001

//FT10F001

Data Bank Print
Utility Program

(SUPERD)

Letters in circles are
Data Set Reference
Numbers; see
APPENDIX E for
record formats

//FT06F001

Printer Dump
All
Data Banks

P

DATA BANK
INITIALIZATION AND
PRINT UTILITY
PROGRAMS

NOTE - PROGRAMS (DBI) AND (SUPERD) MAY
BE RUN INDEPENDENTLY OF ONE
ANOTHER

Figure D-3.

PHASE-1 DATA BANK
MAINTENANCE
PROCESS
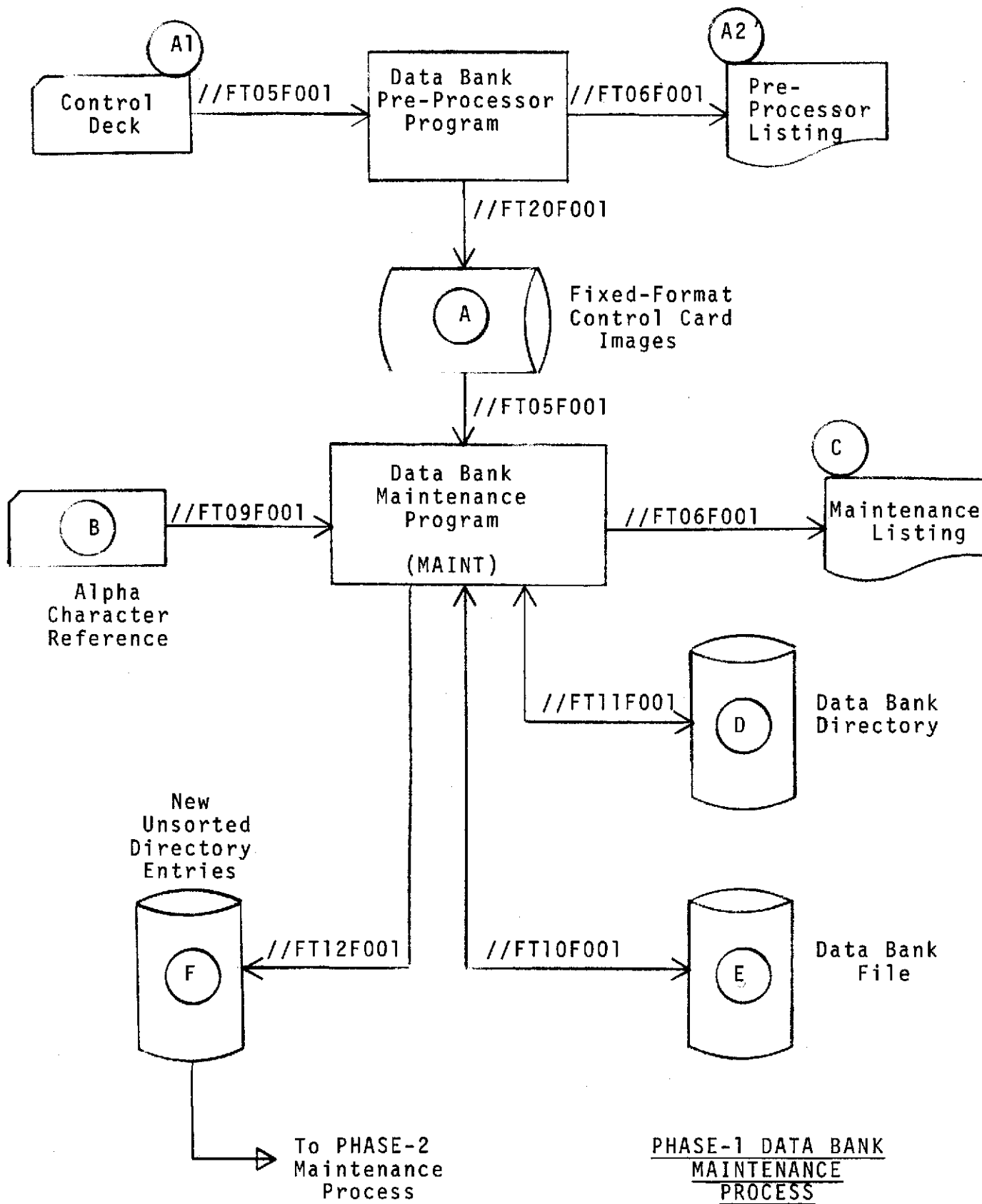
Letters in circles are Data Set Reference
Numbers; see APPENDIX E for record
formats

Figure D-4
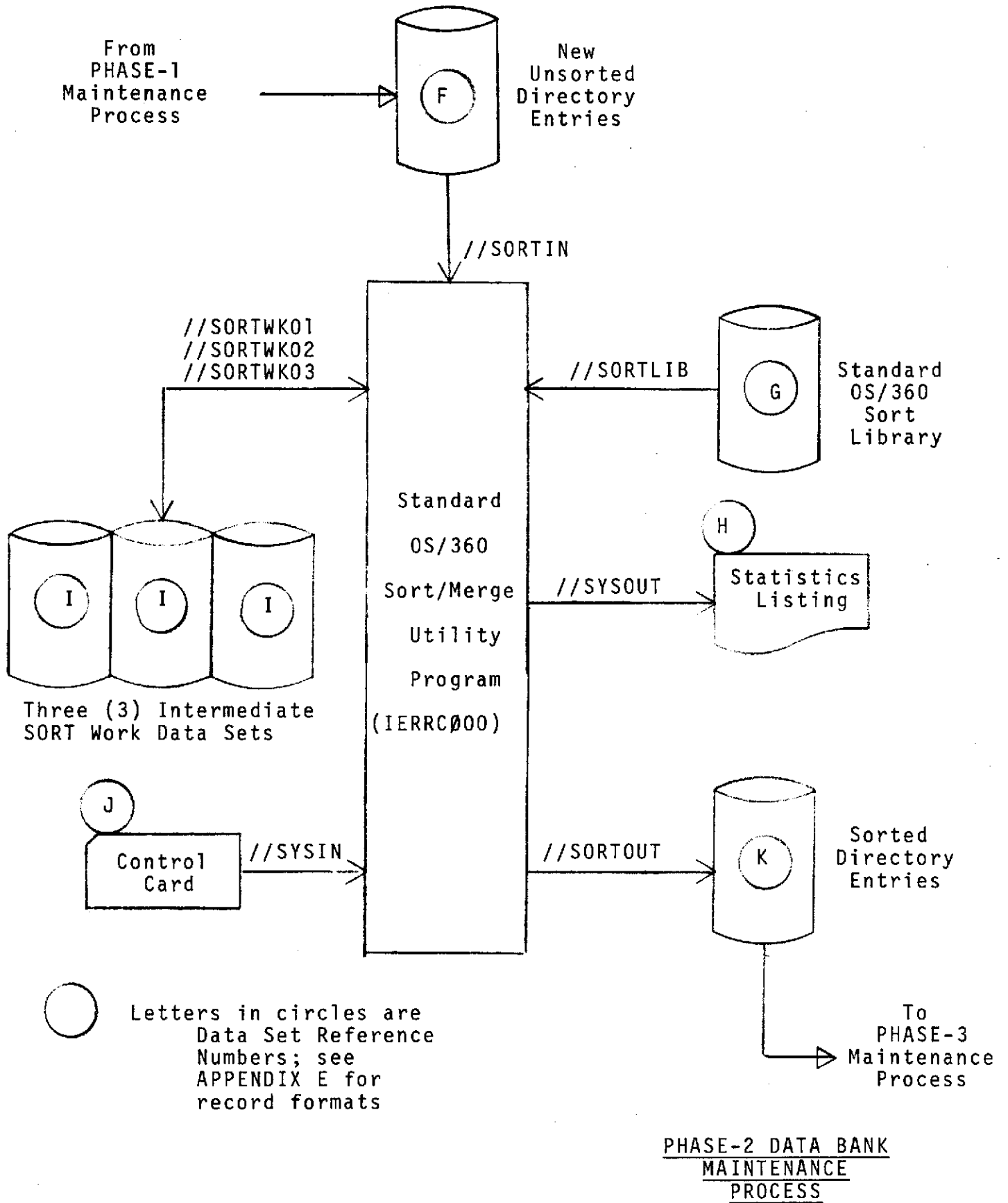
From
PHASE-1
Maintenance
Process
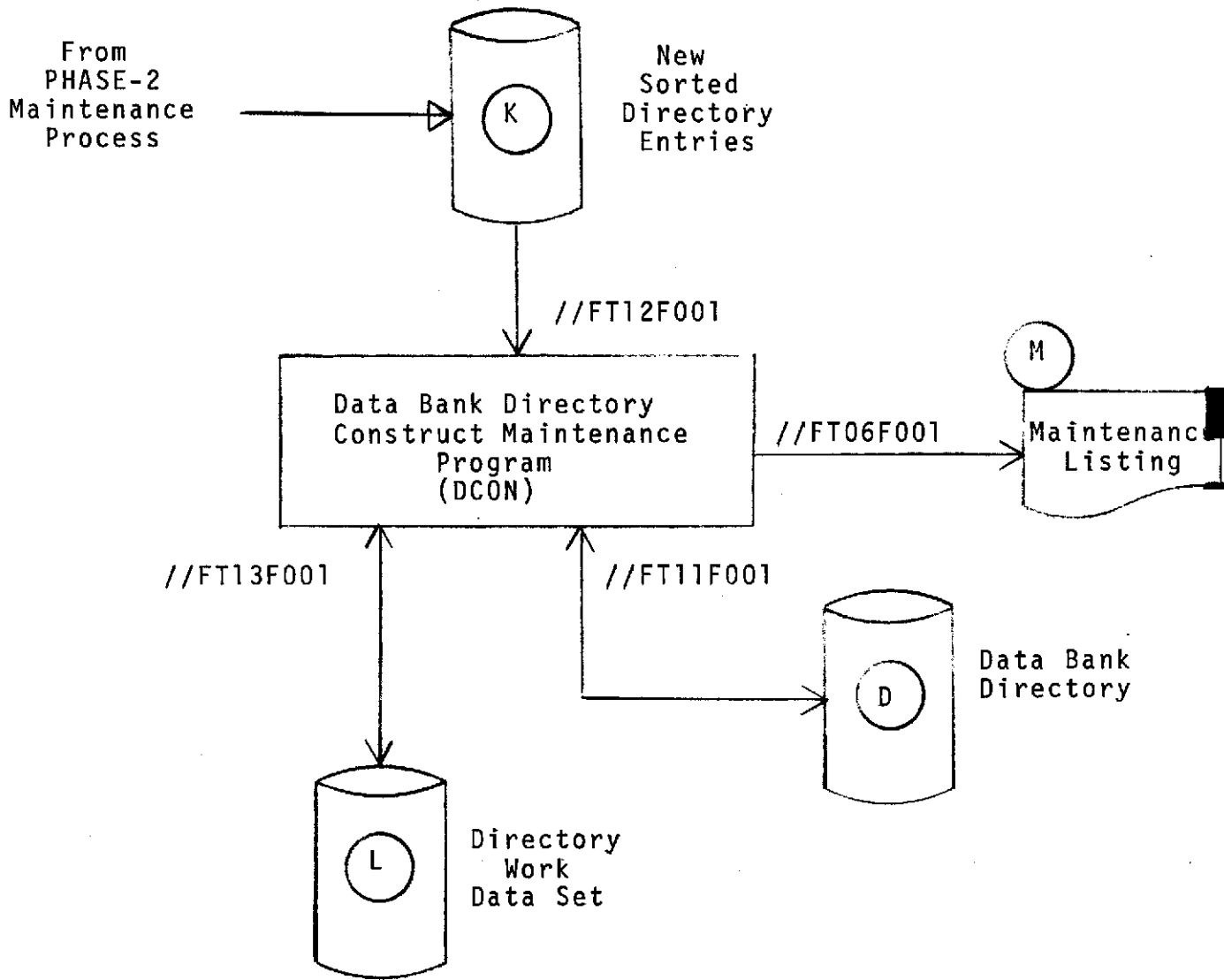
New
Unsorted
Directory
Entries

F

//SORTIN

//SORTWKO1
//SORTWKO2
//SORTWKO3

//SORTLIB

G

Standard
OS/360
Sort
Library

I   I   I

H

Standard

OS/360

Sort/Merge

//SYSOUT

Statistics
Listing

Three (3) Intermediate
SORT Work Data Sets

Utility

Program

(IERRCØOO)

J

Control
Card

//SYSIN

//SORTOUT

K

Sorted
Directory
Entries

Letters in circles are
Data Set Reference
Numbers; see
APPENDIX E for
record formats

To
PHASE-3
Maintenance
Process

PHASE-2 DATA BANK
MAINTENANCE
PROCESS

Figure D-5.

Figure D-6.

NAME   -   SUPERD

TYPE   -   FORTRAN Mainline

FUNCTION   -   Provides printer dump of the current data bank
                    contents

CALLED BY   -   Not Applicable

SUBROUTINES
    CALLED   -   SLEW

DESCRIPTION   -   This module prints the contents of all active
                    data banks. Using the directory, all members of
                    each data bank are listed in alphabetic order by
                    member (record) name.

GOAL DATA BANK ROUTINE

NAME   -   SLEW

TYPE   -   FORTRAN Subroutine

FUNCTION   -   Provides paging and title strips for SUPER data bank
                    print utility program

CALLED BY   -   SUPERD

SUBROUTINES
    CALLED   -   None

DESCRIPTION   -   Provides paging and title strip printing for the
                    SUPERD print utility program.

GOAL DATA BANK ROUTINE

NAME   -   DBI

TYPE   -   FORTRAN Mainline

FUNCTION   -   Data bank file and data bank directory initilization
                (record pre-formatting)

CALLED BY   -   Not Applicable

SUBROUTINES
    CALLED -   None

DESCRIPTION -   Under control-card control, this module pre-formats
                the records of the data bank file and the data bank
                directory by:

                    o   composing and writing the data bank capacity
                        record;

                    o   initializing and writing all data bank reference
                        records; these records also require chaining
                        together;

                    o   filling the remainder of the data bank with
                        a chain of dummy records;

                    o   writing dummy blocks into the data bank
                        directory and directory utility data set.

GOAL DATA BANK ROUTINE

NAME   -   MAINT

TYPE   -   FORTRAN Mainline Program

FUNCTION   -   Mainline root program for all phase 1 maintenance
                    actions; reads control card input and invokes the
                    proper subroutine for further processing

CALLED BY   -   Not Applicable

SUBROUTINES
    CALLED -   DELDB, DBEND, SPECFY, SVSAVE, MACRO, DELETE,
                    BEGDB, NAMESR, BUST

DESCRIPTION -   This module serves to process input cards, determine
                    their nature, and invoke the proper subroutine for
                    completion of the processing. Each subroutine, after
                    accomplishing its required action, returns control to
                    MAINT for further processing of the input stream.

NAME   -   BEGDB

TYPE   -   FORTRAN Subroutine

FUNCTION   -   Process a BEGIN DATABANK control card; open a
                 Data Bank for maintenance

CALLED BY   -   MAINT

SUBROUTINES
   CALLED   -   VNAME, DBSEEK, BUST

DESCRIPTION   - This subroutine is utilized to process a BEGIN
                  DATABANK control card. The named Data Bank is sought
                  within the current Data Bank data set. If it already
                  exists, the Data Bank open flag (DBN) is set with the
                  Data Bank reference number. If the Data Bank is new,
                  the DBN flag is set in the same manner; the new Data
                  Bank Reference Record is composed and written into the
                  Data Bank before return to the caller.

NAME   -   BUST

TYPE   -   FORTRAN Subroutine

FUNCTION   -   Output to the printer a standard error number

                     diagnostic message.

CALLED BY   -   MAINT, BEGDB, DELDB, DELETE, MACRO, NAMESR, SPECFY,

                     DBEND, DBSEEK, FIND, VNAME, SPACE

SUBROUTINES
    CALLED -   RCRETN

DESCRIPTION -   An error message number provided as a parameter by the

                caller module is printed in a standard error message

                format. A non-zero Return Code is established in the

                OS/360 via a call to RCRETN from this subroutine.

NAME   -   DELDB

TYPE   -   FORTRAN Subroutine

FUNCTION   -   Process a DELETE DATABANK control card

CALLED BY -   MAINT

SUBROUTINES
    CALLED -   VNAME, DBSEEK, BUST

DESCRIPTION -   The reference number of the required Data Bank is

                 located via the DBSEEK subroutine. Using this number

                 and the Data Bank Directory, all members of the named

                 Data Bank are returned to the record Free Queue in the

                 Data Bank data set. The existing Directory references

                 to the deleted members are marked for deletion.

NAME   -   DELETE

TYPE   -   FORTRAN Subroutine

FUNCTION   -   Deletes a named member from the current open

                Data Bank; processes a DELETE control card

CALLED BY -   MAINT

SUBROUTINES
      CALLED   - VNAME, FIND, BUST

DESCRIPTION - The Data Bank Directory is used to locate the

                control card named member in the Data Bank currently

                opened. Once the required record is located, it is

                returned to the Data Bank Free Queue. The applicable

                Directory entry is marked for deletion.

GOAL DATA BANK ROUTINE

NAME   -   MACRO

TYPE   -   FORTRAN Subroutine

FUNCTION   -   Processes MACRO control cards; enters the new
                 MACRO into the Data Bank currently open

CALLED BY   -   MAINT

SUBROUTINES
    CALLED   -   VNAME, SPACE, BUST

DESCRIPTION   - Processing of all MACRO input cards is accomplished
                 in this subroutine. Space is retrieved within the
                 currently open Data Bank, and the new MACRO records
                 composed and inserted.

NAME   -   NAMESR

TYPE   -   FORTRAN Subroutine

FUNCTION   -   Provides processing for Subroutine Name Records

CALLED BY   -   MAINT

SUBROUTINES
    CALLED   -   VNAME, SPACE, BUST

DESCRIPTION   - Subroutine name control cards are parsed and processed

                 via this subroutine. Subroutine name record construction

                 is accomplished and the new record written into the

                 currently open Data Bank.

GOAL DATA BANK ROUTINE

NAME   -   SPECFY

TYPE   -   FORTRAN Subroutine

FUNCTION   -   Used to process SPECIFY control cards; creates new
                 Function Designator Records

CALLED BY   -   MAINT

SUBROUTINES
    CALLED -   SPACE, BUST

DESCRIPTION -   New Function Designator records are created by this
                 subroutine. All parsing of the SPECIFY card contents
                 is accomplished within this subroutine.

GOAL DATA BANK ROUTINE

NAME   -   DBEND

TYPE   -   FORTRAN Subroutine

FUNCTION   -   Closes an open Data Bank; this subroutine processes
                 an END DATABANK control card

CALLED BY -   MAINT

SUBROUTINES
      CALLED - BUST

DESCRIPTION -   This subroutine is invoked to process an END DATABANK
                 control card. The data bank open flag (DBN) is set
                 to the zero (off) state. Only DELETE DATABANK control
                 cards will be recognized by the MAINT mainline until
                 another Data Bank is opened (via a BEGIN DATABANK
                 control card).

GOAL DATA BANK ROUTINE

NAME   -   DBSEEK

TYPE   -   FORTRAN Subroutine

FUNCTION   -   Determines if a named Data Bank already exists

CALLED BY   -   BEGDB, DELDB

SUBROUTINES
    CALLED   -   BUST

DESCRIPTION   - Using a Data Bank name (name plus revision label, if
                 appropriate), this subroutine returns one of three
                 pieces of information to the caller:
                    (1) If the Data Bank already exists, the caller is
                        informed of the Data Bank reference number;
                    (2) If the Data Bank does not yet exist, the caller
                        is informed of a Data Bank reference record
                        number that is available for establishing a new
                        Data Bank;
                    (3) If the Data Bank does not exist and all usable
                        Data Bank reference records have been utilized,
                        an appropriate error indication is given.

NAME   -   FIND

TYPE   -   FORTRAN Subroutine

FUNCTION   -   Locate a specific member of a given Data Bank

CALLED BY   -   DELETE

SUBROUTINES
    CALLED   -   COMPAR, BUST

DESCRIPTION   -   Using the Data Bank Directory, this subroutine locates

                  a required member. The name of the member is supplied

                  as a calling parameter;   the required Data Bank reference

                  number is provided as the first four bytes of the name.

                  Upon finding the required member, the sought record is

                  brought into memory and the caller informed of its

                  relative record number within the Data Bank data set.

NAME   -   RCRETN

TYPE   -   Assembler Language Subroutine

FUNCTION   -   The function of this subroutine is to establish the

                   OS/360 Return Code upon termination of the MAINT phase

                   1 maintenance module. Execution (go/nogo) of the

                   programs in subsequent job steps is controlled via this

                   Return Code.

CALLED BY   -   BUST

SUBROUTINES
    CALLED   -   None

DESCRIPTION   -   Non-error execution of the MAINT module results in a

                   zero Return Code indication upon MAINT termination.

                   Subsequent phases of the maintenance process can use

                   this code as an indication that normal execution can

                   continue. Should an error occur during MAINT execution

                   that would render execution of the subsequent maintenance

                   steps inappropriate, a non-zero Return Code is

                   established by call to the RCRETN subroutine from the

                   error diagnostic subroutine BUST.

GOAL DATA BANK ROUTINE

NAME   -   VNAME

TYPE   -   FORTRAN Subroutine

FUNCTION -   Validates and isolates a name bounded by parenthesis

CALLED BY -   BEGDB, DELDB, DELETE, MACRO, NAMESR

SUBROUTINES
     CALLED -   BUST

DESCRIPTION -   This subroutine scans an indicated portion of an input
                 card and attempts to isolate a standard name bounded by
                 parenthesis. The name can be 1 - 32 characters in
                 length. The first character must be a letter; subsequent
                 characters can be letters and/or digits. An appropriate
                 error message is issued if an error is detected in
                 either the name contents or in the delimeters.

NAME   -   COMPAR

TYPE   -   Assembler Language Subroutine

FUNCTION   -   Perform a logical (non-arithmetic) compare of

                    two EBCDIC names

CALLED BY   -   FIND

SUBROUTINES
    CALLED   -   None

DESCRIPTION -   This subroutine is used as a FORTRAN function

                    subprogram. It is designed to be used in an

                    arithmetic FORTRAN statement; the value returned to

                    the caller is either (-1), (0), or (+1). Two named

                    parameters are compared logically with the following

                    implications:

                         Parameter
                         Relation   Return

                          A < B   -1

                          A = B   0

                          A > B   +1

NAME   -   SPACE

TYPE   -   FORTRAN Subroutine

FUNCTION   -   Finds a place to write a new record into the
                 Data Bank data set

CALLED BY   -   MACRO, NAMESR, SPECFY

SUBROUTINES
    CALLED   - BUST

DESCRIPTION   - This subroutine locates the next available singular
                 record space in the Data Bank. The Data Bank Free Queue
                 total and pointer fields are updated and the Data Bank
                 Capacity Record re-written with the adjusted totals.
                 The record selected for availability has been read
                 into memory and is ready for use when return to the
                 caller is effected.

NAME   -   SVSAVE   (Alternate entry point. to RCRETN subroutine)

TYPE   -   Assembler Language

FUNCTION   -   The memory address of the OS/360 Supervisor is
                 established via this entry to the RCRETN subroutine.
                 This address is required for subsequent correct operation
                 of the RCRETN subroutine.

CALLED BY   -   MAINT

SUBROUTINES
     CALLED   - None

DESCRIPTION   - The address of the OS/360 Supervisor is required for
                 operation of the RCRETN subroutine; this address is
                 used as a RCRETN return point. The address is established
                 as a part of the initilization sequence of the MAINT
                 mainline program.

APPENDIX E

DATA RECORD FORMATS

E.1   INTRODUCTION

This Appendix contains record descriptions for all of the data sets re-
quired for data bank and data bank directory initialization, maintenance,
and usage. Each of the data sets is referred to by a "file reference
letter". The reference letters are those used in the functional flow
illustrations of Appendix E. They are itemized in the table following:

File Reference Letter	See Figure	Data Set Description
A1	D-4	Pre-processor program control card input
A2	D-4	Pre-processor program listing
A	D-4	Fixed-format control card images
B	D-4	Maintenance program alphanumeric character set reference
C	D-4	Phase-1 maintenance program listing
D	D-3 D-4 D-6	Data bank directory
E	D-3 D-4 D-6	Data bank file
F	D-4 D-5	New unsorted directory entries
G	D-5	Standard OS/360 Sort/Merge utility program library (SYS1.SORTLIB)
H	D-5	Sort/merge utility program listing
I	D-5	Sort/merge utility program intermediate work data sets
J	D-5	Sort/merge utility program control card

File Reference Letter	See Figure	Data Set Description
K	D-5 D-6	Sorted new directory entries
L	D-6	Directory work data set
M	D-6	Directory construction program listing
N	D-3	Initialization program control card input
Ø	D-3	Initialization program listing
P	D-3	Printer dump of data bank

Figures E-1 through E-4 inclusive, form a sequential sample of the printer listings that might occur as the result of a data bank maintenance operation.

```
RECORD       SOURCE RECORD

   1     BEGIN DATABANK (SAMPLEDB) REVISION A ;
   2     SPECIFY <PRIMARY GSCU ON SWITCH> AS LOAD TYPE DISCRETE ADDRESS 252 ;
   3     SPECIFY <VENT MOTOR FIELD> AS LOAD TYPE ANALOG ADDRESS 0063 ;
   4     SPECIFY <CDC SET CKT> AS LOAD TYPE CLOCK ADDRESS 0040 ;
   5     SPECIFY <MANIFOLD PRESSURE LOW LAMP> AS SENSOR TYPE DISCRETE ADDRESS 4123 ;
   6     SPECIFY <IU COOLANT PRESS> AS SENSOR TYPE ANALOG ADDRESS 0049 ;
   7     SPECIFY <EST> AS SENSOR TYPE CLOCK ADDRESS 6 ;
   8     SPECIFY <LINE PRINTER> ALSO AS <360/PRINTER> AS SYSTEM TYPE PRINTER ;
   9     SPECIFY <CRT2> AS SYSTEM TYPE CRT ADDRESS 1 ;
  10     SPECIFY <LOG> AS SYSTEM TYPE TAPE ;
  11     SPECIFY <PRINTER BUSY> AS SYSTEM TYPE INTERRUPT 0 ;
  12     SPECIFY <PROCEDURE IN PROGRESS FLAG> AS SYSTEM TYPE FLAG 1 ;
  13     SPECIFY <GOAL SR NAME> AS SUBROUTINE (FORT23) ;
  14     NAME (POWER ON) SUBROUTINE (FORT4) ;
  15     NAME (ABCDEFG) PROGRAM (FORT25) ;
  16     BEGIN MACRO ABC (A),(B),(C),<A> ;
  17        (ABC) = (A) ;
  18        (DEF) = (B) ;
  19        (GHI) = (C) ;
  20        (JKL) = <A> ;
  21     END MACRO ;
  22     END DATABANK ;
  23     FINIS
```

E-3

Figure E-1.   SAMPLE DATA BANK
                PRE-PROCESSOR
                LISTING

Refer to File Reference Letter A2

GOAL COMPILER DIAGNOSTIC SUMMARY

```
TOTAL NUMBER OF SOURCE RECORDS:    23
TOTAL NUMBER OF STATEMENTS:    19
TOTAL NUMBER OF WARNINGS:     0
TOTAL NUMBER OF ERRORS :    0
HIGHEST CONDITION CODE WAS   0
```

```
DATABANK  (SAMPLEDB)                                (A)
SPECIFY   <PRIMARYGSCUONSWITCH>                      LOAD        DISCRETE    0252
SPECIFY   <VENTMOTORFIELD>                           LOAD        ANALOG      0063
SPECIFY   <CDCSETCKT>                                LOAD        CLOCK       0040
SPECIFY   <MANIFOLDPRESSURELOWLAMP>                  SENSOR      DISCRETE    4123
SPECIFY   <IUCOOLANTPRESS>                           SENSOR      ANALOG      0049
SPECIFY   <EST>                                      SENSOR      CLOCK       0006
SPECIFY   <LINEPRINTER>
          <360/PRINTER>                              SYSTEM      PRINTER
SPECIFY   <CRT2>                                     SYSTEM      CRT         0001
SPECIFY   <LOG>                                      SYSTEM      TAPE
SPECIFY   <PRINTERBUSY>                              SYSTEM      INTERRUPT   0000
SPECIFY   <PROCEDUREINPROGRESSFLAG>                  SYSTEM      FLAG        0001
SPECIFY   <GOALSRNAME>                               SUBROUTINE  (FORT23)
NAME      (POWERON)                                  SUBROUTINE  (FORT4)
NAME      (ABCDEFG)                                  SUBROUTINE  (FORT25)
MACRO     (ABC)
          (A)
          (B)
          (C)
          <A>
   (ABC) = (A) ;
   (DEF) = (B) ;
   (GHI) = (C) ;
   (JKL) = <A> ;
END       MACRO
END       DATABANK

*** TOTAL DIRECTORY ENTRIES GENERATED =    15 ***

*** TOTAL DIRECTORY ENTRIES DELETED =    0 ***
```

Figure E-2.   SAMPLE PHASE-1 MAINTENANCE PROGRAM
               FIXED-FORMAT CONTROL-CARD IMAGES

Refer to File Reference Letter A

```
IER036I - B =    90
IER037I - G =   700
IER038I - NMAX =    17460
IER045I - END SORT PH
IER049I - SKIP MERGE PH
IER054I - RCD IN        17,OUT       17
IER052I - EOJ
```

Figure E-3.

SAMPLE SORT/MERGE PRINTER OUTPUT

Refer to File Reference Letter $\underline{H}$

NEW DIRECTORY STATISTICS

```
       20: DBF

       14: WMAX

      239: DRTOT
      224: OLDTOT
        0: DELTOT
       15: SORTOT

        0: ZERO BALANCE
```

Figure E-4.

SAMPLE PHASE-3 MAINTENANCE PROGRAM LISTING

Refer to File Reference Letter $\underline{M}$

DATABANK NAME: SAMPLEDB
    REVISION: A
    REFERENCE #:   11

MEMBER NUMBER	RECORD NUMBER	RECORD TYPE	DESCRIPTION	ADDRESS	MEMBER NAME	PRINT EXPANSION
225	509	MACRO	(4 PARAMETERS)		ABC	
225	510	MACRO	SKELETON	(ABC) = &1 ;		
225	511	MACRO	SKELETON	(DEF) = &2 ;		
225	512	MACRO	SKELETON	(GHI) = &3 ;		
225	513	MACRO	SKELETON	(JKL) = &4 ;		
226	508	SUBROUTINE		FORT25	ABCDEFG	
227	497	<FD>	LOAD CLOCK	40	CDCSETCKT	CDCSETCKT
228	502	<FD>	SYSTEM CRT	1	CRT2	CRT2
229	500	<FD>	SENSOR CLOCK	6	EST	EST
230	506	<FD>	SUBROUTINE	FORT23	GOALSRNAME	GOALSRNAME
231	499	<FD>	SENSOR ANALOG	49	IUCOOLANTPRESS	IUCOOLANTPRESS
232	501	<FD>	SYSTEM PRINTER		LINEPRINTER	360/PRINTER
233	503	<FD>	SYSTEM TAPE		LOG	LOG
234	498	<FD>	SENSOR DISCRETE	4123	MANIFOLDPRESSURELOWLAMP	MANIFOLDPRESSURELOWLAMP
235	507	SUBROUTINE		FORT4	POWERON	
236	495	<FD>	LOAD DISCRETE	252	PRIMARYGSCUONSWITCH	PRIMARYGSCUONSWITCH
237	504	<FD>	SYSTEM INTERRUPT	0	PRINTERBUSY	PRINTERBUSY
238	505	<FD>	SYSTEM FLAG	1	PROCEDUREINPROGRESSFLAG	PROCEDUREINPROGRESSFLAG
239	496	<FD>	LOAD ANALOG	63	VENTMOTORFIELD	VENTMOTORFIELD

Figure E-5.   SAMPLE DATA BANK PRINTER DUMP LISTING

Refer to File Reference Letter P

Record Name   : Control Card Input

File Reference Letter   : A1, A2, A

FORTRAN Symbolic Base Declaration : Not Applicable

Record Format Number   : Not Applicable

These three data sets form the control input for the data bank maintenance
process. They are discussed separately below:

(A1) Pre-Processor Program Control Card Input

These control cards direct the maintenance process. They are totally
free-form in conformity with the syntax diagrams presented in this
document as Appendix I. A sample listing of control card variations
is to be found in Figure E-1.

(A2) Pre-Processor Program Listing

The pre-processor program is actually the GOAL compiler module directed
by a suitable syntax table. The listing that is output is an 80-80 list-
ing of the control cards input, plus any errors that may have been
detected in the control-card input. The pre-processor error messages
are to be found in this document in Appendix H. Figure E-1 provides an
example of the listing.

### (A) Phase-1 Maintenance Program Control-Card Input

This data set forms the input control for the Phase-1 data bank main-
tenance program (MAINT). The card images may be punched and entered
directly to MAINT, or, the images may be artificially generated as a
normal output from the data bank pre-processor program.

The control image consists of a total of six (6) distinct fields in
fixed positions. Not all fields are required for any control card type.
Figure E-6 describes the bounds of the six fields.

A sample of the variations of control images possible is illustrated in
Figure E-7. Usage of the various types is discussed in the following
paragraphs:

o A data bank is opened for maintenance by the appearance of a
   DATABANK control card; the same data bank is closed by an END
   DATABANK card.

o All control variations, with the single exception of a delete
   data bank (DELETEDB) option, are legal only between a DATABANK
   and an END DATABANK pair. The DELETEDB option is legal only
   when a data bank has been closed.

o   All function-designator types except for PRINTER, CRT, TAPE,
    and SUBROUTINE require an address field (see "F5" field, Figure
    E-6). This address field is a mandatory four-digit field - lead-
    ing zero(s) to be supplied as applicable.

o   For the examples of function designators indicated in Figure E-7,
    the print expansion field and the function designator name fields
    are identical. If the print expansion field is to be unique, two
    control images will be required. See page 2 of Figure E-7.

o   The total number of formal parameters allowed on a macro will be
    ten (10). Each parameter may be a GOAL name or function designator.
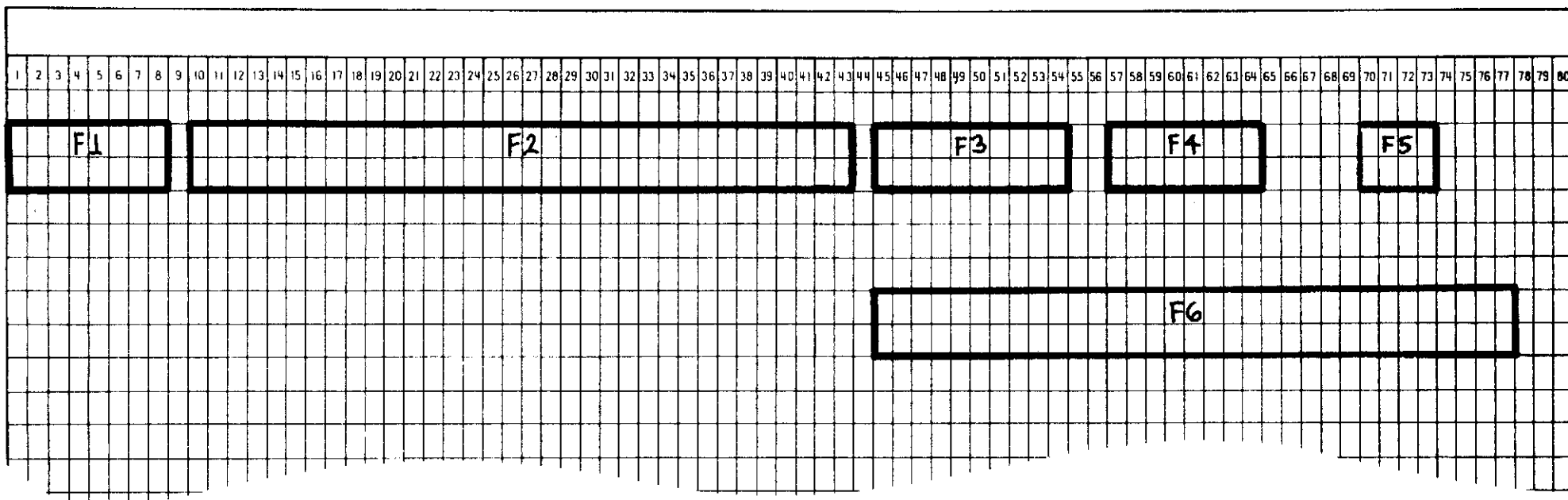
Figure E-6.   FIXED FIELD LAYOUT - DATA SET REFERENCE LETTER "A"

DATABANK (ILLUSTRATIONS)

SPECIFY	<PRIMARY GSCU ON SWITCH>	LOAD	DISCRETE	0252
SPECIFY	<VENT MOTOR FIELD>	LOAD	ANALOG	0063
SPECIFY	<CDC SET CKT>	LOAD	CLOCK	0040
SPECIFY	<MANIFOLD PRESSURE LOW LAMP>	SENSOR	DISCRETE	4123
SPECIFY	<IU COOLANT PRESS>	SENSOR	ANALOG	0049
SPECIFY	<EST>	SENSOR	CLOCK	0006
SPECIFY	<LINEPRINTER>	SYSTEM	PRINTER	
SPECIFY	<CRT2>	SYSTEM	CRT	0013
SPECIFY	<LOG>	SYSTEM	TAPE	
SPECIFY	<EVENT12>	SYSTEM	INTERRUPT	0000
SPECIFY	<REMEMBER	SYSTEM	FLAG	0614
SPECIFY	<GOAL SR NAME>	SUBROUTINE	(FORT23)	
MACRO	(SAMPLEMACRONAME)			
	(PARM1)			

       READ <GMT> AND SAVE AS (PARM1) ;

END   MACRO

END   DATABANK

          Figure E-7.   SAMPLE FIXED-FORMAT CONTROL-CARD IMAGES
          (Page 1 of 2)   (DATA SET REFERENCE NUMBER "A")

```
DELETE    (OLDNAME)

DELETE    <OLD FUNCTION DESIGNATOR>                               (More control-card samples)

DELETEDB  (OLDDATABANK)                          (REVISIONXXX)
```

Note: The print expansion field and the function designator name are identical on
all of the samples presented on page 1 of this figure.

The following example is presented to illustrate the format required to enter
a function designator with unique name and print-expansion formats -   2 cards

```
SPECIFY   <NORMAL FD NAME>        (Note - remainder of first card left blank)

          <FD PRINT EXPANSION NAME - 2ND CARD>   SENSOR        ANALOG        0049
```

Figure E-7.   (Page 2 of 2)   SAMPLE FIXED-FORMAT CONTROL-CARD IMAGES

Refer to File Reference Letter A

Record Name   : Alphabetic Character Reference

File Reference Letter   : B

FORTRAN Symbolic Base Declaration : CHRTAB

Record Format Number   : Not Applicable

This dataset forms the character reference to be used in scanning
control-card records. The dataset consists of a single card image
to be punched as follows:

Card Column	Character	Card Column	Character	Card Column	Character
1	1	21	K	41	<
2	2	22	L	42	(
3	3	23	M	43	)
4	4	24	N	44	' (apostophe)
5	5	25	0	45	+
6	6	26	P	46	-
7	7	27	Q	47	*
8	8	28	R	48	/
9	9	29	S	49	?
10	0 (zero)	30	T	50	#
11	A	31	U	51	$
12	B	32	V	52	¬
13	C	33	W	53	&
14	D	34	X	54		
15	E	35	Y	55	,
16	F	36	Z	56	.
17	G	37	=	57	blank
18	H	38	:	58-80	blank
19	I	39	;		
20	J	40	>		

Record Name   : Databank Maintenance Module Listing

File Reference Letter   : C

FORTRAN Symbolic Base Declaration   : Not Applicable

Record Format Number   : Not Applicable

The output from this module is basically a single-spaced printer listing
of the input card deck (File Reference Letter = A).

If the step terminates due to error, the following message will appear on
the same line as the last input card processed:

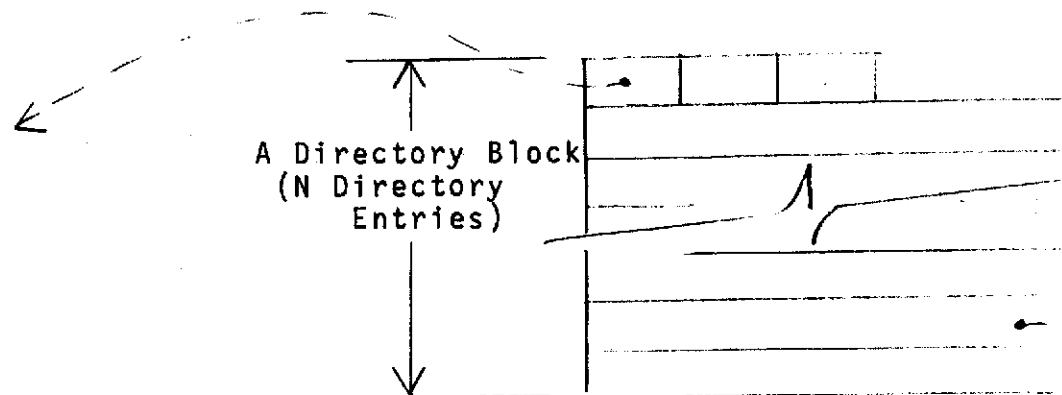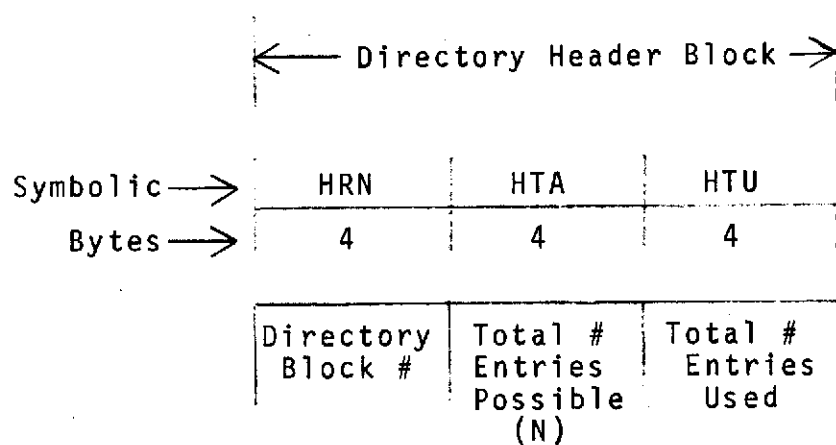<div align="center">***** ERROR # NNN OCCURRED *****</div>

For a description of the error message numbers ("NNN" in the above message),
see the standard error message listing section of this document.

If the step proceeds normally to conclusion (no errors), the following totals
are printed after the last control card:

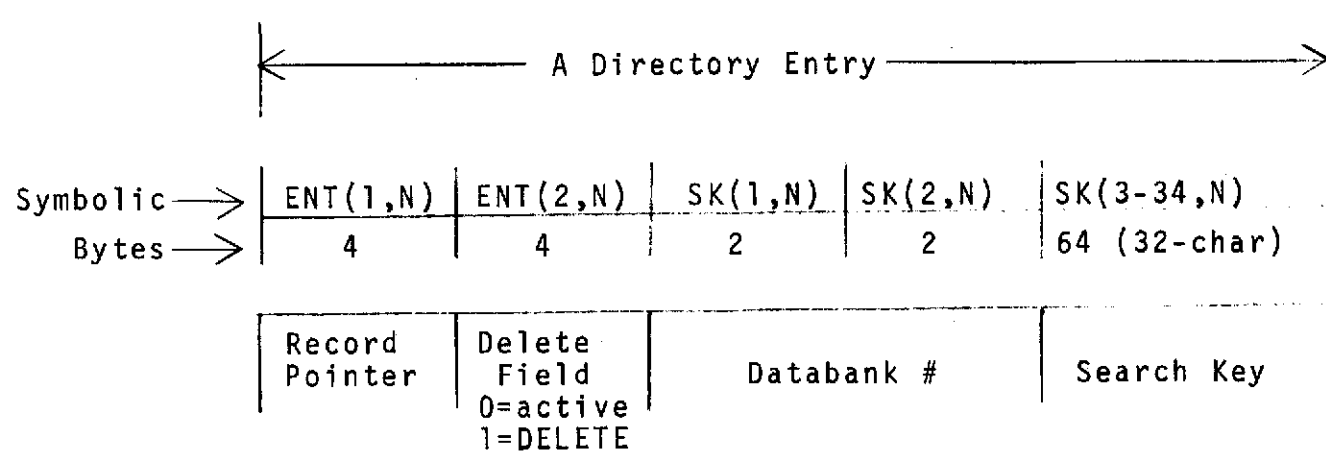<div align="center">*** TOTAL DIRECTORY ENTRIES GENERATED = NNNNNN ***</div>

<div align="center">*** TOTAL DIRECTORY ENTRIES DELETED = NNNNNN ***</div>

For an example of this listing, see Figure E-2.

|←—— Directory Header Block ——→|

				A Directory Block (N Directory Entries)

Symbolic →

HRN	HTA	HTU

Bytes →

4	4	4

Directory Block #	Total # Entries Possible (N)	Total # Entries Used

Record Name   : Directory Block

File Reference Letter   : D, L

FORTRAN Symbolic Base Declaration : DBLOCK (383)
                                           DIR (19)

Record Format Number   : Not Applicable

|←——————— A Directory Entry ———————→|

Symbolic →

ENT(1,N)	ENT(2,N)	SK(1,N)	SK(2,N)	SK(3-34,N)

Bytes →

4	4	2	2	64 (32-char)

Record Pointer	Delete Field 0=active 1=DELETE	Databank #	Search Key

Record Name   : Databank Capacity Record

File Reference Letter   : E

FORTRAN Symbolic Base Declaration   : CAPCTY (43)

Record Format Number   : R1

FW Subscript	HW Subscript	Field Length (Bytes)	Type: A=Alpha N=Numeric	Symbolic Name	Field Description
1	1	4	N	R1	Record number; always = 1 for Capacity Record
2	3	4	N	RAVAIL	Total # record allocated for databank
3	5	4	N	FQTOT	Total # records on FREE-QUEUE
4	7	4	N	FQPTR	Pointer to first record that is on the FREE-Q
5	9	4	N	DBMAX	Total # databanks allowed
6	11	4	N	DBCUR	Total # databanks currently in use
7	13	4	N	DBF	Directory blocking factor
8-43	15-86	144	NOT USED IN THIS RECORD FORMAT		

Record Name   :   Databank Name Reference Record

File Reference Letter   :   E

FORTRAN Symbolic Base Declaration   :   DBREC (43)

Record Format Number   :   R2

FW Subscript	HW Subscript	Field Length (Bytes)	Type: A=Alpha N=Numeric	Symbolic Name	Field Description
1	1	4	N	DBRN	Record Number
2	3	4	N	PNEXT	Pointer to next Databank Reference Record; if this is the last record in the chain, PNEXT = 0
3	5	4	N	RTYPE	= 1 (one) means record currently not in use = 2 (two) means record in use
4	7	4	N	RECTOT	Set to equal the total number of allowed databanks
5	9	4	N	PLAST	Pointer to last Databank Reference Record
6	11	4	N	SDB	Databank reference number; set to equal the current record number (DBRN)
7	13	64	A	SER	32-character databank search key (member name); in FORTRAN 32A1 (halfword) format
23	45	64	A	EPR	32-character databank name REVISION title; if none specified, field is blank; in 32A1 FORTRAN halfword format
39-43	77-86	40	NOT USED WITH THIS RECORD FORMAT		

Record Name : Function Designator Record

File Reference Letter : E

FORTRAN Symbolic Base Declaration : DBREC (43)

Record Format Number : R3

FW Subscript	HW Subscript	Field Length (Bytes)	Type: A=Alpha N=Numeric	Symbolic Name	Field Description
1	1	4	N	DBRN	Record number
2	3	4	N	PNEXT	Always zero with this record type
3	5	4	N	RTYPE	Record type = 3 indicates a Function Designator
4	7	4	N	RECTOT	Always = 1 for this record type
5	9	4	N	PLAST	Pointer to last record in this sequence; for this record type, always set to point to this record, i.e., PLAST = DBRN
6	11	4	N	SDB	Databank reference number of the databank of which this Function Designator is a member
7	13	64	A	SER	32-character Function Designator search key; in FORTRAN (32A1) halfword format
23	45	64	A	EPR	32-character Function Designator print expansion name in FORTRAN (32A1) halfword format
39	77	2	N	GPDATA (1)	Function Designator type: 1 = Load Discrete 7 = System Printer 2 = Load Analog 8 = System CRT 3 = Load Clock 9 = System Tape 4 = Sensor Discrete 10 = Issue Subroutine 5 = Sensor Analog 11 = System Interrupt 6 = Sensor Clock 12 = System Flag

Record Name   : Function Designator Record (Continued)

File Reference Letter   : E

FORTRAN Symbolic Base Declaration   : DBREC (43)

Record Format Number   : R3

FW Subscript	HW Subscript	Field Length (Bytes)	Type: A=Alpha N=Numeric	Symbolic Name	Field Description
* * For Function Designator Types 1 through 9, inclusive, and 11, 12 * *
N/A	78	2	N	GPDATA (2)	Function Designator hardware address
40-43	79-86	16		NOT USED WITH THIS RECORD VARIATION	
* * For Function Designator Types 10 * *
N/A	78	2	N	GPDATA (2)	Subroutine FORTRAN name lengtn (1 to 6, max)
40-42	79-84	2-12	A	GPDATA (3) through (8)	Subroutine FORTRAN name; one to six halfwords
43	85	4		NOT USED WITH THIS RECORD VARIATION	

E-18

Record Name   : MACRO Header Record

File Reference Letter   : E

FORTRAN Symbolic Base Declaration   : DBREC (43)

Record Format Number   : R4

FW Subscript	HW Subscript	Field Length (Bytes)	Type: A=Alpha N=Numeric	Symbolic Name	Field Description
1	1	4	N	DBRN	Record Number
2	3	4	N	PNEXT	Always non-zero with this record type; points to first MACRO Skeleton record (type = 5) of the MACRO
3	5	4	N	RTYPE	Record type - 4 means MACRO Header Record
4	7	4	N	RECTOT	Total # records for this MACRO; sum of all MACRO Skeleton (type = 5) + 1 for MACRO Header record
5	9	4	N	PLAST	Points to last MACRO Skeleton (type = 5) record belonging to this MACRO
6	11	4	N	SDB	Databank reference number of the databank to which this MACRO belongs
7	13	64	A	SER	32-character MACRO name in FORTRAN (32A1) format
23	45	64	A	EPR	32-character blank (field unused in this record type) field in (32A1) FORTRAN format (halfwords)
39	77	2	N	GPDATA(1)	Total # of format parameters declared witn this MACRO; NOTE - it is possible for a MACRO to have no (zero) formal parameters
N/A	78-86	18			NOT USED WITH THIS RECORD TYPE

E-19

Record Name   : MACRO Skeleton Record

File Reference Letter   : E

FORTRAN Symbolic Base Declaration : DBREC (43)

Record Format Number   : R5

FW Subscript	HW Subscript	Field Length (Bytes)	Type: A=Alpha N=Numeric	Symbolic Name	Field Description
1	1	4	N	DBRN	Record Number
2	3	4	N	PNEXT	If not last skeleton record for this MACRO, points to next skeleton image; if last, PNEXT = 0
3	5	4	N	RTYPE	Record type = 5
4-43	7	160	A	IMAGE	Scanned and processed MACRO skeleton card-image in FORTRAN (80A1) format

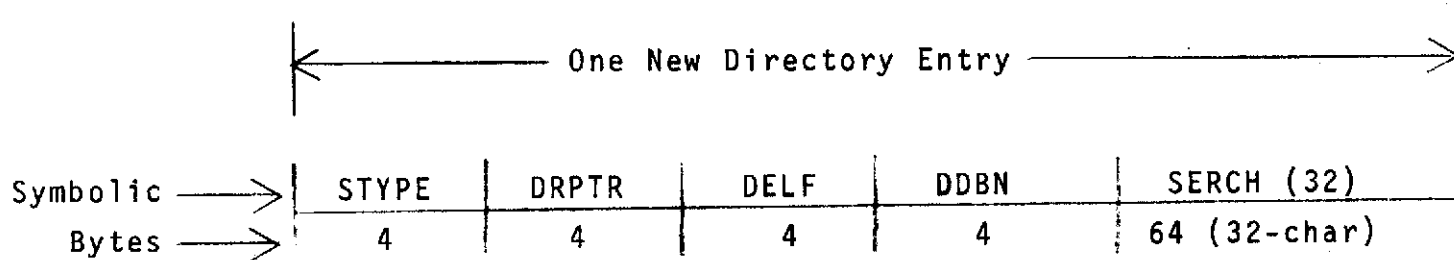Record Name   : Subroutine Name Record

File Reference Letter   : E

FORTRAN Symbolic Base Declaration : DBREC (43)

Record Format Number   : R6

FW Subscript	HW Subscript	Field Length (Bytes)	Type: A=Alpha N=Numeric	Symbolic Name	Field Description
1	1	4	N	DBRN	Record Number
2	3	4	N	PNEXT	Zero
3	5	4	N	RTYPE	Record type = 6
4	7	4	N	RECTOT	Always = 1 for this record type
5	9	4	N	PLAST	Points to this record number (DBRN)
6	11	4	N	SDB	Databank reference number of the databank to which this subroutine belongs
7	13	64	A	SER	32-character subroutine GOAL name in FORTRAN (32A1) halfword format
23	45	64	A	EPR	Unused in this record type
39	77	2	N	GPDATA(1)	Unused in this record format
N/A	78	2	N	GPDATA(2)	Number of characters in subroutine FORTRAN name; range is 1 to 6
40-42	79-84	2-12	A	GPDATA(3) thru (8)	Subroutine FORTRAN name; one to six halfwords with characters in FORTRAN (A1) format
43	85	4		NOT USED IN THIS RECORD FORMAT	

Record Name   :   New Directory Entry

File Reference Number   :   F, K

FORTRAN Symbolic Base Declaration   :   SORT (20)

Record Format Number   :   Not Applicable

```
         |<─────────── One New Directory Entry ───────────>|
         |                                                 |

Symbolic ─→ | STYPE  | DRPTR  |  DELF   |  DDBN    | SERCH (32)        |
Bytes    ─→ |   4    |   4    |   4     |   4      | 64 (32-char)      |

            | Sort    | Record  | Delete  | Databank   | Search Key        |
            | Record  | Pointer | Field   | Reference  | (in (32A1)        |
            | Type    |         | = 0     | Number     | FORTRAN halfword  |
            |  *      |         |(active) |            | format            |
```

1 = Process Record (DRPTR = total # new directory entries)
2 = Normal Directory Entry
3 = EOF (End-of-File) Dummy Record

Record Name   : Standard Sort/Merge Library

File Reference Letter   : G

FORTRAN Symbolic Base Declaration   : Not Applicable

Record Format Number   : Not Applicable

This dataset is the standard Sort/Merge Utility Program library required
for operation un-er OS/360. This dataset is generated as a portion of
the Operating System and is customarily cataloged under the data set name
"SYS1.SORTLIB".

Record Name   : Sort/Merge Utility Printer Output

File Reference Letter   : H

FORTRAN Symbolic Base Declaration : Not Applicable

Record Format Number   : Not Applicable

Indicated below is a sample of the output as produced by a successful run of
this utility program. Space allocation for the three intermediate work
datasets (File Reference Letter = I) was 100 tracks for this run.

```
IER036I - B =    90
IER037I - G =   625
IER038I - NMAX =
IER045I - END SORT PH
IER049I - SKIP MERGE PH
IER054I - RCD IN      30,OUT       30
IER052I - EOJ
```

The exact meaning of the above numbered messages is contained in the following
OS/360 Reference Manual:

IBM System/360 Operating System
Sort/Merge
Form # GC28-6543-6

The only real significance that this message sequence has to the GOAL compiler
is that no error messages are present.

Record Name   : Sort/Merge Intermediate Storage

File Reference Letter   : I

FORTRAN Symbolic Base Declaration   : Not Applicable

Record Format Number   : Not Applicable

These three datasets form the intermediate workspace required for the Sort/
Merge Utility program operation. The total disk space (in tracks) for a
2314 Disk sort is given by the following formula:

$$\text{SPACE} = \frac{3 \ (\text{\# of directory entries to be sorted} - \text{max})}{174} + 6$$

This computed space is to be divided evenly between the three datasets. The
formula above is a condensation of the standard Sort/Merge computation covered
in Section 2, pp 43 of the following OS/360 reference manual:

> IBM System/360 Operating System
> Sort/Merge
> Form # GC28-6543-6

Figure E-8 of this document tabularizes the storage requirements for the sort
work data sets for directory blocking factors ranging from 10 to 50.

THE FOLLOWING IS A TABLE INDICATING THE DISK WORK SPACE
REQUIRED TO SORT NEW DIRECTORY ENTRIES IN THE SECOND
STEP OF THE DIRECTORY-UPDATE PROCESS. THE TOTAL TRACKS
INDICATED BELOW ARE TO BE DIVIDED EQUALLY (ROUNDING UP
IF REQUIRED) BETWEEN THE THREE WORK SPACE ALLOCATIONS.

*** FIXED LENGTH SORT INPUT/OUTPUT RECORD SIZE (BYTES) : 80
*** TOTAL NUMBER INTERMEDIATE SORTWKNN STORAGE AREAS   : 3
*** N = DIRECTORY BLOCKING FACTOR

| | **** 2-LEVEL ***** | | **** 3-LEVEL ***** | |
| | * | * | * | * |
N	TOTAL ENTRIES	SORTWKNN TRACKS	TOTAL ENTRIES	SORTWKNN TRACKS
10	100	8	1000	24
11	121	9	1331	29
12	144	9	1728	36
13	169	9	2197	44
14	196	10	2744	54
15	225	10	3375	65
16	256	11	4096	77
17	289	11	4913	91
18	324	12	5832	107
19	361	13	6859	125
20	400	13	8000	144
21	441	14	9261	166
22	484	15	10648	190
23	529	16	12167	216
24	576	16	13824	245
25	625	17	15625	276
26	676	18	17576	310
27	729	19	19683	346
28	784	20	21952	385
29	841	21	24389	427
30	900	22	27000	472
31	961	23	29791	520
32	1024	24	32768	571
33	1089	25	35937	626
34	1156	26	39304	684
35	1225	28	42875	746
36	1296	29	46656	811
37	1369	30	50653	880
38	1444	31	54872	953
39	1521	33	59319	1029
40	1600	34	64000	1110

```
*** FIXED LENGTH SORT INPUT/OUTPUT RECORD SIZE (BYTES) : 80
*** TOTAL NUMBER INTERMEDIATE SORTWKNN STORAGE AREAS   : 3
*** N = DIRECTORY BLOCKING FACTOR
```

N	**** 2-LEVEL *****		**** 3-LEVEL *****	
	TOTAL ENTRIES	SORTWKNN TRACKS	TOTAL ENTRIES	SORTWKNN TRACKS
41	1681	35	68921	1195
42	1764	37	74088	1284
43	1849	38	79507	1377
44	1936	40	85184	1475
45	2025	41	91125	1578
46	2116	43	97336	1685
47	2209	45	103823	1797
48	2304	46	110592	1913
49	2401	48	117649	2035
50	2500	50	125000	2162

Record Name   : Sort/Merge Utility Control-Card Input

File Reference Letter   : J

FORTRAN Symbolic Base Declaration   : Not Applicable

Record Format Number   : Not Applicable

This dataset contains a single control-card with entry as depicted below.
This entry commences in card-column 2 and extends through card-column 55,
inclusive. The only blank character between column 2 and 55 is that between
the words "SORT" and "FIELDS".

<pre>
    cc   cc5
    2   6

    SORT FIELDS=(1,4,BI,A,13,4,BI,A,17,64,CH,A),SIZE=E1000
</pre>

This causes the Unsorted Directory Entry File (File Reference Letter = F) to
be sorted by the following sequence:

    Primary Sort - Sort Record Type (Ascending)

           2 - Databank Reference Number (Ascending)

           3 - 32-character Entry Name (Search Key) (Ascending)

Record Name   : Directory Construct Listing

File Reference Letter   : M

FORTRAN Symbolic Base Declaration   : Not Applicable

Record Format Number   : Not Applicable

An example of the listing of concern is to be found in Figure E-4 of this
Appendix. In general, the following results from a correct and complete
directory construction operation:

<u>NEW DIRECTORY STATISTICS</u>

                        20: DBF

                        14: WMAX

                       239: DRTOT
                       224: OLDTOT
                         0: DELTOT
                        15: SORTOT

                         0: ZERO BALANCE

The meaning of the symbolic counts depicted above are as follows:

DBF   : Directory blocking factor of the current directory

WMAX   : Maximum number of directory blocks containing any entries

DRTOT   : Total number of named entries currently supported in the
                 directory data set

OLDTOT   : Total number of entries that were supported in the directory
                 before the current additions/deletions were made

DELTOT   : Total number of directory entries that were deleted as a
                 result of the current maintenance action

SORTOT   : Total number of sorted new directory entries that were
                 processed from the input data set (File Reference Letter K)

ZERO BALANCE : A cross check made of the above counts to attempt error
                 checks; the balance will always be zero for a correct
                 maintenance process. Numerically,

$$ZERBAL = DRTOT - (OLDTOT - DELTOT) - SORTOT$$

Record Name   : Databank Initialize Control-Cards

File Reference Letter   : N

FORTRAN Symbolic Base Declaration   : None

Record Format Number   :

The following two control-cards are required for each databank initialization run:

```
      cc    cc
      1     1
      0     5

     8000   TYPE-1 CARD:   TOTAL # RECORDS AVAILABLE IN DATABANK.

      30    TYPE-2 CARD:   TOTAL # DATABANKS TO BE ALLOWED.
```

The first type card indicates via the integer (right-justified to column 10) the total # of records allocated for the databank (File Reference Letter = E). This number must be the cube of the Directory Blocking Factor. Thus the example above indicates the total for a Blocking Factor of 20.

The second control-card indicates the total # of databanks to be allowed. The integer is right-justified to card-column 10.

The commentary shown on both card types as beginning in column 15 is optional and may be omitted. No data beyond card-column 10 is processed by the initialization module.

Record Name   :   Initialization Program Printer Output

File Reference Letter   :   Ø

FORTRAN Symbolic Base Declaration   :   Not Applicable

Record Format Number   :   Not Applicable

The following printer sequence ensues upon successful completion of the
initialization sequence. The example below indicates a databank initialized
with 8000 records total, 30 databanks allowed, and a Directory Blocking Factor
of 20.

DATABANK CAPACITY RECORD

```
                          RECORD # :      1
          TOTAL # RECORDS ALLOCATED :   8000
         TOTAL # RECORDS ON FREE-QUEUE :  7969
         HEAD OF FREE-QUEUE RECORD CHAIN :   32
         MAXIMUM # OF ALLOWED DATABANKS :   30
    TOTAL # DATABANKS CURRENTLY IN USE :    0
```

APPENDIX F

SAMPLE CONTROL-CARD INPUT

F.   INTRODUCTION

The purpose of this Appendix is to provide a sample of the control-card input to the data bank pre-processor program. In this regard, refer to Appendix D, Figure D-4, which pictorially represents the data flow during data bank maintenance. The following is to be noted:

   o   The control-card input, data set reference letter
        Al of Figure D-4 is represented by the sample input
        shown in Figure F-1 of this Appendix.

   o   Figure F-2 is a sample of the fixed-format control
        card images as produced by the pre-processor program.
        Refer to data set reference letter A of Figure D-4.

The exact content of the card-image records is given in Appendix E (Data Set Record Formats) for data set reference numbers Al and A.

SAMPLE PRE-PROCESSOR CONTROL-CARD INPUT

** This example of Pre-Processor Program input results in the output of the
   fixed-format card images shown on Figure F-2.

```
BEGIN DATABANK (SAMPLE) REVISION A ;
SPECIFY <PRIMARY GSCU ON SWITCH> AS LOAD TYPE DISCRETE ADDRESS 252 ;
SPECIFY <VENT MOTOR FIELD> AS LOAD TYPE ANALOG ADDRESS 0063 ;
SPECIFY <CDC SET CKT> AS LOAD TYPE CLOCK ADDRESS 0040 ;
SPECIFY <MANIFOLD PRESSURE LOW LAMP> AS SENSOR TYPE DISCRETE ADDRESS 4123 ;
SPECIFY <IU COOLANT PRESS> AS SENSOR TYPE ANALOG ADDRESS 0049 ;
SPECIFY <EST> AS SENSOR TYPE CLOCK ADDRESS 6 ;
SPECIFY <LINE PRINTER> ALSO AS <360/PRINTER> AS SYSTEM TYPE PRINTER ;
SPECIFY <CRT2> AS SYSTEM TYPE CRT ADDRESS 1 ;
SPECIFY <LOG> AS SYSTEM TYPE TAPE ;
SPECIFY <PRINTER BUSY> AS SYSTEM TYPE INTERRUPT 0 ;
SPECIFY <PROCEDURE IN PROGRESS FLAG> AS SYSTEM TYPE FLAG 1 ;
SPECIFY <GOAL SR NAME> AS SUBROUTINE (FORT23) ;
NAME (POWER ON) SUBROUTINE (FORT4) ;
NAME (ABCDEFG) PROGRAM (FORT25) ;
BEGIN MACRO ABC (A),(B),(C),<A> ;
   (ABC) = (A) ;
   (DEF) = (B) ;
   (GHI) = (C) ;
   (JKL) = <A> ;
END MACRO ;
END DATABANK ;
FINIS
```

Figure F-1.

SAMPLE PRE-PROCESSOR PROGRAM CONTROL-CARD OUTPUT

** The fixed card images shown below are the Pre-Processor program
   output resulting from the input card set shown on Figure F-1.

```
DATABANK   (SAMPLE)                             (A)
SPECIFY    <PRIMARYGSCUONSWITCH>                LOAD        DISCRETE     0252
SPECIFY    <VENTMOTORFIELD>                     LOAD        ANALOG       0063
SPECIFY    <CDCSETCKT>                          LOAD        CLOCK        0040
SPECIFY    <MANIFOLDPRESSURELOWLAMP>            SENSOR      DISCRETE     4123
SPECIFY    <IUCOOLANTPRESS>                     SENSOR      ANALOG       0049
SPECIFY    <EST>                                SENSOR      CLOCK        0006
SPECIFY    <LINEPRINTER>
           <360/PRINTER>                        SYSTEM      PRINTER
SPECIFY    <CRT2>                               SYSTEM      CRT          0001
SPECIFY    <LOG>                                SYSTEM      TAPE
SPECIFY    <PRINTERBUSY>                        SYSTEM      INTERRUPT    0000
SPECIFY    <PROCEDUREINPROGRESSFLAG>            SYSTEM      FLAG         0001
SPECIFY    <GOALSRNAME>                         SUBROUTINE  (FORT23)
NAME       (POWERON)                            SUBROUTINE  (FORT4)
NAME       (ABCDEFG)                            SUBROUTINE  (FORT25)
MACRO      (ABC)
           (A)
           (B)
           (C)
           <A>
    (ABC)  =  (A)  ;
    (DEF)  =  (B)  ;
    (GHI)  =  (C)  ;
    (JKL)  =  <A>  ;
END        MACRO
END        DATABANK
```

Figure F-2.

APPENDIX G

DATA BANK MAINTENANCE MODULE ERROR MESSAGES

G.   INTRODUCTION

Presented in a tabular form in this Appendix is a listing of the possible error messages that can be generated from the data bank maintenance programs. It is to be noted that the error messages for the pre-processor program are listed separately in this document in Appendix H.

Error Number	Program Module	Description
1	MAINT	Control card type unknown - a data bank has not yet been opened (a BEGIN DATABANK statement has not yet been encountered)
2	MAINT	Control card type unknown - a data bank has been opened (a BEGIN DATABANK statement has already been processed)
3	DBEND	END DATABANK control card contents invalid
4	VNAME BEGDB	Too many characters in a name, or, the rightmost name delineator was omitted
5	VNAME BEGDB	Leftmost delineator in a name is not the expected character or is missing
6	VNAME	A character found in a name is not a valid letter of the alphabet or a valid digit (0-9)
7	VNAME	The first character in a name is not a letter
8	BEGDB	Keyword error on a BEGIN DATABANK statement
9	BEGDB	Trying to initilize a new data bank but the maximum number of allowed data banks has already been reached
10	DBSEEK	Wrong record type (not type = 1 or 2) found when searching through the data bank reference record chain in the data bank file
11	DBSEEK	Chaining error in the data bank reference records section of the data bank file
12	SPECFY	Keyword error, missing, or out of sequence on a SPECIFY statement
13	SPECFY	Delineator problem (missing or invalid) on a function designator name
14	SPECFY	End-of-file found on input data set while attempting to read the second card of a SPECIFY statement
15	SPECFY	Error encountered in parsing a subroutine name or in the delineators surrounding a subroutine name
16	SPECFY	Discrete/analog/clock/interrupt/flag address field contains an illegal (non digit) character

Error Number	Program Module	Description
17	FIND	Program logic error when attempting a data bank directory search
18	SUPERD	A data bank function designator record has been found but the function designator type field is zero or negative
19	SUPERD	Same problem as 18, above, however the function designator type field is greater than 11
20	SUPERD	A macro skeleton record has been found in a macro chain in the data bank file, but the record type is not type = 5
21	NAMESR	Keyword error on NAME SUBROUTINE control card
22	NAMESR	Error encountered in parsing a FORTRAN-equivalent subroutine name
23	MACRO	Error in one of the keyword fields on a macro
24	MACRO	Error in a parameter field parse on the BEGIN MACRO statement
25	MACRO	End-of-file found in input data set before the end of a macro was found
26	MACRO	Too many formal parameters found on a BEGIN MACRO statement (more than 10)
27	MACRO	Ran out of space in the data bank file while entering a macro
28	MACRO	Program logic error encountered when processing the macro header record chain
29	MACRO	A macro has no skeleton cards
30	SPECFY	Error encountered in parsing a SYSTEM type keyword, i.e., PRINTER, TAPE, CRT, INTERRUPT, FLAG
31	DELETE	Keyword error found on a DELETE control card
32	DELETE	Delimeter error on the name field
33	DELETE	Member named for deletion is not in the data bank

Error Number	Program Module	Description
34	DELETE	A function designator was scheduled for deletion (the name on the DELETE card was bounded by brackets); the record found in the data bank is not a function designator (type = 3) record
35	DELETE	Name to be deleted was bounded by parenthesis, indicating that the record type was not a function designator; the record found in the data bank has a function designator type field (type = 3, or less)
36	DELETE	Logic error encountered while attempting to make a directory deletion
37	SPECFY	Error in address field - non-numeric characters in a digit field
40	DCON	Record sequencing error in the sorted directory entries data set
41	DCON	End-of-file found on sort input data set before first record read and processed
42	DCON	End-of-file record missing in the sort data set
43	DCON	Duplicate record name in the directory (a single data bank contains two records with the same alphanumeric names)
46	DELDB	Keyword contents invalid on a DELETE DATA BANK control card
47	DELDB	Cannot locate the data bank to be deleted
48	DELDB	Logic error in data bank deletion process; cannot locate the required data bank reference record by using the data bank reference number

APPENDIX H

PRE-PROCESSOR ERROR MESSAGE LIST

H.   INTRODUCTION

As mentioned in Appendix D, Program Module Description, the data bank
pre-processor program is actually the GOAL compiler module driven by a
suitable syntax table. The syntax equations required are covered in
Appendix I. The error messages to be generated are taken from the
standard GOAL compiler message list, a copy of which is included in this
Appendix. The actual message numbers of concern for the data bank pre-
processor program are:

131	844	911
800	847	918
804	903	927
808	904	939
829	909	952
836	910	999

```
100 INVALID ROW DESIGNATOR OR KEYWORD 'ROW' IS MISSING .
101 INVALID COLUMN INDEX NAME OR COLUMN INTEGER NUMBER.
102 INVALID ROW INDEX NAME OR ROW INTEGER NUMBER.
103 INVALID LIST INDEX NAME OR LIST INTEGER NUMBER.
104 INVALID REFERENCE OR KEYWORD FOLLOWING KEYWORD 'SEND' OR 'APPLY'.
106 INVALID OR MISSING EXTERNAL DESIGNATOR -FROM-
108 INVALID OR MISSING EXTERNAL DESIGNATOR - TO -
110 INVALID INTERNAL NAME WHICH MUST BE DECLARED AS A STATE VALUE.
112 INVALID INTERNAL NAME OR STATE WHICH MUST BE DECLARED AS STATE VALUES.
114 INVALID INTERNAL NAME WHICH MUST NOT BE DECLARED AS STATE OR TEXT .
122 INVALID INTEGER NUMBER OF ENTRIES.
124 INVALID INTERNAL NAME OR STATE .
128 INVALID NUMBER NAME.
129 INVALID NUMBER NAME. THIS NAME IS PREVIOUSLY DEFINED.
130 INVALID NUMBER PATTERN OR NUMBER.
131 INVALID NUMERIC VALUE - MUST BE 1-4 DIGITS.
132 INVALID QUANTITY NAME.
133 INVALID QUANTITY NAME. THIS NAME IS PREVIOUSLY DEFINED .
134 INVALID QUANTITY VALUE.
138 INVALID STATE VALUE.
140 INVALID TEXT NAME.
141 INVALID TEXT NAME. THIS NAME IS PREVIOUSLY DEFINED.
142 INVALID NUMERIC LIST NAME.
143 INVALID NUMERIC LIST NAME. THIS NAME IS PREVIOUSLY DEFINED.
144 INVALID NUMERIC TABLE NAME
145 INVALID NUMERIC TABLE NAME . THIS NAME IS PREVIOUSLY DEFINED.
146 INVALID INTEGER NUMBER OF COLUMNS.
147 INVALID INTEGER NUMBER OF COLUMNS. THE LIMITS ARE 0 THROUGH 45.
148 INVALID INTEGER NUMBER OF ROWS.
149 INVALID INTEGER NUMBER OF ROWS. THE LIMITS ARE 1 THROUGH 45.
150 INVALID COLUMN NAME.
151 INVALID COLUMN NAME OR KEYWORD 'COLUMN' IS MISSING.
152 INVALID QUANTITY LIST NAME.
153 INVALID QUANTITY LIST NAME. THIS NAME IS PREVIOUSLY DEFINED.
```

H-2

154 INVALID QUANTITY TABLE NAME.
155 INVALID QUANTITY TABLE NAME. THIS NAME IS PREVIOUSLY DEFINED.
156 INVALID STATE LIST NAME.
157 INVALID STATE LIST NAME. THIS NAME IS PREVIOUSLY DEFINED.
158 INVALID STATE TABLE NAME.
159 INVALID STATE TABLE NAME. THIS NAME IS PREVIOUSLY DEFINED.
160 INVALID INTERNAL NAME OR NUMBER PATTERN .
162 INVALID TEXT LIST NAME.
163 INVALID TEXT LIST NAME. THIS NAME IS PREVIOUSLY DEFINED.
164 INVALID INTEGER NUMBER OF CHARACTERS.
165 INVALID INTEGER NUMBER OF CHARACTERS. THE LIMITS ARE 1 THROUGH 132.
166 INVALID TEXT TABLE NAME.
167 INVALID TEXT TABLE NAME. THIS NAME IS PREVIOUSLY DEFINED.
168 INVALID DELAY STATEMENT FOLLOWING THE VERB DELAY OR WAIT.
172 INVALID REFERENCE OR KEYWORD FOLLOWING THE VERB ISSUE .
173 INVALID LEAVE STATEMENT - LEAVE CAN ONLY BE USED WITHIN A SUBROUTINE
174 INVALID RESUME STATEMENT.
175 INVALID LEAVE STATEMENT.
176 INVALID PERFORM SUBROUTINE STATEMENT FOLLOWING THE SUBROUTINE NAME.
180 INVALID RECORD DATA STATEMENT FOLLOWING THE KEYWORD DISPLAY,PRINT OR RECORD.
182 INVALID STEP NUMBER OR KEYWORD 'ALL' IS MISSING.
184 INVALID TEXT, NAME OR FUNCTION DESIGNATOR FOLLOWING THE VERB REPLACE.
186 INVALID TEXT OR KEYWORD 'ENTRY' IS MISSING FOLLOWING THE VERB REQUEST.
190 INVALID REFERENCE OR KEYWORD 'PRESENT VALUE OF' FOLLOWING THE VERB SET.
195 INVALID WHEN INTERRUPT STATEMENT FOLLOWING THE KEYWORD 'OCCURS'.
200 THE NUMBER OF ENTRIES INITIALIZED EXCEEDS THE NUMBER SPECIFIED.
201 THE NUMBER OF COLUMN TITLES EXCEEDS THE SPECIFIED NUMBER OF COLUMNS.
202 THE NUMBER OF ENTRIES INITIALIZED IS LESS THAN THE NUMBER SPECIFIED.
203 THE NUMBER OF COLUMN TITLES IS LESS THAN THE SPECIFIED NUMBER OF COLUMNS.
204 THE FUNCTION DESIGNATOR SPECIFIED IS NOT DEFINED IN THE DATA BANK.
206 INVALID ROW FUNCTION DESIGNATOR. IT IS PREVIOUSLY DEFINED IN THIS TABLE.
210 INVALID COLUMN TITLE NAME. THIS NAME IS PREVIOUSLY DEFINED IN THIS TABLE.
212 EXECUTION RATE AS SPECIFIED IS GREATER THAN TEN MINUTES.
214 CONCURRENT STATEMENT DOES NOT HAVE A STEP NUMBER.
216 CORRESPONDENCE IS INVALID (SHOULD BE 1 TO 1, 1 TO MANY OR MANY = MANY)
218 INVALID NUMERIC FORMULA (UNBALANCED PARENTHESES)
220 INVALID INTERNAL NAME (NOT DECLARED AS NUMERIC OR QUANTITY)
222 INVALID INTERNAL NAME (NOT A SINGLE ELEMENT)

224 INVALID NUMERIC FORMULA (SIZE EXCEEDS COMPILER CAPACITY)
228 FUNCTION DESIGNATOR SPECIFIED IS NOT A SUBROUTINE PARAMETER.
300 INVALID MACRO LABEL- DOES NOT START WITH A LETTER.
301 INVALID MACRO LABEL- LONGER THAN 32 CHARACTERS.
302 INVALID MACRO LABEL- CONTAINS AN ILLEGAL CHARACTER.
303 INVALID MACRO LABEL- MACRO LABEL IS MULTI-DEFINED.
304 INVALID MACRO PARAMETER - DOES NOT START WITH A LETTER.
305 INVALID MACRO PARAMETER - LONGER THAN 32 CHARACTERS.
306 INVALID MACRO PARAMETER - CONTAINS AN ILLEGAL CHARACTER.
307 INVALID MACRO PARAMETER - MACRO PARAMETER IS MULTI-DEFINED.
308 EXPECTED SEMICOLON ';' NOT FOUND AFTER PROCESSING THE 10 MAXIMUM PARAMETERS.
309 EITHER COMMA ',' OR SEMICOLON ';' WAS OMITTED.
310 LEFT PARENTHESIS '(' MISSING ON PARAMETER FOLLOWING COMMA.
311 MACRO TO BE EXPANDED AND/OR EXECUTED IS NOT DEFINED.
312 MACRO TO BE EXPANDED AND/OR EXECUTED NEEDS PARAMETERS - NONE WERE SUPPLIED.
313 INVALID SUBSTITUTION PARAMETER - CONTAINS AN ILLEGAL CHARACTER.
314 INVALID SUBSTITUTION PARAMETER - CONTAINS NO CHARACTERS.
315 NUMBER OF PARAMETERS IN STATEMENT AND MACRO ARE NOT THE SAME.
316 NUMBER OF PARAMETERS IN STATEMENT EXCEEDS NUMBER OF PARAMETERS IN MACRO.
317 INVALID SUBSTUTUTION PARAMETER - LONGER THAN 79 CHARACTERS.
318 INVALID MACRO BODY - CONTAINS NO CHARACTERS.
350 INVALID CHARACTER STRING - CONTAINS AN ILLEGAL CHARACTER.
351 INVALID CHARACTER STRING - CONTAINS MORE THAN 32 CHARACTERS.
352 INVALID REPLACEMENT CHARACTER STRING. CONTAINS MORE THAN 80 CHARACTERS.
353 INVALID REPLACEMENT CHARACTER STRING. CONTAINS AN ILLEGAL CHARACTER.
354 REPLACEMENT NAME, CHARACTER STRING OR FUNCTION DESIGNATOR IS MULTI-DEFINED.
400 NUMBER OF DATA BANKS IN USE HAS EXCEEDED THE MAXIMUM OF 10.
402 DATA BANK SPECIFIED IS ALREADY IN USE.
406 INVALID DATA BANK NAME. THE DATA BANK NAME IS MULTI-DEFINED.
408 UNABLE TO FREE DATA BANK AS NONE IS BEING USED AT THIS TIME.
410 SPECIFIED DATA BANK NAME DOES NOT EXIST.
412 UNABLE TO FREE DATA BANK AS IT IS NOT IN USE AT THIS TIME.
413 LABEL ERROR - THE STATEMENT FOLLOWING AN UNCONDITIONAL GO TO IS NOT NUMBERED
414 STRUCTURAL ERROR ** PREAMBLE STATEMENT FOUND IN PROCEDURAL BODY.
415 SYMBOL TABLE OVERFLOW HAS OCCURRED. A MAXIMUM OF 9999 ENTRIES IS ALLOWED.
800 INVALID ADDRESS - MUST BE 1-4 DIGITS.
802 INVALID COMPARISON TEST.
804 INVALID DATA BANK NAME.

```
805 INITIALIZATION OF REFERENCED **SUBROUTINE PARAMETER** NAME IS NOT ALLOWED.
806 INVALID OR MISSING EXTERNAL DESIGNATOR.
807 END PROGRAM STATEMENT IS INVALID DURING A SUBROUTINE COMPILATION.
808 INVALID FUNCTION DESIGNATOR.
809 END SUBROUTINE STATEMENT IS INVALID DURING A PROGRAM COMPILATION.
810 INVALID NUMBER,NUMBER PATTERN,QUANTITY,STATE,TEXT OR INTERNAL NAME.
812 INVALID INTEGER NUMBER.
814 INVALID INTERNAL NAME.
816 INVALID OR MISSING REFERENCE FOLLOWING THE COMMA.
826 INVALID NUMERIC FORMULA.
828 INVALID OUTPUT EXCEPTION.
829 INVALID NAME OR FUNCTION DESIGNATOR.
830 INVALID SUBROUTINE PARAMETER (NAME OR FUNCTION DESIGNATOR).
832 INVALID OR MISSING PROGRAM NAME.
834 INVALID QUANTITY OR INTERNAL NAME .
836 INVALID REVISION LABEL.
838 INVALID ROW DESIGNATOR.
841 INVALID STEP NUMBER. THIS STEP NUMBER IS PREVIOUSLY DEFINED.
842 INVALID STEP NUMBER.
843 INVALID PERFORM PROGRAM OR PERFORM SUBROUTINE STATEMENT.
844 INVALID SUBROUTINE NAME.
845 BEGIN PROGRAM OR BEGIN SUBROUTINE FOUND DURING A PROGRAM COMPILATION.
846 INVALID TABLE NAME.
847 INVALID FORTRAN SUBROUTINE NAME.
848 INVALID TEXT CONSTANT.
849 A TEXT CONSTANT ENTRY EXCEEDED THE MAXIMUM NUMBER OF CHARACTERS SPECIFIED.
850 INVALID TIME VALUE.
852 INVALID FUNCTION DESIGNATOR TYPE IN THE SPECIFY STATEMENT.
853 INVALID ROW FUNCTION DESIGNATOR TYPE. MUST BE A LOAD OR SENSOR ANALOG.
854 INVALID ROW FUNCTION DESIGNATOR TYPE. MUST BE A LOAD OR SENSOR DISCRETE.
855 INVALID ROW FUNCTION DESIGNATOR TYPE. MUST BE A SYSTEM FUNCTION DESIGNATOR.
856 THE NUMBER OF ROW FUNCTION DESIGNATORS EXCEEDS THE NUMBER OF ROWS.
857 THE NUMBER OF ROW FUNCTION DESIGNATORS IS LESS THEN THE NUMBER OF ROWS.
900 KEYWORD NOT FOUND - AND.
901 KEYWORD NOT FOUND - RETURN.
902 KEYWORD NOT FOUND - AND SAVE AS.
903 KEYWORD NOT FOUND - ADDRESS.
904 KEYWORD NOT FOUND - AS.
907 KEYWORD NOT FOUND - READINGS OF.
```
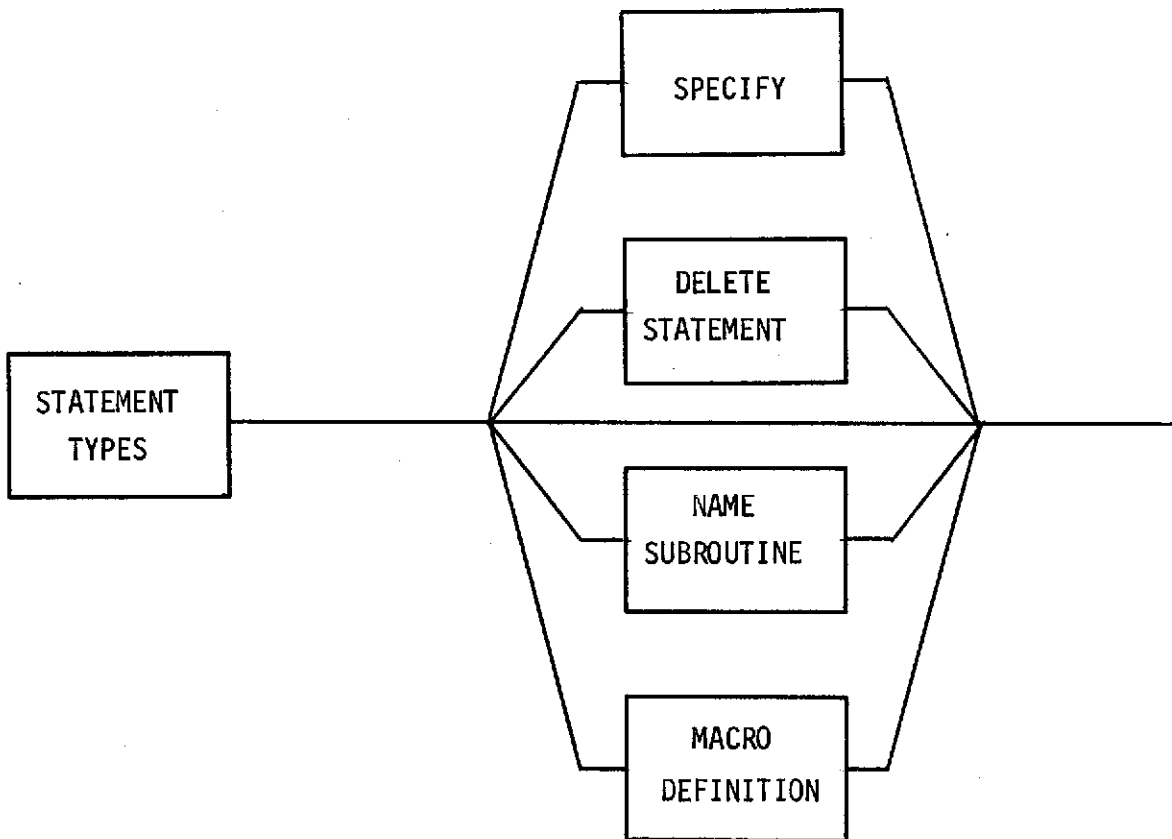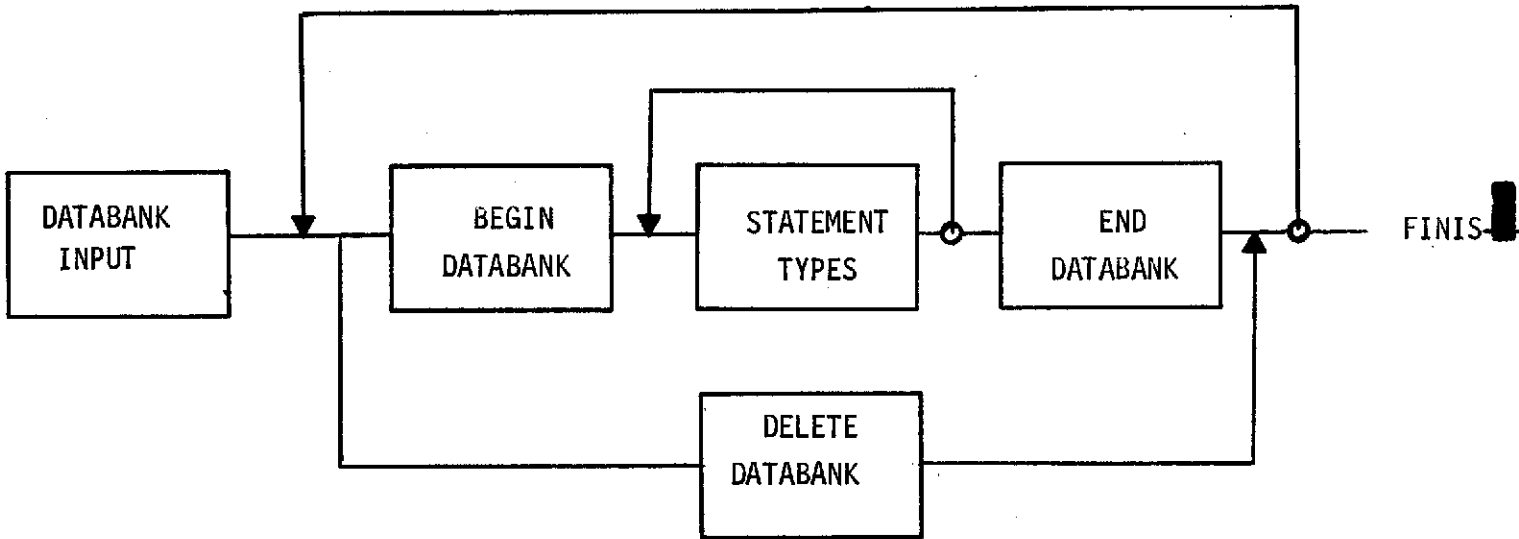
```
908 KEYWORD NOT FOUND - CHARACTERS.
909 KEYWORD NOT FOUND - CRT, PRINTER, TAPE, INTERRUPT, OR FLAG.
910 KEYWORD NOT FOUND - DATABANK OR MACRO.
911 KEYWORD NOT FOUND - ANALOG, CLOCK, OR DISCRETE.
912 KEYWORD NOT FOUND - ENTRIES.
913 KEYWORD NOT FOUND - EXCEPTIONS.
914 KEYWORD NOT FOUND - EQUAL TO OR =.
916 KEYWORD NOT FOUND - FROM
918 KEYWORD NOT FOUND - LOAD OR SENSOR OR SYSTEM.
920 KEYWORD NOT FOUND - OCCURS.
922 KEYWORD NOT FOUND - UNTIL.
924 KEYWORD NOT FOUND - PRESENT VALUE OF.
925 KEYWORD NOT FOUND - COLUMNS.
926 KEYWORD NOT FOUND - ROWS AND.
927 KEYWORD NOT FOUND - REVISION.
930 KEYWORD NOT FOUND - SUBROUTINE.
934 KEYWORD NOT FOUND - TIMES.
938 KEYWORD NOT FOUND - TO
939 KEYWORD NOT FOUND - TYPE.
940 KEYWORD NOT FOUND - WITH.
941 KEYWORD NOT FOUND - WITH ENTRIES.
944 KEYWORD NOT FOUND - WITH A MAXIMUM OF, EQUAL TO OR =.
945 BEGIN PROGRAM OR BEGIN SUBROUTINE FOUND DURING A SUBROUTINE COMPILATION.
946 KEYWORD NOT FOUND - PERFORM PROGRAM, VERIFY, DISPLAY, PRINT, OR RECORD.
948 KEYWORD NOT FOUND - NUMBER, QUANTITY, STATE OR TEXT.
952 KEYWORD NOT FOUND - PROGRAM OR SUBROUTINE.
954 KEYWORD NOT FOUND - AND INDICATE RESTART LABELS OR SEMICOLON ';'.
986 KEYWORD NOT FOUND - THEN OR COMMA ',' .
987 INVALID PAGE NUMBER FOLLOWING THE WORD PAGE. LIMITS ARE 1-999.
988 INVALID LINE SIZE FOLLOWING PAGE SIZE. LIMITS ARE 80-110.
989 INVALID PAGE SIZE FOLLOWING THE WORD LINE. LIMITS ARE 1-32767.
990 INVALID DATE TEXT CONSTANT FOLLOWING THE WORD DATE. LIMITS ARE 1-8.
991 INVALID TITLE TEXT CONSTANT FOLLOWING THE WORD TITLE. LIMITS ARE 1-100.
992 INVALID SEQUENCE FIELD NUMBER FOLLOWING THE WORD SEQ. LIMITS ARE 0-10.
993 INVALID COMPOUND COMPILER CONTROL CARD.
994 INVALID COMPILER CONTROL CARD.
995 THIS STATEMENT IS NOT RECOGNIZED AS A GOAL STATEMENT .
996 EXPECTED DOUBLE DOLLAR SIGN '$$' NOT FOUND
998 EXPECTED COMMA ',' NOT FOUND.
999 EXPECTED SEMICOLON ';' NOT FOUND.
```
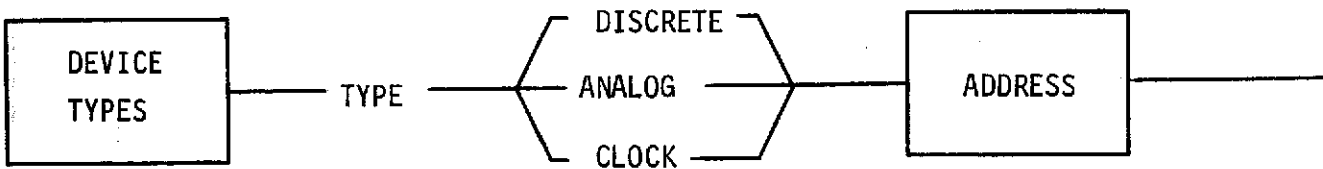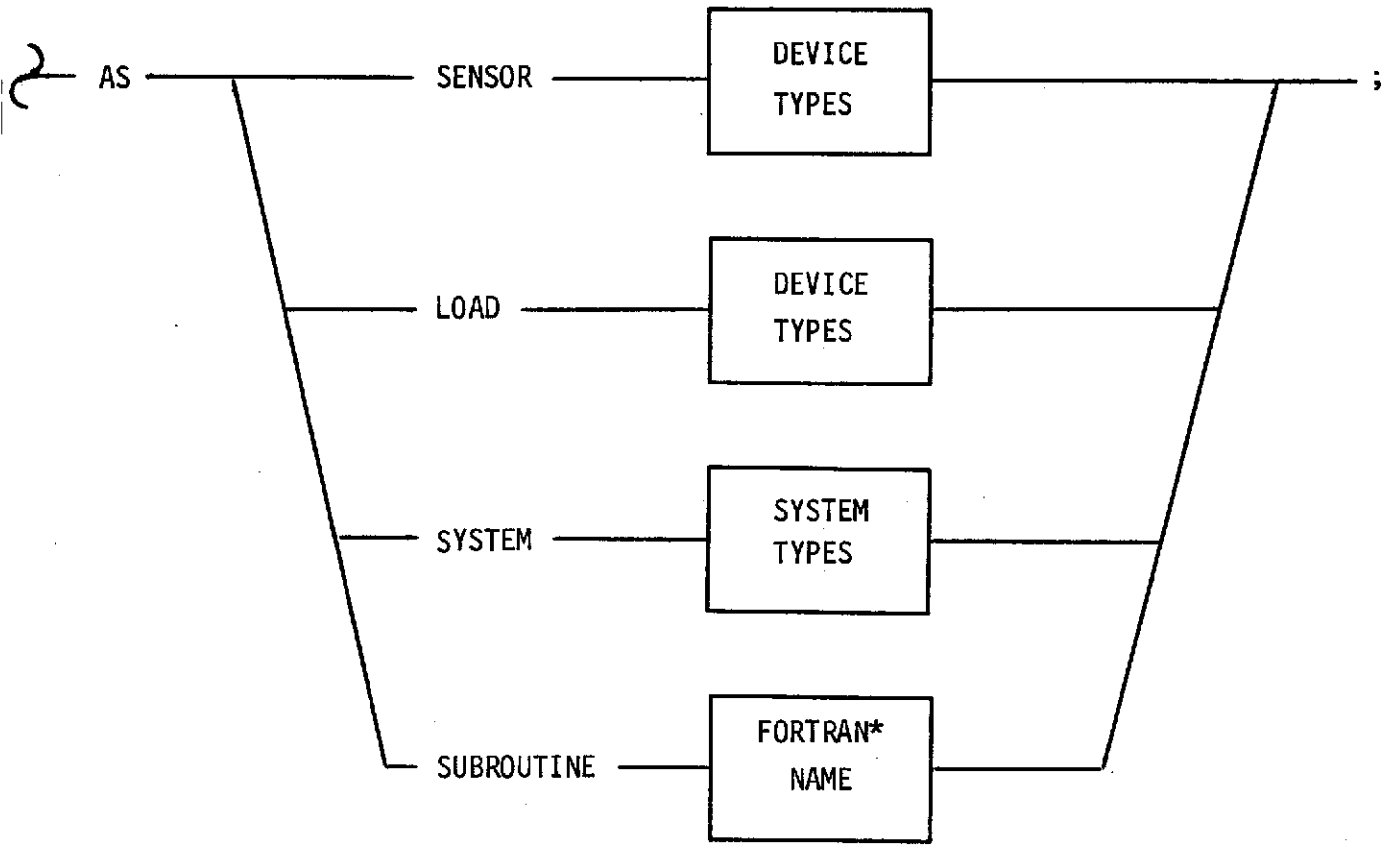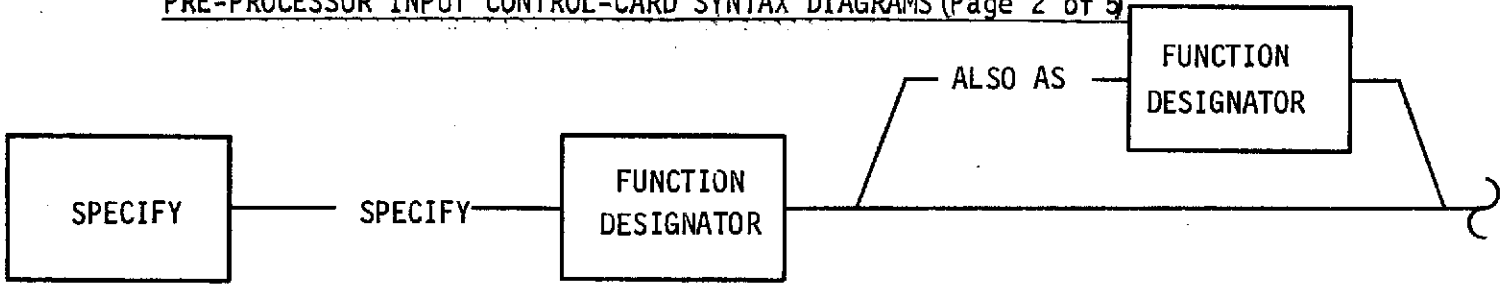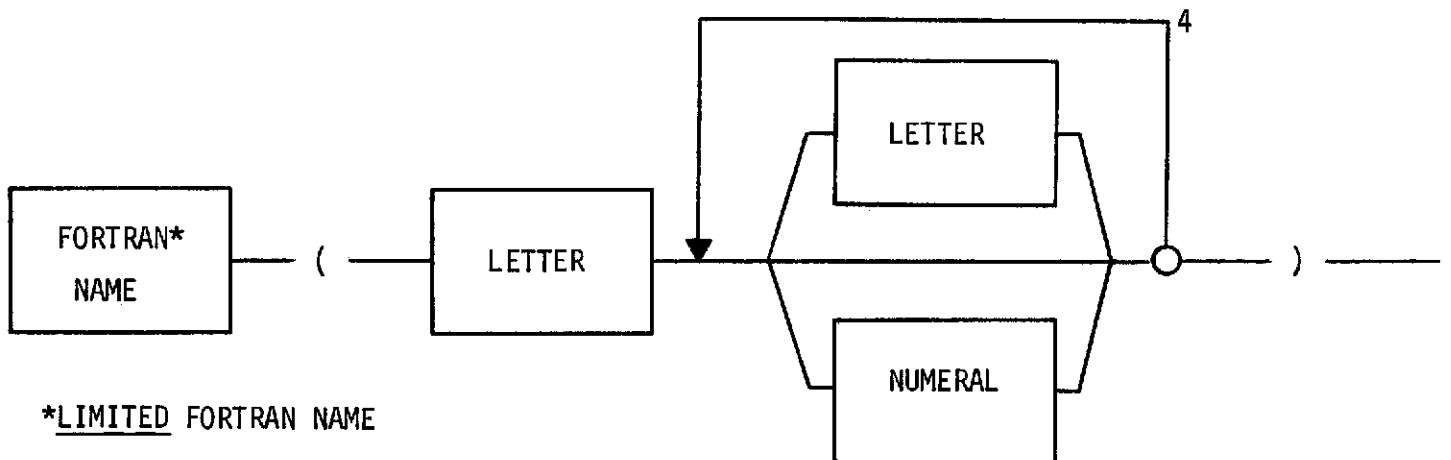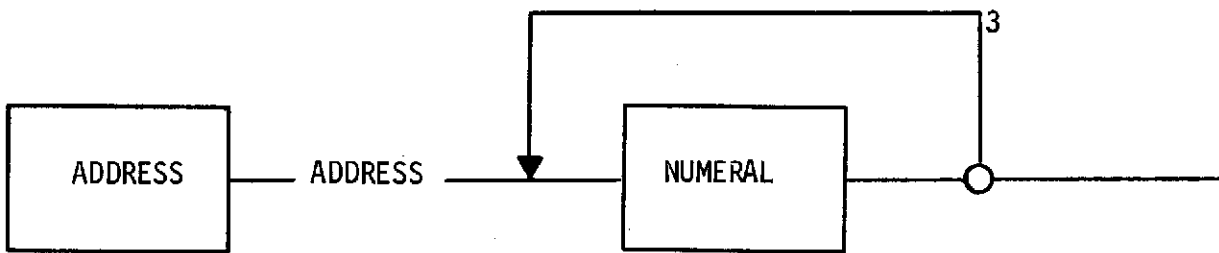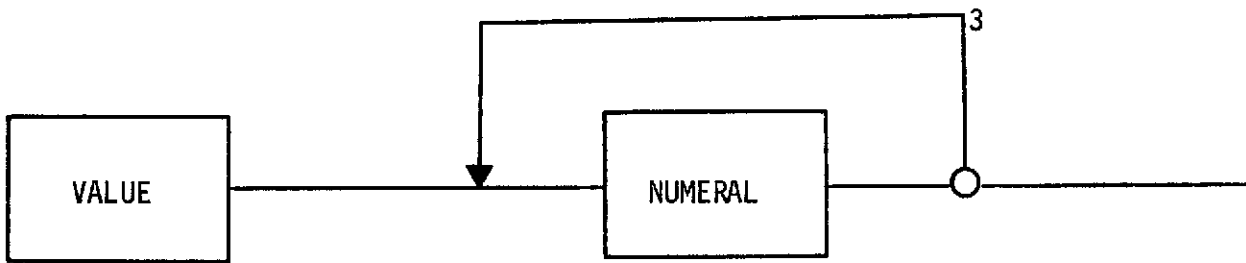
APPENDIX I

DATA BANK PRE-PROCESSOR SYNTAX TABLES

I.   INTRODUCTION

As mentioned in Appendix D, Program Module Description, the data bank pre-
processor program is actually the GOAL language compiler program. Syntax
tables have been provided which direct the GOAL compiler in its action as
pre-processor. The current documentation relative to the compiler and its
syntax tables are to be found in Volumes I and II. The syntax tables
utilized for the release version have been entered as table number 8 in
the syntax table library. These syntax equations are indicated in symbolic
form on the pages following in this Appendix.
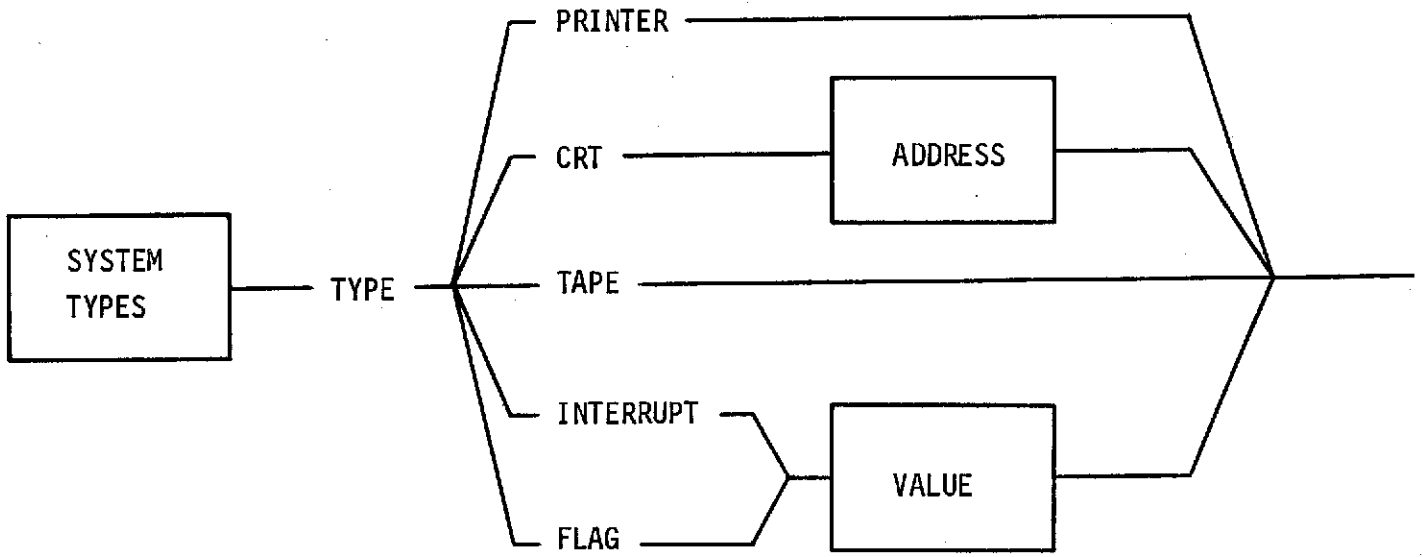
```
┌──────────┐          ┌────────────┐              ┌──────────────┐
│          │          │ FUNCTION   │    ── ALSO AS ── │  FUNCTION    │
│ SPECIFY  │── SPECIFY ─│ DESIGNATOR │─┐              │ DESIGNATOR   │─┐
│          │          │            │ │              └──────────────┘ │
└──────────┘          └────────────┘ │                               ╭
                                     └───────────────────────────────┘
```
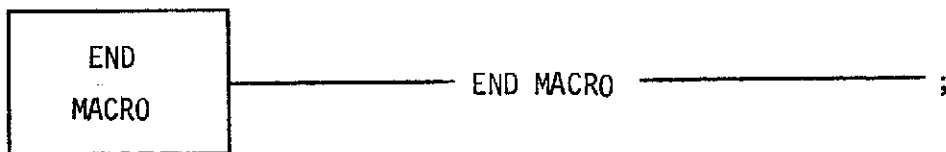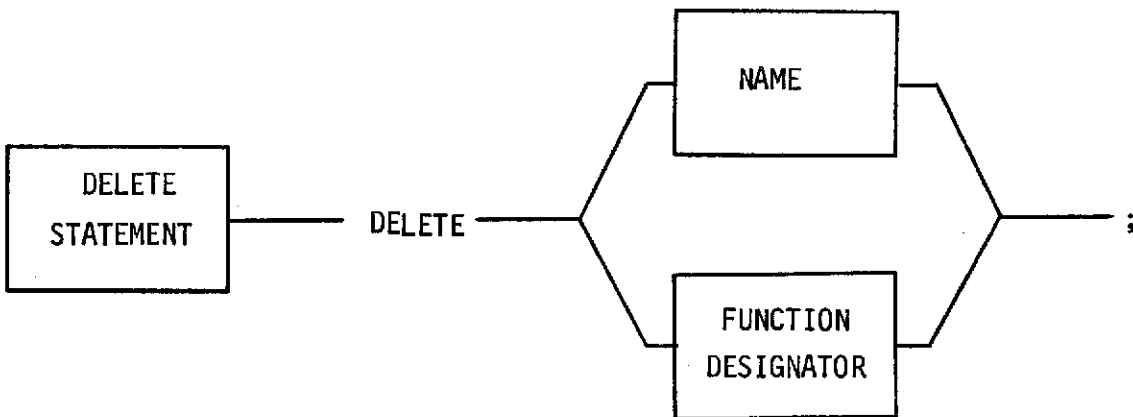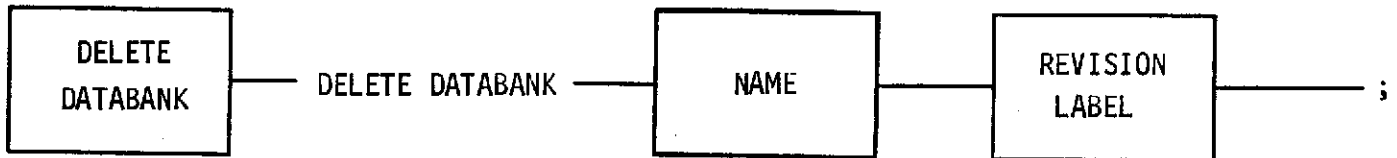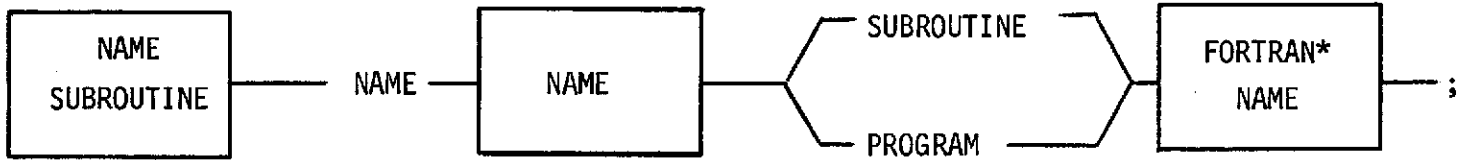
```
╮                              ┌──────────┐
╯── AS ──┬──── SENSOR ─────────│  DEVICE  │──────────────────────────── ;
         │                     │  TYPES   │
         │                     └──────────┘
         │                     ┌──────────┐
         ├──── LOAD ───────────│  DEVICE  │────────────────────
         │                     │  TYPES   │
         │                     └──────────┘
         │                     ┌──────────┐
         ├──── SYSTEM ─────────│  SYSTEM  │────────────────
         │                     │  TYPES   │
         │                     └──────────┘
         │                     ┌──────────┐
         └──── SUBROUTINE ─────│ FORTRAN* │───────────
                               │  NAME    │
                               └──────────┘
```

```
┌──────────┐                ┌─ DISCRETE ─┐        ┌──────────┐
│  DEVICE  │── TYPE ────────┤── ANALOG ──├────────│ ADDRESS  │──────────
│  TYPES   │                └─ CLOCK ────┘        └──────────┘
└──────────┘
```

*LIMITED FORTRAN NAME

I-3

*LIMITED FORTRAN NAME

I-4

```
┌─────────────┐
│   MACRO     │
│             │────────────────
│  SKELETON   │
└─────────────┘
```

o May not include
   words 'END MACRO'

o Must not have more
   than 40 cards without
   a semicolon

```
     1    8    0   -1
*** DATA BANK PREPROCESSOR SYNTAX TABLE -- SYNTAX TABLE #8
*
*
<DB PRODUCTION> = #5225 <DB STMT> #10 ;
<DB STMT> = <MACRO DEFN MODE> | <BEGIN STMT> | <END STMT> | <SPECIFY> |
            <DELETE STMT> | <SUBRTN NAME> | <FINIS> ;
*
    <MACRO DEFN MODE> = #56 <MDM01> ;
        <MDM01> = <MDM02> | #502 ;
        <MDM02> = 'END MACRO' #503 $999 ';' ;
*
    <BEGIN STMT> = 'BEGIN' $910 <B1> ;
        <B1> = <B2> | <B3> ;
          <B2> = 'MACRO' #501 ;
          <B3> = 'DATABANK' $804 <NAME> #5201 $927 <REVISION LABEL>
                 $999 ';' #5202 ;
*
    <END STMT> = 'END DATABANK' $999 ';' #5203 ;
*
    <SPECIFY> = 'SPECIFY' $808 <FUNCTION DESIGNATOR> #5204 <ALT FORM>?
                $904 'AS' $918 <SP1> $999 ';' #5205;
        <SP1> = <SP2> | <SP3> | <SUBRTN> ;
          <SP2> = <SP4> $939 'TYPE' $911 <SP5> $903 'ADDRESS'
                  $800 <SP7> ;
            <SP7> = <ADDRESS> #5217 ;
            <SP4> = <LOAD> | <SENSOR> ;
                <LOAD> = 'LOAD' #5206 ;
                <SENSOR> = 'SENSOR' #5207 ;
            <SP5> = <DISCRETE> | <ANALOG> | <CLOCK> ;
                <DISCRETE> = 'DISCRETE' #5209 ;
                <ANALOG> = 'ANALOG' #5210 ;
                <CLOCK> = 'CLOCK' #5211 ;
          <SP3> = 'SYSTEM' #5208 $939 'TYPE' $909 <SP6> ;
            <SP6> = <PRINTER> | <TAPE> | <CRT> |
                    <INTERRUPT> | <FLAG> ;
                <PRINTER> = 'PRINTER' #5212 ;
                <TAPE> = 'TAPE' #5214 ;
                <CRT> = 'CRT' #5213 $903 'ADDRESS' $800 <SP7> ;
                <INTERRUPT> = 'INTERRUPT' $131 <SP9> ;
                <SP9> = #23 #5222 ;
                <FLAG> = 'FLAG' $131 <SP8> ;
                <SP8> = #23 #5223 ;
          <SUBRTN> = 'SUBROUTINE' $847 <FORTNM> #5218 ;
*
    <DELETE STMT> = <DELETEDB> | <DELETE> ;
        <DELETEDB> = 'DELETE DATABANK' $804 <NAME> #5201 $927
                     <REVISION LABEL> $999 ';' #5220 ;
        <DELETE> = 'DELETE' $829 <FD OR NAME> $999 ';' #5221 ;
```

```
*
        <SUBRTN NAME> = 'NAME' $844 <NAME> #5201 $952 <SN1>
                        $847 <FORTNM> #5218 $999 ';' #5224 ;
              <SN1> = 'SUBROUTINE' | 'PROGRAM' ;
*
*
*
                <ALT FORM> = 'ALSO' $904 <AF1> ;
                <AF1> = 'AS' $808 <FUNCTION DESIGNATOR> #5216 ;
                <ADDRESS> = #23 ;
                <REVISION LABEL> = 'REVISION' $836 #27 #5219 ;
                 <FD OR NAME> =  <FD> | <NM> ;
                    <FD> = <FUNCTION DESIGNATOR> #5204 ;
                    <NM> = <NAME> #5201 ;
                 <FORTNM> = #29 #5226 ;
                <NAME> = #29 ;
                <FUNCTION DESIGNATOR> = #20 ;
*
*
     <FINIS> = 'FINIS' #5215 ;
*
*
* #5201 - MOVE DB NAME TO OUTPUT BUFFER
* #5202 - WRITE FIXED FORM 'DATABANK' RECORD
* #5203 - WRITE 'END DATABANK' STMT
* #5204 - MOVE FUNCTION DESIGNATOR TO OUTPUT BUFFER
* #5205 - WRITE FIXED FORM 'SPECIFY' RECORD
* #5206 - #5214  MOVE FD TYPES INTO OUTPUT BUFFER
* #5215 - SET ENDFLG = 1 AND RETURN
* #5216 - SUPPORT ALTERNATE FORM FD - OUTPUT FIRST LINE
* #5217 - MOVE ADDRESS INTO OUTPUT AREA
* #5218 - MOVE SUBROUTINE NAME INTO OUTPUT AREA
* #5219 - MOVE REVISION LABEL TO OUTPUT AREA
* #5220 - WRITE FIXED FORM 'DELETEDB' RECORD
* #5221 - WRITE FIXED FORM 'DELETE' RECORD
* #5222 - MOVE 'INTERRUPT' AND VALUE TO OUTPUT AREA
* #5223 - MOVE 'FLAG' AND VALUE TO OUTPUT AREA
* #5224 - OUTPUT 'NAME SUBROUTINE' RECORD
* #5225 - SET PREFLG = 1
* #5226 - LIMIT FORTRAN SUB NAME TO SIX CHARACTERS
*
*
        END ;
```