

IBM Confidential

TR 00.2209  
July 19, 1971

## Preparing Method of Operation Diagrams -- The HIPO Method

W. Thomas Wolfe

**IBM**

# Technical Report

Systems  
Development  
Division  
Poughkeepsie Lab

July 19, 1971

TR 00.2209

PREPARING METHOD OF OPERATION  
DIAGRAMS -- THE HIPO METHOD

by

W. Thomas Wolfe

ABSTRACT

This report is for design programmers, developers, coders, and programming technical writers. The focus is on producing method of operation diagrams that picture the internal functions of a programming system.

The main objective is to improve communication procedures and techniques through the effective use of functional operation diagrams.

The report describes how to produce operation diagrams that depict the internal functions of a program system or subsystem. Instructions are given on how to produce single diagrams or sets of diagrams that convey functional information efficiently.

A single method called HIPO (Hierarchy plus Input--Process--Output) is stressed for two reasons:

1. The method is a natural reflection of the development and implementation process and will be relatively easy to master.
2. HIPO is a common base, or graphic language, through which we can understand one another.

Some of the diagramming techniques presented herein will find natural applications in the daily traffic of technical correspondence among all IBM personnel.

**IBM Confidential**

This document contains information of a proprietary nature. ALL INFORMATION CONTAINED HEREIN SHALL BE KEPT IN CONFIDENCE. None of this information shall be divulged to persons other than: IBM employees authorized by the nature of their duties to receive such information, or individuals or organizations authorized by the Systems Development Division in accordance with existing policy regarding release of company information.

**IBM**  
poughkeepsie lab  
Systems Development Division

## TABLE OF CONTENTS

	<u>Page</u>
INTRODUCTION . . . . .	1
WHAT HIPO IS . . . . .	1
THE GROUP PLAN . . . . .	1
VISUAL TABLES OF CONTENTS . . . . .	1
THE FUNCTIONAL PACKAGE . . . . .	2
THE VALUE OF HIPO . . . . .	4
LARGE SYSTEM DOCUMENTATION . . . . .	4
FUNCTIONAL DOCUMENTATION . . . . .	4
EDUCATION . . . . .	5
OTHER FEATURES . . . . .	7
DOING HIPO DIAGRAMS -- THE STEPS . . . . .	8
DEFINING THE OBJECTIVES OF THE DIAGRAMS . . . . .	8
PLANNING THE STRUCTURE OF THE DIAGRAMS . . . . .	18
EXECUTE . . . . .	22
ACKNOWLEDGEMENT . . . . .	29
APPENDIX A: OPERATION DIAGRAMS -- DIALOG . . . . .	31
APPENDIX B: A GROUP PLAN . . . . .	40
APPENDIX C: A HIPO PACKAGE . . . . .	43
APPENDIX D: OTHER PROGRAMMER-GENERATED HIPO DIAGRAMS . . . . .	53
APPENDIX E: WRITER-GENERATED HIPO DIAGRAMS . . . . .	58

## INTRODUCTION

Since the beginning of "software", the programming community has thought of software products as being largely invisible, and having no recordable, functional structures. We have learned to think and to document in terms of the code only, rather than of the functions we support with the code.

So when the system is subsequently modified, we relegate ourselves to working in the dark. At each release we spend much time working with the pieces to get an idea of the whole. And we pay the price in terms of hidden interfaces, integration problems, and slipped schedules.

It does not have to be this way. Functional program structures can be graphically recorded in much the same way as building structures are recorded in blueprints.

## WHAT HIPO IS

HIPO, Hierarchy plus Input--Process--Output, is a method of graphically describing internal function by structuring a presentation from general to detailed levels in a set of method of operation diagrams. A HIPO presentation consists of:

- A group plan.
- Any number of functional packages (determined by the group plan).
- Visual tables of contents to each of the functional packages or, later in the effort, a single table of contents to the entire presentation.

## THE GROUP PLAN

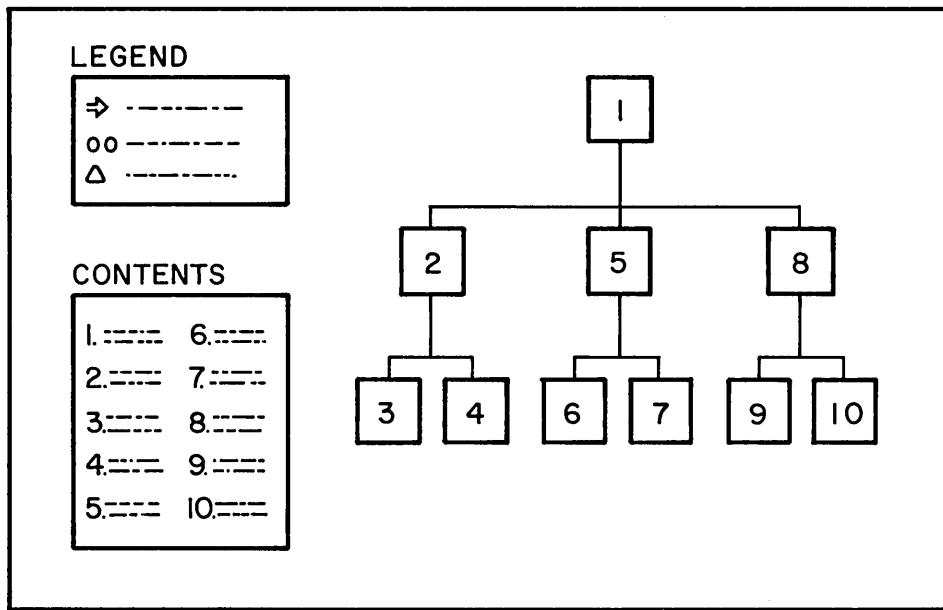
Initial work is done at the project planning stage to develop a plan that specifies the major functional breakouts for the project. It is important to do a group plan before beginning the individual packages to help prevent subsequent duplication of effort on the parts of the programmers or writers doing the packages.

An example of a group plan calling for five functional packages is shown in Appendix B.

## VISUAL TABLES OF CONTENTS

A visual table of contents is prepared for each of the packages. It shows:

- The structural relationships of the diagrams within the package.
- The contents of each of the diagrams.
- A legend applying both to the individual package and the total presentation.



Visual tables of contents appear as the first diagrams in Appendixes C and D.

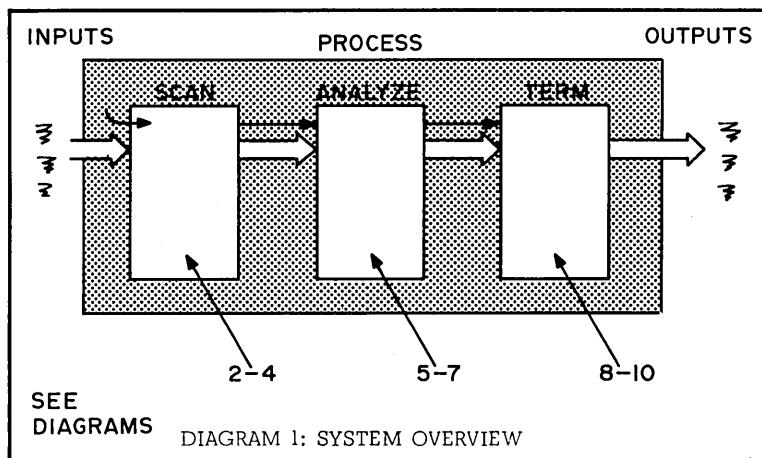
Note: After all packages are firm, their individual tables of contents can be replaced by a single, comprehensive visual table of contents if the size of the total presentation is small enough that a single contents page is practical.

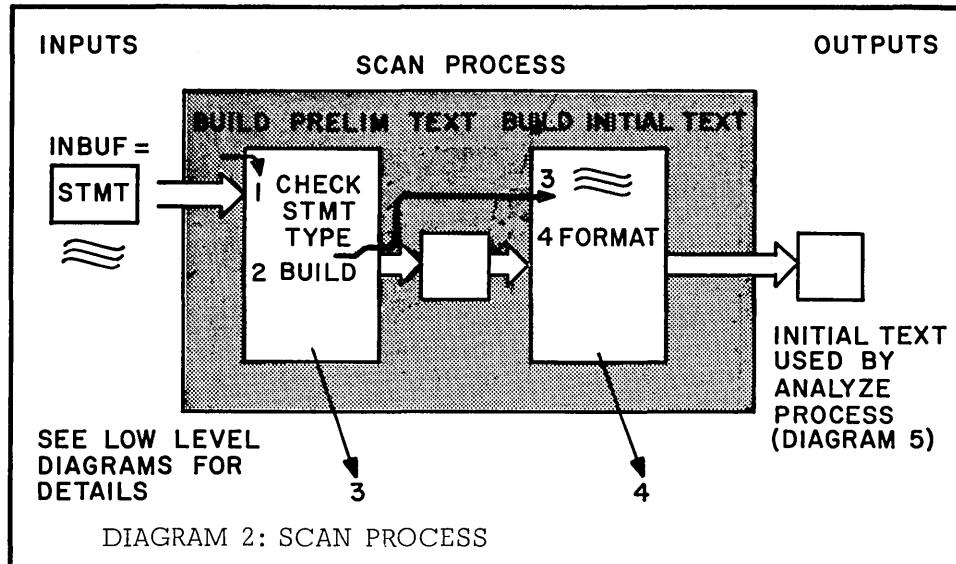
#### A FUNCTIONAL PACKAGE

Each of the packages contains a visual table of contents, one or more overviews, and a number of low level diagrams showing the implementation and/or design of a function.

The number of levels in a package is determined by the number of "functional subassemblies", the complexity of the material, and the amount of information to be documented. Although there are no theoretical limits to the number of levels of detail, a practical limit is five levels, including the overview. Beyond this, the package will become difficult to refer to for information. Usually a package will result in three or, infrequently, four levels of diagrams.

The overview, or upper level, diagrams act as introductions to the functions and directors to the low level, detailed diagrams.

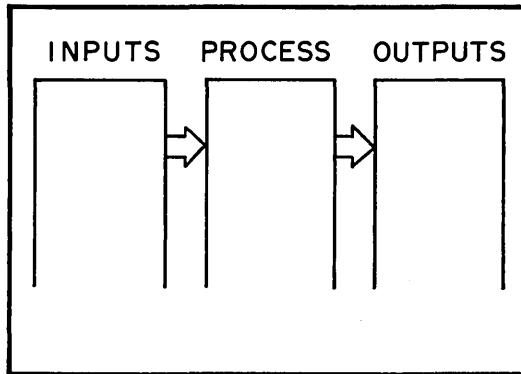




The low level diagrams contain "unit level" information (that level of information reflecting the actual workings of the system). Each low level diagram is arranged to best show:

- A process that supports the function being described.
- Results of the process.
- Requirements for processing.

Stated graphically:



The information in a low level diagram changes from time to time in the development cycle; that is, during early design phases, the unit level deals with basic design plans. During the implementation phase, this information consists of basic implementation points (how the design points have been implemented). This is discussed in more detail under "Define the Objectives of the Diagrams."

A typical functional package appears in Appendix C. Examples of low level diagrams are included in Appendixes C, D, and E. (Appendixes C and D show diagrams done in the design

phase; Appendix E shows diagrams that reflect functional points and the implementation of those points.)

### THE VALUE OF HIPO

In discussing the value of HIPO, it is necessary to discuss a few of our major communication problems:

- We document large systems as though they were small ones.
- We don't document function soon enough.
- We spend inordinate amounts of time on informal education during the development process.

### LARGE SYSTEM DOCUMENTATION

As systems get larger, a time comes when restructuring becomes necessary. A point is reached where systems, "done" in the old way, become inefficient because of their very sizes or degrees of complexity.

Methods of communication are not exceptions to this rule; as the amounts of information increase, systems must be described in terms of more comprehensive subsystems and higher level concepts so that the users of those systems can be better equipped to get to the points of their problems quickly.

Unfortunately, the descriptions of our programming systems have not advanced with increased system complexities, sizes, and technological breakthroughs in the software area. Our documentation methods are sometimes tantamount to describing computer hardware solely in terms of electronic circuit components.

This is not to say that unit level information is unnecessary. After all, it is the unit level information that reflects the true workings of the system. And it is this information that we must, in the end, use to solve our problems. But we have cast ourselves in the roles of low-level part makers, each describing how his part makes the total system go.

HIPO eases this problem in the area of internal function by providing a hierachial framework under which the higher level functions can be pictured. Relationships similar to the country--state--city map structures are established to help the user get quickly to the point of his problem.

### FUNCTIONAL DOCUMENTATION

In the development effort, we neglect program logic throughout the late stages of the cycle

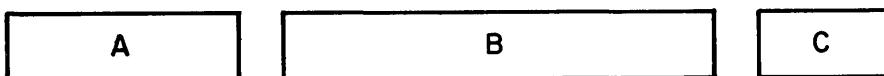
and instead concern ourselves primarily with producing massive amounts of code-oriented information with little or no good pictures of how things really tie together.

At the end of the cycle, an attempt is made to "rediscover" function by analyzing listings and reading various specification material that was generated earlier in the cycle. But this process is very difficult to do because:

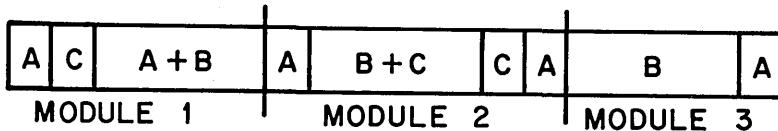
- During implementation, functions become intermixed with other functional structures in the code.

Three major functions -- before implementation.

#### THREE MAJOR FUNCTIONS-BEFORE IMPLEMENTATION.



#### AFTER IMPLEMENTATION.



- Furthermore, as a project moves through the development cycle, our thinking becomes implementation-oriented, and less oriented toward the logic behind what is being done. Thus, late in the cycle, we speak more of routine organizations and the code-oriented jobs we are doing rather than of logic.

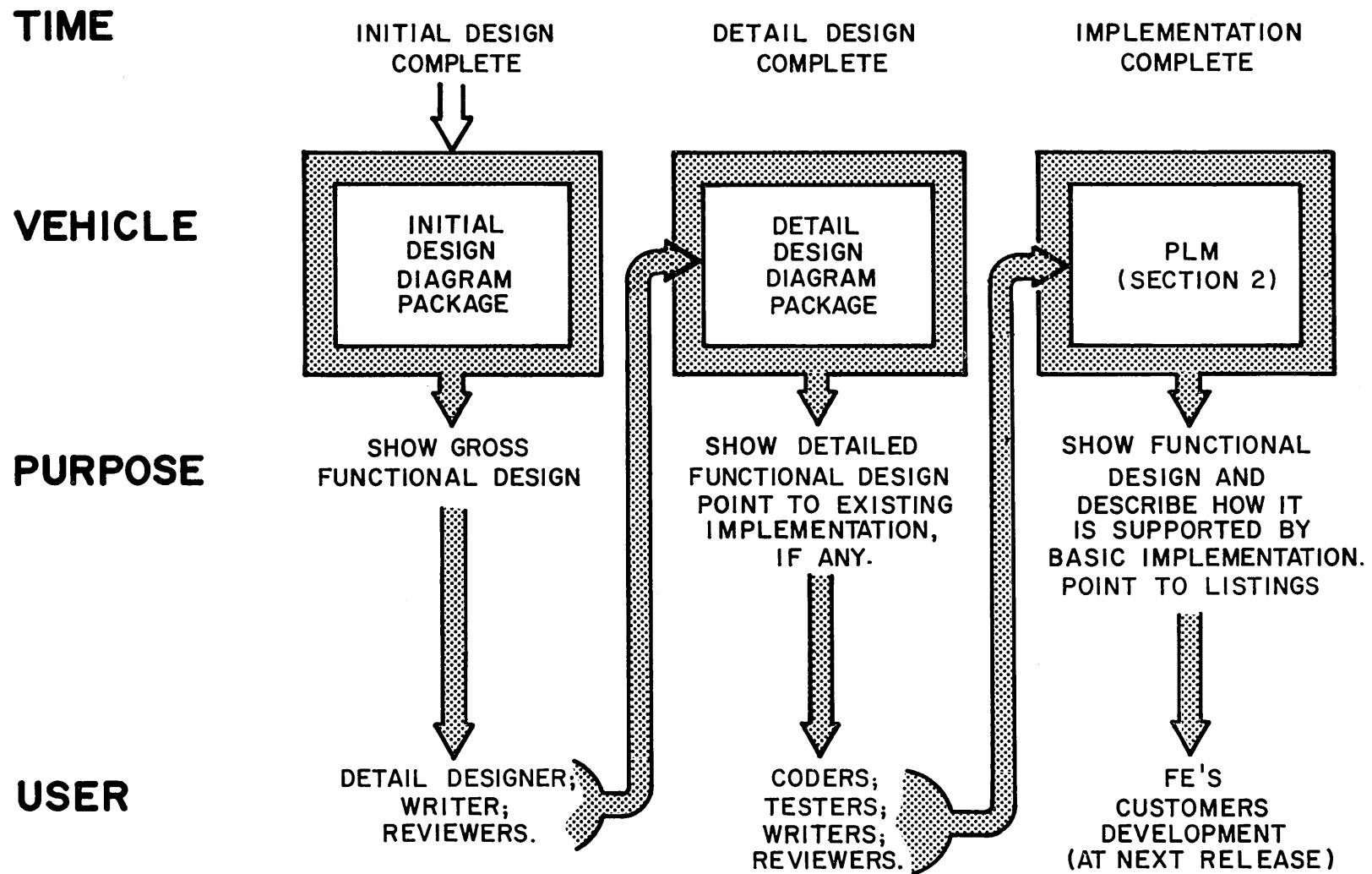
So the task of rediscovering function late in the cycle (or at the beginning of the next release) is made very difficult.

The HIPO approach solves the problem of vanishing functional logic by providing a framework for establishing and documenting these functional structures early in the cycle and then developing the documentation concurrent with the design and implementation of the function. As an added benefit, HIPO will act as a tool to assist the designer/developer in thinking his way through the project and in uncovering logic errors early in the design stage.

The following diagram shows how, ideally, basic functional information can be produced early and then developed through the cycle.

#### EDUCATION

As we move from release to release, the system base is altered to accommodate new or changed functions. In modifying the base, the delta (design change) programmers have PLM's, listings, and (sometimes) base module owners available for the education process.



The PLMs are not currently suitable for helping the programmers come to a quick understanding of the base. And the module owners, if available, have problems of their own. So this leaves the listings, which are at a low level of detail.

In using the listings, the delta programmer is forced to work with large amounts of detailed information in order to reconstruct the base logic to the point where he can understand all of the areas that will be affected by his design. Only after the programmer is familiar with the base can he begin serious work on the new functional design.

This problem is compounded by the fact that in a given release there can very well be more than one function affecting the same base. Furthermore, the designers of the different functions might be geographically separated.

HIPPO can ease some of the education and communication problems by providing a framework for a common, visible base in the form of a structured set of diagrams. Education time will be lessened because the programmer can familiarize himself first at the general levels and then at the lower levels, as applicable.

Integration problems and adverse impact with other programming groups will be easier to check earlier because the different groups can mark their delta changes on copies of the same set of base diagrams.

#### OTHER FEATURES

Making the programming functional structures visible should cause marked improvements in many areas. Look for:

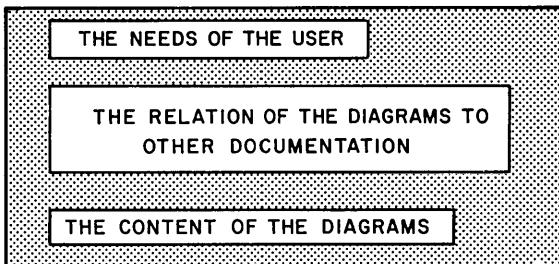
Better control over the design, development, and implementation of a system:

1. In clearer definitions of the points of functional interfaces, resulting in better functional monitoring capabilities.
2. In smoother information transfers and less duplication of effort throughout the development cycle.
3. In better definition of design versus implementation tasks. In apportionment of assignments at the group level.
4. In functional testing.
5. In compressed cycles at n+1 (next release) time. Quicker retrievability of information and faster fixes in the laboratory maintenance environment and in the field.
6. In lessening the impact of the loss of a lead programmer.

## DOING HIPO DIAGRAMS--THE STEPS

1. Define the objectives of the diagrams.
2. Do the group and package plans.
3. Execute the individual packages of diagrams.

### DEFINING THE OBJECTIVES OF THE DIAGRAMS



The ultimate worth of the HIPO presentation will far exceed the time and costs incurred in preparing the diagrams if time is first taken to define and understand what information should be included in the package. Making these definitions will eliminate many problems that would otherwise be experienced in producing a successful set of diagrams.

#### The Needs of the Users

The diagrams should be developed in parallel with the design and development effort, as discussed under "Functional Documentation" in the preceding section. Users will differ throughout the cycle; however, diagrams can be produced to serve the common needs of a great many users.

Here are some of the uses to which diagrams might be put:

EDUCATIONAL AID--Development laboratory and field personnel will use the diagrams to familiarize themselves with internal function.

DESIGN AND DEVELOPMENT AID--You, as the designer, developer, or writer become a user of your own diagrams. The act of doing the diagrams assists your thought processes. In addition, you will be able to quickly retrieve information that you have recorded in the diagrams.

MONITORING AID--The diagrams will provide an excellent facility for keeping track of the design and implementation of function.

MANAGEMENT AID--The diagrams might prove useful in making more accurate estimates of the amount of work involved in a project and in a

better distribution of resources.

TESTING AND INTEGRATION AID--The diagrams will prove useful both to product test and to in-house test groups in determining what functions need to be tested. The diagrams will prove quite valuable for the integration process.

DIAGNOSTIC AID--The diagrams will be used to help diagnose problems and hasten fixes, both in the laboratory maintenance environment and in the field.

OTHER--Other uses will be found for the diagrams. For example, the Software Monitoring Technology group in Poughkeepsie feels that the diagrams might prove valuable for projecting performance estimates earlier in the cycle, tracking performance as the product is developed, and aiding in the area of developing functional models.

#### The Relation of the Diagrams to Other Documentation

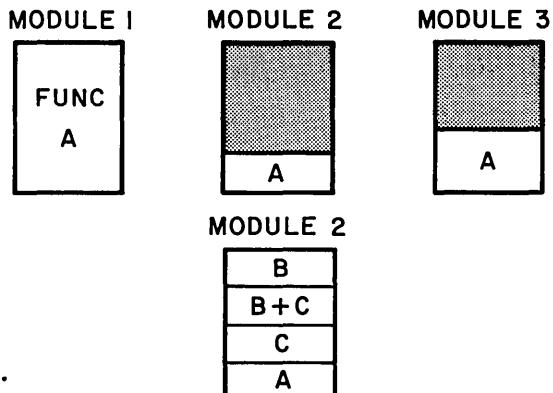
Unless the creator of a diagram package is fully aware of other types of information provided outside of the package, he will probably include items from these areas within his diagrams. This is wrong for the following reasons:

- The diagrams will not work the same way that diagrams prepared by someone else do because each person will have a different idea of the informational needs of the diagrams.
- The diagrams will be subject to change from each of the non-functional areas that are included.
- Duplication of effort will result and redundant documentation will be generated.
- Users will have trouble locating information quickly because they will never be sure of where to look for specific kinds of information.

Functional Versus Program Organization Material: Under current technologies, the single most important relationship is that between functional material and program organization material. The differences are touched on here; a full explanation is provided in the publication PLM Guidelines manual, Form Z28-6673-1.

There is usually a certain amount of confusion as to what program organization and method of operation are, and how they relate to one another. The main reason that the distinction, and the resulting separation of material, is made is because in a large system:

- The implementation of a major function can extend through many modules or routines.
- A specific module might support parts of many functions. Sometimes these functions are not even related.



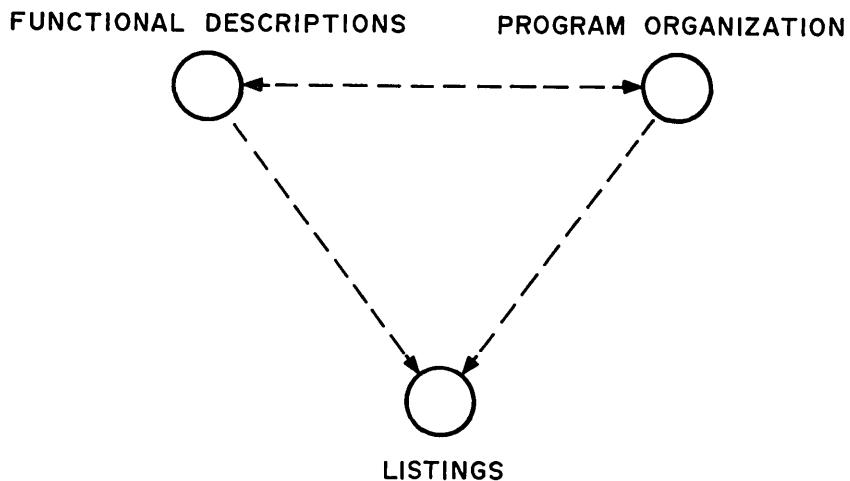
Someone who must modify an existing function, add a new function, test for regression, or fix the system must know:

1. How the function is performed.
2. What other, perhaps non-related, functions use the same code.

1, above, is one of the basic objectives of method of operation diagrams--to show functional logic and to show how the activation of function ties in with the code.

2 is one of the objectives of program organization--to show the code structures pertaining to the combined functions.

To tie together the program organization and functional areas, PLMs use cross references from one area to another and to the listings.



#### Content of the Diagrams

Content can be discussed in terms of the information to be conveyed (informational content) and the graphic means used to express the subject (graphic content).

Graphic Content: The graphic content is determined by the situations to be depicted. Some

typical situations and suggested graphic applications are:

**STARTING POINT** - A heavy black arrow can be used to lead the reader into the diagram at the proper point.

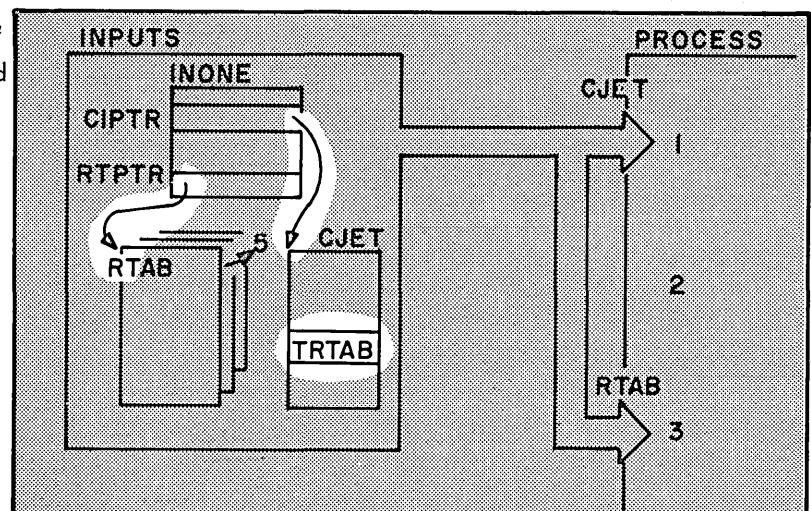
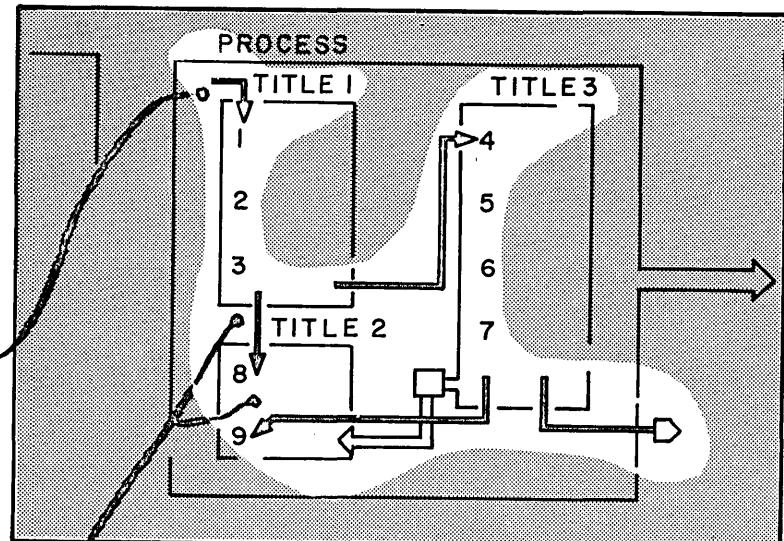
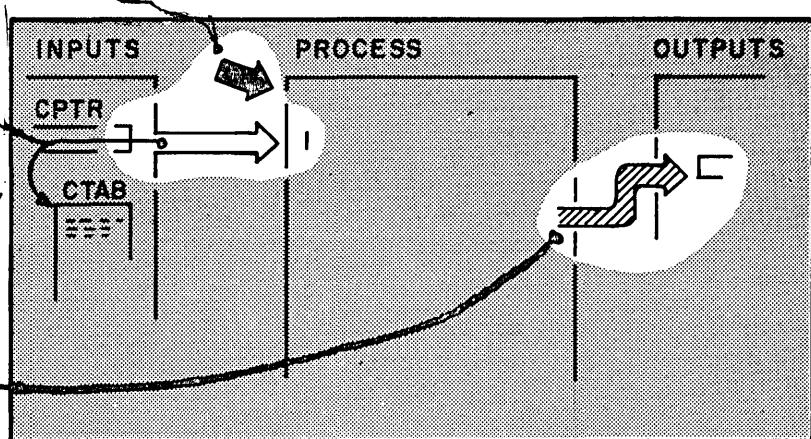
**DATA MOVE** - An open arrow shows movement of data into a temporary work area or initialization of a data area. The open arrow is also used to show the availability of input to a processing step.

**DATA ALTER** - A hatched arrow shows movement of data into an area that has previously been initialized or otherwise used.

**FLOW OF ACTION** - Black arrows used with numbered processing steps show flow of action through the process.

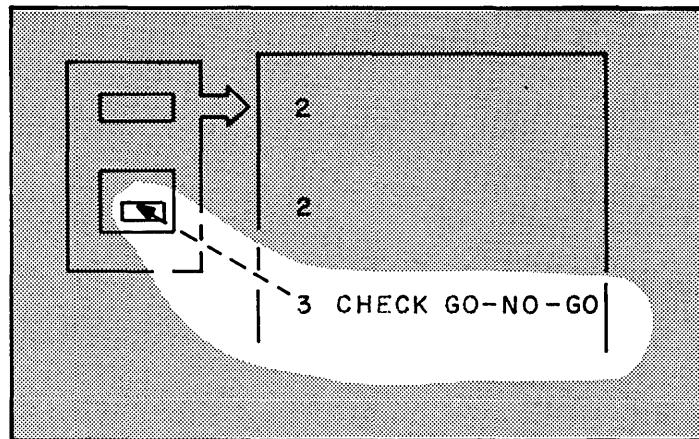
**SUBPROCESSES** - Boxes and titles are used to group related processing steps. Shading is optional. (See Appendix E for examples of titled subprocess blocks.)

**DATA AREAS: PARAMETER LISTS**  
Boxes are used to represent data areas and parameter lists. Light arrows are used to show pointer arrangements, where applicable.

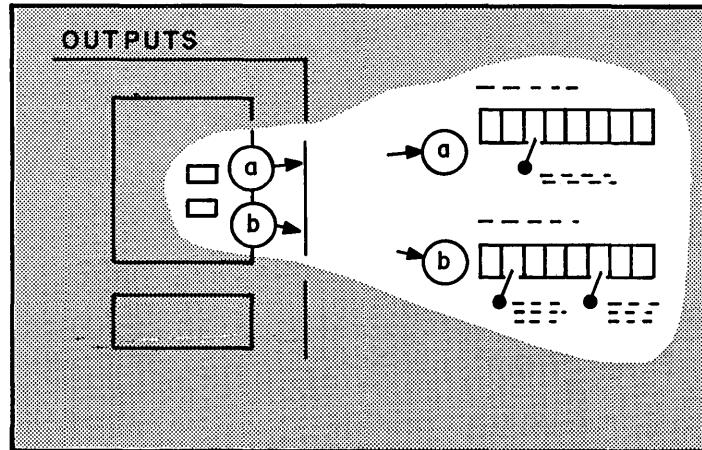


**REFERENCE INDICATOR -**

A dotted arrow means  
"--refer to this item."



**BLOW-UPS** - Blow-ups can be used to expand on areas shown in the context of larger data areas. (See any of the low-level diagrams in Appendix C for examples of blow-ups.)

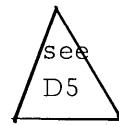
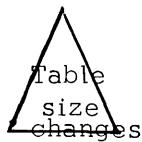


**KEYS** - Keys establish relationships; identify blow-ups; lead to other diagrams.

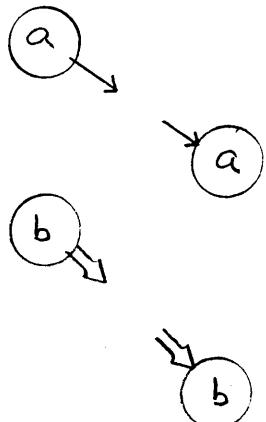


Delta symbol is used to indicate a change to a base diagram. The symbol should either identify a corresponding delta diagram that shows the actual change or it should indicate the nature of the change.

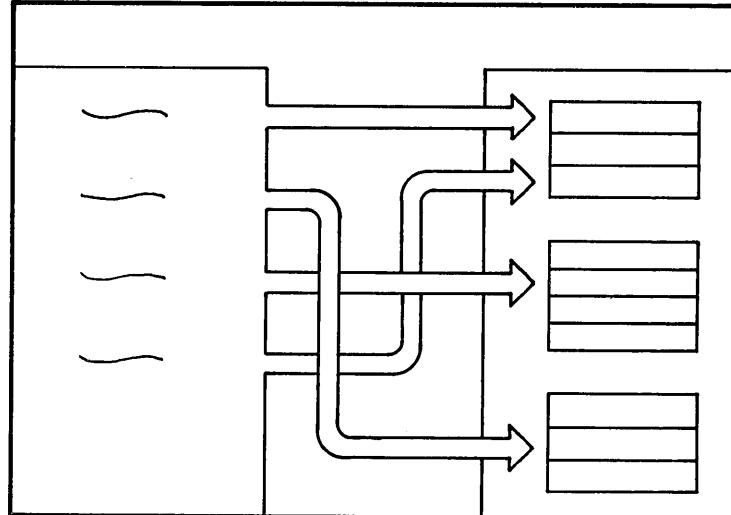
Examples:



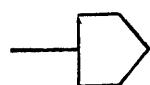
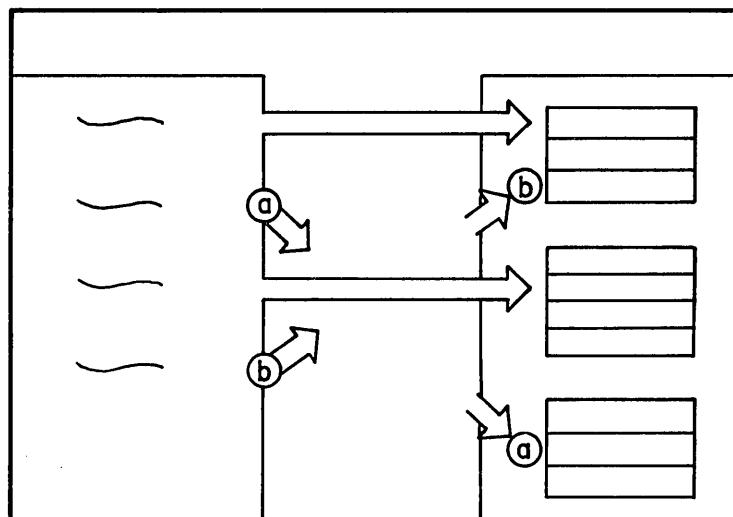
On page connectors can be used a directional keys. The connectors should be used with appropriate arrow types to show data movement, modification, or flow of action. Note that the connectors should



always point in one direction (from--to), never toward one another. The directional key is useful wherever crossing lines would otherwise have to be used. For example, instead of this:



do this:



Off page connectors are used to lead to related diagrams, as in flowcharts.

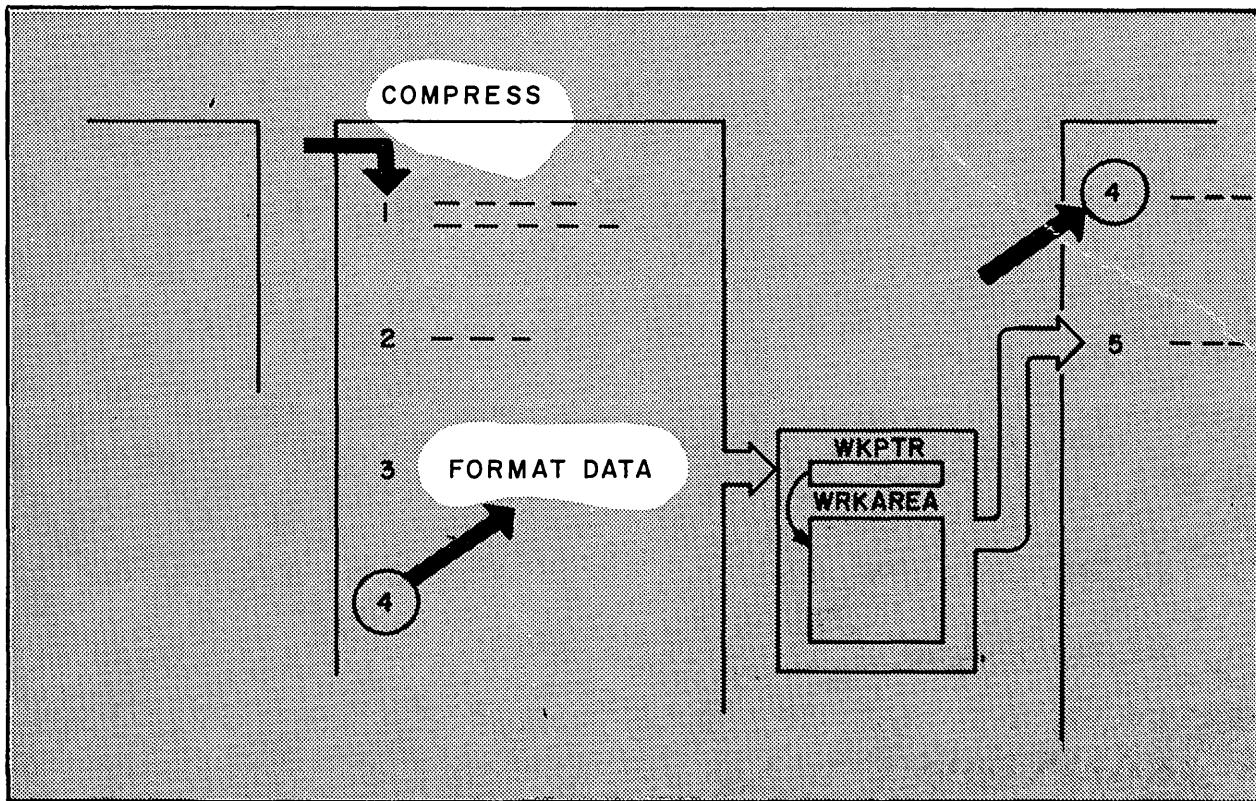
Informational Content: Functional diagrams should discuss inputs, process, and results. For the sake of clarity, these elements should be presented in a normal reading sequence. Simple boxes can be used to block off these three major areas of the diagram.

The "picture area" of the diagrams should contain as few words as possible. There are two reasons for this:

- When the picture becomes cluttered with text, it will lose some value as a recall mechanism.
- The degree of difficulty of maintaining the diagrams increases with increased number of words in the picture area. (Also, it is more time consuming and costly to automate diagrams having many words in the picture area.)

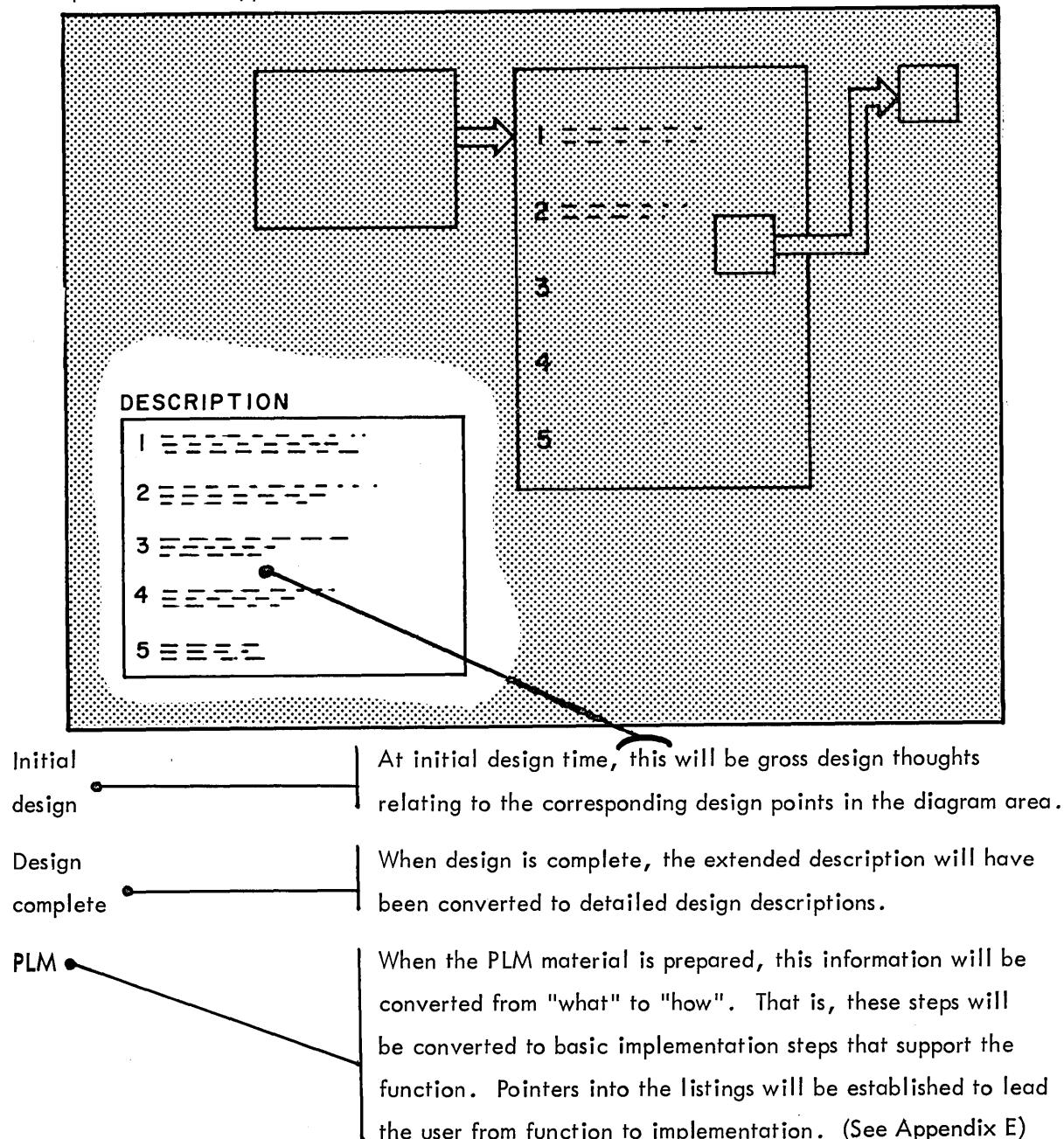
So expand upon the basic points by including extended descriptions away from the primary picture area. Perhaps you might even want to include the descriptions on a separate piece of paper.

The following sample diagram shows how to simplify the picture area by condensing what has to be stated into a few useful words. This is similar to the newspaper technique of summarizing the paragraph before presenting the details.



Step 3, in the previous diagram, is one of the steps showing what has to be done to support the overall function (COMPRESS). The brief caption, "Format data", can be compared to a paragraph title.

The content of the extended description (not shown in the above diagram) that supports step 3 depends on when in the development cycle the diagrams are to be used. The extended descriptions and the type of information to be included are shown in the following diagram.



In the PLMs, functions will be related by cross-references to the module(s) supporting those functions. The addition of cross-references creates a "functional mapping" through the implementation.

DESCRIPTION	ROUT.	LABEL	F.C.
1 ----- ----- -----.	ROUT 1	START	AA
2 ----- ----- -----.		BUILD	AB
3 ----- ----- -----.	ROUT 2	INHRE	BA
4 ----- ----- -----.		GETM	

Implementation steps 1 and 2 are performed in ROUT1. Step 1 can be picked up at key label START in the listings (also found on flowchart AA).

Step 2 can be picked up beginning with key label BUILD (found on flowchart AB).

Prior to step 3, a passing of control is implied by the horizontal line through the routine column. Steps 3 and 4 are performed in routine ROUT2.

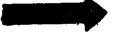
To summarize graphic and informational content:

#### Graphic content

##### Boxes used to

- Localize inputs, process, outputs.
- Represent data areas, tables, fields, etc.
- Isolate related sub-processes within the processing area (reinforced by shading).
- localize extended descriptions.

##### Arrows used to

- Get the reader into the diagram. 
- Lead the reader through the process. 
- Act as pointers. 
- Call the reader's attention to clarifications, blow ups, before-after drawings, etc. 
- Show data flow. 

## Other Symbols



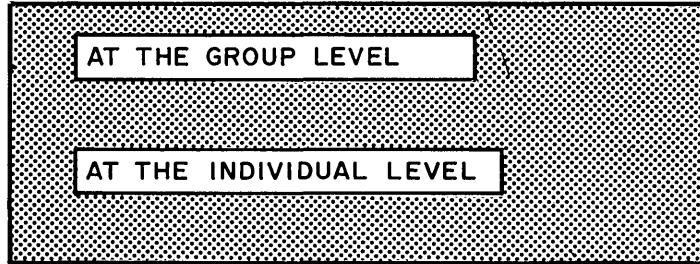
- Used as keys to connect secondary descriptions, explanations, related material, blow ups, etc.
- Used to identify points on master diagrams where changes are applicable.
- Used in conjunction with any of the arrow line weights where crossed lines would be confusing.

## Informational Content

Time	Picture Area	Extended Description
Initial Design Complete	Gross design points; general data flow; gross requirements and results.	Design thoughts; whys.
Detailed Design Complete	More specific design points, data area involvement, requirements and results.	Design plans (more specific information on <u>what</u> should be done); cross-references leading to affected base implementation, if any.
Implementation Complete	Specific design points, data area involvement, requirements and results.	Basic implementation (that is, information on <u>how</u> the design plans have been implemented); cross-references (in the form of routine names, key labels, and flowchart IDS) to the listings, flowcharts, and routine descriptions.

## PLANNING THE STRUCTURE OF THE DIAGRAMS

PLAN



After all concerned have a good understanding of the purpose and content of functional diagrams, work can begin on the group and package plans for the project.

Plan at the Group Level

A plan should be prepared at the group level to determine what functional packages will be needed. The individual programmers, using the plan as a point of departure, can then plan and execute the specified packages of diagrams.

The group plan should be represented in the same form as a visual table of contents. It should specify the individual packages, describe the legend to be used, and note any other conditions, such as responsible parties, peculiar to the project. In some cases, a very high level diagram can be prepared to show how the packages relate to one another. (See Appendix B for an example of a group plan.)

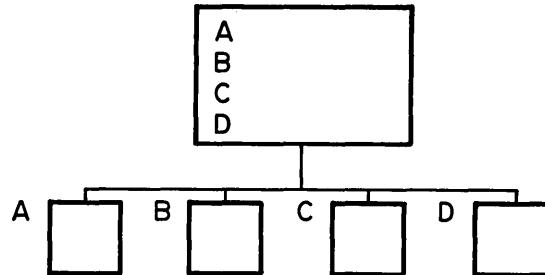
## Who will use the plan?

- Management -- to establish checkpoints for examining and controlling workloads and resultant development.
- The designer/developer -- as a departure point from which he will prepare the functional packages.
- The writer -- to ensure that all areas are adequately documented at release time.

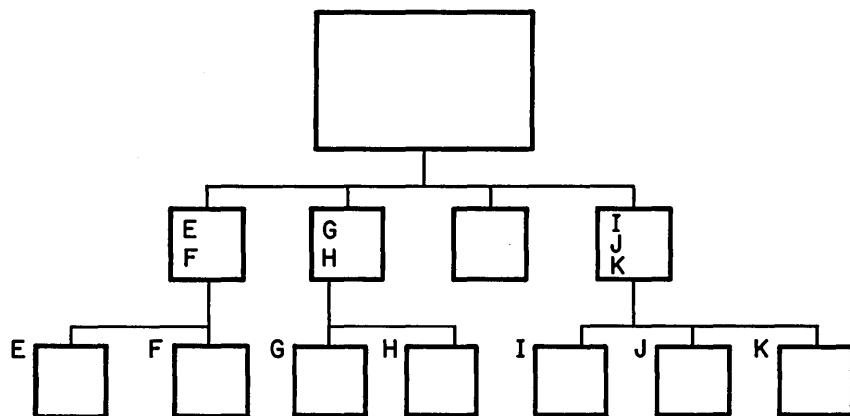
Plan at the Individual Level

Each of the departure points documented in the group plan will become an overview diagram for an area of functional responsibility. Before the overview diagram is prepared, however, the package should be analyzed for logical breakouts into smaller, more controllable areas. This will result in a top down plan (and visual table of contents) for the packages. (See the first diagram in Appendix C for an example of a top down plan.)

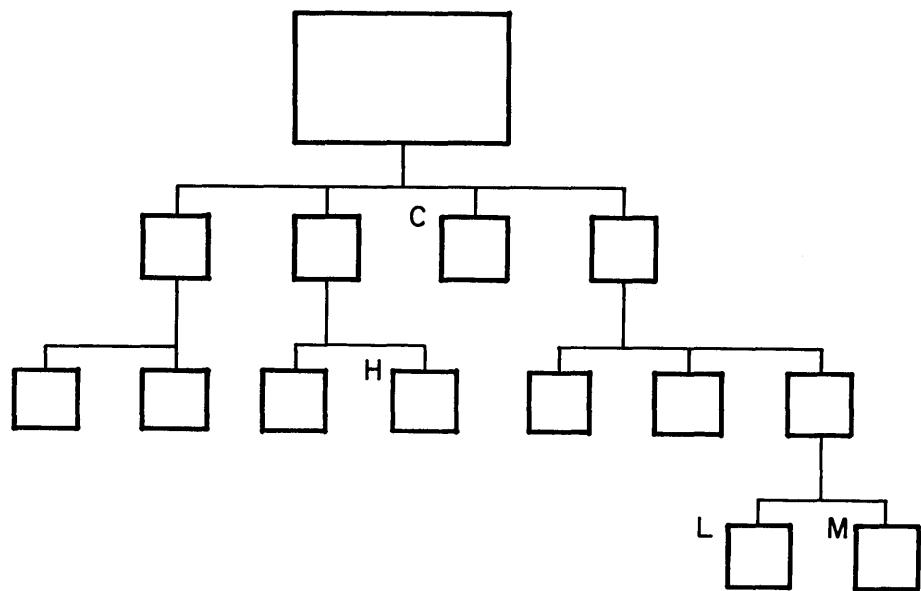
In preparing the top down plan, determine the major areas that will have to be covered. Represent these areas individually as the next level of diagrams under your overview.



Then analyze each of these major areas, in turn, to determine the necessary scope of coverage at lower levels.



Continue this process until the total scope of the package has been determined.



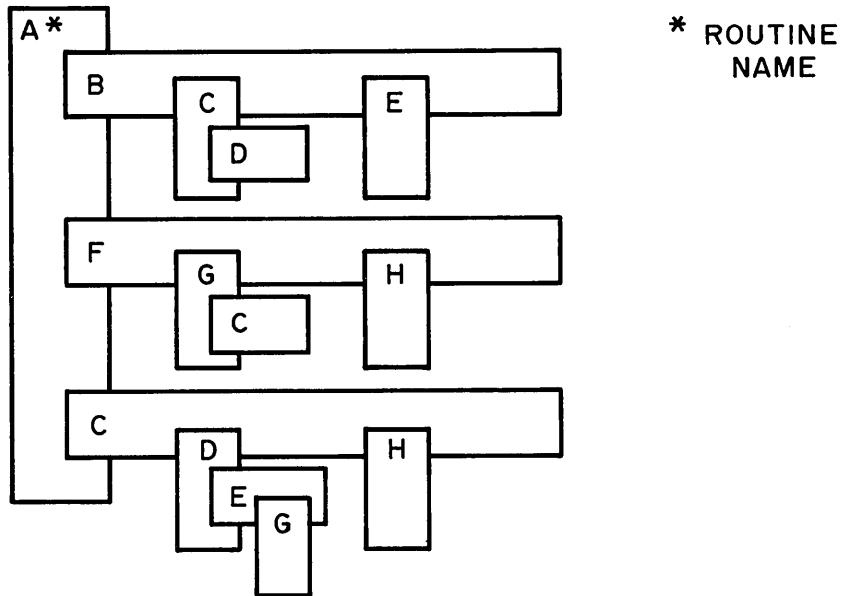
Note in the previous diagram (C, H, L and M) that you need not carry all breakouts to the same level. The breakouts are determined, to a great degree, by the complexity and amount of material to be covered.

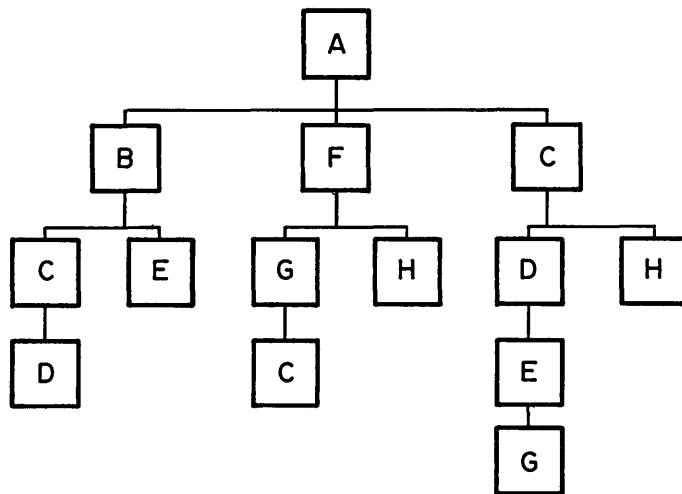
Planning from top to bottom will provide the following benefits:

- You will think through the requirements of your project systematically. This is similar to preparing a topical outline for a presentation or book. Doing the top down plan will help you define the true size of the project.
- You will be better able to estimate the amount of necessary coding because you will have available a diagram plan that parallels the functions to be supported. Your estimates will be more accurate because you will be working to small, controllable scopes.
- When you begin execution of the package, you will have fewer false starts. The diagram package will be less subject to change because of poor organization.
- You will not inadvertently "inflate" the relative importance of given functions.
- Points of functional interface will be preserved for the next release. At that time, you will not have to rediscover the same structures you are now developing.

Special note: If you are attempting to diagram function that has already been coded, the problems in planning the top down package will be compounded. In these cases it will help to do the following:

1. First draw a program organization hierarchy chart (or refer to one if one exists). Show the possible calling sequences of the involved routines. This can be done conveniently in either of the two following ways:



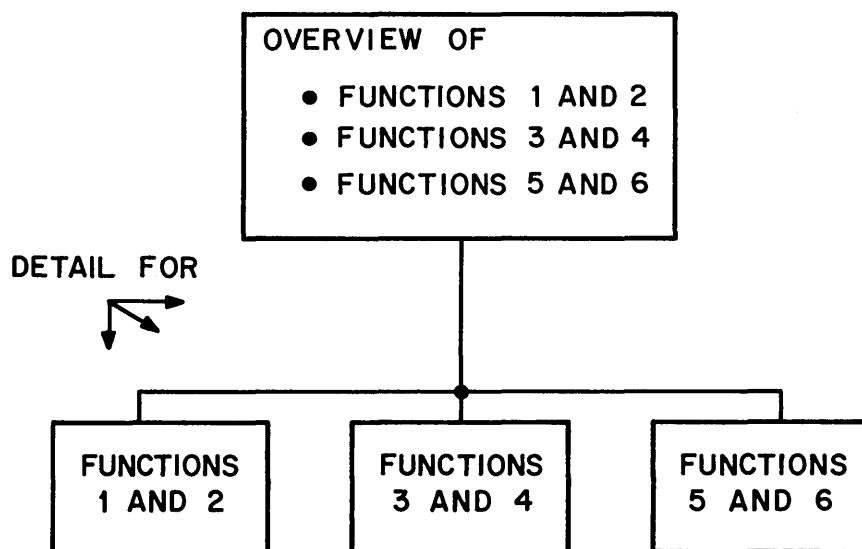


Each of the legs probably reflects the implementation of a function or series of functions.

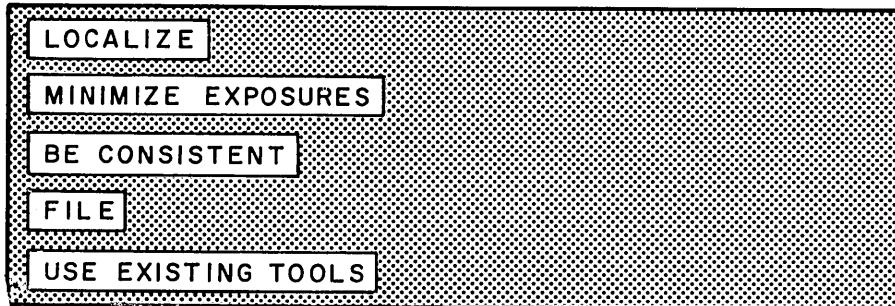
For example:

Sequence through routines	ABCD	= Function 1.
	ABE	= Function 2.
	AFGC	= Function 3.
	AFH	= Function 4.
	ACDEG	= Function 5.
	ACH	= Function 6.

2. The next step is to analyze the program organization hierarchy charts and the listings to begin your top down plan. In developing the plan, first determine what the major functional variations are. You will be able to combine many of the minor variations within the scope of a single chart. The previous example of program hierarchy might result in the following top down plan:



EXECUTE

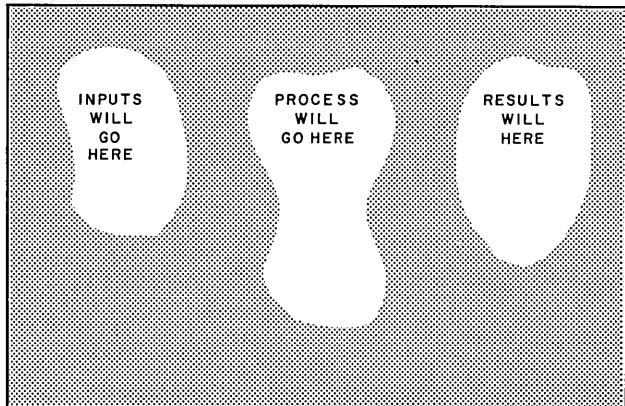


You know what the purpose of the diagrams is. You know what the graphic and informational content should be. What remains is to sit down and begin the diagrams.

How do you start?

Localize

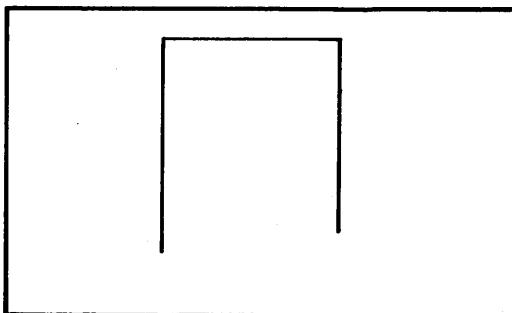
Divide the paper into three areas:



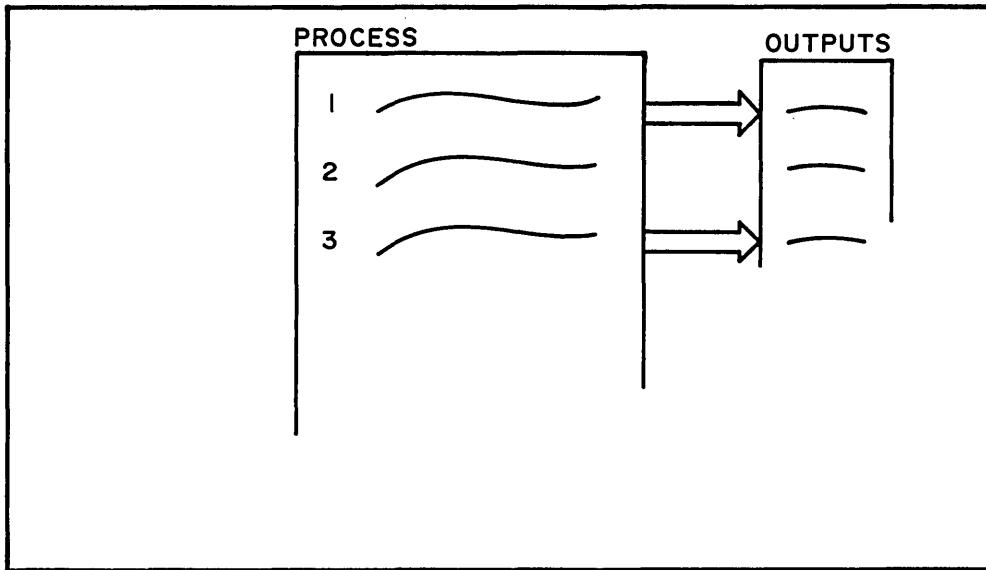
Later, after you have worked out some of the general considerations, you might want to further subdivide to provide an area for blow ups, before/after pictures, tables, or extended descriptions to the processing points.

Remember that initially you are dealing with function. Not with routines.

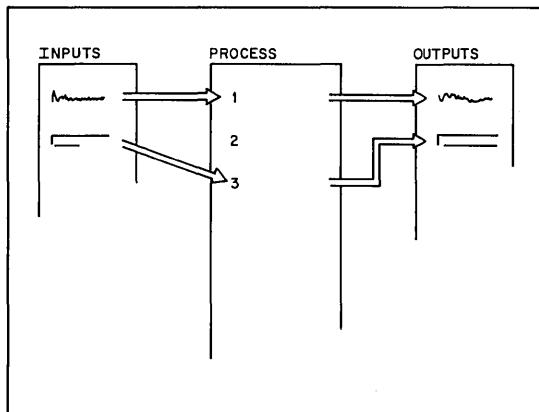
Begin by drawing a large open ended box in the center of the paper.



Work from results to process, and vice versa, jotting down the points as you think of them. Keep the processing points similar in detail to paragraph titles. You can expand on them later.

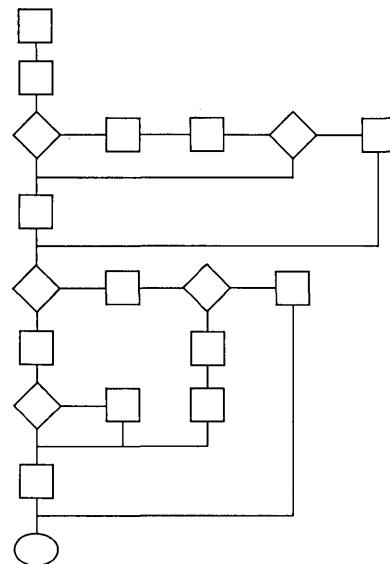


As you do this, input requirements will make themselves known. Jot these down on the left side of the paper and connect them to the processing steps to which they apply.



The time you are now spending on "drawing" is not really drawing time -- it is thinking time. The drawing will evolve as a by-product of your thought processes. Rather than being extra work, this process is actually a tool to help develop the product.

Localizing the inputs, process, and outputs is better than using the standard flowchart approach to show function because inputs and outputs are not readily identifiable in flowcharts.

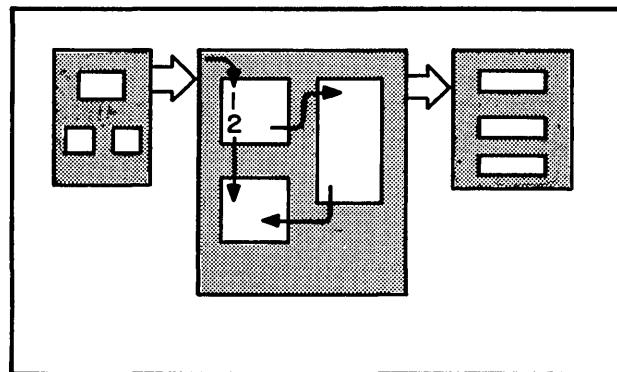


One additional method can be used to help localize diagram components; this is shading.

Although not a requirement of the HIPO technique, shading can add a great deal in making diagrams more understandable.

Use shading as a foil for:

- major process groups.
- inputs.
- results.

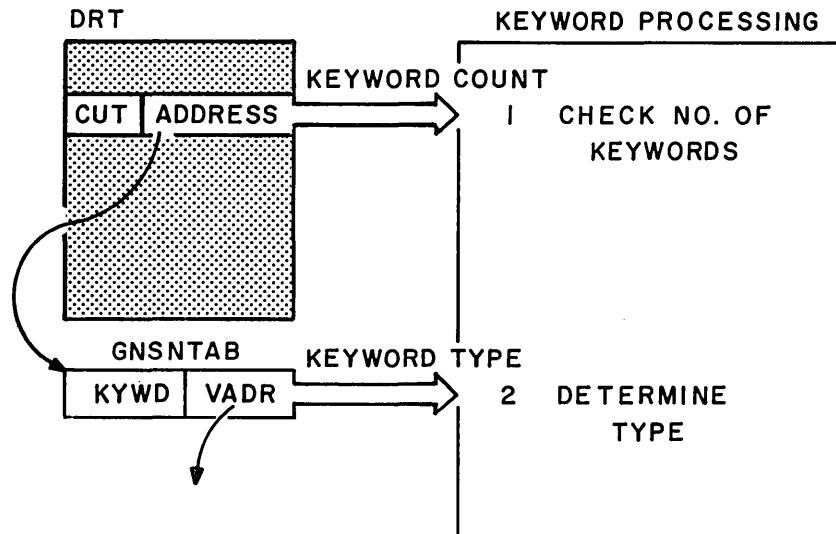


Use the side of a number 2 pencil to apply shading to your final drawings.

#### Minimize Exposures Due to Changes in Logic

There are a few simple considerations that will help you minimize the number and degree of changes to the diagrams.

1. Show only those fields of data with which the function is concerned. Treat the rest as a black box. Do not include displacements.



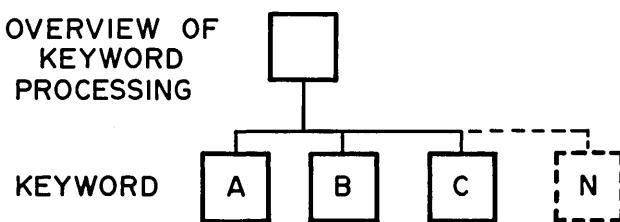
When dealing with data areas belonging to another area of responsibility, use terminology consistently. In addition to eliminating some of the confusion over terminology in the field, this might help you think along the same path that the originator did.

Note: In documenting outputs, always try to state "who" will use what set up. Similarly, if in a subsequent release you use an external data area that you didn't use before, notify the "owner". Someone will have to update the "outputs" portion of that chart.

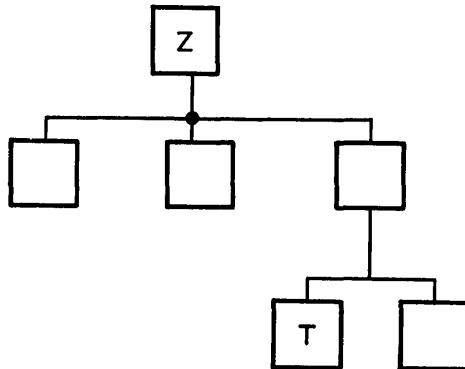
## 2. Find common denominators at the higher levels.

Example: In an overview diagram --

Instead of describing the action for each keyword in the overview, describe in terms of general keyword processing. This way, if you subsequently document a new keyword, you can add it to the package at a lower level without seriously impacting the overview diagram. This ensures that the package is as modular as is possible.

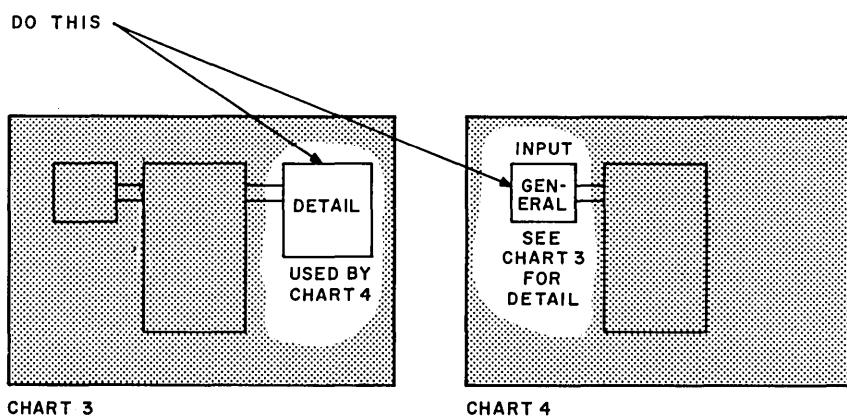
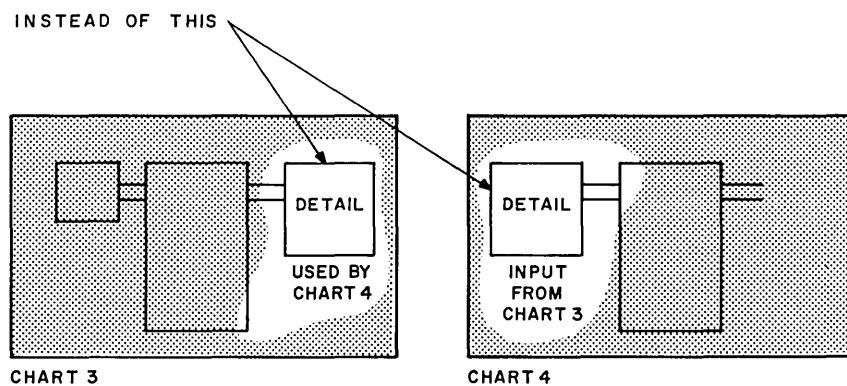


Ideally, a function entering the system at T, below, should not impact Z.



3. Use cross-references from one level of chart to another (vertical references) and from one chart to another at the same level (lateral references) to confine possible changes to as few charts as possible. The cross-references will also help to tie the charts together.

### Example:



Be Consistent

Consistency is one of the more important keys to good communication. If you handle similar problems in similar manners throughout the package, the reader will be able to understand the informational content of the diagrams without having to fight the graphic symbology or structure of the diagram.

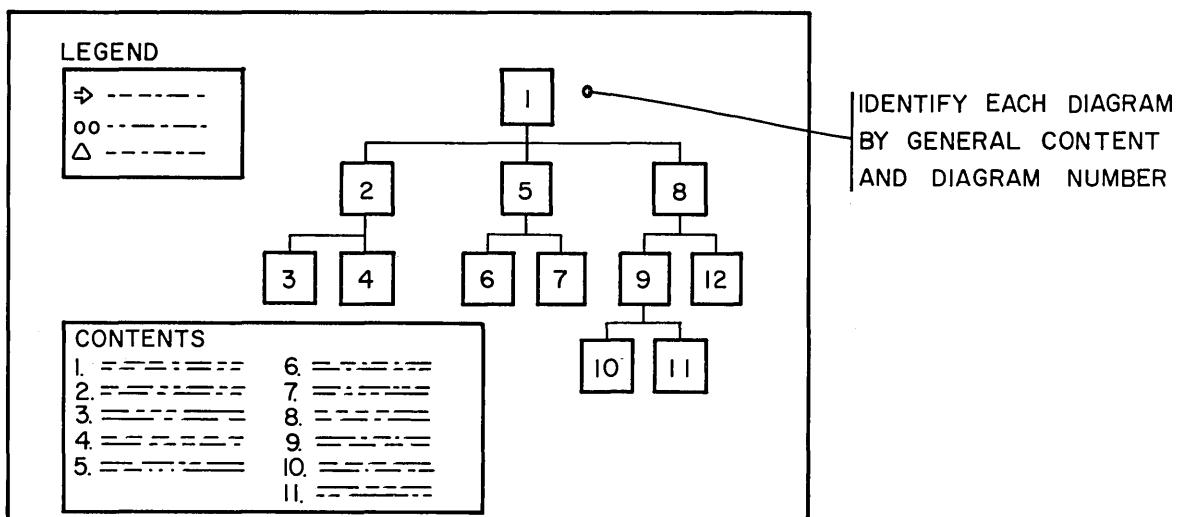
- Remember to localize
- Use a standard symbology and stick to it. If you devise a unique graphic symbol to explain something, you are probably defining a language that only you can understand. In these cases, use text instead.
- Don't change your style in the middle of the package.

File Your Structure

It is quite important to file the structure for subsequent quick access.

1. Use the top down plan.

Your top down plan does not lose its usefulness after you have planned your project. The plan can be used as a visual table of contents to the diagram package.



The table of contents page should show a legend, the structure of the following diagram package, and a summary of the contents of each of the diagrams. With this visual contents page, the reader is not forced to read the diagrams sequential; that is, he can locate a particular level of information or a specific diagram without thumbing through the entire package.

2. Reference from diagram to diagram

High level diagrams should lead the way to the lower level diagrams. Perhaps the best way to do this is to use the off page connector (see Appendix D for examples of this).

Middle level diagrams should refer both up and down to maintain the chain of references.

3. Information retrieval

Another type of filing that will be necessary is that of recording the identification of those diagrams that are affected by design changes. It will be necessary to establish a means of control over retrieving and updating diagrams from release to release.

Use Existing Tools

As was pointed out under "Localize", the time to do initial drawings should not be thought of as drawing time, but rather as thinking time. Jotting down your thoughts in a systematic way will help formalize them.

1. Template and grid

But there comes a time when you will want to make, or have someone else make, a better copy of your diagrams. Currently there are two tools that will help you do neat, uncluttered diagrams with a minimum of fuss. These are:

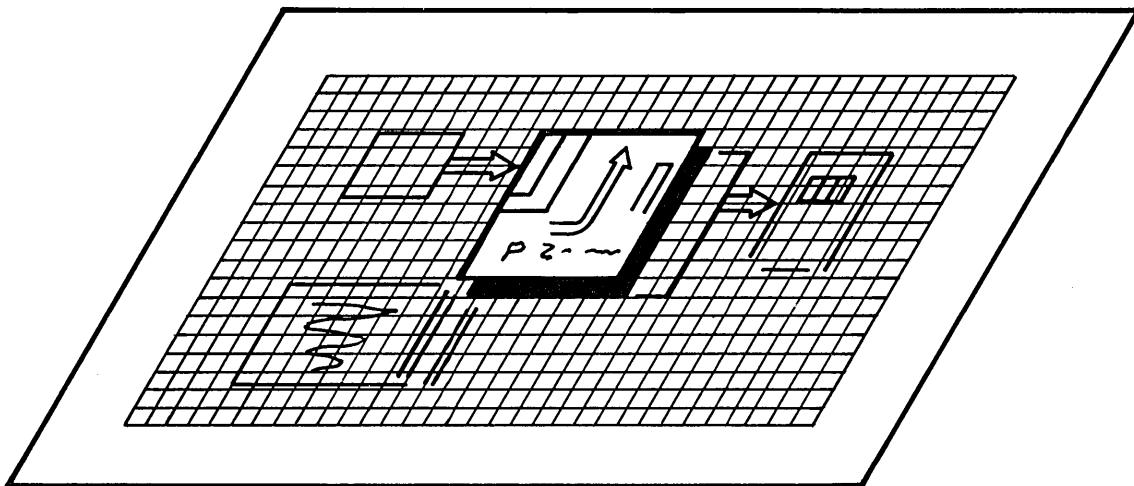
- the Program Logic Template and Jacket (Form Nos. zx28-6734-0 and zx28-6735-0).
- the drawing grid -- 18" by 24" vellum on which a one inch, non-reproducible blue grid has been printed (Form No. zx28-6736-0 -- 1 pad of 25 sheets).

Using these tools, writers have been able to realize a savings of from \$17 to \$40 per diagram (over previous vendor costs) in producing draft copy for final preparation of art.

Furthermore, there is no need to take time to measure or rule the drawing; the grid and template do this automatically.

All of the diagrams in the Appendixes were prepared using the template and grid. As an example of the speed with which final drawings can be done:

The nine diagrams in Appendix C, done by a design programmer, were done in  $4\frac{1}{2}$  hours. The programmer had no prior experience with formalized drawings.



TEMPLATE AND GRID PAPER

ACKNOWLEDGEMENT

A great many people have supplied valuable input and assistance in the development of this document. The writer would like to take this opportunity to thank them for their efforts.



## APPENDIX A

### OPERATION DIAGRAMS -- DIALOG

The appendixes are divided into two parts. First is an interview conducted by the author, Tom Wolfe, with a group of programmers who created method of operation diagrams during the design phase of their project. The programmers involved are: Pete Bowen, Barbara Butler, Donna Dethoff, Terry Elliott, Dave Fishlock, and Sue Montano.

These people were among the first programmers to document design with method of operation diagrams.

The second half is samples of their diagrams.

Note: All examples in these appendixes are photographs of actual pencil drawings done by programmers and, in Appendix E, by the writer.

A second report is being prepared by the OAK programmers. The report will deal with the benefits and problems of diagram packages as seen by the parties involved in the OAK effort.

The appendixes included herein do not necessarily reflect all of the points made in the front section of this report.

- Tom: How many diagrams were in the packages that you prepared?
- Pete: I did eighteen large charts and four foil charts. The foils were drawn on 8½" x 11" pieces of paper and were not duplicated on the large charts. These were general functional charts; general enough so that they would fit on a small piece of paper like that.
- Sue: I did nine diagrams including the overview. I did the drawings in two phases, first initial roughs, and then the full set. I should probably mention that when I began, the design was already quite firm. There were only a few areas that I had to change. For those first drawings, I probably drew the whole set in about 2 hours at the most; but they weren't really that detailed. For the second set, the detailed charts, I had to make all the little decisions -- what's this bit going to be, what am I going to call this field and that -- so it took me a little longer, especially planning out the format of the diagrams. First I sketched what I wanted on a piece of scrap paper. That probably took me about ten minutes. Then it took me a good half hour to do each of the detailed diagrams. That would be about 4½ hours for actual drawing time.
- Donna: I did five diagrams myself. I did one chart that showed what my other charts were going to be. Then I did four charts which were a breakdown of four particular areas I was working on.
- Tom: This next question is in two parts. First of all, did you enjoy doing the diagrams, and then do you think that they help you think through your project?
- Sue: I guess I can answer yes to both of those questions. I did enjoy doing the diagrams because I like to do detailed work. I definitely think that they help you think through the project.
- Donna: I'd say yes to both questions. It was enjoyable working with diagrams rather than trying to write out what I was doing. Any by doing the diagrams, I had to think what the next step would be and put it down concisely; I think this helped my thinking and also my design.
- Sue: Before starting the charts I had most of the major concepts already planned out, but when it came down to the bits and bytes the diagrams made me get my ideas down concretely.

Tom: Well, then would you think that the diagrams are a good way to show function? Are they better than text or flowcharts?

Barbara: Yes. As you know, I was a tester on this project and not a developer. In the past I've had to come up with functional variations from functional specs, and the diagrams are

far superior to specs as far as finding the variations.

- Donna: I definitely like this method better than text -- it's not that I dislike writing, I just think the diagrams are easier for someone to understand.
- Pete: I think this method is a large improvement over just plain text. With this method you can put in the text what you feel is necessary and usually, if you're showing a good deal of processing and logic in the diagrams, you find that the amount of text you need is cut down quite a bit. I think the diagrams are an excellent intermediate step between regular text and flowcharts.
- Terry: Another think became obvious when we gave design reviews with the diagrams. A programmer really had to understand what he was presenting because he couldn't hide anything from the reviewers with MO diagrams. The entire design is fully exposed and can be seen at a single glance.

Tom: Any other comments on this point?

Barbara: Yes. One of the big problems in OS is the fantastic complexity. I think the diagrams are a really good way to get an overall look at the system so that new function can be added in a much more logical way.

Tom: I believe you said you did 18 diagrams, is that right, Pete?

Pete: Yes.

Tom: Can you give me a rough estimate of the amount of text your diagrams replace?

Pete: Well, if I tried to provide the amount of documentation that I think my charts provide, I would have had to write probably about one hundred pages. I don't know if the comparison can be made directly. If I was just writing text and had to sit down and write one hundred pages, I would probably tend to leave information out simply because it would be too tedious to write that much.

Dave: I'd like to emphasize that point. Our job is to specify change to existing function in order to support a new hardware device. This could be expressed as a delta specification on top of a base design. Since no one has used method of operation diagrams before us, we've had to diagram the base system and then superimpose the delta on top of it. So we've spent a great deal of time documenting existing design in addition to specifying changes and additions.

But part of the problem with the more conventional method of specification is that the designer doesn't do the job of defining his base. He just defines his delta and the

reviewer often has a great deal of difficulty putting the design into context. The method of operation diagram forces you to put the design into context. So to present the information that was on any one of our diagrams could require five or six pages of written text.

- Tom: Then have you any suggestions on how existing documentation requirements in the development cycle, for example logic specs or functional specs, could be modified to prevent redundancies or eliminate duplication of effort as a result of the diagrams?
- Terry: As I see it, the MO diagrams serve as a design tool from which functional specs can be written. I guess the MO diagrams can also serve as part of the logic specs and PLM documentation. I think what's happened in the past is that the initial designers have been drawing things that look a lot like MO diagrams, but they get thrown away somewhere in the process. Usually different people write functional specs, logic specs and flowcharts. Then to complete the cycle the pubs people make MO diagrams out of logic specs and flowcharts to put in the PLM. I see no reason why we couldn't save the diagrams originally done by the designers.
- Dave: I see the diagrams as a central document in an overall revamping of the design specification phase of a project. We're using them to supplement pre-functional spec design work. I think the question of whether this is additional work is really not the issue. It may be the formalization of work that, if not going on today, should be going on. I think that diagramming is just a natural step that a good program designer goes through. But he usually has design diagrams on scraps of paper that he throws in the waste basket once the design is converted to prose. The diagrams merely standardize that initial phase of the project.
- Tom: Terry, you mentioned before that you used the diagrams in design review meeting. How did the reviewers react?
- Terry: I have heard reactions from several people. It's all been positive. They think the diagrams are a great improvement over the past. Programmers can clearly explain their work at reviews. I haven't heard any negative response.
- Dave: We presented the method of operation diagrams as projection foils. Since we had been using them for several weeks in our design, we just naturally started presenting the data this way without really preparing the audience for method of operation diagrams. We had the reviewers understanding the design immediately. The review meetings were very productive. Reviewers could see the whole thing in one picture, and we spent much

less time explaining what the design was and could spend more time analyzing it. The feedback was very positive.

Tom: Aside from design reviews, what uses do you anticipate for the diagrams? What are your plans for them?

Pete: We plan to use the MO diagrams to write final functional specifications from, and eventually it looks like MO diagrams will lead to system plans and logic specifications. I guess the MO diagrams will be the basic input to the Pubs people.

Tom: How about in the testing area, Barbara?

Barbara: Well, in my specific area there will be other groups doing testing for different specific functions. It would help them to know what changes were made for our project, especially people that are responsible for maintaining the regression libraries, where our test cases will eventually end up. The diagrams will help in documenting the test cases.

Dave: In the future I hope that the charts could form the nucleus of a base document for the whole system. As we go through the project and bring various people on board -- for example, publications people, test case people, and new programmers -- the diagrams will be a good tutorial to describe the project.

Of course the diagrams will be used as implementation input. The people who produced the diagrams are not necessarily the ones who are going to do the coding, although they may do some of it. So the diagrams can help maintain the integrity of the design when it gets into the implementation phase.

Tom: From a programmer's point of view, what were the toughest problems in picking up the diagramming technique?

Donna: I didn't really find that many problems in picking up the technique. Every once in a while I would have trouble getting my ideas down concisely and trying to put them down in just four or five steps that would still get the idea across. But after I conquered my obsession with words, I managed to get the diagram down fairly accurately and concisely.

Pete: The toughest problem I've had is that the logic I'm concerned with is quite a bit more complex than I can fit on one chart. I don't think that it's a problem with the charts, I think that I just haven't been able to break complex logic into more than one diagram. I tend to take a whole functional area and attempt to fit it into one diagram and sometimes it gets to be too much. I think it's a matter of experience in deciding what can be broken down into subfunctions.

Terry: The big problem is that you really need these method of operation diagrams across the entire system. Also, we found out when Pete was doing his diagrams that when he started them at a low level and tried to build back up that it really couldn't be done with method of operation diagrams. You have to start at the top, at a high level, and then go down to the detailed level.

Pete: Another problem with the charts has been with reproducing them for handouts. They're large and sort of cumbersome, and the size of the paper makes it difficult to reproduce. Reproducing is somewhat dangerous since the equipment tends to destroy your originals once-in-a-while.

Tom: Any other problems?

Sue: One of the biggest problems I had was making changes. We did the diagrams in pencil and we could erase, but after awhile some of the changes were just too much. The diagrams were fairly large and sometimes we had to draw them over.

Tom: One of the problems, then, is updating. Assuming that an automated facility was available to update your drawings so that you could red-pencil your changes in and give it to a coder-keypuncher, would you use such a facility? Would such a facility be useful in the development cycle?

Sue: Yes, very useful.

Pete: Such a facility would be very useful when changes are slowing down a little bit and are no longer major. At the beginning, perhaps, we would still stay with hand-drawn diagrams simply to have faster turnaround on them. If they had to be large charts, when we get to that level of detail, hopefully we'd go right away to an automated method. I think if base diagrams were available as Dave mentioned, automation wouldn't come into play until we were ready to make a permanent update to the base. We would mark up the base pictures to show our new design, and we would have reviews on that marked up base. As the design became firmer, then we would automate the changes.

Tom: Did you find that the grid paper and template were helpful in laying out the diagrams?

Sue: Yes, I thought the grid paper was very good as far as getting proportions and things like that, and the template too. I thought it was especially good for redrawing a diagram once I had it laid out.

Donna: I found them very helpful and easy to use. I have trouble with straight lines and rounded corners, so I made good use of the template.

- Pete: I agree. The template certainly was very good, and the grid paper too.
- Tom: OK. Let's go back for a minute and talk about base diagrams. Since they don't exist, how did you learn existing logic so that you could apply your change logic to it? And how much time did the learning take?
- Dave: In the initial month of the project practically all of our time was spent in learning. We read existing documentation, looked at some video tapes, and talked with the recognized experts in the area.
- Donna: I used mainly the Job Management PLM and I talked quite a bit with module owners and viewed video tapes. I think I've learned the most by discussing things with the module owners and talking to other people. I get more out of talking to them than I would trying to read the PLM.
- Pete: Yes, I certainly agree with that. But I want to bring out another point. It's OK to go back to people if they are still around, but quite often they have gone on to other projects.
- Tom: If base diagrams of OS were available, how would they have cut down on the design time, on the amount of design time you had to spend figuring out existing OS logic?
- Donna: I can only guess, but I think I could have spent maybe a half or a quarter of the time trying to figure out existing logic, and spent more time working on my design.
- Dave: I agree. I think we probably could have learned the material in half of the time.
- Pete: I think base diagrams would help a lot if the diagrams were done as I understand they are suppose to be done, that is, with sufficient levels of detail. There are certain levels that I'm looking for that just are not contained in the documentation that is presently available. I think at the different points in the design cycle different amounts of detail are necessary. For instance, at the beginning you don't have to go all the way down to the very detailed level. Base diagrams would allow us to get into our design faster.
- Tom: To sum up briefly, then, you feel that base diagrams, if they existed, would greatly cut down your learning time and would allow you to specify some of your design quickly as delta changes to base diagrams. But now that your diagrams are done, can you see ways in which these diagrams could fit into the existing phase plan or some similar system of checkpoints?

Terry: I think they should be in the design cycle very early for the people who actually do the initial design and planning. I think that a development group can pick up these MO diagrams almost at any point once the initial designers have laid out a function. Once the developers pick them up, they can continue to put the details in the MO diagrams. While this is happening, we can give them to people like development test, publications, and product test. We can let these people see early just what we're doing.

As far as phase reviews and what level of diagrams would be useful at each checkpoint, I don't know. But it seems like much of the material could be available a lot earlier than has happened in the past.

Tom: I see. Could the diagrams help define the point where design ends and implementation begins more readily than is currently possible?

Barbara: I think they'd help. The reason that design carries on through implementation is you find out while you're implementing, and especially while you're testing, that there's a hole in the design. With diagrams it will be easy to see holes sooner so you can patch them during design. One reason you can catch more design holes, I feel, is that it's more productive to review say, 10 diagrams than to read 50 pages of specs.

Terry: Another reason why some designs change during implementation is that the early planners write the functional specs, and then when the implementer gets them he replaces them with his own design because he really doesn't understand what the specifications say. I think this kind of communication problem could be alleviated with MO diagrams.

Pete: Also, the method of operation diagrams are a good media for simply writing down the reasons why things have been done; a lot of my design reasoning is contained in there. So if a flaw is found during implementation, they would be able to decide on changes. You can't really pick up somebody else's design and implement it unless you completely understand it.

Terry: I agree with that. The MO diagrams can be done in such a way that, by the time the implementer gets them, he doesn't really have much choice in what to do. I think this is good. This is different from what happens today. With MO diagrams the coder's interpretations should be identical to what the designer intended.

Tom: After doing the diagrams, do you feel that it would now be easier for you to do a package for another project?

- Sue: Yes, I think it would be a lot easier although it really wasn't extremely difficult in the first place. I think it would be a lot easier to get the feel of exactly how things should be laid out and what level of detail should be included -- things like that.
- Donna: It probably would be easier because I have the basic knowledge of method of operation diagrams now. And I certainly hope that the next package that I get to work on we'll be using the method of operation diagrams. I've enjoyed working with them.
- Pete: I think it would be easier, but we need a little more experience in diagramming different types of projects -- more complicated logic, easier logic.
- Terry: It didn't really take that much time to learn how to do the diagrams. Once you get away from flowchart thinking or code thinking and get to thinking in terms of hierarchy, input, process and output, the diagrams are not difficult to do.

APPENDIX B

A GROUP PLAN

The following two charts are a group plan. These charts were prepared prior to preparing any of the individual diagram packages.

Chart 0 shows the package assignments.

Chart 1 shows the major functional areas of concern, with emphasis on the interfaces between the areas.

# STRUCTURE OF STAGING/DESTAGING DOCUMENTATION

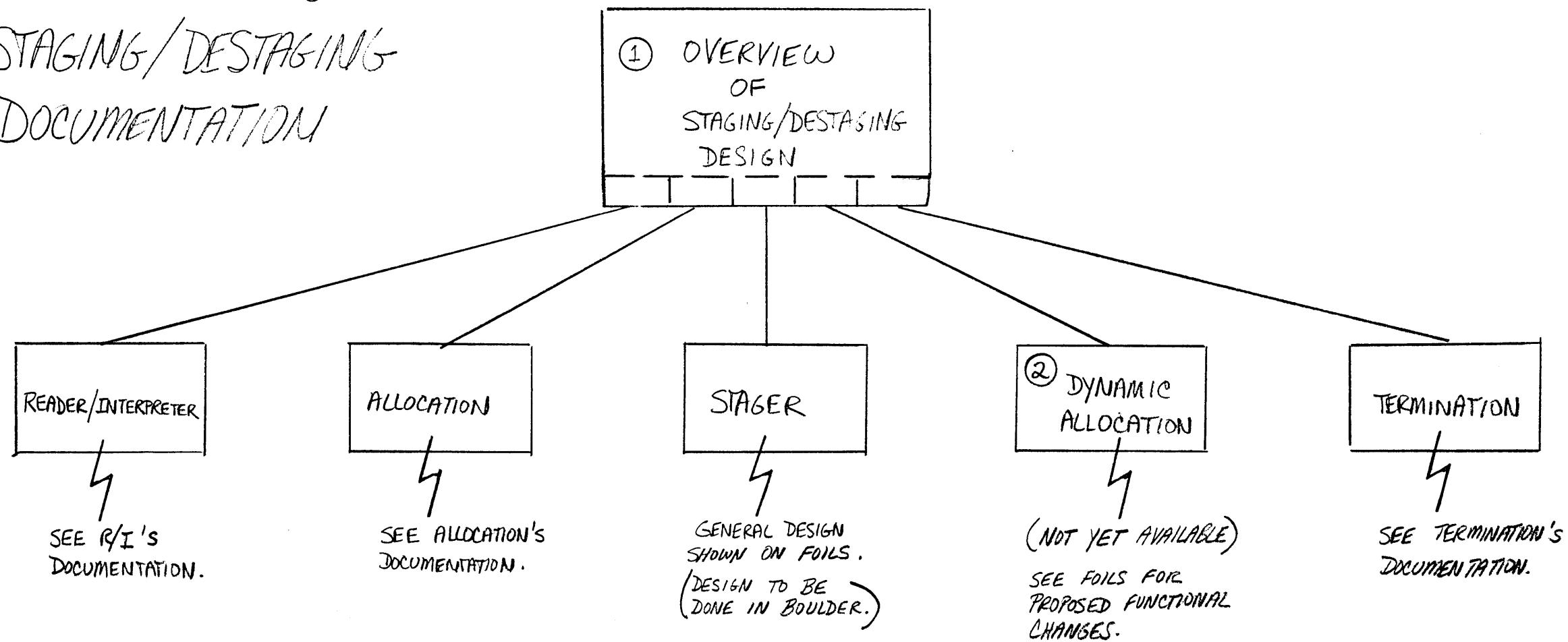


CHART NUMBERS	DESCRIPTION
①	THIS CHART SHOWS THE MAJOR FUNCTIONAL AREAS CONCERNED WITH STAGING/DESTAGING. EMPHASIS IS ON THE INTERFACES BETWEEN THESE FUNCTIONAL AREAS.
②	(NOT YET AVAILABLE)

CHART #Φ

IBM CONFIDENTIAL

2/6/71 RB

# OVERVIEW OF STAGING/DESTAGING

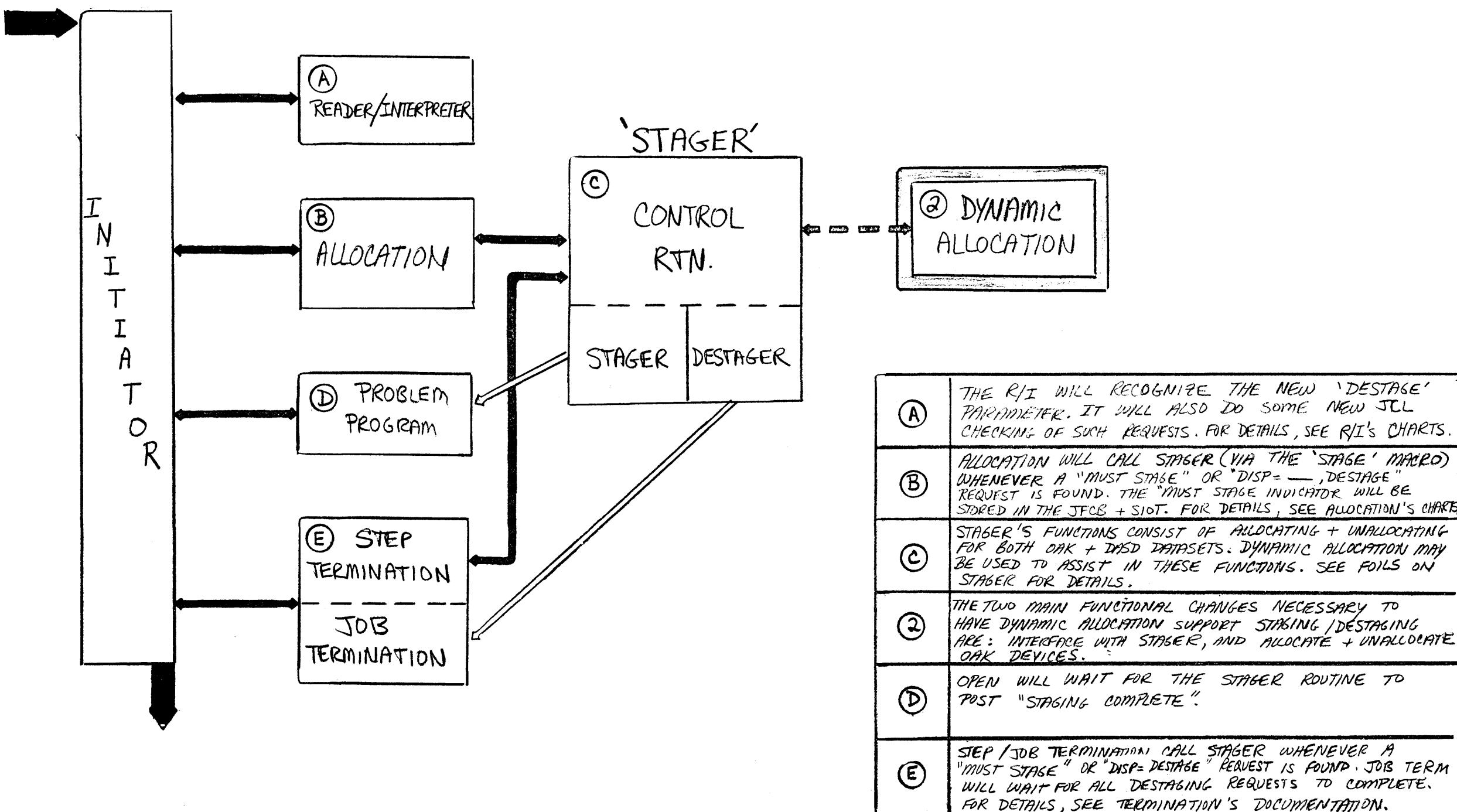


CHART #1

IBM CONFIDENTIAL

2/6/71 PB

## APPENDIX C

## A HIPO PACKAGE

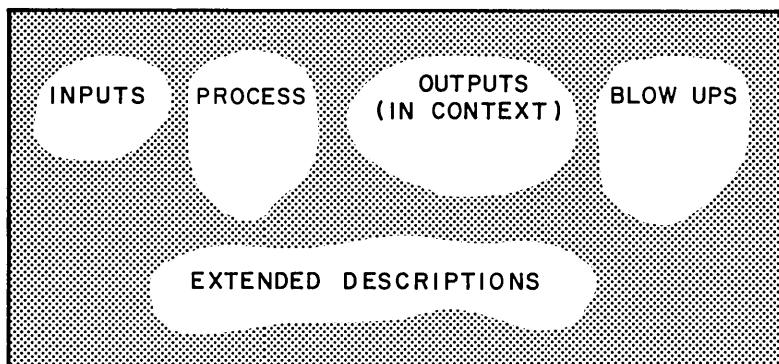
The next nine diagrams\* depict OAK impact to the Reader/Interpreter area.

The first diagram (unnumbered) is a visual table of contents. It shows the structural relationships of the remainder of the diagrams. Included on this chart is a chart description for each of the charts, and a legend.

The next diagram (Chart 1) shows, at a very general level, basic inputs, process, and results of Reader/Interpreter operation.

Diagrams 2 and 6 show the two major areas of concern within the Reader/Interpreter. These diagrams are overviews, which lead, via vertical cross references, to lower level breakouts in each of the two areas.

Charts 3-5 and 7, 8 are the low level charts in this particular package. Each of the low level charts is subdivided into five areas:



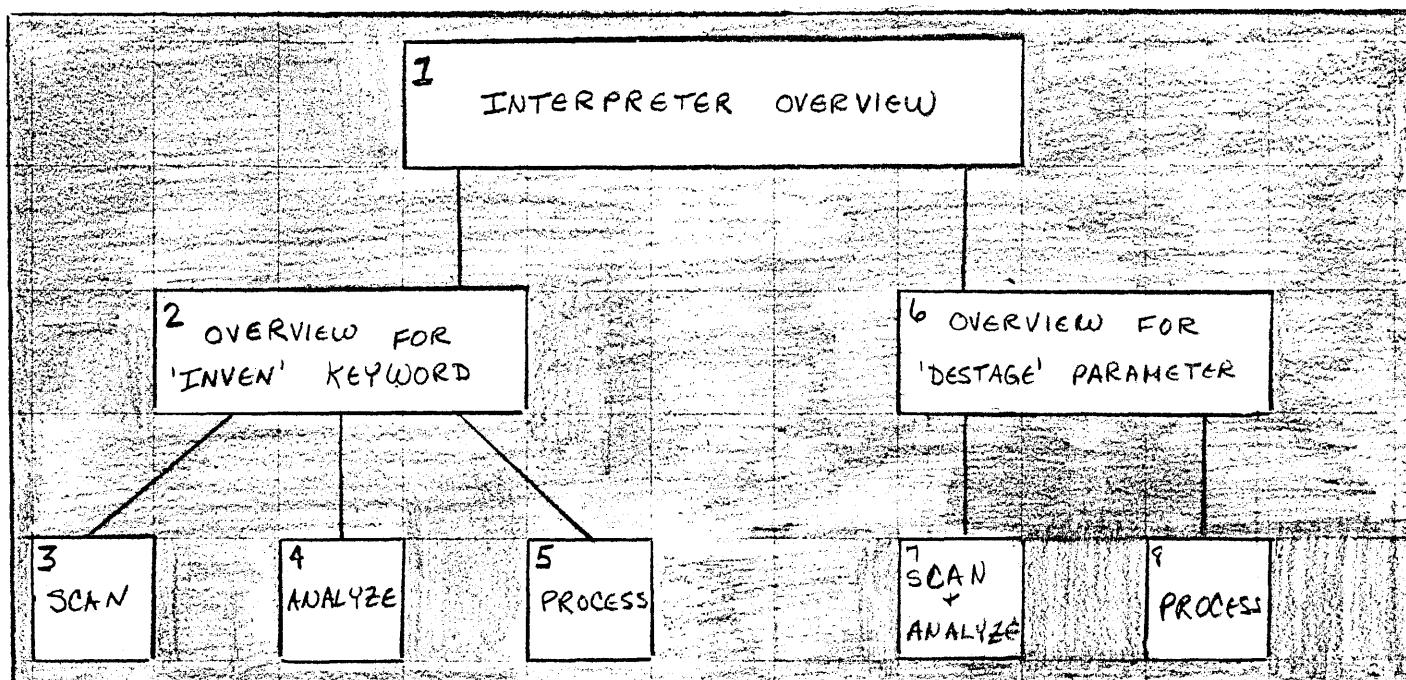
The consistency of treatment of this package makes the diagrams easy to understand; that is, the use of graphics aids the information to be conveyed -- it does not hide it.

Note in these diagrams that the programmers were working to an Operating System base, while interfacing with AM1 functional specifications. Existing OS logic is presented in as much as is required to show the points of impact by OAK.

---

\*Prepared by Susan Montano.

READER/INTERPRETER  
DESIGN FOR OAK



LEGEND

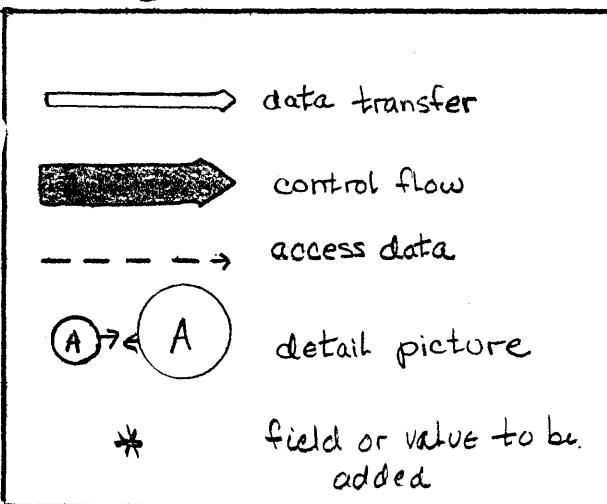


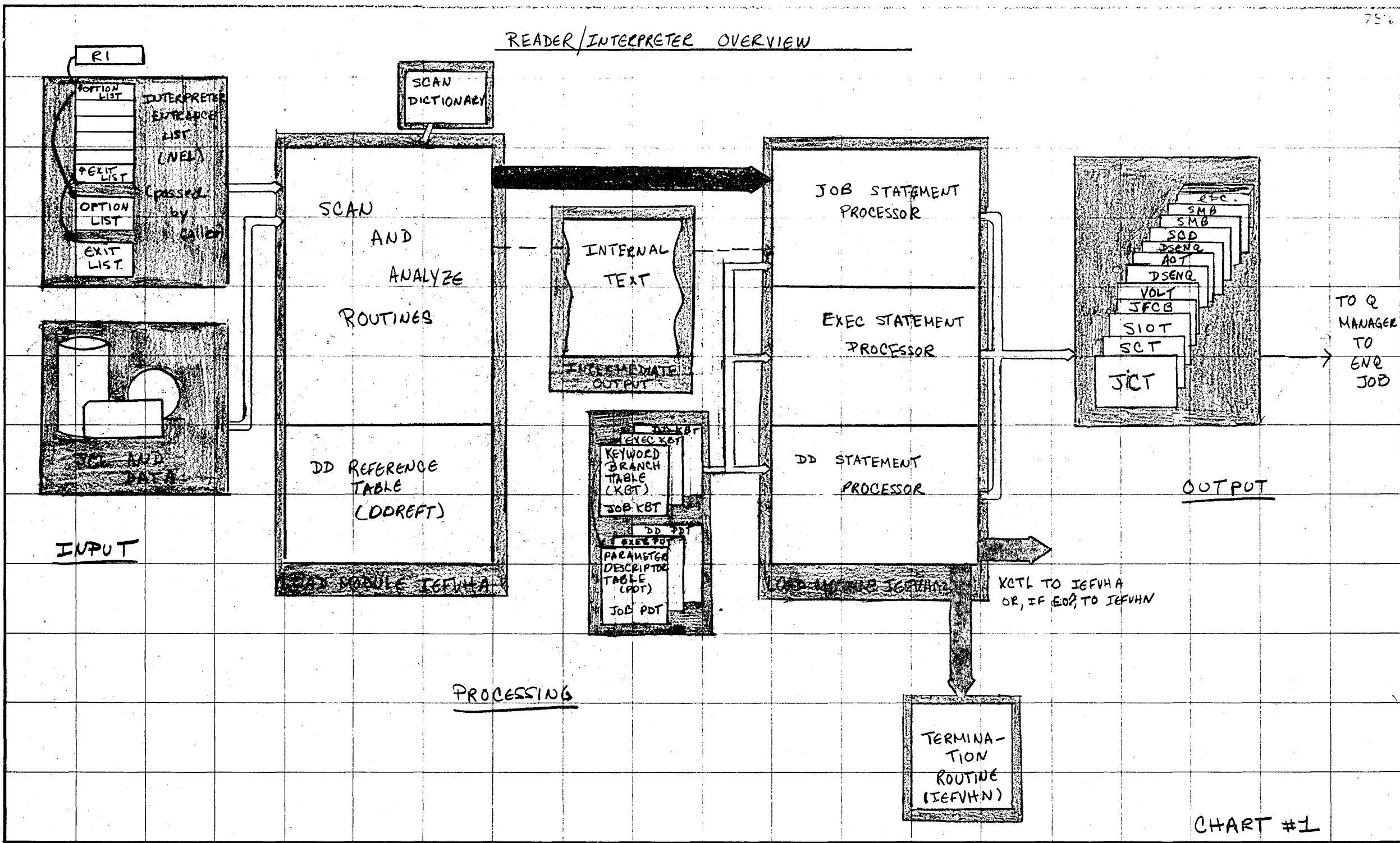
CHART DESCRIPTION

I	Reader/Interpreter overview according to redesign for AM/z
2	OVERVIEW FOR design of new 'INVEN=' keyword into Interpreter
3-5	detail of INVEN design 3-recognize INVEN keyword 4-process Internal text 5-chain associated data set SLOT's
6	OVERVIEW for design of new disposition 'DESTAGE' into Interpreter
7-8	detail of DESTAGE design 7-recognize DISP keyword (current processing); do error checking for DESTAGE specification 8-recognize DESTAGE disposition

Susan Montano  
D81 - 706 Bldg  
X-37189

1/7/71

SM

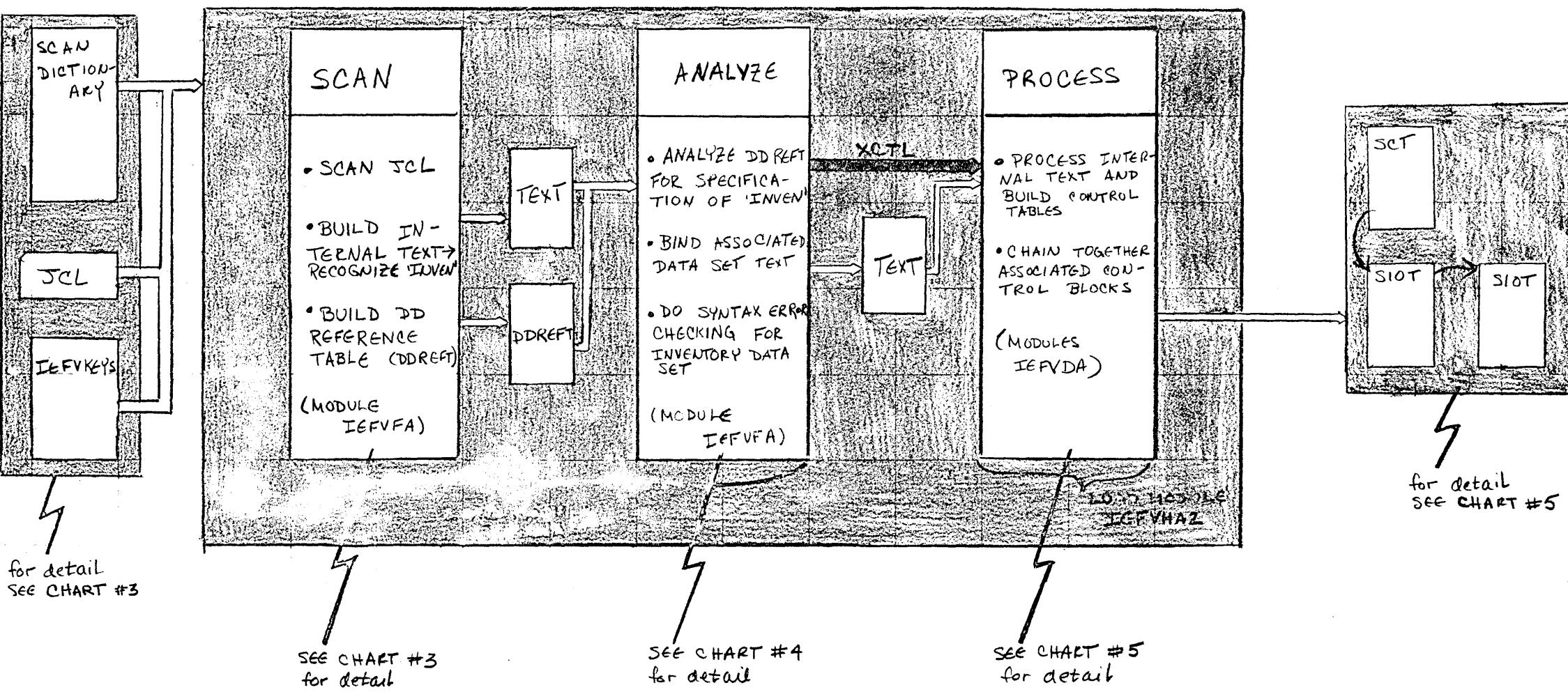


IBM CONFIDENTIAL

1/25/71

Sm

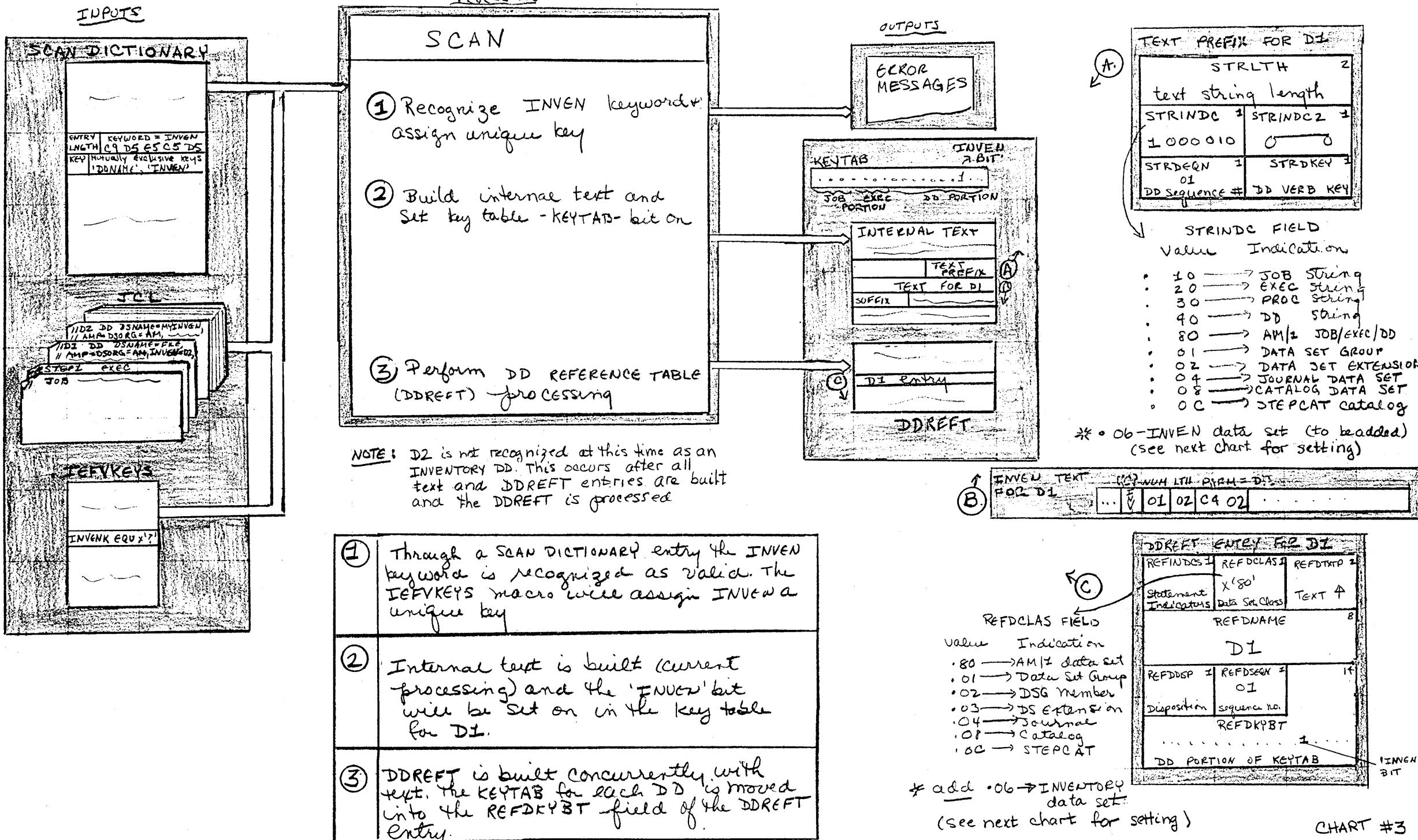
OVERVIEW FOR 'INVEN' KEYWORD



To support private inventories for OAK, a new keyword, INVEN, will be added to the list of valid DD parameters. So that ALLOCATION will be able to mount the user's private inventory, the user must submit a DD describing his inventory data set. Each data set which belongs to this inventory must have INVEN=ddname on its own DD where 'ddname' refers to this inventory DD. This inventory DD must be submitted in every step where data sets belonging to this inventory will be defined. (See example on next chart)

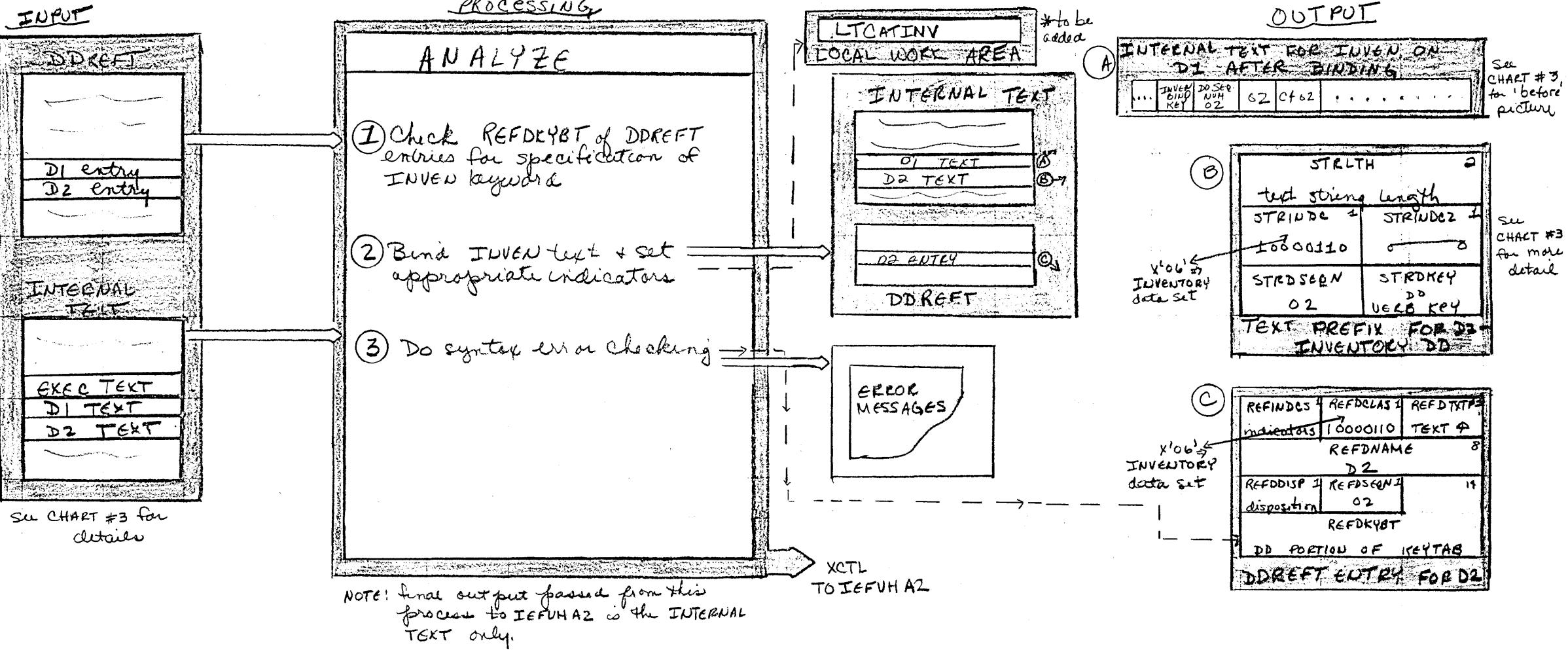
CHART #2

INVEN=d&name SUPPORT FOR OAK  
PROCESSING



INVEN = ddname SUPPORT FOR OAK

PROCESSING



- (1) after all JCL for a job is scanned, converted into text and entered into the DDREFT, DDREFT is processed. the REFDKYBT field will be checked for each entry to see if 'INVEN=ddname' was specified on the DD. if so, the named DD and its text are found.
- (2) INVEN binding will be done from data set to associated inventory and from previously specified catalog or inventory to currently specified inventory. this is done by overlaying INVEN text of the data set D1 with the INVEN binding key + sequence number of the INVENTORY DD, or appending the binding key + sequence number to the previous inventory or catalog text. (use LTCATINV field to point to the text of the last catalog or inventory specified, for this function) also, set INVEN data set indicators in text prefix + DDREFT
- (3) check the REFDKYBT field of the INVENTORY's DDREFT entry for required + invalid keyword specification.

CHART #4

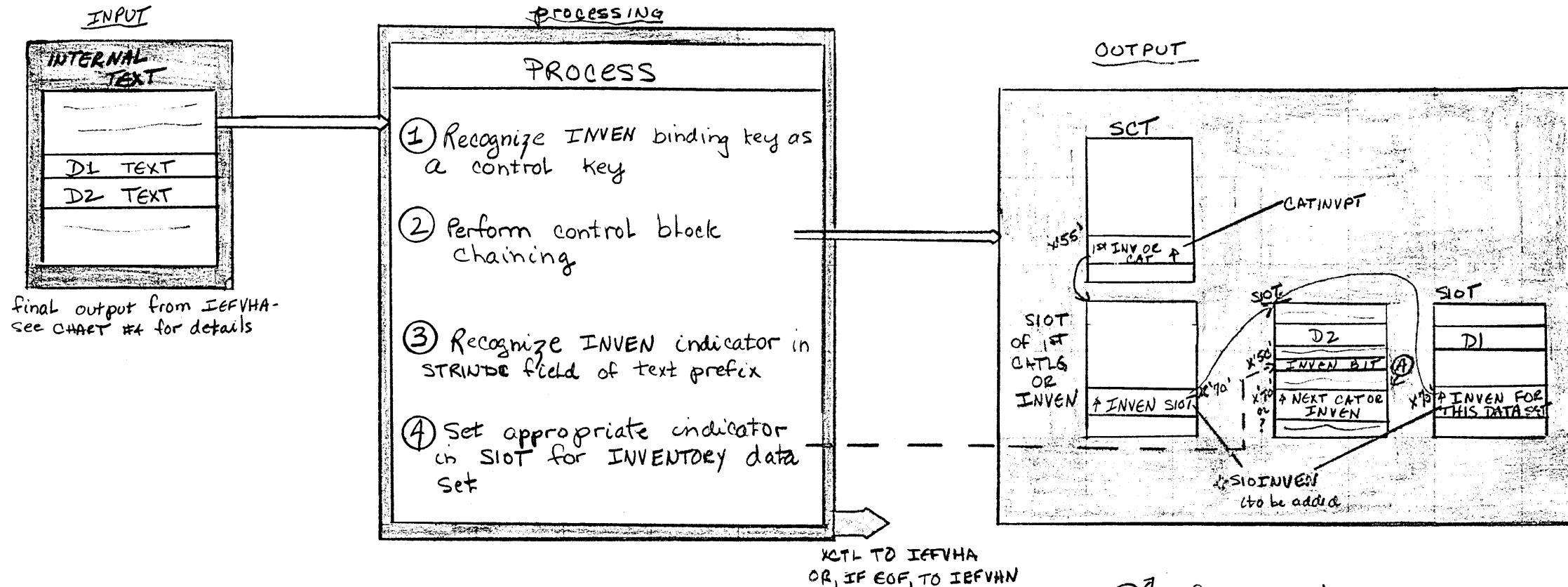
1/12/71

Sm

IBM

CONFIDENTIAL

INVEN= ddname SUPPORT FOR OAK



- |   |  |
|---|--|
| ① | The INVEN binding key will be added to the list of control keys which the GET KEY routine will recognize.  |
| ② | Instead of using the TEST AND STORE module, the control key will cause GETKEY to pass control to AMCTKP - the control key processing routine. This routine will use the sequence number of the DD which was inserted after the binding key to find the TTR of the associated SLOT. This TTR will be moved into the S10INVN field of SLOT which named INVEN or of this CATALOG or INVENTORY SLOT being processed. |
| ③ | The INVENTORY data set indicator will be recognized in the text prefix and the S10INVN bit will be set in the SLOT for the INVENTORY DD.   |

A → offset x'5C'

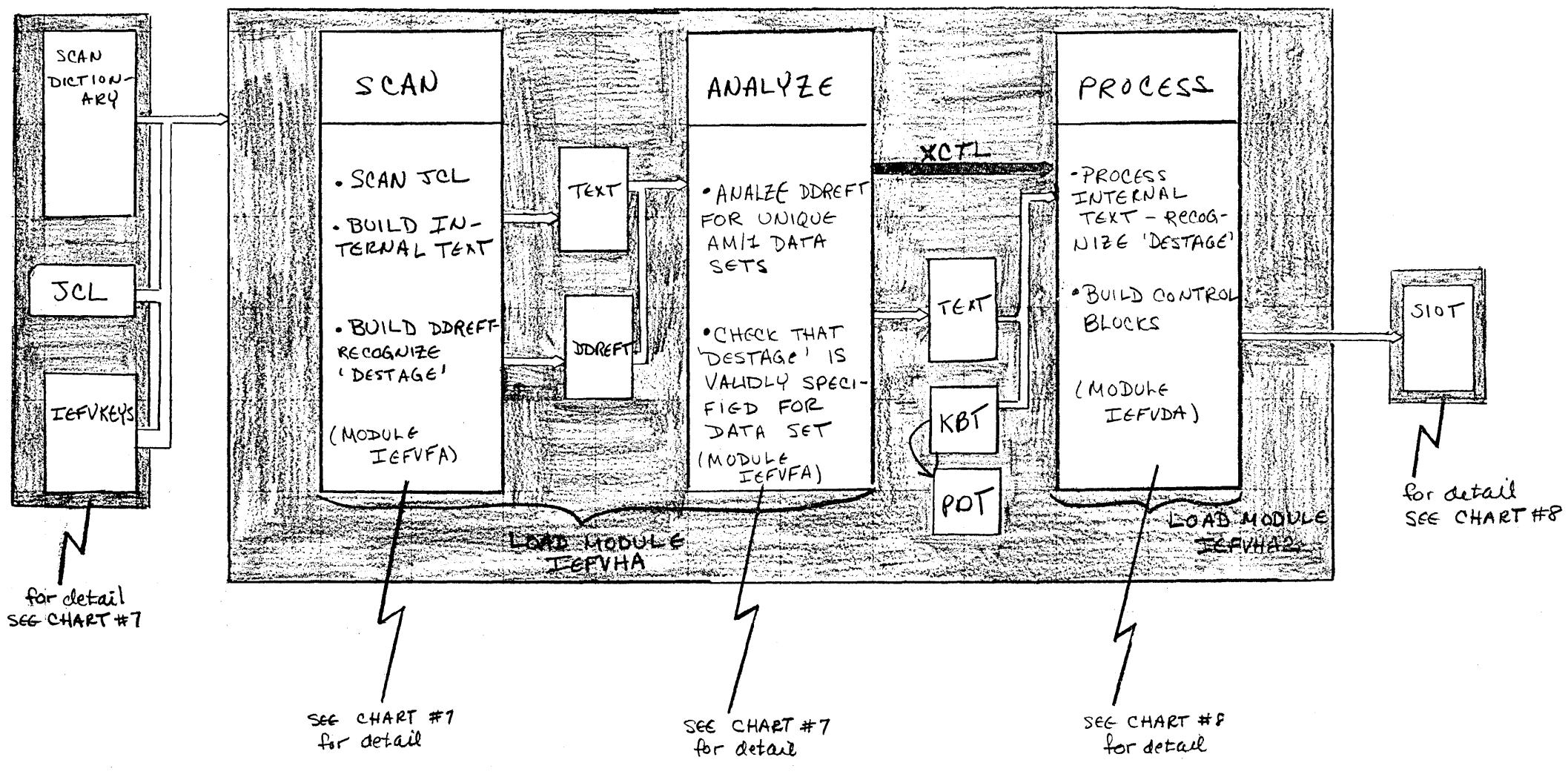
\* { 1... .... } — INVEN indicator - S10INV  
   { 1..1.... } DESTAGE indicators  
   { .1.... } (see CHARTS 6-8)  
   { ..1... }  
   { ...1.. }  
   { ....1.. } used  
   { .....1.. }  
   { .....1.. }

CHART #5

IBM CONFIDENTIAL

2/2/71  
Sm

OVERVIEW FOR 'DESTAGE' DISPOSITION



To support STAGING for OAK, a new disposition, 'DESTAGE' will be added to the list of valid dispositions on DD's for AM/I data sets. 'DESTAGE' will cause DASD data sets to be stored on OAK devices and 'staged' to DASD whenever they are to be processed. DESTAGE is valid only for AM/I data sets with the exception of unique AM/B data sets such as CATALOGS, INVENTORIES and JOURNALS!

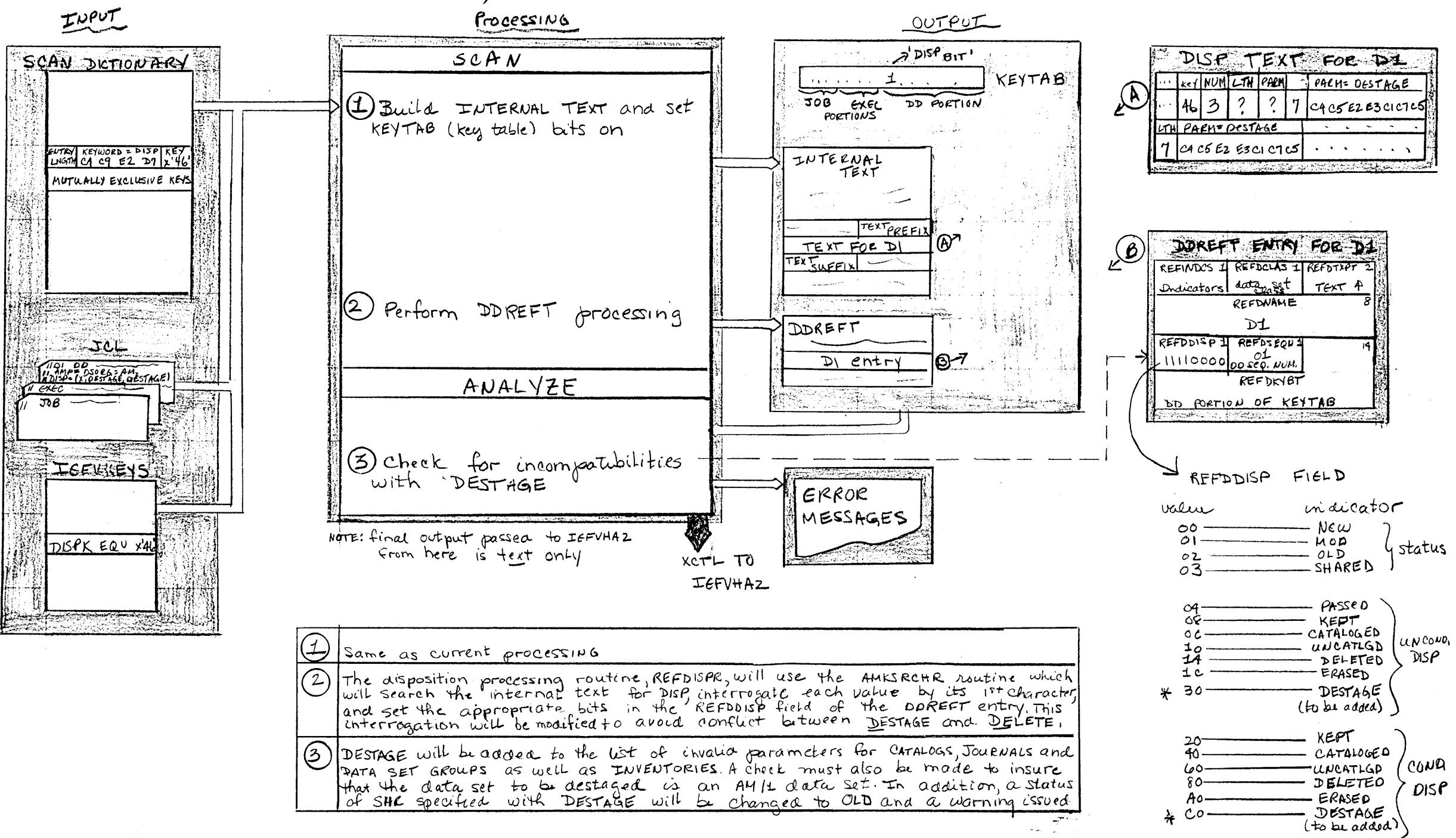
CHART #6

IBM

CONFIDENTIAL

1/11/71  
SM

DISP = (X, DESTAGE, DESTAGE) SUPPORT FOR OAK



IBM CONFIDENTIAL

1/11/71  
Sm

DISP=(Y, DESTAGE, DESTAGE) SUPPORT FOR OAK

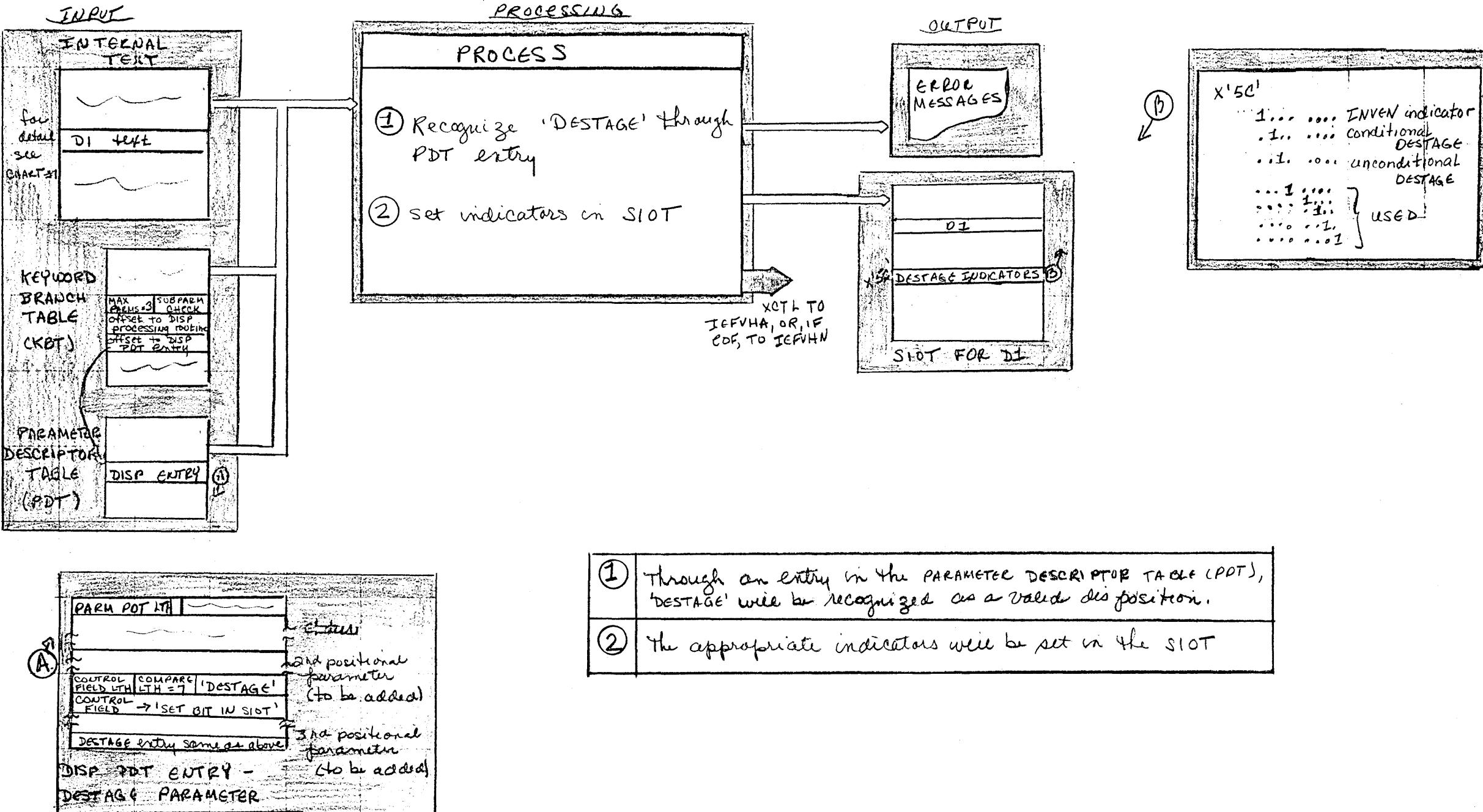


CHART #8

IBM CONFIDENTIAL

1/11/71  
Sm

APPENDIX D

OTHER PROGRAMMER-GENERATED HIPO DIAGRAMS

This Appendix contains four diagrams\* from the Allocation package.

The first diagram, Chart 0, is a visual table of contents. Note the similarity to the corresponding contents diagram in Appendix C.

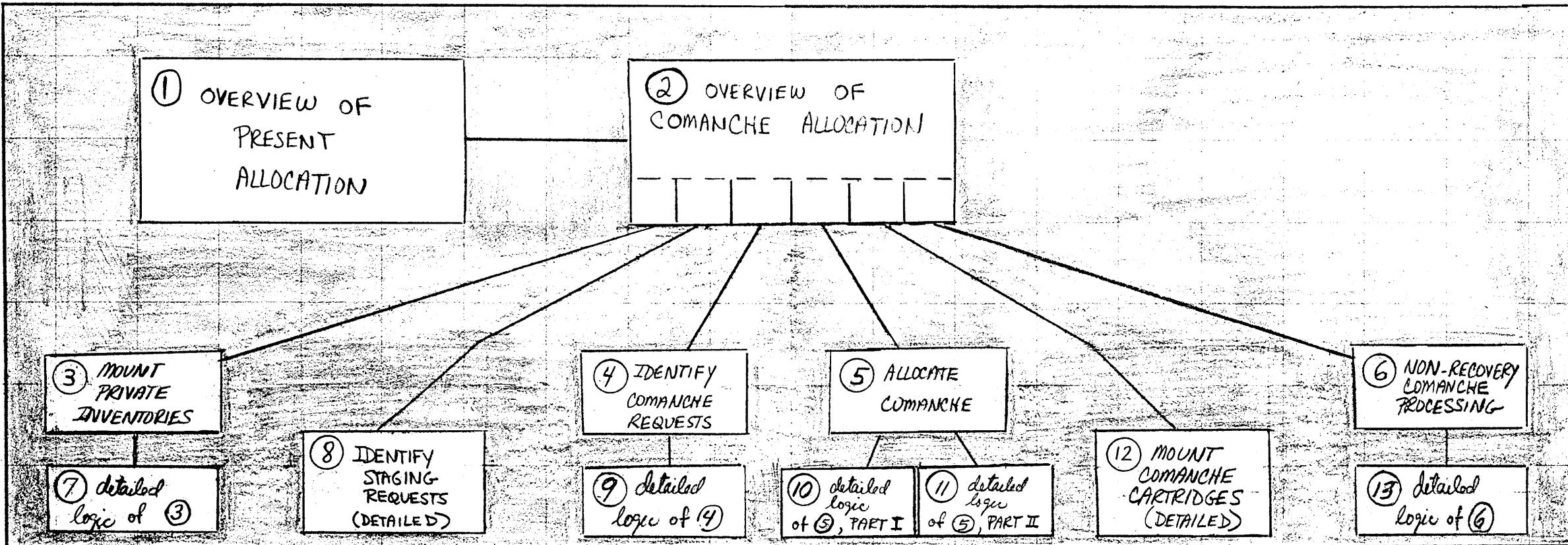
The remaining three diagrams illustrate the adherence to the basic Input--Process--Output format.

Note particularly on Charts 7 and 9 how the input structures were pictured without confusing them with the function being described. To do a similar job with a text description would take more pages, and the task of documenting these items in an understandable manner would be more difficult.

---

\*Prepared by Pete Bowen

# STRUCTURE OF ALLOCATION DOCUMENTATION



## LEGEND

	DATA TRANSFER
	CONTROL FLOW
	ACCESS DATA AND POSSIBLY SMALL MODIFICATION
(a)	reference INDICATOR

## CHART NUMBERS

	DESCRIPTION
①	CONCEPTUAL VIEW OF DEVICE ALLOCATION, INCLUDING AM/I FUNCTIONS. EMPHASIS ON SECTIONS IMPORTANT TO COMANCHE ADD-ON.
②	OVERVIEW DEPICTING THE MAJOR FUNCTIONS OF THE NEW COMANCHE SUPPORT. EMPHASIS IS ON THE POSITION OF THE INTERFACES BETWEEN THE EXISTING CODE AND THE NEW COMANCHE SUPPORT.
③ - ⑥	THE LOGIC OF EACH FUNCTIONAL AREA IS DEPICTED IN SEPARATE DIAGRAMS. EMPHASIS IN THESE DIAGRAMS IS ON THE ORDER OF PROCESSING. (NOTE: THIS LEVEL OF CHARTS WAS SHOWN ON FOILS — NOT AVAILABLE ON LARGE CHARTS.)
⑦ - ⑬	DETAILED LOGIC OF EACH OF THE MAJOR FUNCTIONAL AREAS OF THE NEW COMANCHE SUPPORT. EMPHASIS IS ON THE ORDER OF PROCESSING AND ON CONTROL BLOCKS, TABLES, ETC. THAT ARE USED TO ACCOMPLISH EACH FUNCTION.

CHART #Φ

IBM CONFIDENTIAL

2/17/71 PB

# MOUNT PRIVATE INVENTORIES

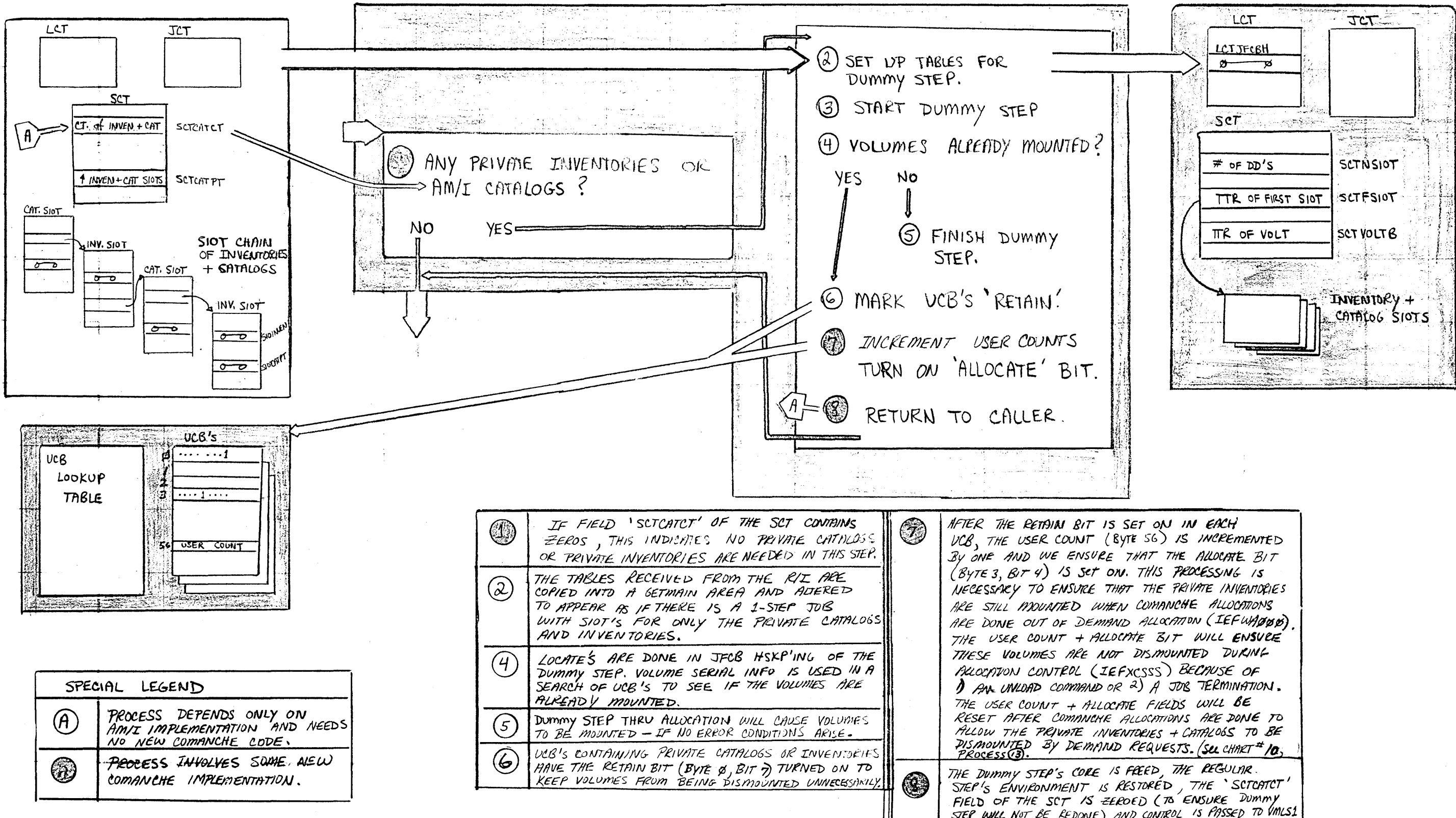


CHART # 7

IBM CONFIDENTIAL

2/17/71 PB

# IDENTIFY COMANCHE REQUESTS

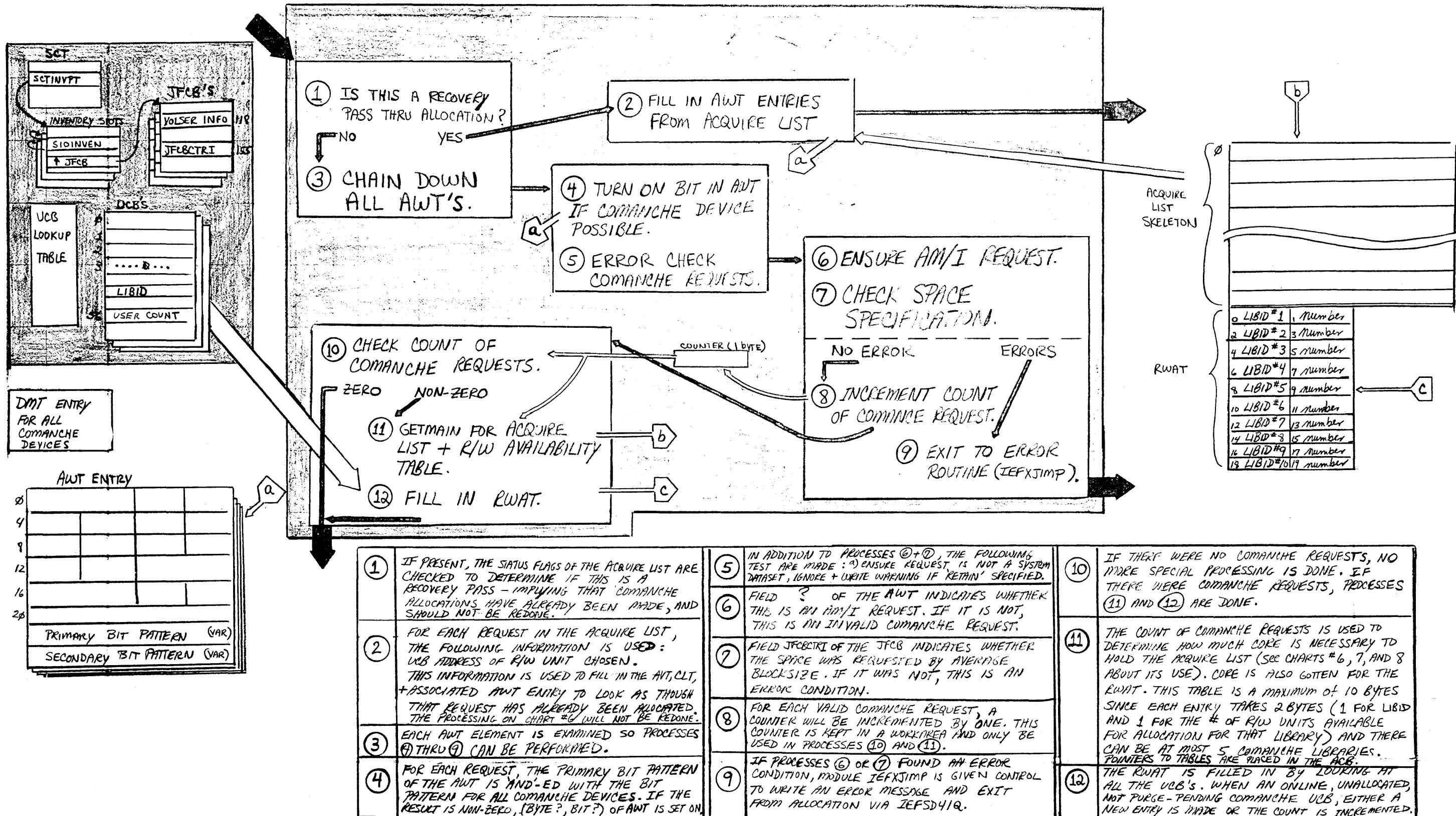


CHART # 9

IBM CONFIDENTIAL

2/17/71  
RB

# NON-RECOVERY ERROR PROCESSING

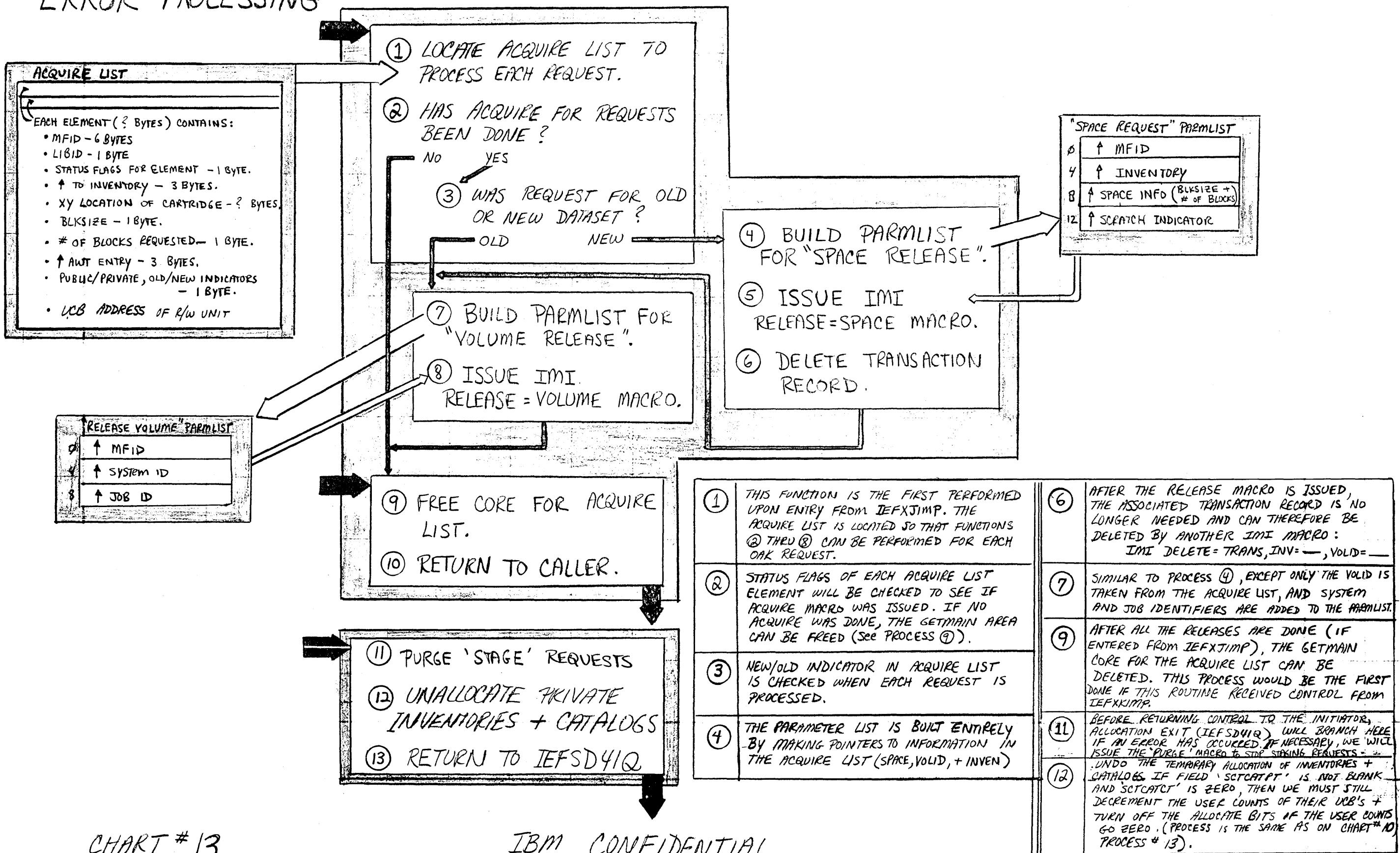


CHART # 13

IBM CONFIDENTIAL

2/17/71 PB

APPENDIX E

WRITER-GENERATED DIAGRAMS

The diagrams in this Appendix\* are not affiliated with the same project as the diagrams in the other Appendixes. These diagrams represent a functional data Reduction package, prepared for inclusion in an existing PLM.

No visual table of contents diagram is included.

Diagram 27 is the overview, leading the way to the lower level diagrams.

The remainder of the diagrams, which represent about 50K bytes of code, depict the different data reduction functions, describe how the design points were implemented, and provide pointers into the listings.

Particularly noteworthy are:

- The grouping of the processes and sub-processes. For example.

Diagram 29 -- The "Scan-for-START-card process consists of three steps. Step 3 is actually supported by many different routines, as shown through the cross-reference provided in the extended description area.

- The simplicity of treatment of the major data areas.

Only those fields that are used by the function are discussed.

No displacements are included.

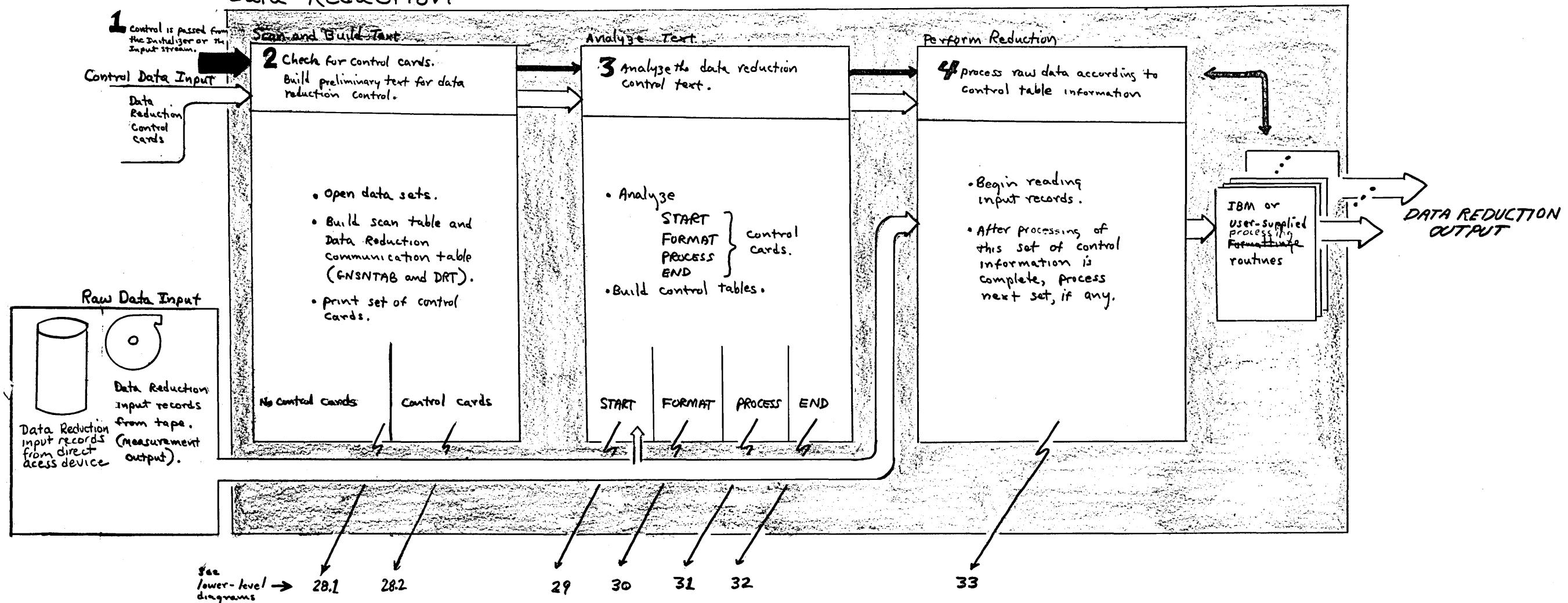
- The Input--Process--Output format of each of the diagrams.

- The overall consistency of handling.

---

\*Prepared by Gil Moore

## Data Reduction

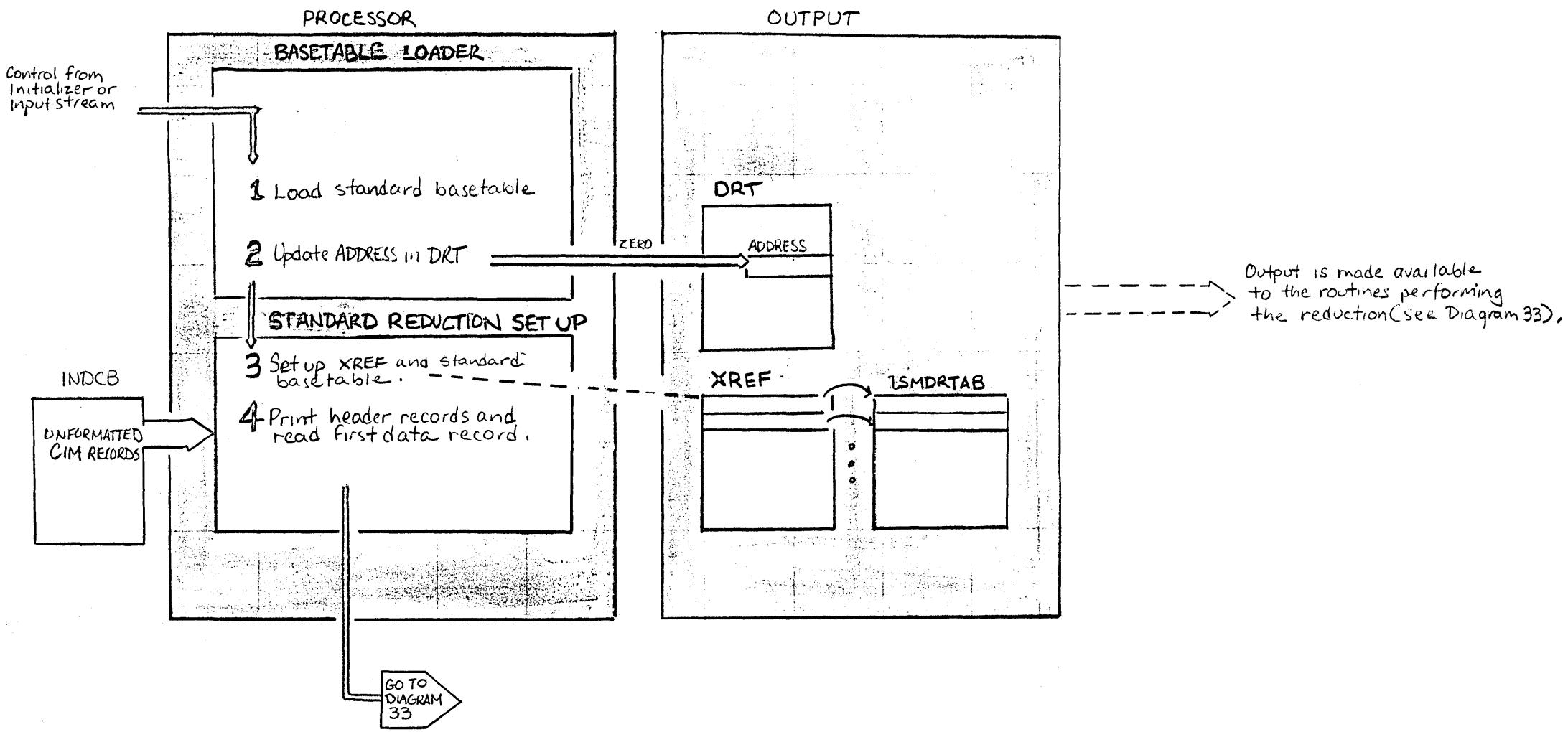


- 1 Data reduction can be called as a separate job or run at the end of a measurement.
- 2 If a set or sets of data reduction control statements are provided, preliminary text consisting of information from the statements is built to control the data reduction run. If no control statements are provided, all input records whose format identifications (FIDs) are represented in the standard base table are processed.

3 A set of control statements is analyzed and used to build control tables.

4 As the input records are read in, the control tables are checked for processing instructions such as sampling rates, conditional operations, and branch locations of formatting routines, etc.

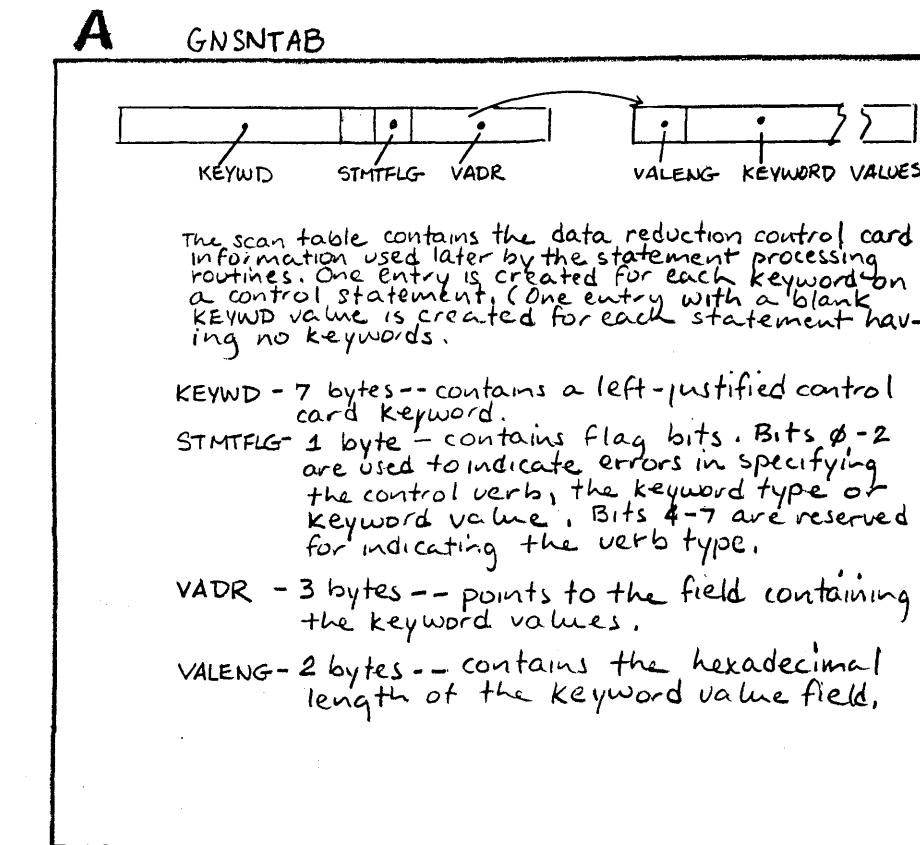
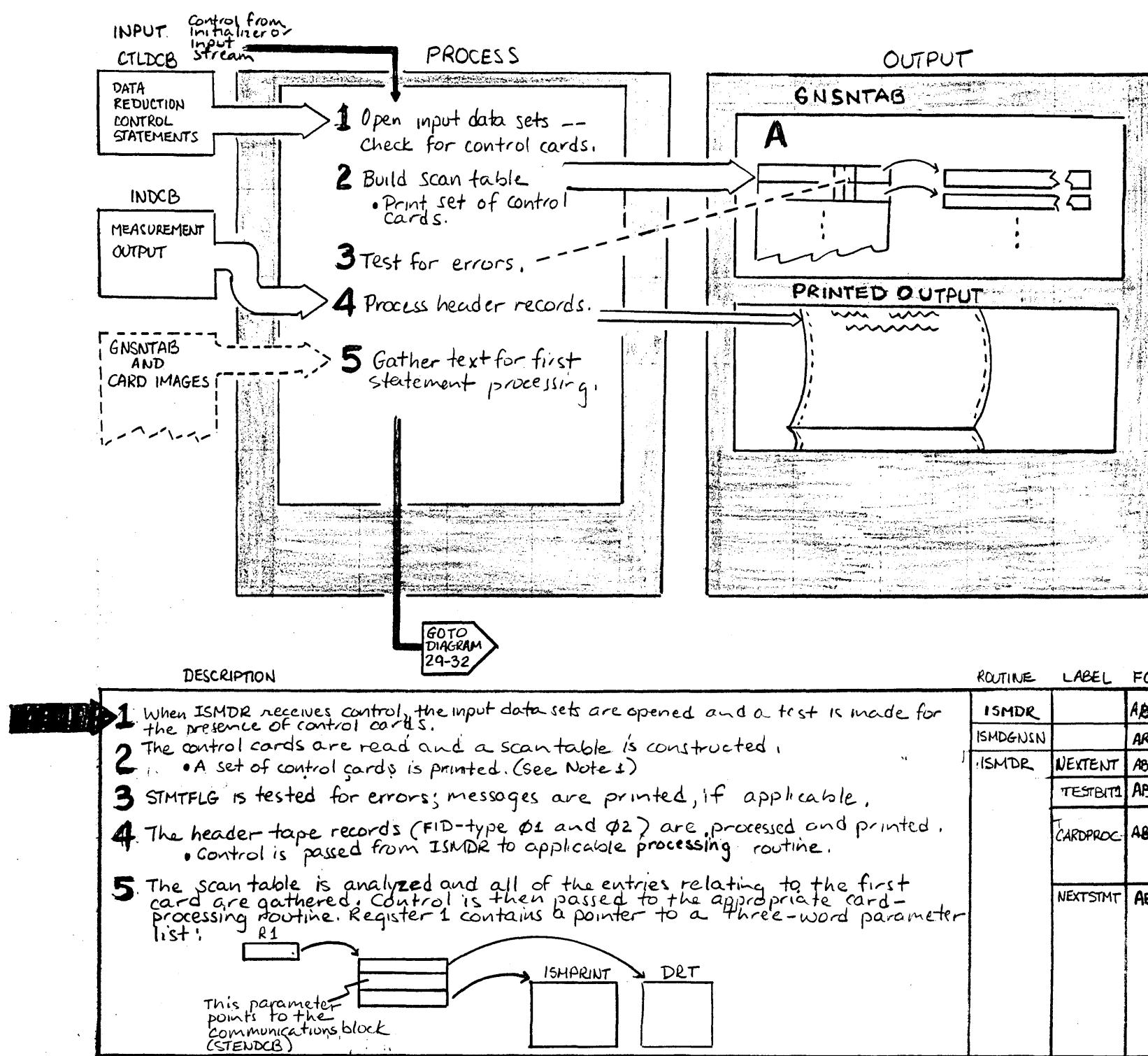
IBM CONFIDENTIAL



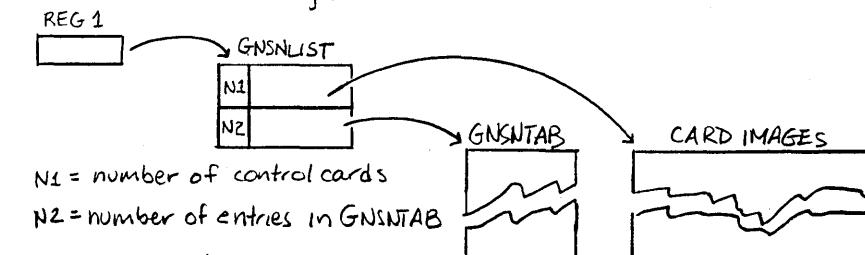
- 1 After ISMDR receives control and finds an empty control card data set, it loads the standard basetable (ISMDRTAB)
- |        |         |    |
|--------|---------|----|
| ISMDR  | LOADTAB | AA |
| ZEROAD | AA      |    |
- 2 The ADDRESS field in the data reduction communicator (DRT) is zeroed to indicate the absence of control statements.
- |          |         |    |
|----------|---------|----|
| ISMDRPRO | BLDXREF | AH |
|----------|---------|----|
- 3 The XREF table (used as an index into the base-table), is set up by pointing each XREF slot to an entry in the basetable. All ISMDRTAB entries are primed for use.
- |       |          |    |
|-------|----------|----|
| ISMDR | CARDPROC | AA |
|-------|----------|----|
- 4 The header tape records are printed (control is passed to the applicable formatting routine). The first CIM record is read and standard data reduction begins. (See Diagram 33)

IBM CONFIDENTIAL

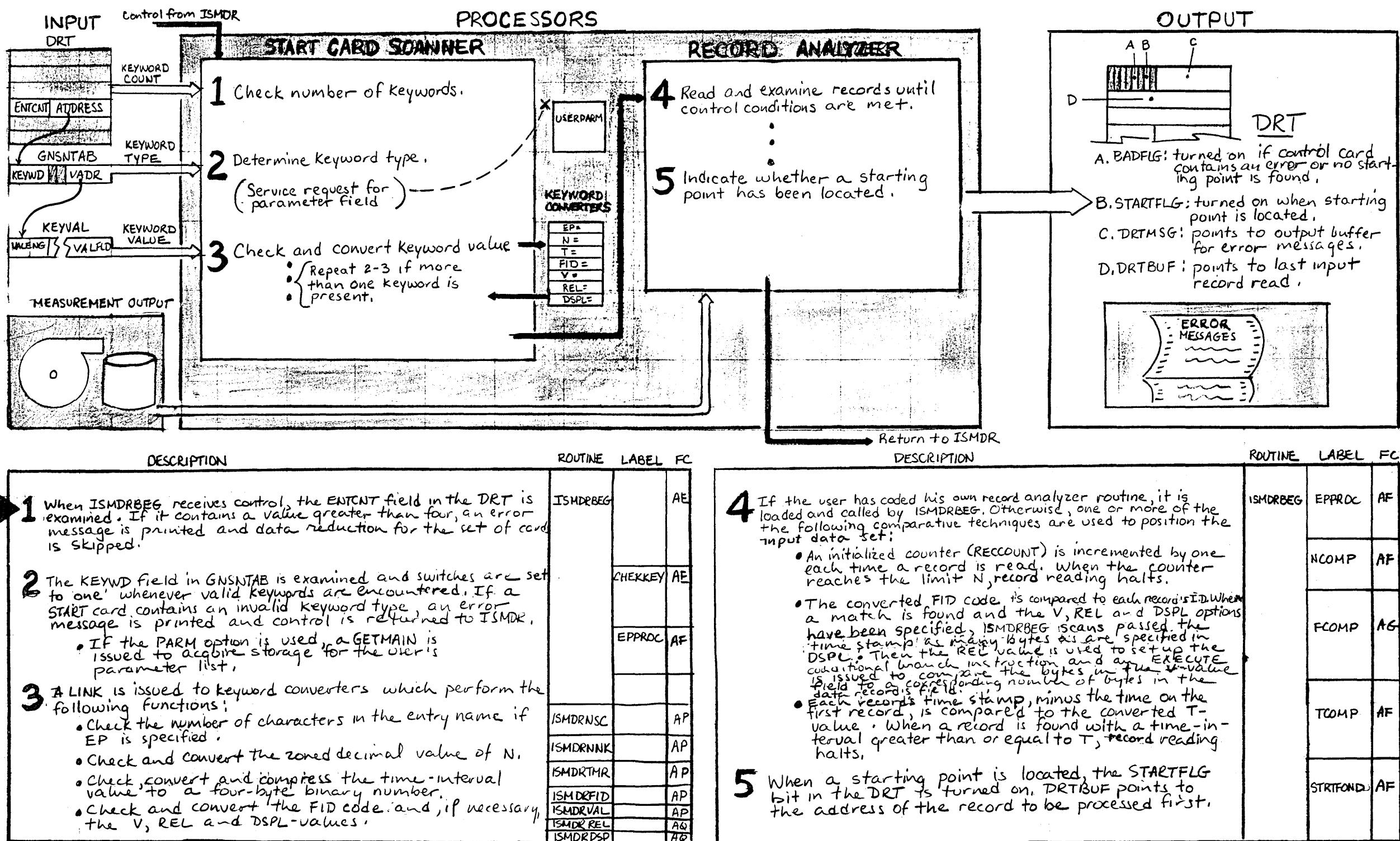
## 28.1 SCANNING DATA REDUCTION CONTROL INPUT-- NO CONTROL STATEMENTS



Note 1: When ISMDGNSN passes control back to ISMDR, register 1 contains the following:

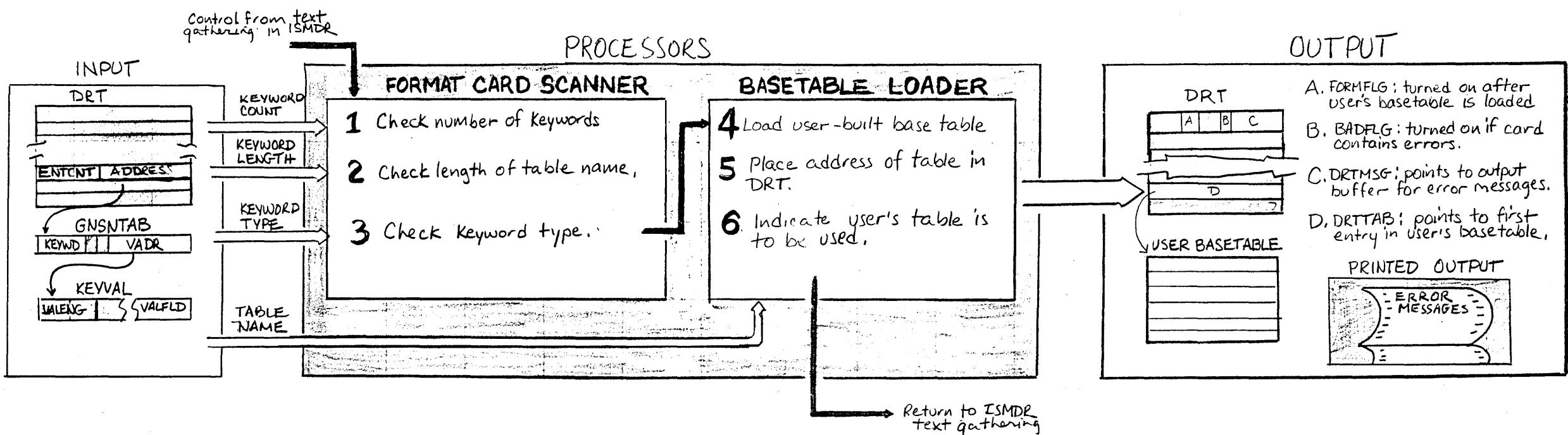


## 28.2 SCANNING DATA REDUCTION CONTROL INPUT-- CONTROL STATEMENTS PRESENT

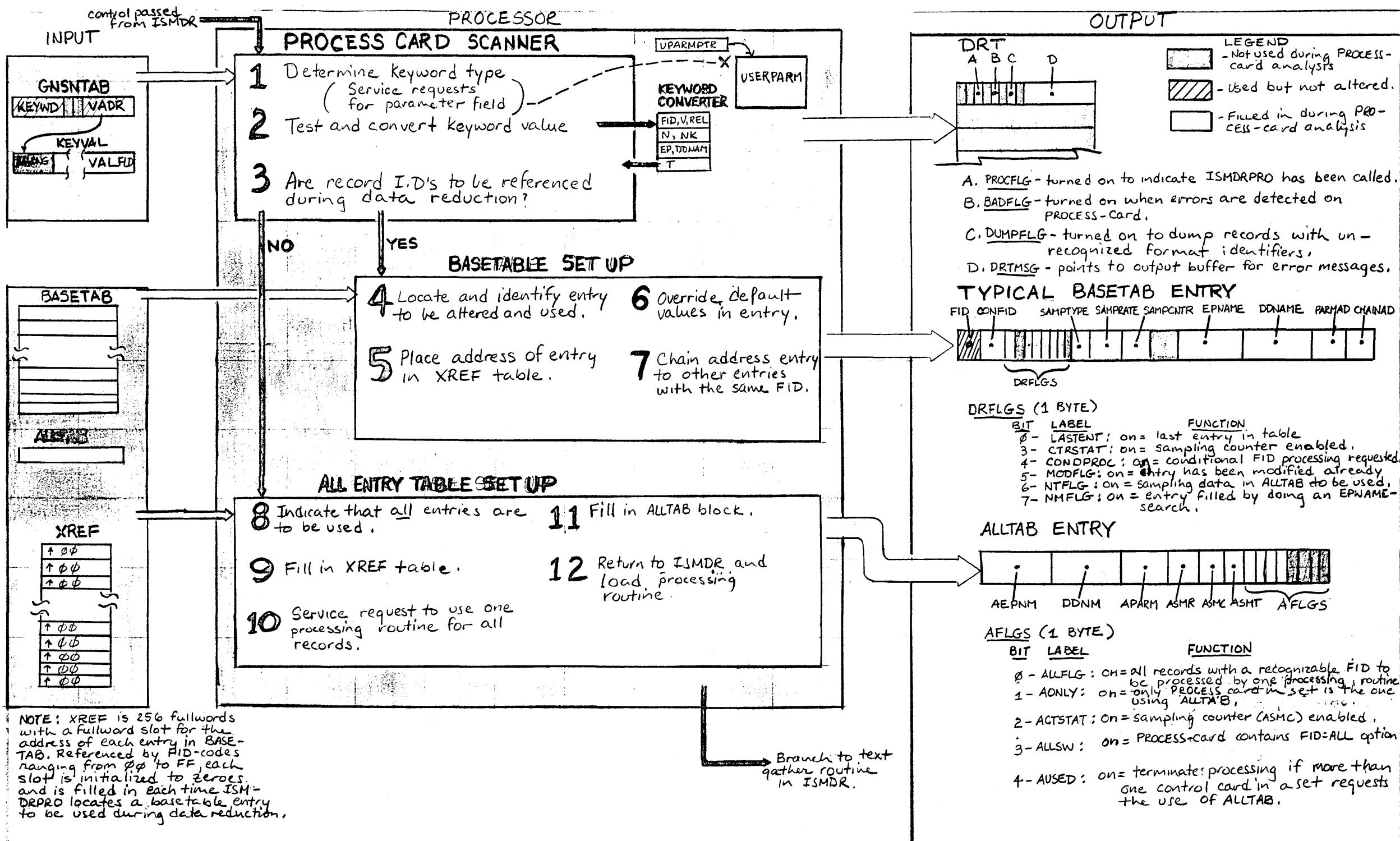


## 29 START CARD PROCESSING

IBM CONFIDENTIAL

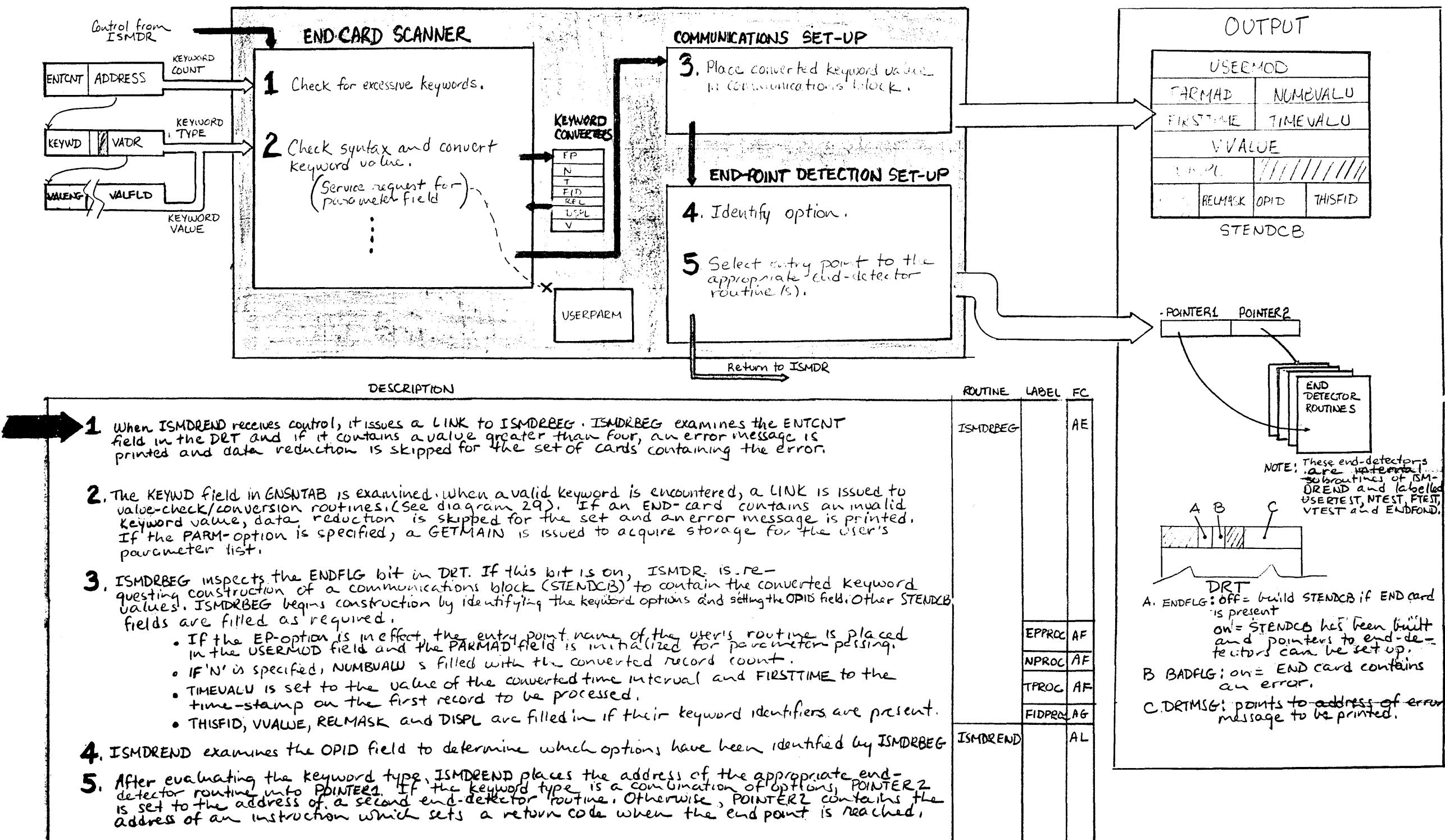


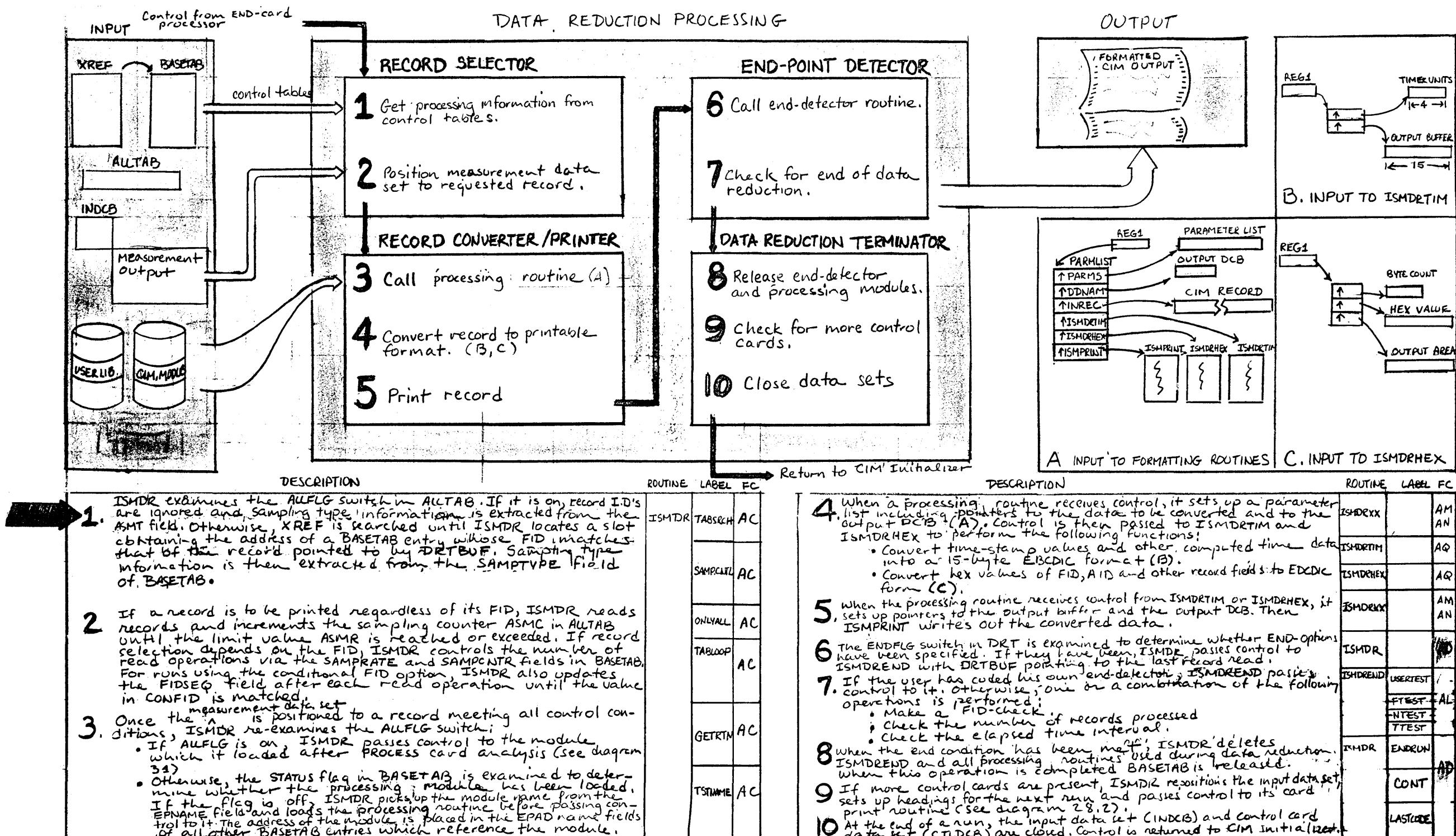
DESCRIPTION	ROUTINE	LABEL	FC	DESCRIPTION	ROUTINE	LABEL	FC
<p>1 The ENTCNT field in the DRT is examined. If it contains a value greater than one, an error message is printed and data reduction is skipped for the set of control cards.</p> <p>2 VALENG, the keyword length indicator is tested. If the keyword field contains more than eight characters, an error message is printed and data reduction is skipped.</p> <p>3 The KEYWD field in GNSNTAB is examined and if it does not contain the word 'TAB', an error message is printed and data reduction is skipped.</p>	ISMDR	FORMOPT	AB	<p>4 ISMDR moves the name of the user-built basetable to TABNAME and issues a LOAD to bring a copy of the table from SYS1.LINKLIB or a library defined by the user on a STEPLIB card.</p> <p>5 After the table is loaded, the address returned in register 0 is saved in the DRTTAB field of the DRT.</p> <p>6 The FORMFLG field in the DRT is turned on to indicate that the user's basetable has been loaded and is available for use.</p>	ISMDR	FORMOPT	AB



## 31 PROCESS CARD ANALYSIS (CONT.)

DESCRIPTION	ROUTINE	LABEL	FC
<p>1 When ISMDRPRO receives control, it initializes all pointers, counters, and switches to be used during PROCESS-card analysis. The KEYWD field in GNSNTAB is examined and one of the initialized switches in ISMDRPRO is flipped whenever a valid keyword type is encountered.</p> <ul style="list-style-type: none"> <li>If the user has specified the PARM option, a GETMAIN is issued for core storage. The address of this area is placed in UPARMPTR and all parameter information is transferred from the VAL field to USERPARM.</li> </ul>	ISMDRPRO	STRTSCRT	AH
		PMPPROC	AI
	ISMDRFID		AP
	ISMDRNSC		AP
	ISMDRNK		AP
	ISMDRTMR		AP
<p>2 ISMDRPRO issues a LINK to value-check/conversion routines which perform the following functions:</p> <ul style="list-style-type: none"> <li>Check and convert the FID-code.</li> <li>Check number and type of characters in VAL-field if EP or DDNAME options are used.</li> <li>Check and convert zoned decimal value of N or NK to binary.</li> <li>Check and convert time-interval value.</li> </ul>	ISMDRPRO	FIDSRCRT	AH
		TABSRCH	AI
<p>3 If the user has specified FID= code,[code], ISMDRPRO executes the instructions which locate and modify BASETAB entries(4-7). For FID=ALL,[NO,DUMP] ISMDRPRO turns on ALLSW and executes its ALTAB set-up instructions (8-12).</p> <p><b>BASETABLE SET UP</b></p>	ENTHERE	AI	
<p>4 BASETAB is searched for a FID code or EP name to match the one located in the VAL field of KEYVAL. When one is found, the MODFLG bit in DRFLGS is turned on to indicate the entry is to be used and cannot be altered by any other PROCESS-card in the set. (NOTE: If EP and FID appear on the card, the FID-code is used as a search argument and when a match is found in BASETAB the EPname is placed in the EPNAME field.)</p>	BLDBASE	AJ	
<p>5 The algorithm 'current FID code + 1' is used as an index into the XREF table. If the fullword slot contains zeroes, the address of the BASETAB entry is moved there. Otherwise, the address is placed in the CHAINAD field of the last entry in BASETAB to be created with the same FID code. (See 7).</p>	ENTHERE	AI	
<p>6 The initial values of SAMPTYPE, SAMPRATE, CONFID, PARMAD and DDNAME fields are overridden as requested. If the user has specified a time or record count sampling technique, the CTRSTAT flag in DRFLGS is turned on to indicate the sampling counter SAMPCNTR is to be used during data reduction. When the conditional FID option is used, the CONDPROC flag is turned on.</p>	PMPPROC	AI	
	DDNOK	AJ	
<p>7 Whenever a control card has a FID code matching the FID of an entry which has been altered already, ISMDRPRO creates a new entry at the tail-end of BASETAB and points the CHAINAD field of the old entry to the address of the new. The CHAINAD field of the new entry is zeroed out to indicate that it is the last entry belonging to this FID class.</p>	ISMDRPRO	NEWENT	AI
	SETMOD		AH
<p>8 The MODFLG field in every baseable entry is turned on. To indicate that ALLTAB contains the required sampling information, NTFLG is also turned on.</p>	BLDXREF		AH
<p>9 DRRTAB points to the address of the first entry in BASETAB. This address is placed in BPTR and moved to the corresponding FID slot in XREF. BPTR is incremented to the next baseable entry and the next corresponding FID slot in XREF is filled. This process continues until the address of every entry in BASETAB has been placed in XREF.</p>	TSTAUSED		AK
<p>10 The EP name is copied from the VAL field in KEYVAL to the AEPNM field of ALLTAB. Other fields in ALLTAB are assigned default values at this time. The ALLFLG bit is turned on after the AEPNM field has been set. The AUSED flag is also turned on. If more than one control card in the set requires the use of ALLTAB, the AUSED flag routes the request to the error printout routine and further processing of the set is terminated.</p>	DDNOK	AJ	
<p>11 Default values of DDNM, APARM, ASMR and ASMT are overridden as requested. If the user specifies a sampling rate, the ACTSTAT bit is turned on to enable the all entry sampling counter ASMC.</p>	PMPPROC	AI	
	CKAUSED		AK
	BLDAALL		AK
<p>12 When ISMDR receives control, it examines the ALLFLG bit. If this bit is on, ISMDR loads the user-specified processing routine.</p>	ISMDR	CLRXREF	AA





### 3.3. PERFORMING ACTUAL DATA REDUCTION

IBM CONFIDENTIAL