UNCLASSIFIED

AD NUMBER

AD489667

NEW LIMITATION CHANGE

TO

Approved for public release, distribution unlimited

FROM

Distribution authorized to U.S. Gov't. agencies and their contractors; Administrative/Operational Use; Aug 1966. Other requests shall be referred to Rome Air Development Center, Attn: EMLI, Griffiss AFB, NY 13440.

AUTHORITY

RADC, USAF ltr, 17 Sep 1971

THIS PAGE IS UNCLASSIFIED

489657

RADC-TR-66-474, Volume II
Final Report

RELIABILITY CENTRAL AUTOMATIC DATA PROCESSING SUBSYSTEM

Design Specification Report (Cont'd)

Auerbach Corporation

TECHNICAL REPORT NO. RADC-TR-66-474
August 1966

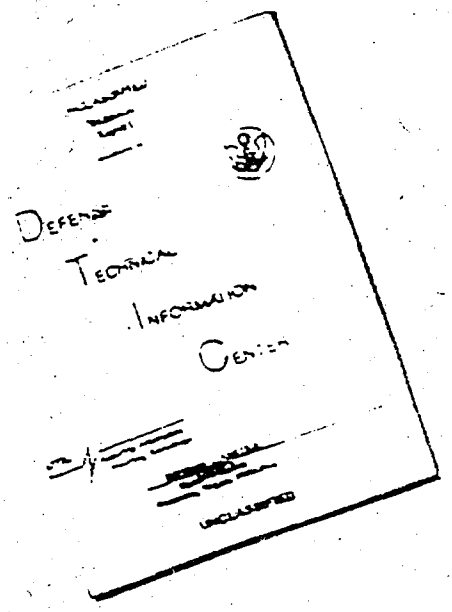
This document is subject to special export controls and each transmittal to foreign governments or foreign nationals may be made only with prior approval of RADC (EMLI), GAFB, N.Y. 13440.

Rome Air Development Center
Research and Technology Division
Air Force Systems Command
Griffiss Air Force Base, New York

When US Government drawings, specifications, or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded, by implication or otherwise, as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacturer, use, or sell any patented invention that may in any way be related thereto.

Do not return this copy. Retain or destroy.

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

REPRODUCED FROM
BEST AVAILABLE COPY

RELIABILITY CENTRAL AUTOMATIC DATA PROCESSING SUBSYSTEM

Design Specification Report (Cont'd)

Auerbach Corporation

This document is subject to special export controls and each transmittal to foreign governments or foreign nationals may be made only with prior approval of RADC (EMLI), GAFB, N.Y. 13440.

MAINLINE ENGINEERING MAPLE SHADE, N.J.
9/23/66 240

FOREWORD

This three-volume final technical report was prepared by the Auerbach Corporation, Philadelphia 3, Pennsylvania under Contract AF 30(602)-3620, Project 5519. It is identified by the contractor as 1280-TR. The authors were Dr. J. Sable, W. Crowley, M. Rosenthal, S. Forst, and P. Harper. The Rome Air Development Center Project Engineer was Casper DeFiore, EMIID.

This technical report contains information embargoed from release to the Clearinghouse for Federal Scientific and Technical Information, Department of Commerce, by AFR 400-10.

This technical report has been reviewed and is approved.

Approved:
FRANK J. TOMAINI
Chief, Information Processing Branch

Approved:
JAMES J. DIMEL, Colonel, USAF
Chief, Intel & Info Processing Division

FOR THE COMMANDER
IRVING J. GABELMAN
Chief, Advanced Studies Group

ABSTRACT

This is Volume II of the three-volume final report produced for the Rome Air Development Center (RADC) under Contract AF30(602)-3926. Volumes I and II constitute the Design Specification Report for the Automatic Data Processing Subsystem (ADPS) of Reliability Central, known as Data Manager-1 (DM-1). Volume III is a survey of major, computer-oriented, on-line information and fact retrieval systems.

This volume contains a detailed description of the data pool and directories, and the technical documentation and flow charts for the system program and jobs. It presents the system components which provide for the following functions of DM-1:

(1) Programming Services. The DM-1 Service Package acts as an intermediary between the data pool and running programs which need to access and store data.

(2) System Supervision. The Request Processor responds to requests for the execution of jobs in the DM-1 library. Job Manager supervises the exchanges of control among the operational, system, and user programs.

(3) Job Library Maintenance. A set of system jobs provides for entering programs with their parameter descriptions into the library, describing jobs as combinations of programs and jobs, deleting programs and job descriptions from the library, and displaying job descriptions from the library for review.

(4) Directory and Data Manipulation. A set of system jobs provides for adding and deleting item structure definitions in the directory, and adding, deleting, replacing, and modifying data in the data pool.

(5) User Utility Support. A set of system jobs provides for querying the data pool, developing new data items from combinations of existing data, and displaying the results of queries and the values of data pool items.

(6) Dialogue Procedure A set of system jobs which allow the user to approach the DM-1 data pool without knowing the complete specification for the information

TABLE OF CONTENTS

Paragraph	Title	Page
	SECTION I. INTRODUCTION	
	SECTION II. SYSTEM DATA POOL	
2.1	ITEM STRUCTURES	2-1
2.2	DATA POOL	2-4
2.3	ITEM LIST	2-5
2.4	TERM LIST	2-7
2.5	TERM ENCODING TABLE	2-7
2.6	INDEX TABLES	2-8
2.7	LINKAGE TABLE	2-10
2.8	USER ACCESS/MODIFICATION RIGHTS TABLES	2-11
2.9	PROGRAM AND JOB DESCRIPTION LIBRARY	2-13
2.10	DATA SEGMENTS	2-16
2.10.1	Segment Head	2-16
2.10.2	Segment Index	2-17
2.10.3	Segment Body	2-18
2.10.4	Partition between Segments	2-18
2.11	SEGMENT NAME LIST	2-18
2.12	DIRECTORY FORMAT	2-20
	SECTION III. DM-1 SERVICE PACKAGE	
3.1	BOOKKEEPING	3-2
3.1.1	Level Pushdown List	3-2
3.1.2	Access Parameter Table	3-5
3.1.3	Item List Table	3-5
3.1.4	Segment Index	3-7
3.1.5	Missing Data Indicators	3-7
3.1.6	Bookkeeping Service Routines	3-8
3.2	SEGMENTATION	3-9
3.3	TRANSLATION	3-11
3.4	MAINTENANCE	3-12
3.5	BASIC USER SERVICES	3-13
3.6	COMPOUND USER SERVICES	3-14

TABLE OF CONTENTS (CONTD.)

PARAGRAPH	TITLE	PAGE
3.7	DETAILED DESCRIPTION OF SERVICE ROUTINES	3-16
3.7.1	Fetch Segment	3-17
3.7.2	Name Segment	3-19
3.7.3	Locate IL Entry	3-22
3.7.4	Step Lists	3-25
3.7.5	Discount Item	3-27
3.7.6	Step Item	3-30
3.7.7	Define Field	3-33
3.7.8	Get Next SX	3-35
3.7.9	Skip Item	3-37
3.7.10	Define Segment	3-39
3.7.11	Read	3-41
3.7.12	Write	3-45
3.7.13	Locate Item	3-49
3.7.14	Seek	3-51
3.7.15	Seek with Copy	3-54
	SECTION IV. JOB SUPERVISOR	
4.1	DYNAMIC TASK LIST	4-1
4.2	REQUEST PROCESSOR	4-2
4.3	JOB MANAGER	4-5
4.4	JOB EXTENSION	4-6
4.5	TEMPORARY ITEMS REQUIRED BY JOB SUPERVISOR	4-8
4.5.1	Reserved Core Items	4-9
4.5.2	Request File	4-9
4.5.3	Request Directory	4-11
4.5.4	Scratch items	4-11
4.6	DETAILED DESCRIPTION OF JOB SUPERVISOR PROGRAMS	4-11
4.6.1	RQ Bootstrap	4-12
4.6.2	Task Terminate	4-16
4.6.3	RQ Scan	4-18
4.6.4	Specify Item	4-22
4.6.5	Update DTL	4-27
4.6.6	RQ Terminate	4-31
4.6.7	JX Processor (Resident Portion)	4-34
4.6.8	JX Scan	4-37

TABLE OF CONTENTS (CONTD.)

PARAGRAPH	TITLE	PAGE
	SECTION V. JOB LIBRARY MAINTENANCE JOBS	
5.1	PROGRAM ENTRY	5-1
5.1.1	Functional Description	5-2
5.1.2	Inputs	5-3
5.1.3	Results	5-4
5.1.4	Directories Used	5-4
5.1.5	Services Used	5-4
5.1.6	Jobs Used	5-4
5.1.7	Method of Operation	5-5
5.2	JOB DESCRIPTION	5-32
5.2.1	Functional Description	5-35
5.2.2	Inputs	5-36
5.2.3	Results	5-37
5.2.4	Directories Used	5-37
5.2.5	Services Used	5-37
5.2.6	Jobs Used	5-38
5.2.7	Method of Operation	5-38
5.3	JOB AND PROGRAM DELETION	5-68
5.3.1	Functional Description	5-68
5.3.2	Inputs	5-68
5.3.3	Results	5-68
5.3.4	Directories Used	5-68
5.3.5	Services Used	5-69
5.3.6	Jobs Used	5-69
5.3.7	Method of Operation	5-69
5.4	DISPLAY JOB DESCRIPTION	5-84
5.4.1	Functional Description	5-84
5.4.2	Inputs	5-84
5.4.3	Results	5-84
5.4.4	Directories Used	5-84
5.4.5	Services Used	5-84
5.4.6	Jobs Used	5-85
5.4.7	Method of Operation	5-85
	SECTION VI. DATA POOL MAINTENANCE JOBS	
6.1	ITEM DEFINITION MANIPULATION	6-2
6.1.1	Define Item	6-3
6.1.2	Delete Definition	6-11
6.1.3	Delete Node	6-15
6.1.4	Renovate Item	6-18

TABLE OF CONTENTS (CONTD.)

PARAGRAPH	TITLE	PAGE
6.2	DATA MANIPULATION	6-25
6.2.1	Add Data	6-26
6.2.2	Replace Data	6-39
6.2.3	Modify Data	6-43
6.2.4	Update Data	6-47
6.2.5	Delete Data	6-52
6.2.6	Compress File	6-62
6.3	INDEXING	6-71
6.3.1	Index	6-71
6.3.2	Remove Index	6-79
6.4	LINKAGE	6-82
6.4.1	Link	6-82
6.4.2	Delete Link	6-87
6.5	OTHER MAINTENANCE JOBS	6-91
6.5.1	Data Security	6-91
6.5.2	External to Internal Data Conversion	6-93
	SECTION VII. UTILITY JOBS	
7.1	QUERY	7-1
7.1.1	Functional Description	7-2
7.1.2	Inputs	7-2
7.1.3	Results	7-2
7.1.4	Directories Used	7-3
7.1.5	Services Used	7-3
7.1.6	Jobs Used	7-3
7.1.7	Method of Operation	7-4
7.2	CONDITIONAL REFORMAT	7-91
7.2.1	Functional Description	7-91
7.2.2	Inputs	7-91
7.2.3	Results	7-91
7.2.4	Directories Used	7-91
7.2.5	Services Used	7-91
7.2.6	Jobs Used	7-92
7.2.7	Method of Operation	7-92
	SECTION VIII. THE INPUT SCANNER	
8.1	ACTION-GRAPHS	8-1
8.2	ADDRESS LISTS	8-3

TABLE OF CONTENTS (CONTD.)

PARAGRAPH	TITLE	
8.3	INSCAN	
8.3.1	Functional Description	
8.3.2	Inputs	
8.3.3	Results	
8.3.4	Indicator Cells Required	
8.3.5	Tables Read	
8.3.6	Tables Modified	
8.3.7	Method of Operation	
8.4	INSCAN EXAMPLES	
	SECTION IX. THE DIALOGUE PROCEDURE	
9.1	INTRODUCTION	
9.1.1	Need for a Dialogue	
9.1.2	The Dialogue Principle	
9.2	THE DIALOGUE IN DM-1	
9.2.1	Use of the Data-Pool Structure	
9.2.2	Object of the Dialogue Procedure	
9.2.3	Uses of the Dialogue Procedure	
9.3	THE DIALOGUE PROCEDURE	
9.3.1	Structure of the Dialogue Query	
9.3.2	Console Dialogue Example	
9.3.3	Display Development in the Example	
9.4	EXPLANATION OF LOGICAL DESIGN	
9.4.1	Phase 1 Display and Select	
9.4.2	Phase 1: Termination	
9.4.3	Phase 2: Display and Select	
9.4.4	Phase 2: Value Selection	
9.4.5	Phase 2: Condition Development	
9.4.6	Phase 2: Termination	
9.5	DETAILED DESIGN	
9.5.1	Phase 1: Initialization and DWT Setup	

Best Available Copy

TABLE OF CONTENTS (CONTD.)

PARAGRAPH	TITLE	PAGE
9.5.2	Phase 1: Selection Diagnosis	9-46
9.5.3	Phase 1: Console Display	9-48
9.5.4	Phase 1: Homograph Dialogue – I	9-49
9.5.5	Phase 1: Homograph Dialogue – II	9-50
9.5.6	Phase 1: Homograph Dialogue – III	9-50
9.5.7	Phase 1: Option	9-51
9.5.8	Phase 1: Transfer or Delete	9-52
9.5.9	Phase 1: Termination	9-52
9.5.10	Phase 2: Initialization and DWT Setup	9-53
9.5.11	Phase 2: Selection Diagnosis	9-54
9.5.12	Phase 2: Console Display	9-55
9.5.13	Phase 2: Option	9-56
9.5.14	Phase 2: Terminal Selection – I	9-56
9.5.15	Phase 2: Terminal Selection – II	9-57
9.5.16	Phase 2: Termination – I	9-58
9.5.17	Phase 2: Termination – II	9-59
9.5.18	Retrieve Node IL Entry (Subroutine)	9-59
9.5.19	Create ICC's and Obtain IL Entries (Subroutine)	9-61
9.5.20	Setup Output Buffer – I (Subroutine)	9-62
9.5.21	Setup Output Buffer – II (Subroutine)	9-63
9.5.22	Display (Delete) Selected Items – I (Subroutine)	9-63
9.5.23	Display (Delete) Selected Items – II (Subroutine)	9-64
9.5.24	Set NIV = Number Indexed Values (Subroutine)	9-65
9.5.25	Display Eight Ranges – I (Subroutine)	9-66
9.5.26	Display Eight Ranges – II (Subroutine)	9-66
9.5.27	Display Eight Ranges – III (Subroutine)	9-67
9.6	EXPLANATION OF TERMS AND PARAMETERS	9-95
9.6.1	General	9-95
9.6.2	Display Work Table (DWT)	9-95
9.6.3	Display Work Table (DWT1)	9-95

PARAGRAPH	TITLE	PAGE
9.6.4	Want List (WL)	9-97
9.6.5	Condition Value Table (CVT)	9-97
9.6.6	Control Word (CW)	9-97
9.6.7	Option Control Word (OCW)	9-98
9.6.8	Value Range Table (VRT)	9-98
9.6.9	Term Encoding Table (TET)	9-98
9.6.10	Item List (IL)	9-98
9.6.11	Term List (TL)	9-98
9.6.12	Field Value Table (FVT)	9-99
9.7	TIMING ESTIMATES	9-107
9.8	TECHNICAL NOTE ON POPULATION ESTIMATES	9-113
9.8.1	Population Estimate With a One-Level File	9-114
9.8.2	The Situation With Multilevel Structures	9-115

APPENDIX A. RELIABILITY CENTRAL TEST OPERATION

APPENDIX B. RELIABILITY CENTRAL SCHEDULED OUTPUTS

LIST OF ILLUSTRATIONS

FIGURE	TITLE	PAGE
2-1	System Data Pool	2-3
2-2	An Indexed Field in the Data Base	2-9
4-1	Dynamic Task List	4-3
4-2	Overview of Request Processor	4-4
4-3	Static Task List	4-4
4-4	Flow of Task in a Job	4-7
4-5	Request File	4-10
5-1	The Program Entry Job	5-6
5-2	Unit Job, Internal Job Description	5-19
5-3	Job Description Job, Internal Job Description	5-38
5-4	Binding Job, Internal Job Description	5-50
5-5	Job Deletion Job, Internal Job Description	5-70
5-6	Auxiliary Deletion Job, Internal Job Description	5-80
7-1	Query Job, Internal Job Description	7-4
7-2	Typical Data Structure	7-8
7-3	Set Calculus Definitions	-10
7-4	Set Calculus Theorems	7-10
7-5	Conditional Search, Internal Job Description	7-13
7-6	Reformat Job, Internal Job Description	7-54
7-7	Conditional Reformat Job, Internal Job Description	7-93
8-1	An Action-Graph	8-2
8-2	Shape Codes	8-3
8-3	The Address Lists	8-4
8-4	Inscan Flow Chart	8-5
8-5	Inscan Example	8-8
8-6	Infix to Suffix Operator Translator	8-10
9-1	Structure of Purchasing Data Base	9-16
9-2	Display and Select	9-39
9-3	Phase I: Termination	9-40
9-4	Phase 2: Display and Select	9-41
9-5	Phase 2: Value Selection	9-42
9-6	Phase 2: Condition Development	9-43

LIST OF ILLUSTRATIONS

Figure	Title	Page
9-7	Phase 2: Termination	9-44
9-8	Phase 1: Initiatization and DWT Setup	9-68
9-9	Phase 1: Selection Diagnosis	9-69
9-10	Phase 1: Console Display	9-70
9-11	Homograph* Dialogue – I	9-71
9-12	Phase 1: Homograph Dialogue – II	9-72
9-13	Phase 1: Homograph Dialogue – III	9-73
9-14	Phase 1: Option	9-74
9-15	Phase 1: Transfer or Delete	9-75
9-16	Phase 1: Termination	9-76
9-17	Phase 2: Initialization and DWT Setup	9-77
9-18	Phase 2: Selection Diagnosis	9-78
9-19	Phase 2: Console Display	9-79
9-20	Phase 2. Option	9-80
9-21	Phase 2: Terminal Selection – I	9-81
9-22	Phase 2: Terminal Selection – II	9-82
9-23	Phase 2: Termination – I	9-83
9-24	Phase 2: Termination – II	9-84
9-25	Retrieve Node IL Entry	9-85
9-26	Create ICC's and Obtain IL Entries	9-86
9-27	Set Up Output Buffer – I	9-87
9-28	Set Up Output Buffer – II	9-88
9-29	Display (Delete) Selected Items – I	9-89
9-30	Display (Delete) Selected Items – II	9-90
9-31	Set NIV = Number Indexed Values	9-91
9-32	Display Eight Ranges – I	9-92
9-33	Display Eight Ranges – II	9-93
9-34	Display Eight Ranges – III	9-94

LIST OF TABLES

TABLE	TITLE	PAGE
1-1	DESCRIPTION OF DM-1 COMPONENTS	1-3
2-1	DERIVATION OF LOGICAL NAME FROM ITEM POSITION	2-2
2-2	ITEM TYPES	2-2
2-3	STRUCTURE OF THE DATA POOL	2-21
3-1	FORMAT OF LEVEL PUSHDOWN LIST	3-3
3-2	DEFINITION OF PURCHASING ITEM	3-4
3-3	LPL FOR PRICE FIELD	3-5
3-4	LPL FOR YEAR FIELD	3-5
3-5	ITEM LIST TABLE	3-6
7-1	LOGICAL CONDITIONS	7-7
9-1	DIRECTORY SCHEMATIC	9-31
9-2	PHASE 2: TERMINATION	9-60
9-3	ABBREVIATIONS/NAMES USED IN FLOW CHARTS	9-96
9-4	DWT (DISPLAY WORK TABLE)	9-100
9-5	DWT 1 (DISPLAY WORK TABLE 1)	9-101
9-6	WL (WANT LIST)	9-102
9-7	CVT (CONDITION VALUE TABLE)	9-103
9-8	CONTROL WORD	9-104
9-10	VRT (VALUE RANGE TABLE)	9-105
9-11	TERM ENCODING TABLE (TET)	9-106
9-12	TIMING ESTIMATES	9-112

SECTION I. INTRODUCTION

Data Manager-1 (DM-1), a computer software system, is designed for the Rome Air Development Center to operate as the Automatic Data Processing Subsystem (ADPS) of Reliability Central. DM-1 consists of a data pool, a series of system programs and jobs that control the system and provide for the management and manipulation of data, and provisions for tailoring the system to the requirements of an application by the specification of descriptive and declarative information and the addition of programs to the DM-1 complement.

The data pool contains a data base, a set of directories which include a library of jobs and programs, and logical provisions for retaining work data and scratch data under system control. The data pool is described in detail in Section II. The structure of the data pool is described, along with the rules for structuring the application data base and the work items. Each of the directory elements is defined in terms of the structure and its uses in the system. The DM-1 approach to segmentation, which provides for the storage of the data and directories on a variety of random-access storage devices, is also explained in Section II.

The operational programs of the DM-1 system are presented in Sections III and IV. Section III contains the technical documentation for the DM-1 Service Package. This consists of a set of system routines which function like an I/O control system. The routines respond to requests from programs for the storage and access of data in the data pool. Section IV contains the technical documentation for the DM-1 Supervisor. This consists of a Request Processor and a Job Manager which work together to control the execution of system and user jobs.

The programs of the Service Package and the Supervisor are presented in a standard format which is used throughout this volume. The standard format contains paragraphs in the following order:

(1) Functional Description,

(2) Inputs,

(3) Results,

(4) Directories Used,

(5) Services Used,

(6) Jobs Used,

(7) Method of Operation (followed by flow chart or charts)

Sections V, VI, and VII contain the technical documentation for the system jobs which perform the job-library maintenance, data-pool maintenance and user-utility functions of the system. Section VIII contains the description of the Input Scan Routine which is used by the language processors to perform a syntax-directed scan of text in one of the system languages. These jobs and routines are presented in the standard format.

Table 1-1 describes the DM-1 components and explains the role played by each component in the system. Many of these components contain subelements at a lower level of detail. For example, the Service Package consists of many small routines and the system jobs are frequently composed of a series of task programs. These details are presented with the technical documentation in the appropriate sections.

TABLE 1-1. DESCRIPTION OF DM-1 COMPONENTS

Function	Name	Description
Programming Services	Service Package	Acts as intermediary between running programs and the data in the data pool. Handles the mechanics of interpreting the data in terms of the directory, segment packing and unpacking, and conversions between the machine-independent segment format and computer-oriented format. Transmits data between the data pool and the program's buffer.
System Supervision	Request Processor	Prepares a request record for job execution.
	Job Manager	Manages transitions of control among operational system and user programs.
Job Library Maintenance	Program Entry	Adds a new program to the job library with a description of its input and output items.
	Job Description	Adds a new job, consisting of a sequence of defined jobs and entered programs, to the job library.
	Job Deletion	Deletes a job, defined in an earlier program entry or job description, from the job library.
	Display Job Description	Displays the components of a job, showing the relationships among their input and output parameters.
Item Structure Manipulation	Define Item	Adds the definition of a new item to a node in the data-pool structure.
	Delete Definition	Deletes the item definition at a specified node in the data-pool structure.
	Delete Node	Deletes the item definition at a specified node in the data-pool structure, eliminates the node, and adjusts internal item identifiers.
	Renovate Item	Deletes all unused nodes subsumed by a specified item as in Delete Node.
Data Manipulation	Convert External Data	Converts data from an external medium into an item in the data pool.
	Convert IDL	Converts data in the Internal Data Language into an item in the data pool.

TABLE 1-1. DESCRIPTION OF DM-1 COMPONENTS (Contd.)

Function	Name	Description
Data Manipulation	Add Data	Maps data from a source item into the set of empty data positions defined by a name and a condition.
	Replace Data	Replaces the data in the data positions defined by a name and a condition with the data in a source item.
	Modify Data	Calculates a value for a designated attribute and replaces the values of the field, defined by its name and a condition, with the calculated value.
	Update Data	Replaces values in the records of a master file with values obtained from a transaction file identifying the records to be updated.
	Delete Data	Deletes the values for a named item at the data positions defined by a condition.
	Compress File	Eliminates null records (from earlier deletes) from a file containing indexed fields, and updates the record lists in the index tables.
Storage and Retrieval Efficiency	Index	Creates an inverted file relating the values taken on by a field to the record numbers of the data positions where the values occur.
	Remove Index	Eliminates the inverted file for a previously indexed field.
	Link	Establishes a conditional, logical link between data on separate stems of the data-pool tree structure.
	Delete Link	Eliminates the logical connection created by a previous link request.
Data Security Control	Add User	Establishes and identifies a new user of DM-1 services and assigns his security restriction level for data access and modification.
	Delete User	Eliminates a user from the active list.
	Add Access Rights	Defines specific items, outside the user's security restriction level, which may be accessed by a specific user.

TABLE 1-1. DESCRIPTION OF DM-1 COMPONENTS (Contd.)

Function	Name	Description
Data Security Control	Delete Access Rights	Eliminates specific access rights previously assigned to a user.
	Add Modification Rights	Defines specific items, outside the user's security restriction level, which may be modified by a specific user.
	Delete Modification Rights	Eliminates specific modification rights previously assigned to a user.
User Utility	Query	Retrieves and displays the data relevant to a user's information need The need is expressed by naming the attributes which describe the object of interest and providing an arbitrary logical condition which specifies the characteristics of the individual objects of interest.
	Conditional Reformat	Retrieves selected data, restructures it, and maps it into an output item in the data base or work area. The desired data is specified as with Query.
	Display	Displays the data values for a specified set of items under the control of a format statement.
	Dialogue Query	Guides a user to specify the attributes and condition that define his information need, through a multistage dialogue with the system.
System Utility	Conditional Search	Develops a record number list which defines the item positions specified by an arbitrarily complex logic condition.
	Input Scan Routine	Scans a stream of alphanumeric characters in a language defined by an action-graph. Performs a syntax check and permits language processors to execute appropriate routines at action points in the scan.

SECTION II. SYSTEM DATA POOL

DM-1 is a data processing system, and, in common with all other such systems, its elements are programs and data. This section is devoted to a description of the data, and the succeeding sections describe the programs.

The most undiscriminating term used for data is the data pool. The data pool includes the data base, working and scratch data, and the directories. A structural diagram of the data pool, before any data has been added to the data base, is illustrated in Figure 2-1. The ovals and rectangles show the primary items which subsume all other items of the data-pool structure.

This section will describe item structures, segment formats for the physical storage of data, and complete substructures for each directory item named in Figure 2-1.

2.1 ITEM STRUCTURES

Item is a generic term in the data description language of DM-1. Figure 2-1 provides term names (e.g., Data Pool) for nineteen items, and it partially defines the logical structure of the data pool. The method of deriving logical names from the

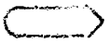
relative position of the item within the structure is also illustrated. Several examples are given in Table 2-1.

TABLE 2-1. DERIVATION OF LOGICAL NAME FROM ITEM POSITION

Term Name	Logical Name Item Class Code (ICC)
Directory	1.2
Job List	1.2.10
Scratch Area	1.4

Figure 2-1 also introduces two of the specific item types, the statement and the file. The complete list of item types is shown in Table 2-2.

TABLE 2-2. ITEM TYPES

Type	Abbreviation	Symbol
Statement	S	
File	F	
Record	R	
Field	f	
Null Node	N	
Source Link	L	
Target Link	L	

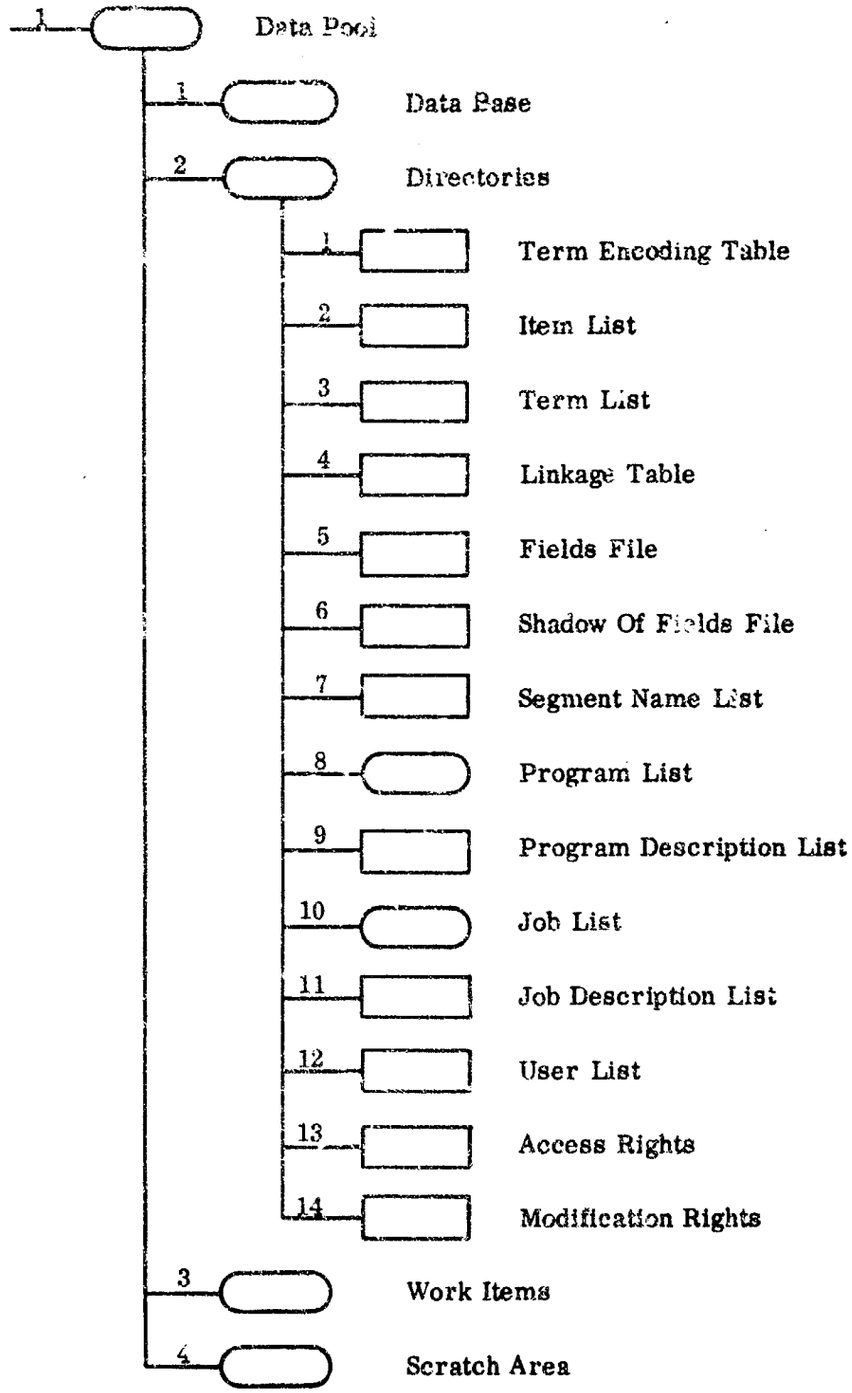


Figure 2-1. System Data Pool

Fields are further subdivided by type as follows:

alphanumeric	(A)
floating point	(E)
integer	(I)
decimal	(D)
octal	(O)
binary	(B)
B-S format	(H)

An item definition is a description of a data-pool item with its subsumed items. In computer input form, the item definition is called an item image. The relationship of items in a structure is quite flexible, and there are certain rules which must be observed.

(1) A record is the only item which may be directly subsumed by a file.

(2) Link items must subsume fields.

(3) Fields and null nodes are terminal items (they subsume nothing).

2.2 DATA POOL

The item identified as the data pool in Figure 2-1 has four subsumed items that are identified as statements.

The data base is the repository for the great mass of data which is to be retrieved and maintained by DM-1. The structure is defined only through the data-pool maintenance jobs described in Section VI. For speed of retrieval, this data may be indexed and linked.

Work items may be written by user programs. Indexing and linkage are not permitted. This section of the data pool is designed to receive outputs written by a job which must be preserved until another job is run, but after which the data is of no further use. Queries will be satisfied, while this data exists, but since the data may not be indexed, random retrieval will be slow.

The scratch area is even more temporary. It is used for intermediate data in the course of executing a job. At the termination of a job request, both the data and its definition are destroyed.

All of the data in these three, separate, logical sections of the data pool is defined in the directories, and all data (including the directories) is formatted into data segments as described in Paragraph 2.10.

2.3 ITEM LIST

When items are defined, all of the information provided in the definition, except for term name and units, is stored in the Item List. No data can be stored or retrieved without the Item List to serve as a key. The Item List is a file ordered by the logical name, the ICC, of its items. The ICC is not included in the records, because it can be calculated. The Item List records contain nine fields which provide a definition of the structure of the data pool and a set of system parameters describing each item. The Item List file has the following structure:

```
ITEM LIST, F
    RESERVED, B, 10
    SRL - ACCESS, 0, 1
    SRL - MODIFICATION, 0, 1
    INDEX CODE, B, 2
    ITEM TYPE, B, 6
    OPTION CODE, B, 1
    ITEM SIZE, I, 11
    NO DATA FLAG, B, 2
    RECORD NUMBER, I, 16
```

The fields of the Item List are described as follows:

(1) Reserved. This is a ten-bit field that is reserved for future use.

(2) SRL-Access. This is a three-bit field which specifies the security restriction level of the item for access. The code $\emptyset$ stands for an unrestricted item, and the code 6 is the highest restriction. To access an item without special access rights, a user must have an SRL greater than the SRL of the item.

(3) SRL-Modification. This is a three-bit field that works like the preceding field for modification of the item by a user.

(4) Index Code. This is a two-bit field which specifies the kind of indexing associated with a field item. The codes have the following meanings:

00 - not indexed
01 - indexed by list
10 - indexed by range
11 - indexed by all values

(5) Item Type. This is a six-bit field which is coded to represent the item type. The codes correspond to the following item types:

000000 - null node
000001 - statement
000010 - file
000011 - record
000100 - source link
000101 - target link
100000 - floating point field
100001 - integer field
100011 - octal field
100100 - decimal field
100110 - B-S format field
101001 - alphanumeric field
110001 - binary field

(6) Option Code. This is a one-bit field. It contains a Ø if the item is required, or a 1 if the item is optional.

(7) Item Size. This is an eleven-bit field. It contains a sign and ten bits which represent item sizes of 1 through 1023. A size of Ø means that the item is variable in length.

(8) No Data Flag. This is a two-bit field which is used by the system to specify whether a definition has corresponding data or not.

(9) Record Number. This is a sixteen-bit field. It is used as a record number to associate an indexed field with the corresponding Field Value Table and to associate link items with the appropriate record of the Linkage Table.

Item List segments follow the data segment conventions (see Paragraph 2.10) except for the following special requirements.

(1) No record may be split across two segments.

(2) An Item Goal Code (IGC), required for the calculation of ICC's for the records of the segment, must be included within the segment.

(3) If Item List segments are to be processed as data, a method of identifying the segments as data (e.g., 1.2.2.R) needs to be provided. The special identification of Item List segments is a prefix of 1.0.2 followed by the ICC of the first Item List entry within the segment (see Paragraph 2.11.1).

2.4 TERM LIST

When items are defined, the information provided in the item definition is split between the Item List and the Term List. Term names and unit codes (for numeric fields) are retained in the Term List so they will not encumber the Item List which deals with logical names rather than term names.

The Term List is a file ordered by the logical name, the ICC, of its items. Its records are always paired with Item List records so that the system can retrieve the term name from the logical name whenever this is necessary for communication with a system user. The format of the Term List is as follows:

```
TERM LIST, F
    TERM NAME, A, V
    UNITS, B, 6
    RESERVED, I, 18
```

Term List segments are completely standard data segments. They are identified by the prefix 1.2.3 followed by the Item List record number and a .1 for Term Name or .2 for Units.

2.5 TERM ENCODING TABLE

The Term Encoding Table contains the term names from the Term List arranged in alphabetic order by term name and equated to the logical names (ICC) assigned to each term.

The Term Encoding Table (TET) is an ordered file maintained in ascending order by term name. Its format is as follows:

```
TERM ENCODING TABLE, F, ORDERED (1)
    TERM NAME, A, V
    ICC FILE, F
        ICC, H, V
```

A file of ICC's is equated to each term name because the same term name may be used more than once in the data pool. Sometimes names will be intentionally repeated in several different structures because the corresponding items have the same meaning. At other times, term names are homographs (words spelled the same but with different meanings) and the user will have to use qualifiers (term names for parent items) to distinguish between them.

TET segments follow the data segment conventions except for the following special requirements.

(1) No record of the TET file may be split across two segments.

(2) If TET segments are to be processed as data, the record identifiers are 1.2.1.R. When a service routine accesses the TET segments, it uses as an identifier a prefix of 1.0.1 suffixed by the term name. This means that segments of the TET have dual identifiers in the Segment Name List (see Paragraph 2.11.1)

2.6 INDEX TABLES

Any field in the data base can be indexed. An indexed field is one whose field values are important enough to justify storing them in the directory in a table called a Field Value Table (FVT). As shown in Figure 2-2, each value in an FVT points to an R-Value Index Table (RVIT) which contains the record numbers of records containing the field with the given field value.

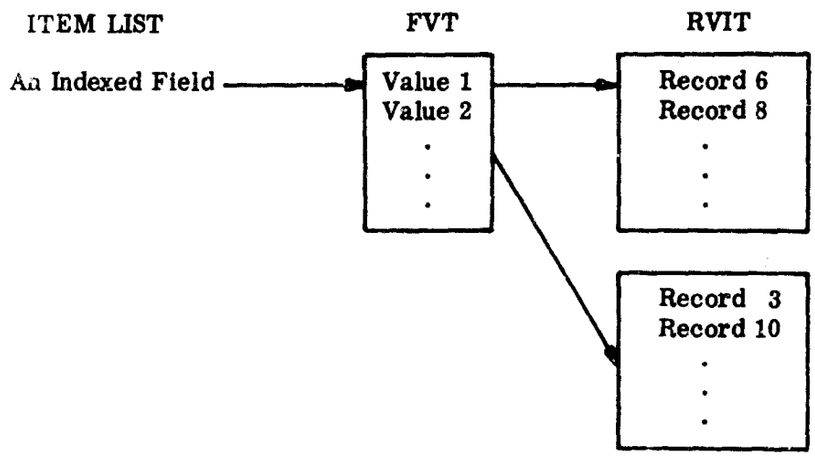


Figure 2-2. An Indexed Field in the Data Base

Since there is an FVT and a set of RVIT's for each indexed field, these elements are subsumed by the records of two parallel files, the Fields File and the Shadow of Fields File. The structure of these files is as follows:

```
FIELDS FILE, F
   INDEX CODE, B, 2
   USAGE, I, 18
   RESERVED, I, 18
   FVT FILE, F, ORDERED (1)
        FIELD VALUE, B, V
        END OF RANGE, B, V
        VALUE/RANGE OCCURRENCES I, 18
        USAGE OF VALUE/RANGE,  I, 18
        FIRST R-VALUE, H, V
SHADOW OF FIELDS FILE, F
   SHADOW OF FVT FILE, F
       RVIT FILE, F, ORDERED (1)
          R-VALUE, H, V
```

The Fields File has a record for each indexed field in the data base. As fields are indexed (see Index job, Paragraph 6.3.1), the next available record number of the Fields File is assigned and placed in the Item List entry for the field. This record number

serves as a link from the Item List to the Fields File. The record itself contains the following information:

(1) Index Code. This is a two-bit field with same coding as the corresponding field in the Item List; i.e.:

01 - indexed by LIST

10 - indexed by RANGE

11 - indexed by ALL

(2) Usage. This is an eighteen-bit field, which contains a count of the number of times the index table is used.

(3) Reserved. This is an eighteen-bit field which is reserved for future use.

(4) FVT File. This is an ordered file containing records for each list or range of values provided by the user as input to the Index job or for each different field value within the data if indexed by all values.

(5) The Shadow of Fields File is a parallel file, in that, for each record of the Fields File, there is an equivalent record in the Shadow of Fields File.

(6) The RVIT File is an ordered file containing records for each R-Value which points to a specific field position in the data base. That field position has a value corresponding to the value in the associated FVT record.

The segments of the index tables follow standard data segment conventions. The Fields File records are identified by 1.2.5.R, where the value for R is obtained from the record number stored in the Item List. The equivalent Shadow of Fields File record is identified by 1.2.6.R, with the same value for R. An RVIT File is identified by 1.2.6.R.1.R.1, with the value for the first R coming from the Item List entry and the value for the second R coming from the record number of the FVT File.

2.7 LINKAGE TABLE

Link items are never defined by the user when he presents item definitions to the system for the purpose of defining his structure. The Define-Item job, the Item Image Translation subroutine, and the Fix Service routine do not recognize the link as an item type. The reason for this restriction is that item definition is by nature a

sequential process, and a source or target link item in one structure is meaningless until the partner link has been defined in another structure. To avoid problems, both ends of a link are defined in parallel through the Link job (see Paragraph 6.4.1). When the Link job is executed, link items are inserted into the Item List, and they remain there until a Delete-Link job is executed.

The Linkage Table is a file which contains a record for each link item in the Item List. Its structure is as follows:

```
LINKAGE TABLE, F
   S/T CODE, B, 1
   RELATED ICC, H, V
   USAGE, I, 18
```

Each record in the Linkage Table contains the following fields:

(1) S/T Code. This is a one-bit binary field which is ∅ if the record corresponds to a source link, or 1 if the record corresponds to a target link.

(2) Related ICC. This is a variable length field containing the ICC of the partner link.

(3) Usage. This is an eighteen-bit integer field which contains a count of the number of times the link is traversed.

When the Link job is executed, the next two available record numbers of the Linkage Table file are assigned. These record numbers go into the new Item List entries which are inserted into the Item List. The ICC of the partner link is placed in the Linkage Table records.

The segments of the Linkage Table follow standard data segment conventions. Records are identified by 1.2.4.R, where the value for R is obtained from the record number stored in the Item List.

2.8 USER ACCESS/MODIFICATION RIGHTS TABLES

Security Restriction Level (SRL) codes for data are recorded in the Item List. If data is restricted, a system user must have a clearance level greater than the security

restriction level of the data, or he must have specific access or modification rights to the item before he may read or write this data.

The directory tables containing user access and modification rights are produced when the appropriate maintenance jobs are executed (see Paragraph 6.5.1). The structure of the User List file, Access Rights file, and Modification Rights file is as follows:

```
USER LIST, F, ORDERED (1)
    USER NAME, A, V
    USER CLASS, A, V
    PRIORITY, O, Z
    CLEARANCE LEVEL - ACCESS, 0, 1
    CLEARANCE LEVEL - MODIFICATION, 0,1
ACCESS RIGHTS, F, ORDERED (1)
    USER CLASS, A, V
    OPEN CLASS LIST, F
        ICC, H, V
    FIELD CONDITION LIST, F
        ICC, H, V
        ICC, H, V
        FIELD VALUE, B, V
MODIFICATION RIGHTS, F, ORDERED (1)
    USER CLASS, A, V
    OPEN CLASS LIST, F
        ICC, H, V
    FIELD CONDITION LIST, F
        ICC, H,V
        ICC, H, V
        FIELD VALUE, B, V
```

Records are added to the User List file each time the Add-User job is executed. The information for the record is input to the Add-User job.

Users are assigned a User Class code so that in the Access Rights and Modification Rights tables all users with identical rights can be combined. These tables list data items which may be read or written even though their SRL codes are equal to or higher than the user's clearance level. These tables are consulted only after failure of the initial security check of user clearance level against the security restriction level of the data. The information for these tables is input to the Add-Access-Rights or Add-

Modification-Rights jobs. There are two kinds of access or modification rights:

(1) Open Class List. This is a file giving the logical names of items which a class of users may read or write, even though the initial security check fails.

(2) Field Condition List. This is a file giving the logical names of items which a class of users may read or write, even though the initial security check fails, if a field within the item has a specified value.

These tables follow standard data segment format rules. The identifiers are:

(1)	User List	1.2.12
(2)	Access Rights	1.2.13
(3)	Modification Rights	1.2.14

A specific record can be retrieved quickly through binary search techniques because these files are maintained in order by User Name or User Class.

2.9 PROGRAM AND JOB DESCRIPTION LIBRARY

The programs which make up the constituent elements of DM-1 are not stored in the data pool. The operating system stores the DM-1 system coding and loads it into memory when called upon to do so. DM-1 stores descriptions of these programs in a set of directory tables collectively called the library.

After a program is written and filed with the operating system, a Program Entry job (see Paragraph 5.2) is executed to identify and describe the program to DM-1. At this time, the program name and a description of the program's inputs and outputs are filed in the program description library. An entry is also made for the program in the job description library.

Programs are the building blocks of jobs. The Job Description job (see Paragraph 5.3) may be used to name and describe a job. This places the job name and a list of the programs which constitute the job into the job description library.

For convenience, the program description library and the job description library are each subdivided into two separate directory tables. The four directory tables which make up the library are identified as:

(1) Program Statement (program names)

(2) Program Description List

(3) Job Statement (job names)

(4) Job Description List

The elements of the program description library have the following structure:

```
PROGRAM STATEMENT, S, 3
    PROGRAM NULL LIST, F
        PROGRAM NULL R-NO, I, 18
    PROGRAM LAST R-NO., I, 18
    PROGRAM NAME LIST, F, ORDERED (1)
        PROGRAM NAME, A, V
        PROGRAM R-NO., I, 18
PROGRAM DESCRIPTION LIST, F
    PROGRAM BINDING LIST, F
        PROGRAM ITEM LIST, F
            RESERVED, B, 10
            SRL - ACCESS, 0, 1
            SRL - MODIFICATION, 0, 1
            RESERVED, B, 2
            ITEM TYPE, B, 6
            OPTION CODE, B, 1
            ITEM SIZE, I, 11
    PROGRAM TERM LIST, F
        TERM NAME, A, V
        UNITS, B, G
        RESERVED, I, 18
```

The elements of the job description library have the following structure:

```
JOB STATEMENT, S, 3
    JOB NULL LIST, F
        JOB NULL R-NO., I, 18
    JOB LAST R-NO., I, 18
    JOB NAME LIST, F, ORDERED (1)
        JOB NAME, A, V
        JOB R-NO., I, 18
JOB DESCRIPTION LIST, F
    JOB ID, A, V
    JOB ITEM LIST, F
        CLASS, I, 3
        I/O LIST F
            TYPE B, 3
            I/O NAME, A, V
            PIL R-VAL, H, V
STATIC TASK LIST, F
    TYPE B, 3
    TASK ID, I, 12
    NO. FLOATS, I, 3
    INPUT LIST, F
        FORMAL NAME, A, V
        CLASS, I, 3
        I/O R-NO., I, 15
    OUTPUT LIST, F
        FORMAL NAME, A, V
        CLASS, I, 3
        I/O R-NO., I, 15
        FORMAL NAME, A, V
        CLASS, I, 3
        I/O R-NO., I, 15
JOB COMPONENT LIST, F
    COMPONENT NAME, A, V
    COMPONENT R-NO., I, 18
    COMPONENT I/O LIST, F
        TYPE, B, 3
        I/O NAME, A, V
        CLASS, I, 3
        I/O R-NO., I, 15
USAGE LIST, F
    JOB NAME, A, V
```

Segments of all four of these directory tables follow standard data segment format rules. The identifiers are:

(1) Program Statement - 1.2.8

(2) Program Name List - 1.2.8.3

(3) Program Description List - 1.2.9

(4) Job Statement - 1.2.10

(5) Job Name List - 1.2.10.3

(6) Job Description List - 1.2.11

2.10 DATA SEGMENTS

A segment is the unit of data transfer between DM-1 and the operating system. The segment is 9216 bits long (512 x 18). The bits are numbered from 1 to 9216. Each segment consists of four parts: the segment head, the segment index, the body, and the slack.

2.10.1 Segment Head

The elements of the segment head are:

(1) Segment Name . . . . . 54 bits

(2) Pointer to Body . . . . . 15 bits
(The bit number of the last bit occupied by index, expressed in binary.)

(3) Pointer to Slack . . . . . 15 bits
(The bit number of the last bit occupied by body, expressed in binary.)

(4) Reserved for future use. . . . 12 bits

TOTAL 96 bits

The Segment Name is a unique symbolic name user by DM-1 in data transfer calls to the operating system. DM-1 expects the operating system to control relative and absolute mass-storage addressing. The Segment Name List (see Paragraph 2.11) cross-references the DM-1 segment identifiers to Segment Names.

2.10.2 Segment Index

The final step in retrieving a data item is taken with the aid of the segment index. Information in the directories serves to identify the particular segment containing the desired data item. The Item List provides a pattern for "stepping down" a data segment to the desired item, but there are several variables which cannot be stored in the Item List because they pertain to a unique position (IPC) rather than to a class of items (ICC). Information about these variables is stored in the segment index. It contains the following directory-like information:

(1) File-Continuation Counters. For files that began in a previous segment, these counters specify the number of records or partial records that exist in this segment. There is one counter for the deepest level of the file and one for each higher level up to that level where the record number does not change. Stepping down through a file of data requires looping through the Item List each time a record begins. The file-continuation counters control the looping back in the Item List.

(2) File-Size Entries. These are counters which specify the number of records or partial records that exist in this segment for files that begin in the segment. Like the file continuation counters, the file-size counters control the looping back in the Item List when stepping through the data segment.

(3) Optional Item Entries. The Item List (see Paragraph 2.3) contains a code which may indicate that an item is optional. This means that at any single position (IPC), the item may be missing. For optional items, the segment index contains an entry which will be zero if the item is missing. If the item is present, the entry will contain the item size.

(4) Variable Length Fields. An item size of zero in the Item List means that the field is of variable length. For variable length fields, there is an entry in the segment index which gives the actual length at this position. Field size, whether given in Item List or segment index, is stored as the number of bits or bytes in the field. It must be multiplied by 3, 4, etc., based on the item type of the field.

The segment index is of variable length, depending on the number of entries required. All entries are recorded in B-S format.

2.10.3 Segment Body

The Segment Body contains the data as a stream of bits with no field separators. The Item List and segment index provide the information necessary to identify the fields within this stream of bits.

All fixed-length required fields have a most significant bit (the null bit), in addition to the size as given in the Item List. If this leading bit is zero, the field is present in the following bits. If this leading bit is one, the field is not present, and the following bits represent the space which has been reserved for the field.

In placing fields in the body of a segment, a field will not be split across two segments. If the last field will not fit, it will be placed in the next segment and the wasted space will be designated as Slack. This limitation on the length of a field to approximately 9,000 bits is not considered to be serious.

2.10.4 Partition between Segments

Since some items appearing in the Item List never consume any space in the data segment (e.g., record, null node), the identification of the second segment becomes a problem when the partition occurs at one of these nonspace-consuming items. For convenience, segments are identified by the IPC of the first space-consuming item in either the index or the body of the segment.

2.11 SEGMENT NAME LIST

The Segment Name List (SNL) relates the logical segment identifiers of DM-1 to the segment names known to the operating system. The SNL is an ordered file which

is used by the Name-Segment Routine of the Service Package (see Section III) to convert segment identifiers to segment names. The format of the SNL is as follows:

SEGMENT NAME LIST, S, ORDERED (1)
ELEMENT IDENTIFIER, H, V
SEGMENT NAME, A, 9
FLAC, B, 1
ACCESS USAGE, I, 18
EDITION NUMBER, I, 18
CURRENT EDITION FLAG, B, 1
BUSY BIT, B, 1

All segments of the data pool are identified by the IPC of the first data item contained in the segment. Since the SNL is ordered by segment identifiers, a search on the SNL will provide the segment name for the segment which contains the desired item. There is no need to access any data segment except the one containing the desired item.

There are two directory tables which are identified in the SNL by special identifiers as well as by IPC's. These are:

(1) Term Encoding Table. The special identifier used for TET segments consists of a prefix of 1.0.1 followed by the term name of the first record within the segment.

(2) Item List. The special identifier used for Item List segments consists of a prefix of 1.0.2 followed by the ICC of the first record within the segment.

These special segment identifiers are used by routines of the Service Package which treat these directory elements as special tables rather than as standard data segments.

Segment-Name List segments follow standard data segment format rules. The file can be processed as data through the same input-output routines which service the rest of the data pool, using a prefix of 1.2.7. The only restriction on these segments is that no record will be split across two segments. The Term Encoding Table and Item List are like the SNL in this respect.

2.12 DIRECTORY FORMAT

The detailed structure of the data pool, before the definition of any application-oriented items, is shown in Table 2-3. This table shows the entire data pool as the statement DATA POOL, which subsumes four items. These are the statements DATA BASE, DIRECTORY, WORK AREA, and SCRATCH AREA. The substructure for the data base, work area, and scratch area are defined as the data pool evolves through system use. The substructure for the directory is fixed by the DM-1 system. All elements of the directory are shown in Table 2-3.

TABLE 2-3. STRUCTURE OF THE DATA POOL

1	DATA POOL, S, 4
1.1	DATA BASE, S, ø
1.2	DIRECTORY, S, 14
1.2.1	TERM ENCODING TABLE, F, ORDERED (1)
1.2.1.R.1	TERM NAME, A, V
1.2.1.R.2	ICC FILE, F
1.2.1.R.2.R.1	ICC, H, V
1.2.2	ITEM LIST, F
1.2.2.R.1	RESERVED, B, 1ø
1.2.2.R.2	SRL-ACCESS, O, 1
1.2.2.R.3	SRL-MODIFICATION, O, 1
1.2.2.R.4	INDEX CODE, B, 2
1.2.2.R.5	ITEM TYPE, B, 6
1.2.2.R.6	OPTION CODE, B, 1
1.2.2.R.7	ITEM SIZE, I, 11
1.2.2.R.8	NO DATA FLAG, B, 2
1.2.2.R.9	RECORD NUMBER, I, 16
1.2.3	TERM LIST, F
1.2.3.R.1	TERM NAME, A, V
1.2.3.R.2	UNITS, B, 6
1.2.3.R.3	RESERVED, I, 18
1.2.4	LINKAGE TABLE, F
1.2.4.R.1	S/T CODE, B, 1
1.2.4.R.2	RELATED ICC, H, V
1.2.4.R.3	USAGE, I, 18

TABLE 2-3. STRUCTURE OF THE DATA POOL (Continued)

1	DATA POOL, S, 4 (continued)
1.2	DIRECTORY, S, 14 (continued)
1.2.5	FIELDS FILE, F
1.2.5.R.1	INDEX CODE, B, 2
1.2.5.R.2	USAGE, I, 18
1.2.5.R.3	RESERVED, I, 18
1.2.5.R.4	FVT, F, ORDERED (1)
1.2.5.R.4.R.1	FIELD VALUE, B, V
1.2.5.R.4.R.2	END OF RANGE, B, V
1.2.5.R.4.R.3	OCCURRENCES, I, 18
1.2.5.R.4.R.4	VALUE/RANGE USAGE, I, 18
1.2.5.R.4.R.5	FIRST R-VALUE, H, V
1.2.6	SHADOW OF FIELDS FILE, F
1.2.6.R.1	SHADOW OF FVT, F
[illegible]	RVIT, F, ORDERED (1)
[illegible].R.1	R-VALUE, H, V
[illegible]	SEGMENT NAME LIST, F, ORDERED (1)
[illegible]	SEGMENT IDENTIFIER, H, V
[illegible]	SEGMENT NAME, A, 9
[illegible]	FLAG, B, 1
[illegible]	ACCESS USAGE, I, 18
[illegible]	EDITION NUMBER, I, 18
[illegible]	CURRENT EDITION FLAG, B, 1
[illegible]	BUSY BIT, B, 1

TABLE 2-3. STRUCTURE OF THE DATA POOL (Continued)

1	DATA POOL, S, 4 (continued)
1.2	DIRECTORY, S, 14 (continued)
1.2.8	PROGRAM STATEMENT, S, 3
1.2.8.1	PROGRAM NULL LIST, F
1.2.8.1.R.1	R-NUMBER, I, 18
1.2.8.2	PROGRAM LAST R-NUMBER, I, 18
1.2.8.3	PROGRAM NAME LIST, F, ORDERED (i)
1.2.8.3.R.1	PROGRAM NAME, A, V
1.2.8.3.R.2	PROGRAM R-NUMBER, I, 18
1.2.9	PROGRAM DESCRIPTION LIST, F
1.2.9.R.1	PROGRAM BINDING LIST, F
1.2.9.R.1.R.1	PROGRAM ITEM LIST, F
1.2.9.R.1.R.2	RESERVED, B, 10
1.2.9.R.1.R.3	SRL-ACCESS, O, 1
1.2.9.R.1.R.4	SRL-MODIFICATION, O, 1
1.2.9.R.1.R.5	RESERVED, B, 2
1.2.9.R.1.R.6	ITEM TYPE, B, 6
1.2.9.R.1.R.7	OPTION CODE, B, 1
1.2.9.R.1.R.8	ITEM SIZE, I, 11
1.2.9.R.2	PROGRAM TERM LIST, F
1.2.9.R.2.R.1	TERM NAME, A, V
1.2.9.R.2.R.2	UNITS, B, 6
1.2.9.R.2.R.3	RESERVED, I, 18

TABLE 2-3. STRUCTURE OF THE DATA POOL (Continued)

1	DATA POOL, S, 4 (continued)
1.2	DIRECTORY, S, 14 (continued)
1.2.1Ø	JOB STATEMENT, S, 3
1.2.1Ø.1	JOB NULL LIST, F
1.2.1Ø.1.R.1	R-NUMBER, I, 18
1.2.1Ø.2	JOB LAST R-NUMBER, I, 18
1.2.1Ø.3	JOB NAME LIST, F, ORDERED (1)
1.2.1Ø.3.R.1	JOB NAME, A, V
1.2.1Ø.3.R.2	JOB R-NUMBER, I, 18
1.2.11	JOB DESCRIPTION LIST, F
1.2.11.R.1	JOB ID, A, V
1.2.11.R.2	JOB ITEM LIST, F
1.2.11.R.2.R.1	CLASS, I, 3
1.2.11.R.2.R.2	I/O LIST, F
1.2.11.R.2.R.2.R.1	TYPE, B, 3
1.2.11.R.2.R.2.R.2	I/O NAME, A, V
1.2.11.R.2.R.2.R.3	PIL R-VALUE, H, V
1.2.11.R.2.R.3	STATIC TASK LIST, F
1.2.11.R.2.R.3.R.1	TYPE, B, 3
1.2.11.R.2.R.3.R.2	TASK ID, I, 12
1.2.11.R.2.R.3.R.3	NO. FLOATS, I, 3
1.2.11.R.2.R.3.R.4	INPUT LIST, F
1.2.11.R.2.R.3.R.4.R.1	FORMAL NAME, A, V
1.2.11.R.2.R.3.R.4.R.2	CLASS, I, 3
1.2.11.R.2.R.3.R.4.R.3	I/O R-NO., I, 15
1.2.11.R.2.R.3.R.5	OUTPUT LIST, F
1.2.11.R.2.R.3.R.5.R.1	FORMAL NAME, A, V
1.2.11.R.2.R.3.R.5.R.2	CLASS, I, 3
1.2.11.R.2.R.3.R.5.R.3	I/O R-NO., I, 15
1.2.11.R.2.R.4	JOB COMPONENT LIST, F
1.2.11.R.2.R.4.R.1	COMPONENT NAME, A, V
1.2.11.R.2.R.4.R.2	COMPONENT R-NO., I, 18
1.2.11.R.2.R.4.R.3	COMPONENT I/O LIST, F
1.2.11.R.2.R.4.R.3.R.1	TYPE, B, 3
1.2.11.R.2.R.4.R.3.R.2	I/O NAME, A, V
1.2.11.R.2.R.4.R.3.R.3	CLASS, I, 3
1.2.11.R.2.R.4.R.3.R.4	I/O R-NO., I, 15
1.2.11.R.2.R.5	USAGE LIST, F
1.2.11.R.2.R.5.R.1	JOB NAME, A, V

TABLE 2-3. STRUCTURE OF THE DATA POOL (Continued)

1	DATA POOL, S, 4 (continued)
1.2	DIRECTORY, S, 14 (continued)
1.2.12	USER LIST, F, ORDERED (1)
1.2.12.R.1	USER NAME, A, V
1.2.12.R.2	USER CLASS, A, V
1.2.12.R.3	PRIORITY, O, 1
1.2.12.R.4	CLEARANCE LEVEL-ACCESS, O, 1
1.2.12.R.5	CLEARANCE LEVEL-MODIFICATION, O, 1
1.2.13	ACCESS RIGHTS, F, ORDERED (1)
1.2.13.R.1	USER CLASS, A, V
1.2.13.R.2	OPEN CLASS LIST, F
1.2.13.R.2.R.1	ICC, H, V
1.2.13.R.3	FIELD CONDITION LIST, F
1.2.13.R.3.R.1	ICC, H, V
1.2.13.R.3.R.2	FIELD ICC, H, V
1.2.13.R.3.R.2	FIELD VALUE, B, V
1.2.14	MODIFICATION RIGHTS, F, ORDERED (1)
1.2.14.R.1	USER CLASS, A, V
1.2.14.R.2	OPEN CLASS LIST, F
1.2.14.R.2.R.1	ICC, H, V
1.2.14.R.3	FIELD CONDITION LIST, F
1.2.14.R.3.R.1	ICC, H, V
1.2.14.R.3.R.2	FIELD ICC, H, V
1.2.14.R.3.R.3	FIELD VALUE, B, V
1.3	WORK AREA, S, Ø
1.4	SCRATCH AREA, S, Ø

SECTION III. DM-1 SERVICE PACKAGE

The DM-1 Service Package encompasses all of the routines involved in storage and retrieval, both for data and for system directories. In Volume I of this report, the services are treated from the user's viewpoint. Volume I addresses such questions as: Which service or set of services must a user call on to accomplish a particular information processing activity? What does each service do for the user? What parameters must the user supply to the service? Volume I, then, is directed towards those who will use the Service Package, while Volume II, in which the services are analyzed in much greater detail, is aimed at those who will implement the Service Package.

The first characteristic which can be observed in the DM-1 system is that it consists of a large number of relatively small routines. This fact reflects a basic design decision. The user services, such as Read and Write, could have been designed as individual, monolithic programs. That approach was rejected as being wasteful and inflexible. Instead, a modular design was selected. Consequently, the DM-1 system contains many routines, each of which performs a single function. The user services consist of combinations of the more elementary services. All of the services can be divided into six categories, which are treated in the following paragraphs. These categories are:

(1) Bookkeeping

(2) Segmentation

(3) Translation

(4) Maintenance

(5) Basic User Operations

(6) Compound User Operations

3.1 BOOKKEEPING

Some fundamental aspects of the DM-1 system imply an important bookkeeping activity while the data pool is being used. The design is characterized by the following facts:

(1) The item identifiers are not stored explicitly but are simply implied in certain counters.

(2) The structure definitions, which allow considerable variations in the data, are stored separately from the data, and

(3) The data is stored as a continuous, unformatted stream of bits.

Volume I has explained the reason for this design; the following paragraphs describe the record keeping which the design implies.

3.1.1 Level Pushdown List

The system keeps track of the identification of the item currently being processed by means of a device called the Level Pushdown List (LPL). Refer to Table 3-1.

The first column of the list contains the Item Position Code (IPC) of the current item. Each row of this column consists of the digit value of the IPC at that level. The level gets larger to represent a deeper position in the data hierarchy. The letter l, without qualification, represents the level of the current item.

The second column of the LPL contains the subsumed item count (SIC). On each row, this count serves as a bound which limits the IPC digit on that level. As a rule, the SIC on level l contains the total number of items subsumed by the parent of the current item. An exception occurs if the current item is a record, which requires special processing.

TABLE 3-1. FORMAT OF LEVEL PUSHDOWN LIST

Level	IPC	SIC	R	J
1	IPC (1)	SIC (1)	0/1	J (1)
2	IPC (2)	SIC (2)	0/1	J (2)
'	'	'	'	'
'	'	'	'	'
'	'	'	'	'
l	IPC (l)	SIC (l)	0/1	J (l)

In the third column, R is a simple flag indicating whether the IPC digit at a given level is a record number.

In the fourth column, J is an index to a nonterminal item in the Item List Table. At level 1, J is an index to the parent of the current item.

The Purchasing Item, defined in Table 3-2, provides the basis for the two examples of the Level Pushdown List given in Tables 3-3 and 3-4. The LPL in Table 3-3 assumes that the system is currently processing the field PRICE in the eighth record of the PO Item File.

The first column of Table 3-3 contains the IPC for PRICE (1.1.8.3). The bottom row of the list reflects the fact that PRICE is the third subitem under a parent which directly subsumes four items. The R-bit is zero because PRICE is not a record. The J-value (3) points to the Item List Table where more information is available on the parent of the PRICE field.

The 14 on level 3 of the SIC column in Table 3-3 is an example of the exception previously mentioned. The 14 does not represent the total number of records in the file as the rule for SIC would suggest. Rather 14 is the number of records from the beginning of the file to the end of the data segment currently being processed.

The LPL in Table 3-4 assumes that the current item is the field YEAR in the statement DUE DATE of the seventh record of the ORDER File. YEAR is a 5th level item with an IPC of 1.2.7.2.1. The bottom row shows that YEAR is the

TABLE 3-2. DEFINITION OF PURCHASING ITEM

J	ICC	ITEM DEFINITION
1	1	PURCHASING, S, 3
2	1.1	PO ITEM, F
3	1.1.R	(PO ITEM, R, 4)
4	1.1.R.1	ITEM NO., I, V
5	1.1.R.2	VENDOR NO., I, 4
6	1.1.R.3	PRICE, I, 12
7	1.1.R.4	*DESCRIPTION, A, V
8	1.2	ORDER, F
9	1.2.R	(ORDER, R, 6)
10	1.2.R.1	PO NO., I, 6
11	1.2.R.2	DUE DATE, S, 3
12	1.2.R.2.1	YEAR, D, 2
13	1.2.R.2.2	MONTH, D, 2
14	1.2.R.2.3	DAY, D, 2
15	1.2.R.3	REQUESTOR, A, V
16	1.2.R.4	VENDOR NO., I, S
17	1.2.R.5	VALUE, E, V
18	1.2.R.6	ITEM DETAILS, F
19	1.2.R.6.R	(ITEM DETAILS, R, 3)
20	1.2.R.6.R.1	ITEM NO., I, V
21	1.2.R.6.R.2	QUANTITY, I, S
22	1.2.R.6.R.3	COST, E, V
23	1.3	VENDOR, F
24	1.3.R	(VENDOR, R, 2)
25	1.3.R.1	VENDOR NO., I, 4
26	1.3.R.2	VENDOR NAME, A, V

* Optional item

TABLE 3-3. LPL FOR PRICE FIELD

Level	IPC	SIC	R	J
1	1	-	0	-
2	1	3	0	1
3	8	14	1	2
4	3	4	0	3

TABLE 3-4. LPL FOR YEAR FIELD

Level	IPC	SIC	R	J
1	1	-	0	-
2	2	3	0	1
3	7	11	1	8
4	2	6	0	9
5	1	3	0	11

first in a group of three items. The zero in the R column simply means that the item is not a record. The 11 in the J-column indicates that the parent (DUE DATE) of the current item is defined in entry 11 of the Item List Table.

3.1.2 Access Parameter Table

For each input or output data stream which the user is processing, the DM-1 system must maintain a Level Pushdown List as well as other control information which is described in the remainder of this section. The user supplies an area of memory to hold this information. The total group of lists and control pointers relating to a particular input or output stream is called the Access Parameter Table (APT). The address of the APT is used as an input parameter for user services (e.g., Read and Write) which need to know which data stream to process.

3.1.3 Item List Table

The second list which the service routines employ to interpret the data streams is the Item List. For the sake of efficiency, the DM-1 system extracts from the entire Item List that information which is frequently required for the processing of an opened item and its subitems. The extracted information makes up the

Item List Table (ILT) that is set up during the OPEN operation, which is part of both the storage and retrieval services. Table 3-5 shows the Item List Table that corresponds to a portion of the Purchasing Item which is defined in Table 3-3.

TABLE 3-5. ITEM LIST TABLE

	j	Item Type	Item Size	Option Code	Index Code	No Data Flag
	1	S	3	0	0	0
	2	F	V	0	0	0
	3	R	4	0	0	0
	4	I	V	0	0	0
	5	I	4	0	0	0
Price	6	I	12	0	0	0
	7	A	V	1	0	0
	8	F	V	0	0	0
	9	R	6	0	0	0
	10	I	6	0	0	0
	11	S	3	0	0	0
Year	12	D	2	0	0	0

Table 3-5 lists Item Type, Item Size, Option Code, Index Code, and No Data Flag. A detailed explanation of these fields is presented in Section IV of Volume I. The letter j is the index which the system keeps set at the current item. The fields which were used in the examples for the Level Pushdown List are shown in this example. PRICE (shown at j = 6) is a 12-bit integer, and YEAR (at j = 12) is a 2-digit decimal value. Neither field is optional or indexed. The No Data Flag is clear because both items are actually represented in the data pool (either by data or by null indicators).

3.1.4 Segment Index

The third source of information which helps to define the items in a data stream is the Segment Index. Although this index is carried in data segments, it is a logical extension of the Item List. The size of a fixed length field can be contained in the Item List because it is the same for every occurrence of that field. Obviously the same cannot be said for the size of a variable length field. Consequently, a different mechanism must be used to define the size of variable length fields; the mechanism is the Segment Index. Each time a variable length field occurs, its size is defined in the Segment Index. In a similar manner, the index contains record counters for files. These counters, the File Continuation Counter and the File Size Entries, are described in greater detail in Paragraph 2.10. The final area in which the Segment Index supplements the Item List is the area of optional items. The Item List declares whether an item is optional or required. But, an optional item may be present in some records and absent in other records. Therefore, an indicator is needed for each occurrence of the item. These indicators are contained in the Segment Index.

3.1.5 Missing Data Indicators

To understand the interpretation of the data streams, it is necessary to know the various methods of indicating the absence of data. The following list summarizes these indicators:

(1) Null Node and No-Data Flag. The Null Node and the No-Data Flag are devices used by the Maintenance Jobs under the direction of job requestors. Although they appear in the Item List, these indicators do not relate to any data existing in the data pool. Therefore, the service routines ignore them and immediately step to the next item.

(2) Option Code. The option code in the Item List plus an indicator in the Segment Index represent an optional item which is absent. If a nonterminal item is absent, all of its subitems are assumed to be missing.

(3) Missing Parent. Even required, i.e., nonoptional, items have no data in the data pool if a parent item is missing.

(4) Files with Zero Records. An empty file is represented by a file size of zero in the File Size Entry of the Segment Index.

(5) Fields with Zero Size. A variable length field may have a size of zero.

(6) Null Bits. If there is no data for a fixed length field, the null bit is set to one and a null value (corresponding to the item type) is stored in the space allotted for that field.

3.1.6 Bookkeeping Service Routines. The individual service routines in the Bookkeeping category are defined in the following paragraphs.

(1) Get Next IL Entry. This routine steps the pointers of the Item List segment to the next entry. If there are no more entries in the current segment, this routine calls for the next segment and initializes the pointers to the first entry. Other services used are: Name Segment and Fetch Segment.

(2) Locate IL Entry. This routine skips over unwanted entries in an Item List Segment and sets the pointers at an entry which corresponds to a given ICC (Item Class Code). Other services used are: Stem and Skip Subitems.

(3) Skip Subitems. This routine uses the item type codes in the Item List segment to skip over "T" items and all of their subsumed items, including those subsumed directly and indirectly. Other service used: Get Next IL Entry.

(4) Retrieve IL Entry. This routine retrieves the Item List Segment containing the entry for a given ICC and moves the pointers to that entry. Other services used: Get Segment and Locate IL Entry.

(5) Build IL Table. This routine extracts information from Item List segments and produces the Item List Table. The table, which was described earlier in Paragraph 3.1.3, begins with the entry corresponding to the stem of the data segment containing the desired item. The table includes all items subsumed directly or indirectly by the desired item. Other services used: Stem, Retrieve IL Entry, and Get Next IL Entry.

(6) Stem. This routine compares two identifiers, digit by digit, and produces a number (X) which tells how many consecutive digits match. This number identifies the level of the common parent of the two items.

(7) Define Segments. This routine steps the Item List Table from a parent item to the first item in a segment (Segment Index Left). The parent item may be the opened item or the stem of the segment of the opened item. Define Segment also initializes the pointers of the data segment. Other service used: Step Item.

(8) Step Item. This routine steps all of the control pointers and data pointers to the next item and retrieves the next data segment when necessary. This is a key routine. Other services used are: Step Lists, Get Next SX, and Define Field.

(9) Step Lists. This routine steps the Level Pushdown List once and manipulates the index (j) of the Item List Table accordingly. Other service used: Discount Item.

(10) Discount Item. This routine adds one to the deepest level of the LPL which has not already reached the limit expressed in the Subsumed Item Count. Discount Item also steps the ILT index (j) taking special care to reset j for each record in a file.

(11) Skip Item. This routine uses the item type code in the Item List Table to skip over "T" items and all of their subsumed items. It is similar to the Skip Subitems routine which operated on the Item List segment.

3.2 SEGMENTATION

This category of service routine pertains primarily to segments, not to logical data items. The segmentation routines provide for packing and unpacking segments, for fetching and storing segments, and for searching and updating segment name lists. The routines depend on the segment name list and on the format of a segment, both of which are described in Section II.

The individual routines in the segmentation category are defined in the following paragraphs:

(1) Name Segment. This routine searches the Segment Name List to find the segment name corresponding to a given identifier (ID). The ID may be an IPC, an ICC, or a term name. This service also obtains the identifier of the next segment, i.e., the Segment Identifier Right (SIR). Other service used: Fetch Segment.

(2) Fetch Segment. This routine calls on the Executive Program to read the desired segment into core. When the segment arrives, Fetch initializes the segment pointers.

(3) Get Segment. This routine finds the segment name for a given identifier and brings the desired segment into core. Other services used: Name Segment and Fetch Segment.

(4) Define Field. This routine saves the current bit pointer (Bit Pointer Left) and then steps the pointer past the current item. This makes available two pointers which delimit the current field. If the current segment is exhausted, Define Field calls for the next segment. Other service used: Get Segment.

(5) Get Next SX. This routine unpacks the next entry of the segment index and moves it to temporary storage. If the current segment is exhausted, the routine calls for the next segment. Other service used: Get Segment.

(6) Extract. This routine moves the current field from the input segment to the user's buffer. Extract is a key service. It uses the size field (TSIZE) developed by Step Item, the size field of the Buffer Description List, the type in the Item List Table, and the bit pointers developed by Define Field.

(7) Compose. This routine is the inverse of Extract. It takes a field from the user's buffer, edits it, and stores it into the next available bits of the output body. This service begins by calculating whether the field will fit in the output segment. If not, Compose calls on another service to terminate the current segment and initialize a new one. Other service used: Step Segment.

(8) Pack SX. This routine takes the values which have been stored in the temporary segment index by the Write routine and moves them to the next available bits of the output segment index. Then, Pack SX tests to determine whether the limit of available storage has been reached. If so, this service calls on another routine to terminate the current segment and initialize a new one. This space test is made after the segment index is packed to assure that a field is never separate from its associated SX entry. Other service used: Step Segment.

(9) Move. This routine takes a field from the input area and stores it into the next available bits of the output body. If the segment does not have sufficient space available, Move calls on Step Segment to terminate the current segment and initialize a new one. Other service used: Step Segment.

(10) Initialize TSNL (Temporary Segment Name List). This routine does the initial bookkeeping to begin a list which will act as the segment name list for an output item until the item is closed.

(11) Initialize Output Segment. This routine begins a new segment in the output area. It sets up the segment header, assigns a segment name, and initializes the segment pointers.

(12) Terminate Segment. This routine completes the packing of the output segment and inserts terminate symbols in the body and in the segment index. Then, the segment is stored, and its identification is entered into the temporary segment name list. The active record counters, whose contents have been written on the file size entries and the file continuation counters, are reset to zero. Other services used: Store Segment and Catalog Segment.

(13) Store Segment. This routine calls on the executive program to store a segment into the space corresponding to the assigned segment name.

(14) Catalog Segment. This routine puts an entry into the Temporary Segment Name List for a completed output segment.

(15) Step Segment. This routine simply calls on Terminate Segment and then executes Initialize Output Segment.

(16) Incorporate TSNL. This routine uses the temporary segment name list to update the permanent segment name list.

3.3 TRANSLATION

The DM-1 system permits several kinds of identification for the items in the data pool, including generalized names, specific names, internal names, external names, IPC's, etc. The translation services provide for the conversion of one type of identification to another.

(1) Formal Name Translation. General-purpose task programs are written with formal names as their inputs and outputs. This allows different data items to be selected and bound to the task at execution time. The Request Processor binds the formal name to a specific Item Position Code and stores both in the Binding List of the task. It is during the running of the task that the Formal Name Translation comes into play. Whenever the program uses the formal name, e. g., in an OPEN call, the Formal Name Translation looks up the corresponding IPC.

(2) Term Name Translation. This routine supplies the ICC corresponding to a given term name or term name with qualifiers. This routine uses TET Search.

(3) TET Search. This routine looks in the Term Encoding Table and opens the file of ICC's associated with the given term name.

(4) Retrieve Term List Entry. This routine translates a given ICC to a term name by retrieving a Term List segment and searching within the segment for the entry corresponding to the ICC.

(5) Relative Item Number Translation. The relative item number is a convenience which permits a subsumed item to be identified by its distance from the parent. The number is simply the total number of items between the parent and the desired item regardless of item types. The base item can be the direct parent or a higher parent. The Relative Item Number Translation steps from the base item to the desired item in order to develop the IPC of the desired item.

3.4 MAINTENANCE

In the DM-1 system, the modification of system directories and other maintenance functions are accomplished primarily by system jobs which are described in Section VI. However, service routines have been provided for a few maintenance operations which are needed dynamically by several general-purpose jobs.

(1) Assign IPC. This routine binds a given name to a unique node in the scratch area or the work area of the Data Pool.

(2) Fix Item. This routine permits dynamic definition of the structure of a scratch item or a work item. Fix Item enters the given item definition into the system directories. The operation is similar to that of the system job, Define-Item (see Section VI).

(3) Assign Item. This routine executes Assign IPC and Fix Item. The total effect is to bind a given name to a unique node and to enter the definition of the named item into the system directories.

3.5 BASIC USER SERVICES

The user services are described in Volume I with an emphasis on what they do for the user. Here, they are repeated, but the emphasis is on how they accomplish their work.

Many of the user services operate under the control of the Buffer Description List (BDL). In this list, the user supplies an action code, a size entry, and space for a status indicator. A sample BDL is shown in the following example; it is fully explained in Volume I.

Action	Size	Indicator
T	-	
T	24	
S	-	
S	-	
T	16	

Notes:
T = Transmit
S = Skip

Size is given in number of units; the unit depends on the item type in the Item List

The following paragraphs describe individual, basic user services.

(1) Read. This routine edits selected data fields from data segments into the user's buffer area under the control of the Buffer Description List. Other services used: Step Item, Extract, and Seek.

(2) Seek. This routine sets the data pointers and control pointers to the item desired by the user. If the item is not in the current segment, Seek gets the segment which contains the desired item. Other services used: Name Segment, Fetch Segment, Define Segment, and Locate Item.

(3) Locate Item. This routine steps through a segment until the system pointers are set at a desired item. Other service used: Step Item.

(4) Write. This routine causes the data in the caller's buffer area to be edited, packed into an output segment(s), and written into the RC Data Pool, the routine interprets the Buffer Description List and the Item List. Other services used: Step List, Discount Item, Skip Item, Pack SX, and Compose.

(5) Seek With Copy. This routine causes a string of items in an input segment(s) to be packed into an output segment(s) and written into the RC Data Pool. The routine interprets the input segment index and the Item List. All input items are copied, up to but excluding the item identified in the request. Other services used: Stem, Step Item, Step Lists, Move, and Pack SX.

(6) Insert. This routine writes an item from the user's buffer. Insert is used on an item which is open for updating. It implies that the pointers of the input item are not to be moved. Other service used: Write.

(7) Replace. This routine writes an item from the user's buffer and moves the system pointers over the current input item. Other services used: Write and Locate Item.

(8) Delete. This routine marks the current output item as missing and moves the system pointers over the current input item. Other services used: Write and Locate Item.

3.6 COMPOUND USER SERVICES

(1) Retrieve Item Segment. This routine is required by each of the OPEN services. The functions of Retrieve Item Segment can be summarized as follows:

(a) Translate the given formal name to an IPC.

(b) Determine which data segment contains the data for this IPC.

(c) Stem the IPC's which delimit the desired segment (Segment Identifier Left and Segment Identifier Right).

(d) Retrieve the Item List entry for the stem item.

(e) Build an IL Table beginning with the stem item down to the last item subsumed by the opened item.

(f) Get the data segment containing the opened item.

(g) Initialize the Level Pushdown List by stepping from the stem item to the first item in the data segment and set all

pointers to this item. Other services used: Formal Name Translation, Name Segment, Fetch Segment, Stem, Retrieve IL Entry, Build IL Table, Get Segment, and Define Segment.

(2) Open for Input. This routine performs all of the functions of Retrieve Item Segment and is followed by the moving of all system pointers from the first item in the segment to the opened item. Other services used: Retrieve Item Segment and Locate Item.

(3) Close for Input. This routine simply releases the storage areas which the system was using for the opened item.

(4) Open for Writing. This routine executes Retrieve Item Segment; then, it initializes a temporary segment name list for the output item. Next, a data segment is set up, and the items which precede the opened item in the input segment are copied. Other services used: Retrieve Item Segment, Initialize TSNL, Initialize Output Segment, and Seek with Copy.

(5) Close for Writing. This routine begins by taking data from the segment which contains the item which logically follows the item to be closed. That item and the remainder of its segment are copied into the current output segment(s). The final segment is terminated, and the Segment Name List is updated with all the segments which have been created since the OPEN operation. Then the storage areas which the system was using for the opened item are released. Other services used: Open for Input, Seek with Copy, Terminate Segment, Incorporate TSNL.

(6) Open for Update. This service routine by executing the Open for Writing routine. Then the Level Push-down List and the system pointers are duplicated to permit the input item to be moved independently from the output item. Other service used: Open for Writing.

(7) Close for Update. This service begins by copying any remaining data in the input item. Next, the Close for Writing routine is executed. Then, the control storage areas required for the update procedure are released. (Close for Writing has released some, but not all of these.) Other services used: Seek with Copy and Close for Writing.

(8) Retrieve Item. This routine fetches data from any area of the data pool and delivers it to the user's buffer. It essentially consists of an Open, followed by Read and Close. Other services used: Open for Input, Read, and Close for Input.

(9) Insert Data. This routine provides an Open, Write, and Close in response to a single request. Other services used: Open for Writing, Write, and Close for Writing.

(10) Replace Item. This routine logically substitutes data from the user's buffer for existing data at any point in the data pool. It is composed of Open, Replace, and Close. Other services used: Open for Update, Replace, and Close for Update.

(11) Delete Item. This service routine is similar to Replace Item, but Delete Item replaces the named item with a mark indicating that the item is missing. The various kinds of indicators for a missing item are discussed in Paragraph 3.1.5. Other services used: Open for Update, Delete, and Close for Update.

3.7 DETAILED DESCRIPTION OF SERVICE ROUTINES

Each of the following paragraphs describes a service routine and is followed by the appropriate flowchart or charts.

3.7.1 Fetch Segment

3.7.1.1 Functional Description. This service routine calls on the executive program to read the desired segment into core.

3.7.1.2 Inputs. The inputs are:

(1) Name of the desired segment.

(2) Address of the beginning of the area into which the segment is to be read.

(3) Address of the area reserved for system control information; i.e., for the Access Parameter Table (APT).

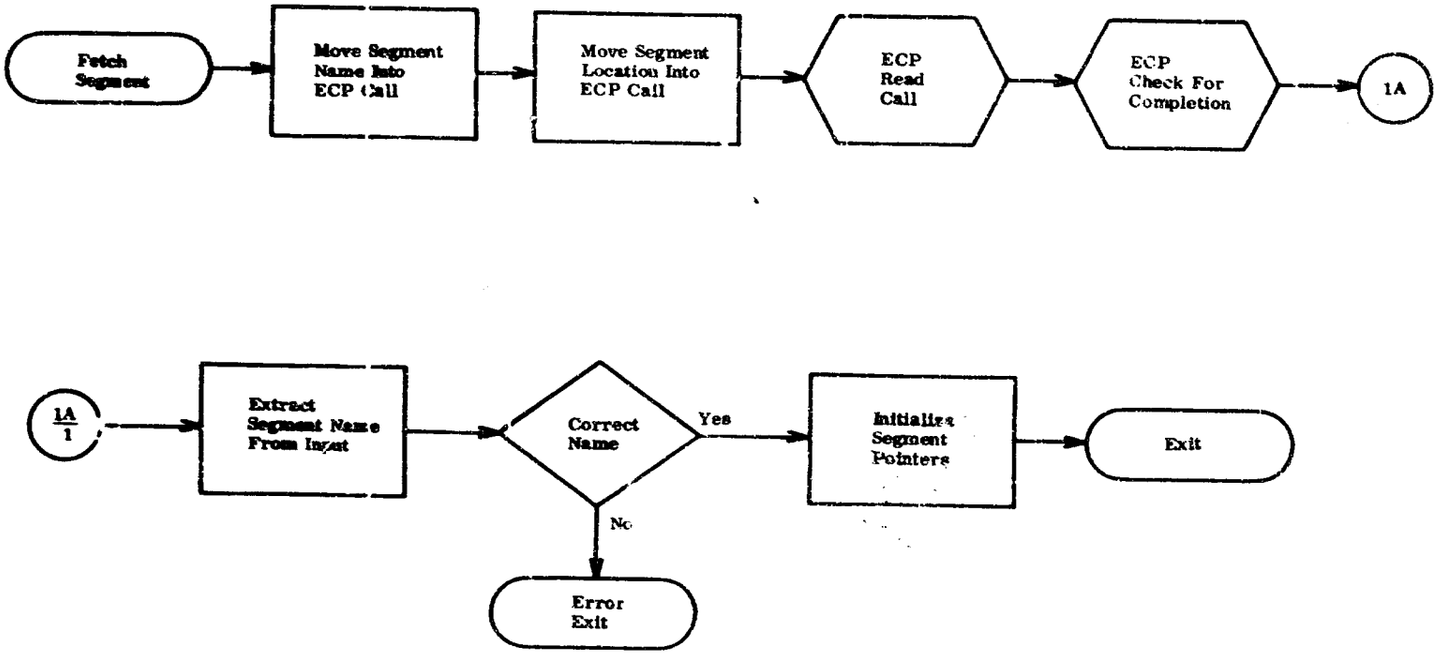
3.7.1.3 Results. The desired segment is brought into core.

3.7.1.4 Directories Used. None.

3.7.1.5 Services Used. None.

3.7.1.6 Jobs Used. None.

3.7.1.7 Method of Operation. Fetch inserts the variables into a Read request and issues this request to the executive program. This is followed by another request on the executive to check for the completion of the Read operation. When the executive returns control, Fetch checks the segment name against the requested name. The pointers for the segment index and the body are initialized and control is returned to the caller.



Fetch Segment
Sheet 1 of 1

3.7.2 Name Segment

3.7.2.1 Functional Description. This routine searches the Segment Name List (SNL) to find the name of the segment which contains the information associated with the given identifier.

3.7.2.2 Inputs. The inputs are:

(1) The caller supplies the address of the identifier for the desired item. This identifier can be an Item Position Code, an Item Class Code, or a term name. The latter two types of identifiers are prefixed by unique codes which cause them to fall into separate areas of the SNL.

(2) Address of the beginning of the area into which segments of the SNL can be read.

(3) Address of an area reserved for system control information (APT).

3.7.2.3 Results. The results are:

(1) Segment name of the segment containing the desired item.

(2) Segment Identifier Left (SIL). The identifier for the first item is the segment which contains the desired item.

(3) Segment Identifier Right (SIR). The identifier for the first item is the segment which logically follows the segment of the desired item.

(4) The segment pointers. The Bit Position Left and the index for the Segment Index are initialized.

3.7.2.4 Directories Used

(1) Segment Name List (SNL), see Section II.

(2) Private SNL. This is a small index which provides the search program with the initial breakdown of the SNL. It has the same format as the SNL; each entry contains an identifier, a segment name, and a flag.

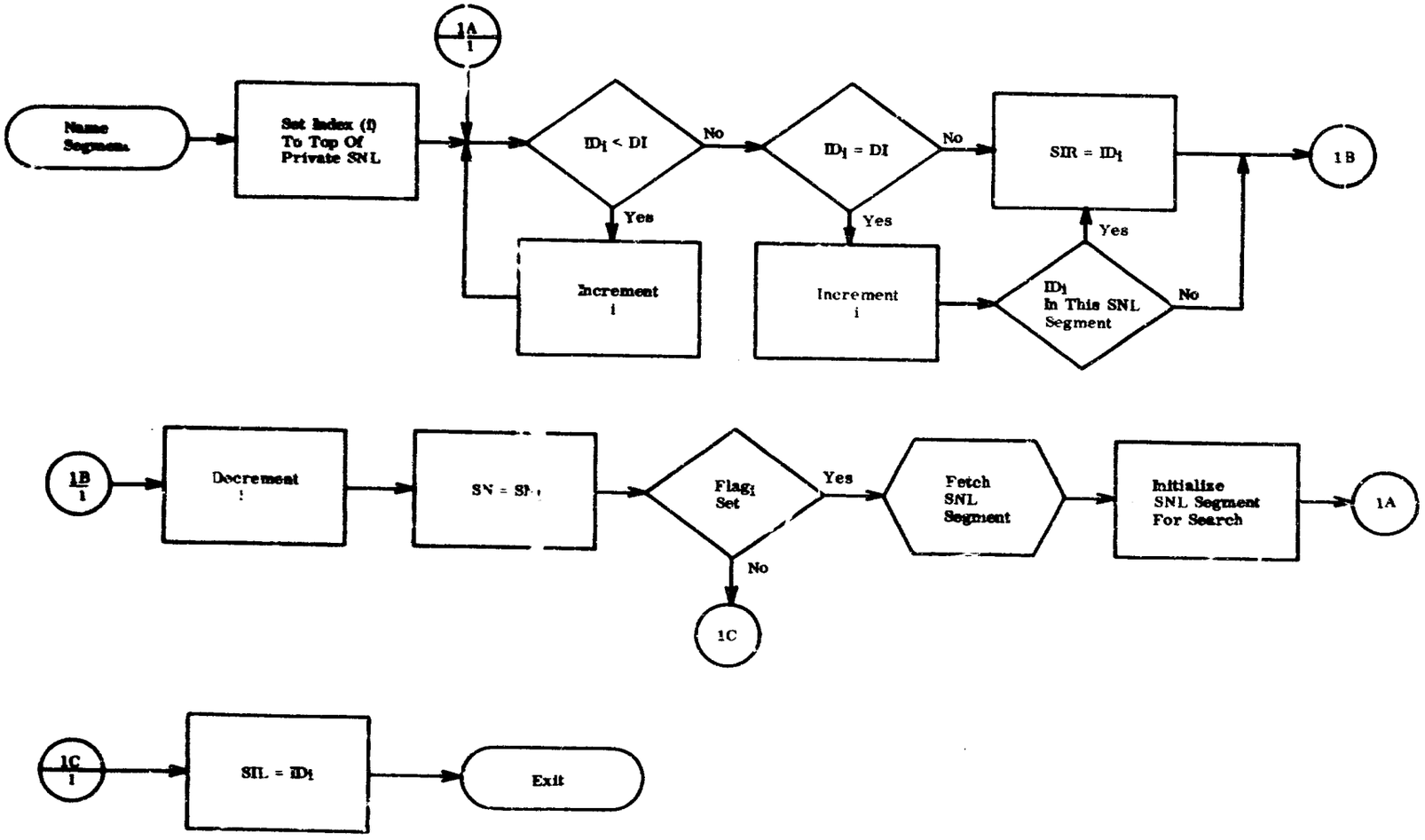
3.7.2.5 Services Used. Fetch Segment.

3.7.2.6 Jobs Used. None

3.7.2.7 Method of Operation. The following abbreviations are used in the flowcharts:

i	=	the index for the Segment Name List.
IDi	=	the current identifier in the Segment Name List.
DI	=	the identifier of the desired item.
SIR	=	Segment Identifier Right.
SIL	=	Segment Identifier Left.

Name Segment searches the private SNL to find the first ID that is greater than the identification of the desired item. This ID is saved because it might become the Segment Index Right of the desired segment. Then the segment name associated with the preceding ID is saved. If the flag associated with this segment name is set, it is merely naming a segment of the SNL. This segment is retrieved and searched in the same manner as the original search. Eventually, the routine comes to a terminal segment name, i.e., one which has a clear flag. This is the name for the segment containing the desired item. The ID associated with this SN becomes the Segment Index Left. Then, control is returned to the caller.



Name Segment
Sheet 1 of 1

3.7.3 Locate IL Entry

3.7.3.1 Functional Description. This routine skips over unwanted entries in an Item List segment and sets the pointers at an entry which corresponds to a given ICC.

3.7.3.2 Inputs. The inputs are:

(1) An Item List segment.

(2) The Item Goal Code. This is contained in the IL segment.

(3) The Access Parameter Table for that segment.

(4) An Item Class Code. This is the identifier of the desired IL entry.

3.7.3.3 Results. The results are:

(1) An index (i). This points to the desired entry in the IL segment.

(2) A pointer (p). This is set at the bit which begins the desired entry.

3.7.3.4 Directories Used. Item List.

3.7.3.5 Services Used. Stem, Skip Subitems.

3.7.3.6 Jobs Used. None.

3.7.3.7 Method of Operation. The following symbols are used in the flowchart:

SIL = Segment Identifier Left

DI = the identifier of the desired item

X = level of the stem

k = temporary storage for the level number of the highest level at which a difference exists between the DI and the current item.

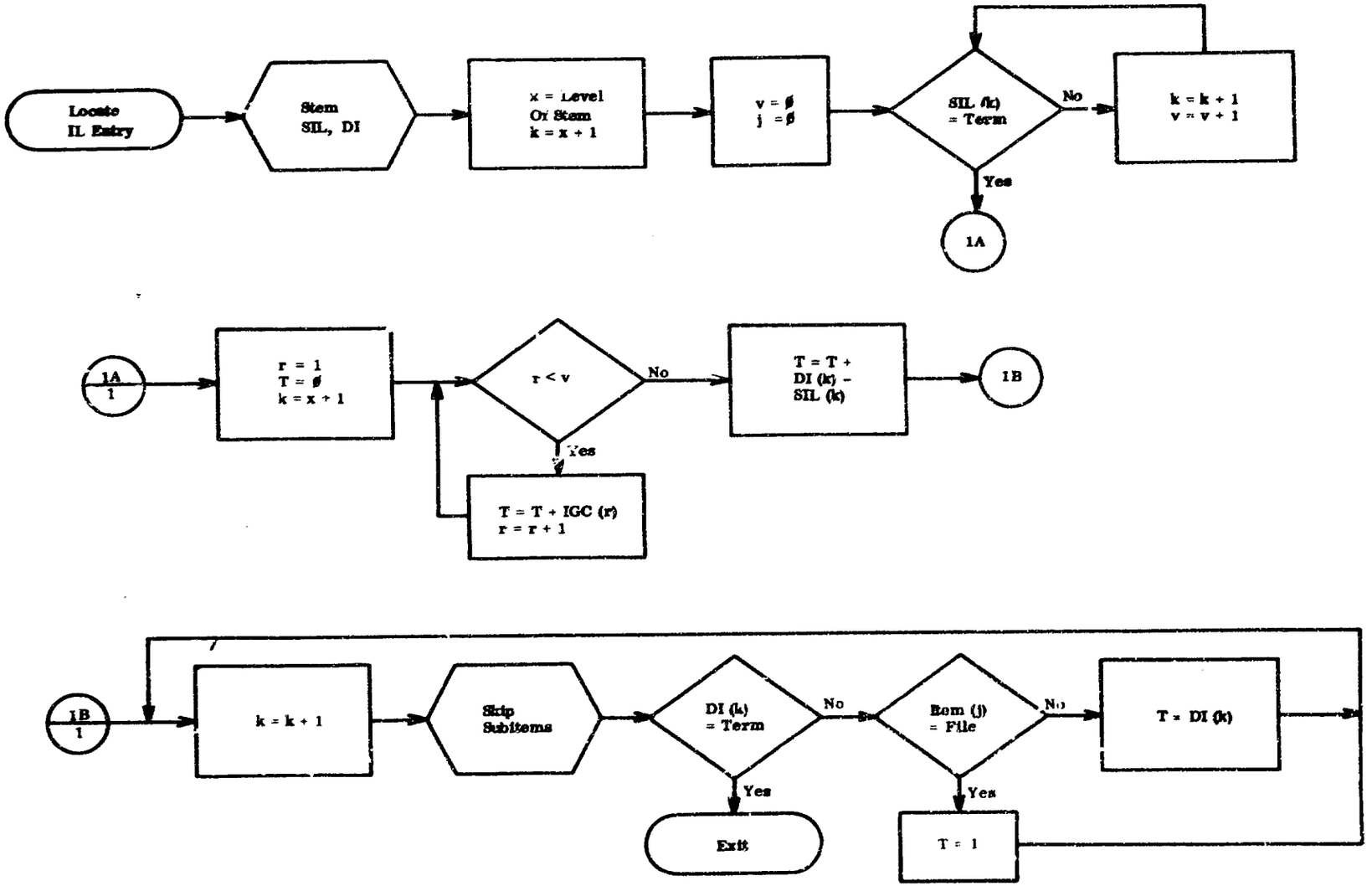
v = the number of Item Goal Code digits needed.

Flowchart symbols (Contd.)

j	=	index to the Item List segment
T	=	a temporary counter which reflects the number of items to be skipped.
r	=	a pointer which keeps track of the current Item Goal Code digit.
IGC	=	Item Goal Code

Locate IL Entry executes the Stem Routine. This determines the leftmost level (k) at which the DI differs from the SIL. Now the number of skips required to cause a step in the IPC at the level k is computed. The number of skips is equal to the sum of the IGC digits associated with all the SIL digits to the right of level k. To do this computation, the routine steps through the SIL counting the number of digits to the right of level k. Then the routine totals the proper number of IGC digits. Assume the SIL is 1.3.5.2.4.1 and the DI is 1.3.5.6.2.4. In that case, the use of the IGC digits moves the IPC from 1.3.5.2.4.1 to 1.3.5.3. Next the SIL at level k subtracted from the DI at level k. The remainder reflects more IL entries which be skipped. This would take the example from 1.3.5.3 to 1.3.5.6.

The k index is incremented by one, and all of the IL skips which have been ted to this point are performed. Then the actual digit values of the DI (from k to the end) are used to direct the skipping. This is done in the last loop of the owchart. This final operation would move the example from 1.3.5.6 to 1.3.5.6.2.4.



Locate IL Entry
Sheet 1 of 1

3.7.4 Step Lists

3.7.4.1 Functional Description. This routine steps the Level Pushdown List once and manipulates the index (j) of the Item List Table accordingly.

3.7.4.2 Inputs. The inputs to Step Lists are:

(1) A Level Pushdown List and its associated index, l. See Description in Paragraph 3.1.1.

(2) An Item List Table and its associated index, j.

(3) TSIZE. A temporary storage field which has been set by Step Item or by Write.

(4) EFFLAG. An end-of-file indicator which is set by Discount Item.

3.7.4.3 Results. The results are:

(1) The LPL and l are updated.

(2) The index j is updated.

(3) TSIZE is sometimes updated.

3.7.4.4 Directories Used. None.

3.7.4.5 Services Used. Discount Item.

3.7.4.6 Jobs Used. None

3.7.4.7 Method of Operation. If the current item is a record and the end-of-file indicator is set, Step Lists performs the following operations:

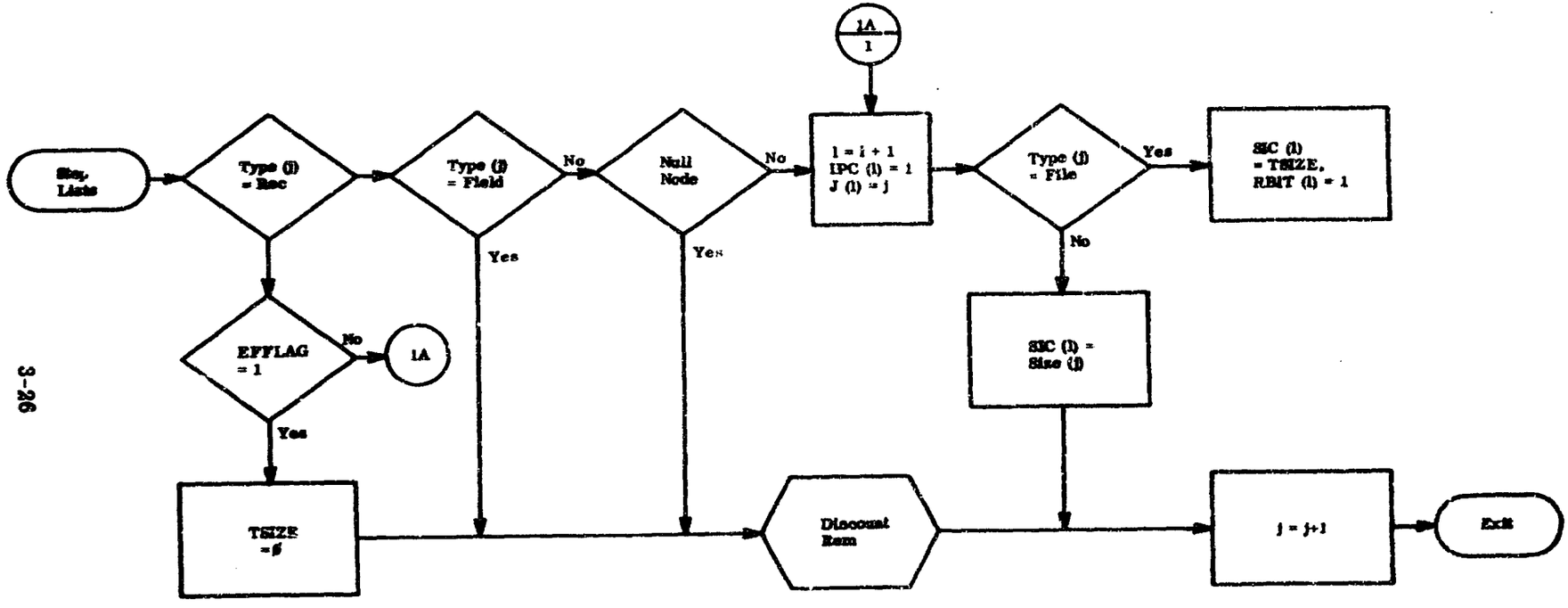
(1) The data size (TSIZE) is set to zero, thereby indicating end-of-file status.

(2) Discount Item is executed.

(3) The index j is stepped once, and the routine exits.

If the previous circumstances do not apply, Step Lists determines whether the current item is a nonterminal item. If it is, index l is incremented and a new entry is created for the LPL. The index j is stepped once, and the routine exits.

If the current item is a terminal item, the routine simply executes Discount Item, steps j, and exits.



Step Lists
Sheet 1 of 1

3.7.5 Discount Item

3.7.5.1 Functional Description. This routine adds one to the deepest level of the LPL which has not already reached the limit expressed in the Subsumed Item Count.

3.7.5.2 Inputs. The inputs are:

(1) An LPL and its index l. See Paragraph 3.1.1.
(2) An Item List Table and its index j.
(3) EFFLAG, an end-of-file indicator.

3.7.5.3 Results. The results are:

(1) The LPL, l, and j are updated.
(2) The EFFLAG is updated.

3.7.5.4 Directories Used. None.

3.7.5.5 Services Used. Skip Item.

3.7.5.6 Jobs Used. None.

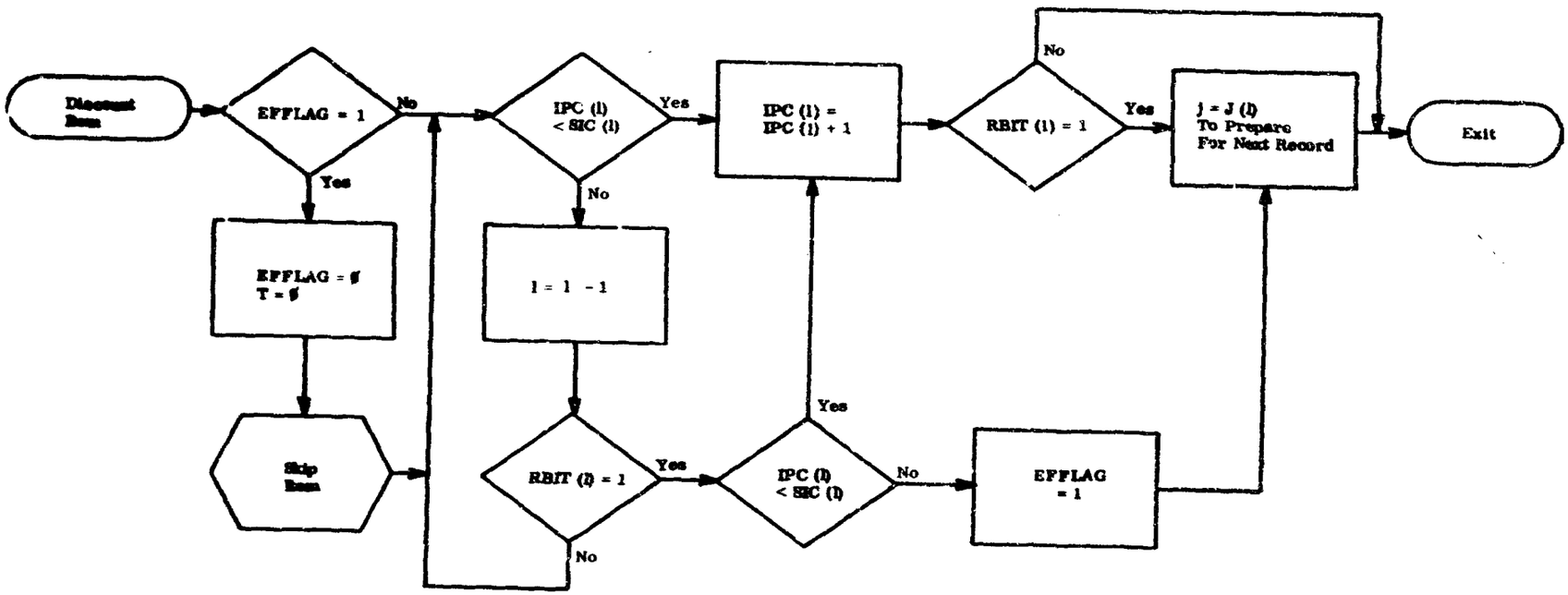
3.7.5.7 Method of Operation. The symbol T represents the count of IL entries to be skipped. The other symbols have been defined earlier in Paragraph 3.7.3.

The simplest path through this routine occurs as follows. The EFFLAG is 0, IPC (l) is less than SIC (l), and the current item is not a record. In such a case, Discount Item simply adds one to IPC (l) and exits.

A more complex case occurs when EFFLAG is 0 but IPC (l) is not less than SIC (l). In this case, the level is reduced by one. If the new level is not a record, the routine checks whether the new IPC (l) is less than SIC (l). If it is not, the loop continues until an IPC is found to be less than the corresponding SIC. Then the IPC is incremented and the RBIT (l) is examined. If RBIT (l) is zero, the routine exits; otherwise, j is reset to permit processing of the next record of the file. The index (j) is actually reset to the file entry of the ILT, but it (j) is stepped to the record entry before it is used.

A different case occurs if, when the level is reduced, the new IPC (l) is a record. In that case, the IPC (l) is compared to SIC (l): If SIC (l) is not greater, there are no records remaining in the file. The EFFLAG is set, and j is reset to the file entry. This causes the system to give an end-of-file status on the next call. If SIC (l) is greater than IPC (l), IPC (l) is incremented and j is reset to prepare for processing the next record.

A special case occurs when EFFLAG is /. This means that the caller will receive an end-of-file status on this request. EFFLAG is cleared, the entire record is skipped in the ILT, and the normal LPL updating is performed.



Discount Item
Sheet 1 of 1

3.7.6 Step Item

3.7.6.1 Functional Description. This routine steps all of the control pointers and data pointers to the next item, retrieving the next data segment when necessary.

3.7.6.2 Inputs. The inputs are:

(1) The current level number, l.

(2) The Item List Table and its index, j.

(3) A temporary level number (Y). This identifies the level of a missing data item and implies that no data exists at deeper levels.

(4) The data segment.

3.7.6.3 ..esults. The results are:

(1) TSIZE. The size of the current data item.

(2) TNULL. This is a binary switch which indicates whether a field is fixed length.

(3) g. This is a temporary storage for the preceding value of j.

(4) An updated, LPL, l, j and Y.

(5) BPA, BPL. These are two data pointers which delimit the current item if it is a field.

3.7.6.4 Directories Used. None.

3.7.6.5 Services Used. Step Lists, Get Next SX, and Define Field.

3.7.6.6 Jobs Used. None.

3.7.6.7 Method of Operation. Step Item always clears TSIZE and TNULL. Therefore, these fields contain zero unless explicitly set elsewhere in the routine. The routine skips all null nodes and all items with a No Data Flag set.

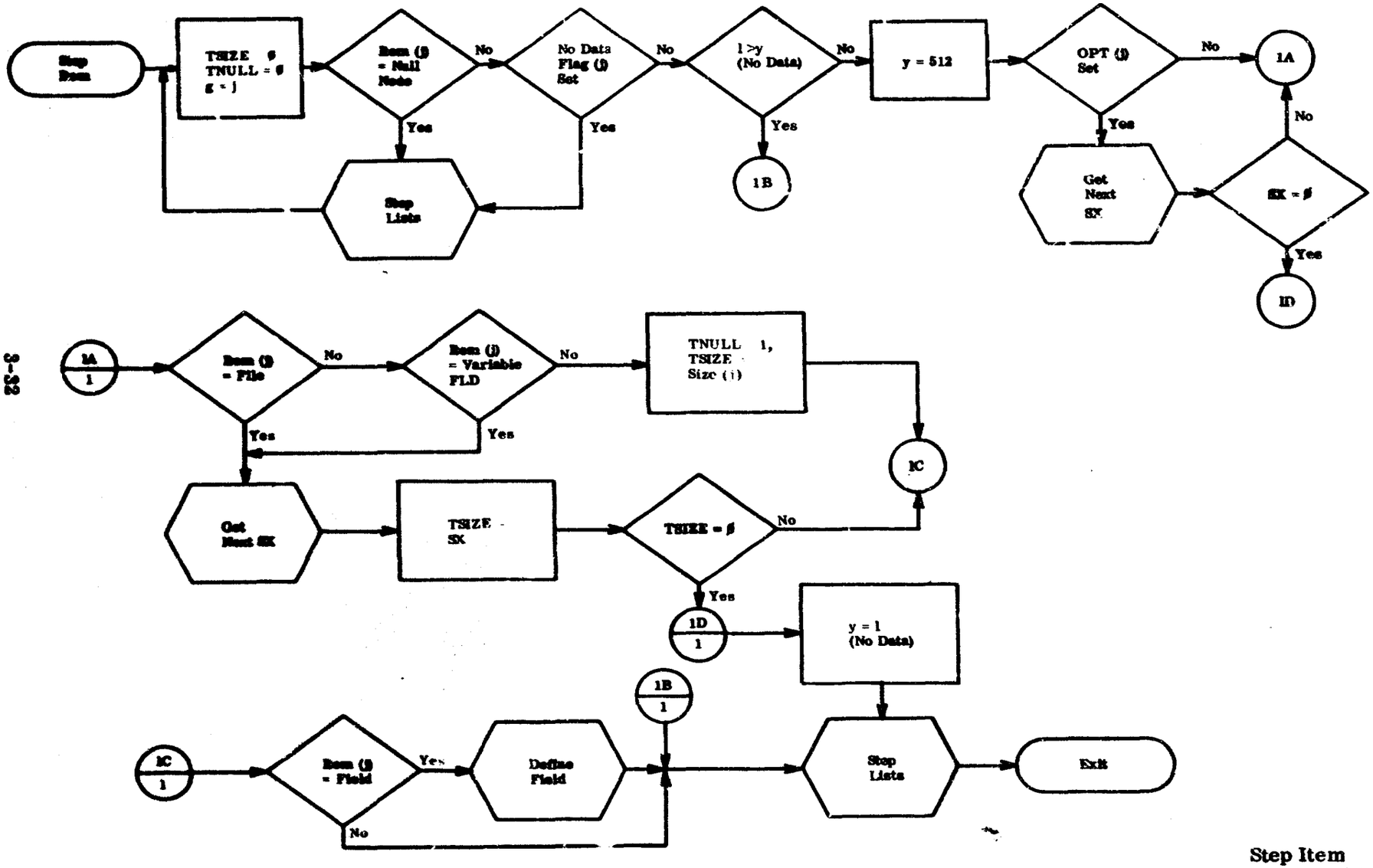
If the current level (l) is greater than Y, this indicates that the parent of the current item is missing. The routines executes Step Lists and exits.

If the segment index declares that an optional item is missing, Step Item puts the current level (l) into Y, executes Step Lists, and exits.

If none of the above exits have occurred, Step Item sets the TSIZE field. For files and variable length fields, the size is taken from the segment index. Otherwise, the size entry of the Item List Table is moved to TSIZE. If the item is not a variable length field, TNULL is set to 1.

After TSIZE is set, it is tested. If it is zero, there is no data for the current item. The current level (1) is put into Y, Step Lists is executed, and the routine exits. If the data is present, the current item type is checked. If it is not a field, Step Lists is executed, and the routine exits.

For fields, Define Field is used to compute the delimiting field pointers. Then Step Lists is executed and the routine exits.



Step Item
Sheet 1 of 1

3.7.7 Define Field

3.7.7.1 Functional Description. This routine saves the current body pointer (Bit Pointer Left) and steps the pointer over the current item. This makes available two pointers which delimit the current field.

3.7.7.2 Inputs. The inputs are:

(1) The data segment.

(2) The current body pointer (BPL).

(3) The Item List Table and its index, j.

(4) TNULL. This is an indicator which implies the need for a null bit.

(5) TSIZE. This is the size of the current data item.

(6) Segment Index Right (SIR).

3.7.7.3 Results. The results are:

(1) BPA and BPL. These are the two pointers which delimit the current field.

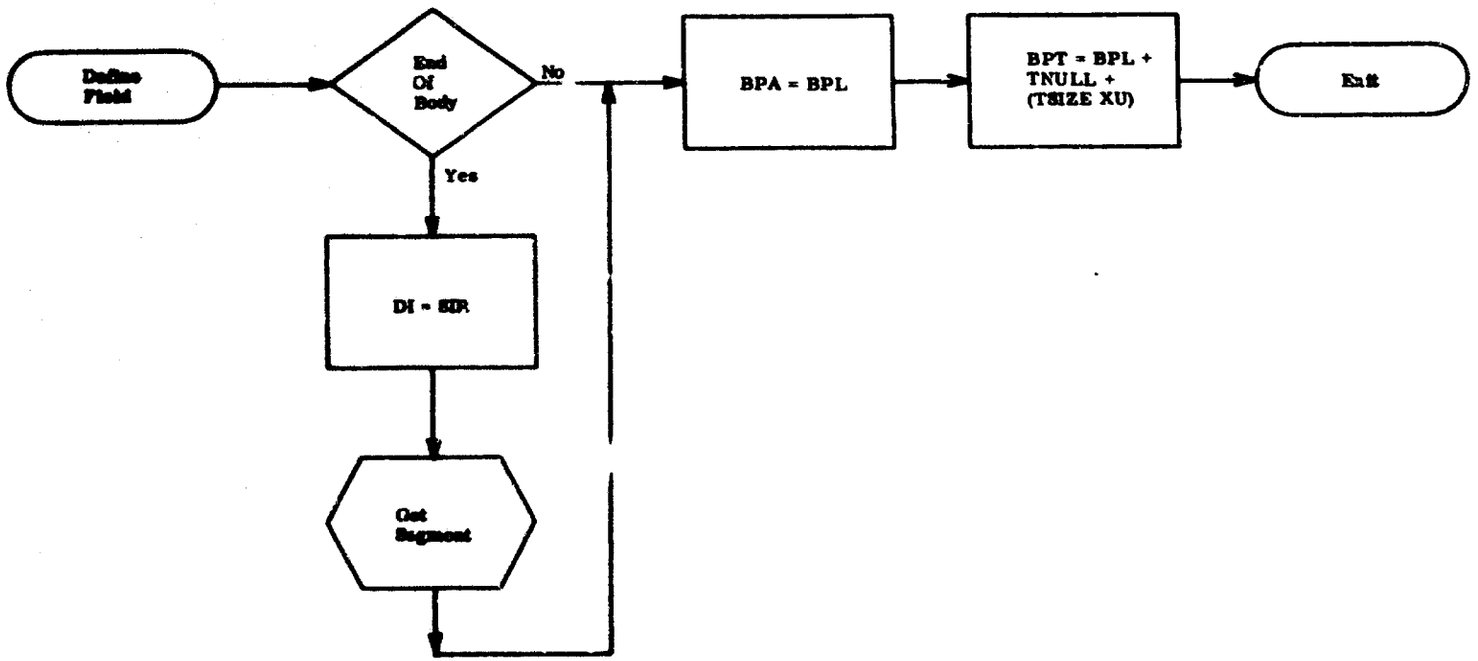
(2) A new data segment. If the field was not in the current segment, a new data segment is retrieved.

3.7.7.4 Directories Used. None.

3.7.7.5 Services Used. Get Segment.

3.7.7.6 Jobs Used. None.

3.7.7.7 Method of Operation. If the current segment is already exhausted, Define Field calls on Get Segment to retrieve the next data segment and to initialize the data pointers. Define Field saves the current body pointer, then computes a new one. The computation is primarily the multiplication of TSIZE, which has been set by Step Item, times U, the number of bits in one unit of the field. Define Field selects the appropriate U-value, based on the type of the field as defined in the ILT. TNULL is added to this product to provide space for the null bit if the field is fixed length.



Define Field
Sheet 1 of 1

3.7.8 Get Next SX

3.7.8.1 Functional Description. This routine unpacks the next entry of the segment index and moves it to temporary storage.

3.7.8.2 Inputs. The inputs are:

(1) The data segment.

(2) The current SX pointer.

(3) Segment Index Right (SIR).

3.7.8.3 Results. The results are:

(1) An SX entry in temporary storage.

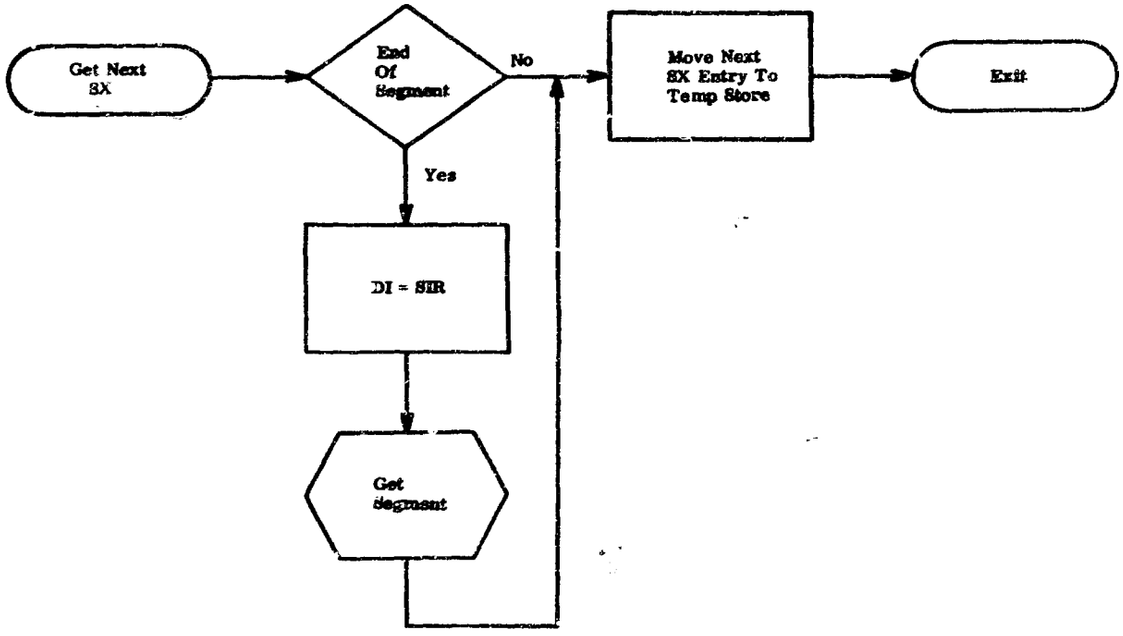
(2) A new data segment, if the current SX was exhausted.

3.7.8.4 Directories Used. None.

3.7.8.5 Services Used. Get Segment.

3.7.8.6 Jobs Used. None.

3.7.8.7 Method of Operation. If the current segment is already exhausted, Get Next SX calls on Get Segment to retrieve the next data segment and to initialize the data pointers. Get Next SX unpacks the segment index entry, moves it to temporary storage, and updates the SX pointer.



Get Next SX
Sheet 1 of 1

3.7.9 Skip Item

3.7.9.1 Functional Description. This routine moves the index (j) of the Item List Table over "T" items and all of their subsumed items.

3.7.9.2 Inputs. The inputs are:

(1) The Item List Table and its index, j.

(2) T. This is the number of items to be skipped.

3.7.9.3 Results. The results are:

(1) Index j is updated.

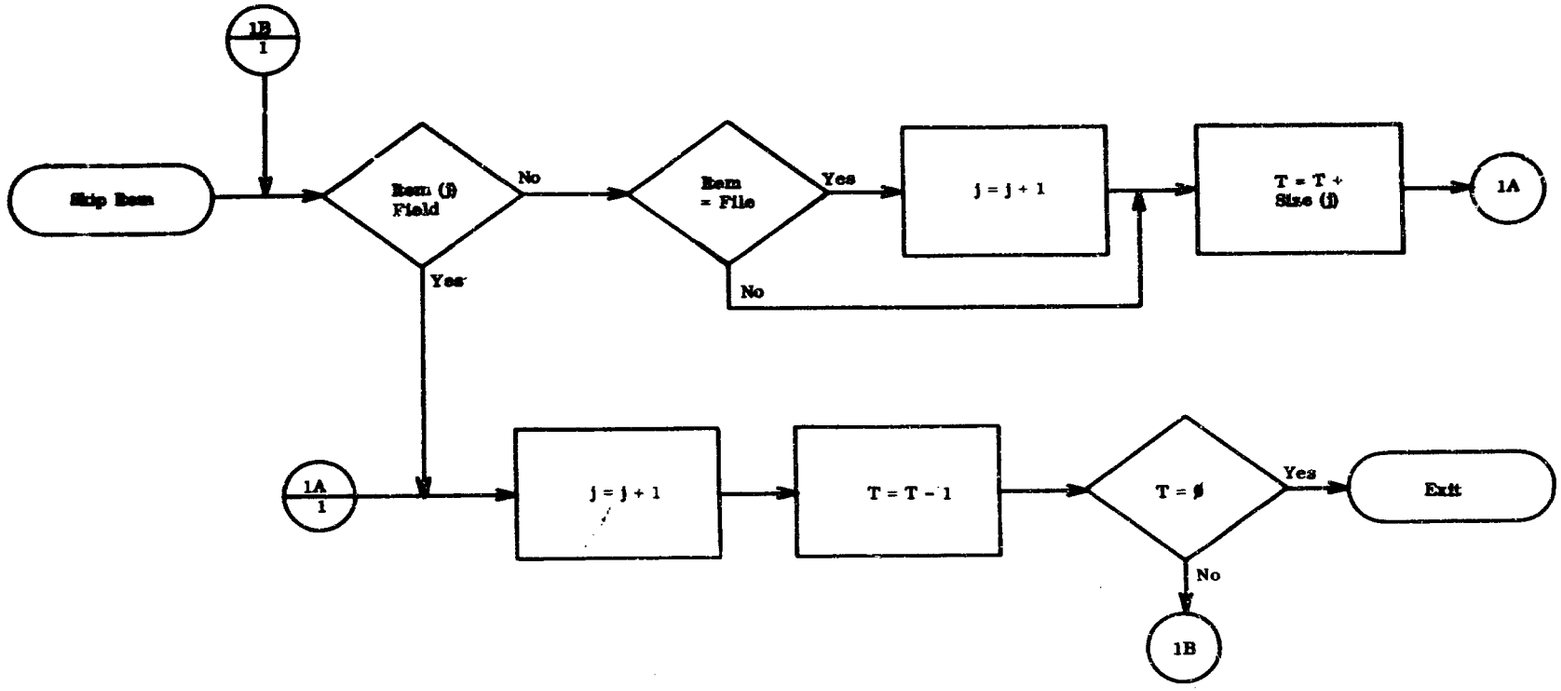
(2) T is reduced to zero.

3.7.9.4 Directories Used. None.

3.7.9.5 Services Used. None.

3.7.9.6 Jobs Used. None.

3.7.9.7 Method of Operation. For records and statements, Skip Item adds the size entry from the Item List Table to the skip counter, T. Otherwise, the routine amounts to a loop which increments j and decrements T until T is zero.



Skip Item
Sheet 1 of 1

3.7.10 Define Segment

3.7.10.1 Functional Description. This routine steps the Item List Table from a parent item to the first item in a segment, and it initializes the data pointer of the segment.

3.7.10.2 Inputs. The inputs are:

(1) The identifier of the desired item. In this case, the ID is the Segment Index Left.

(2) k. This is the level of the parent item. This item can be the opened item or the stem of the segment of the opened item.

(3) An Item List Table.

(4) An area for the Level Pushdown List. The LPL already contains at least the IPC digit on level k and the J entry on level k + 1.

(5) The data segment.

3.7.10.3 Results. The results are:

(1) An updated LPL, l, and j.

(2) Y. This control word is reset to its maximum value, which renders it inoperative until it is set.

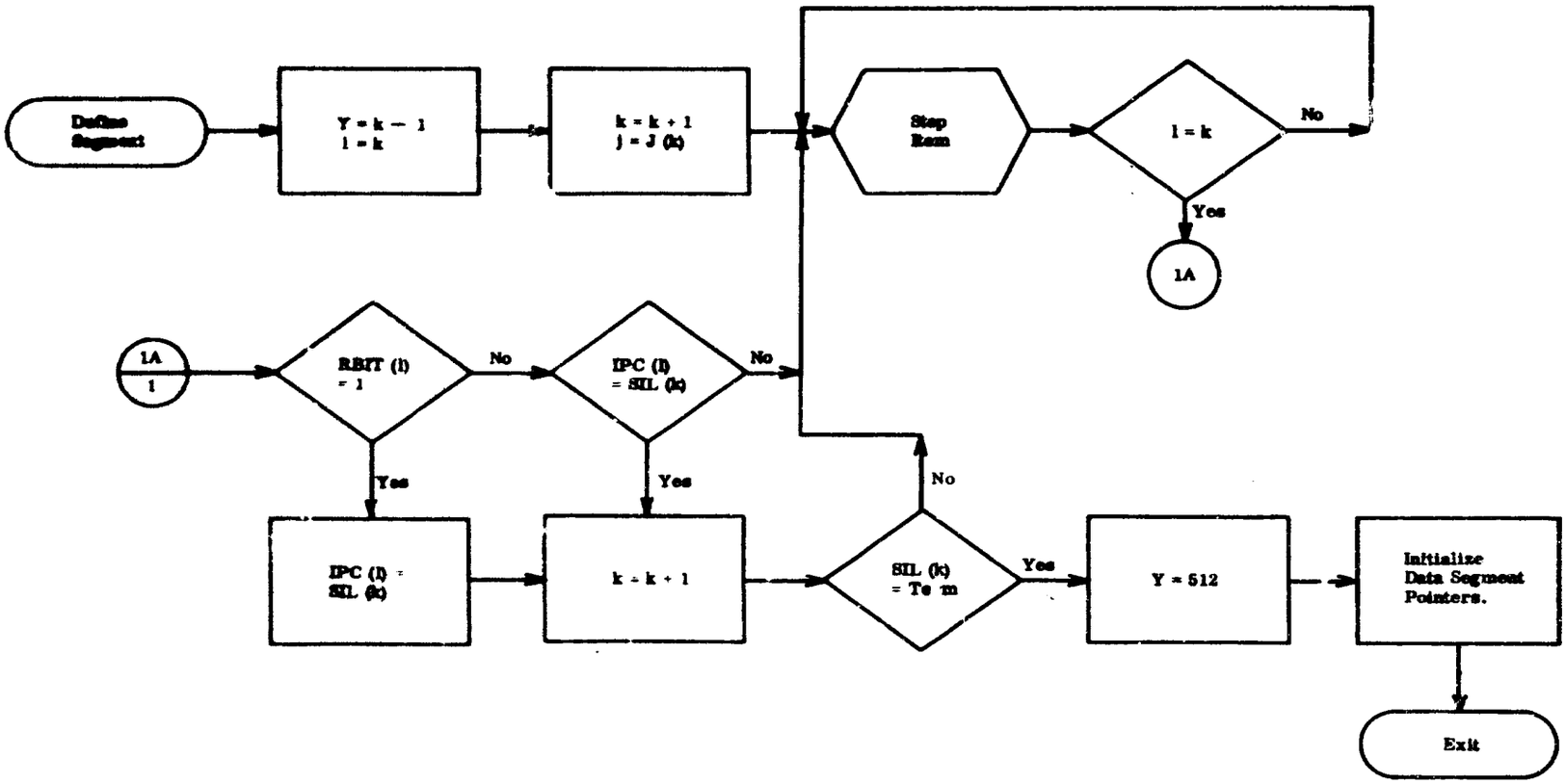
(3) The data pointers. These are initialized to the beginning of the body and the beginning of the SX.

3.7.10.4 Directories Used. None.

3.7.10.5 Services Used. Step Item.

3.7.10.6 Jobs Used. None.

3.7.10.7 Method of Operation. Define Segment sets Y so that no data is manipulated when Step Item is executed. The current level (l) is forced equal to k. (See input description). Index j is set to the parent item at which the stepping procedure is to begin. After this, Define Segment simply keeps calling on Step Item until the IPC of the current item is equal to the given IPC.



Define Segment
Sheet 1 of 1

3.7.11 Read

3.7.11.1 Functional Description. This routine edits selected data fields from a data segment into the user's buffer.

3.7.11.2 Inputs. The inputs are:

(1) The Buffer Description List (BDL). See Paragraph 3.5.

(2) The address of the user's buffer.

(3) Access Parameter Table. This contains the LPL and the system control information for a given data segment.

3.7.11.3 Results. The results are:

(1) The data fields. These are delivered to the user's buffer.

(2) Status indicators. These are put into the BDL as required.

(3) System Pointers. The data pointers and the control pointers in the APT are updated for each item processed.

(4) New Data Segment. If required to satisfy the request, new data segments are retrieved.

3.7.11.4 Directories Used. None.

3.7.11.5 Services Used. Step Item, Extract, and Seek.

3.7.11.6 Jobs Used. None.

3.7.11.7 Method of Operation. The following symbols used in the flowcharts have not been previously explained:

(1) TL. This is a temporary storage for the level number of the item being processed.

(2) TIPC. This is a temporary storage for an IPC, which is used as an input to Seek.

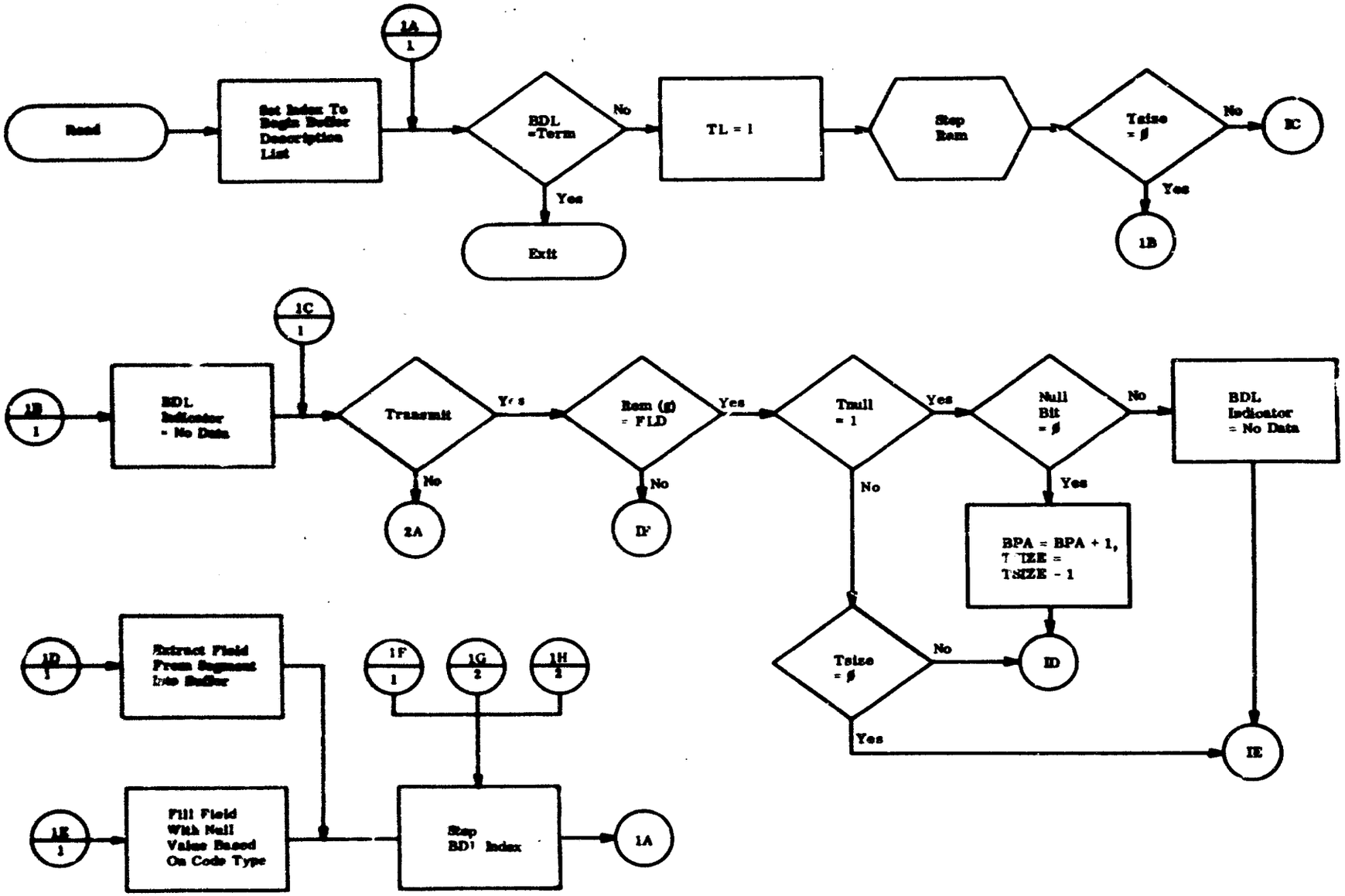
(3) g. This is a temporary storage where the old value of j is saved before j is stepped. The Read routine uses Step Item to identify the type and size of the next input item. Read itself interprets the Buffer Description List. There are two principal operations: Transmit and

Skip. For nonterminal items, Transmit simply requires stepping to the next item. Actually, this stepping is done at the very beginning of the routine; this is the reason for using index g instead of j.

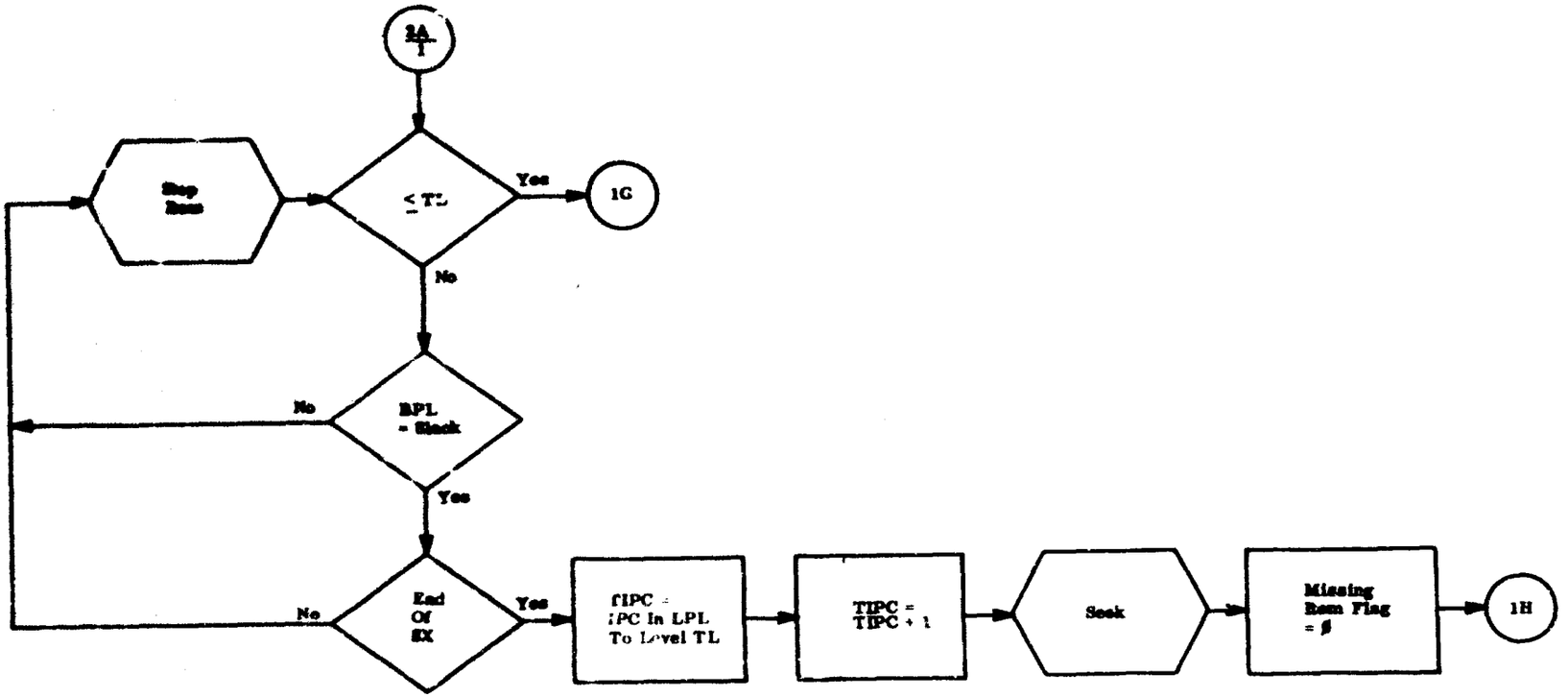
For fields, Transmit requires that data be put in the user's buffer. If the item is missing, the buffer field is filled with a null value. If the item is present, the Extract routine is used to edit the field from the data segment to store it in the buffer field.

For a Skip operation, the Read routine executes Step Item until a step occurs at level TL indicating that all subitems of the item being processed have been skipped. If the end of the data segment is reached before this operation is completed, Read turns to a random retrieval. It calls on the Seek routine to move the data pointers and the control pointers to the end of the item being processed. To use Seek, an IPC is generated which might not correspond to a real item, but which causes Seek to do the required work. Since the item might be nonexistent, the Read routine clears the missing item indicator.

The Read routine continues stepping through the BDL and performing Transmit or Skip operations until all entries of the BDL have been processed.



Read
Sheet 1 of 2



Read
Sheet 2 of 2

3.7.12 Write

3.7.12.1 Functional Description. This routine causes the data in the caller's buffer area to be edited, packed into an output segment, and written into the data pool.

3.7.12.2 Inputs. The inputs are:

(1) The Buffer Description List (BDL), see Paragraph 3.5.

(2) The address of the user's buffer.

(3) The Access Parameter Table. This contains the LPL and the other system control information for a given data segment.

3.7.12.3 Results. The results are:

(1) Output Segment. The user's data is put into the output body, and the required control values are put into the output segment index.

(2) System Pointers. The data pointers and the control pointers in the APT are updated for each item processed.

(3) EFFLAG. The end-of-file indicator is set when the BDL entry for a record item is zero.

(4) TSIZE. The data size is set. This is used either by the Compare routine or by the Step Lists routine.

3.7.12.4 Directories Used. None.

3.7.12.5 Services Used. Step Lists, Discount Item, Skip Item, Pack SX, and Compose.

3.7.12.6 Jobs Used. None.

3.7.12.7 Method of Operation. The following new symbols are used in the flowcharts:

(1) TSX. This is the temporary segment index, where SX values are kept temporarily to be sure that the go into the same segment as the associated field.

(2) NDF. This is the No Data Flag of the Item List.

The Write routine itself steps through the user's Buffer Description List. It calls on the Step Lists routine when the system lists must be stepped.

For each item, Write tests the option code in the ILT. If the code is set, Write tests the BDL entry to determine whether the item is present in the buffer. The option code in the temporary segment index is set accordingly. If the item is missing, and if it is a field, Write goes into its Item Completion Procedure. This procedure includes:

(1) The stepping of the system lists.

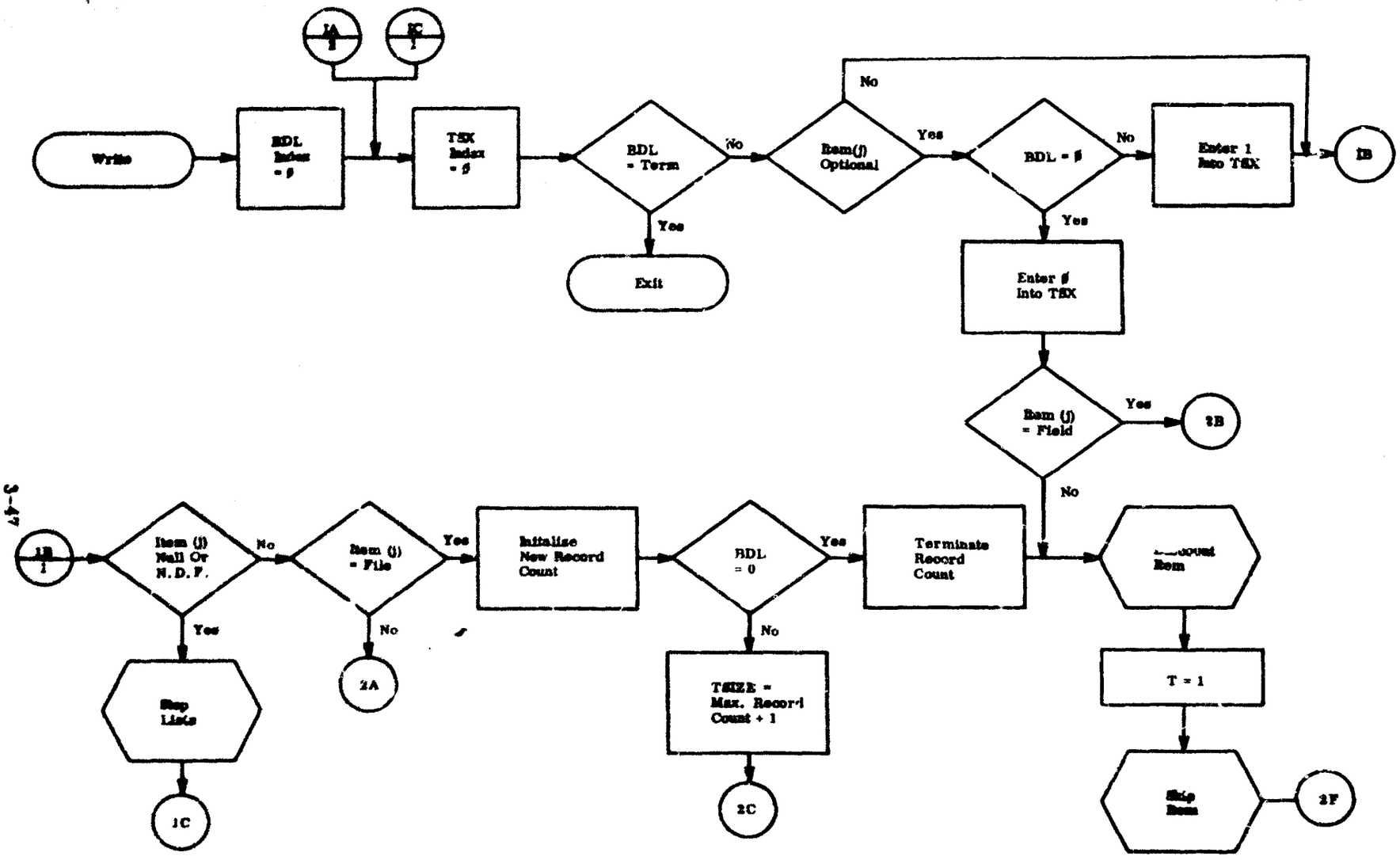
(2) The packing of the temporary segment index into the output segment.

(3) The stepping of the BDL to the next entry.

If the item is missing, and if it is a nonterminal item, Discount Item is executed to account for this item in the system lists. Next, Skip Item is executed to pass over any subsumed items. Then the segment index is packed and the BDL is stepped.

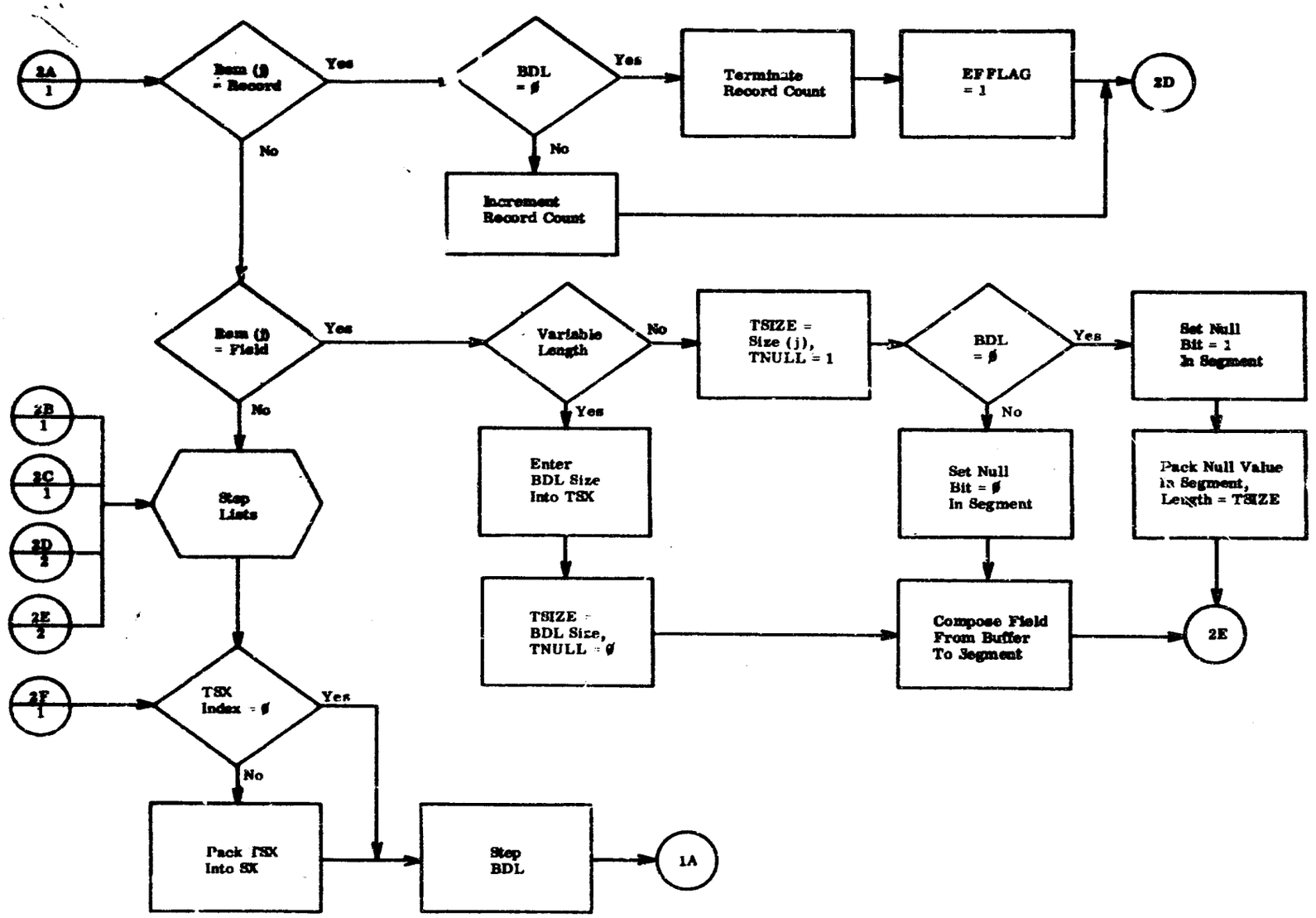
For nonoptional items and optional items which are present, the Write routine is subdivided according to item type. After the unique processing, the branches come back together for the item completion procedure.

The Write routine repeats the above sequence until it encounters the termination symbol in the BDL, whereupon the end of the request is signaled.



S-47

Write
Sheet 1 of 2



Write
Sheet 2 of 2

3.7.13 Locate Item

3.7.13.1 Functional Description. The routine steps through a data segment until the system pointers are set at a desired item.

3.7.13.2 Inputs. The inputs are:

(1) The Access Parameter Table.

(2) The IPC of the desired item.

3.7.13.3 Results

(1) The Access Parameter Table. The LPL and all control pointers and data pointers are set at the desired item.

(2) Missing Item Code. If the given identifier does not correspond to any node in the data pool, the routine sets this error code.

3.7.13.4 Directories Used. None.

3.7.13.5 Services Used. Stem and Step Item.

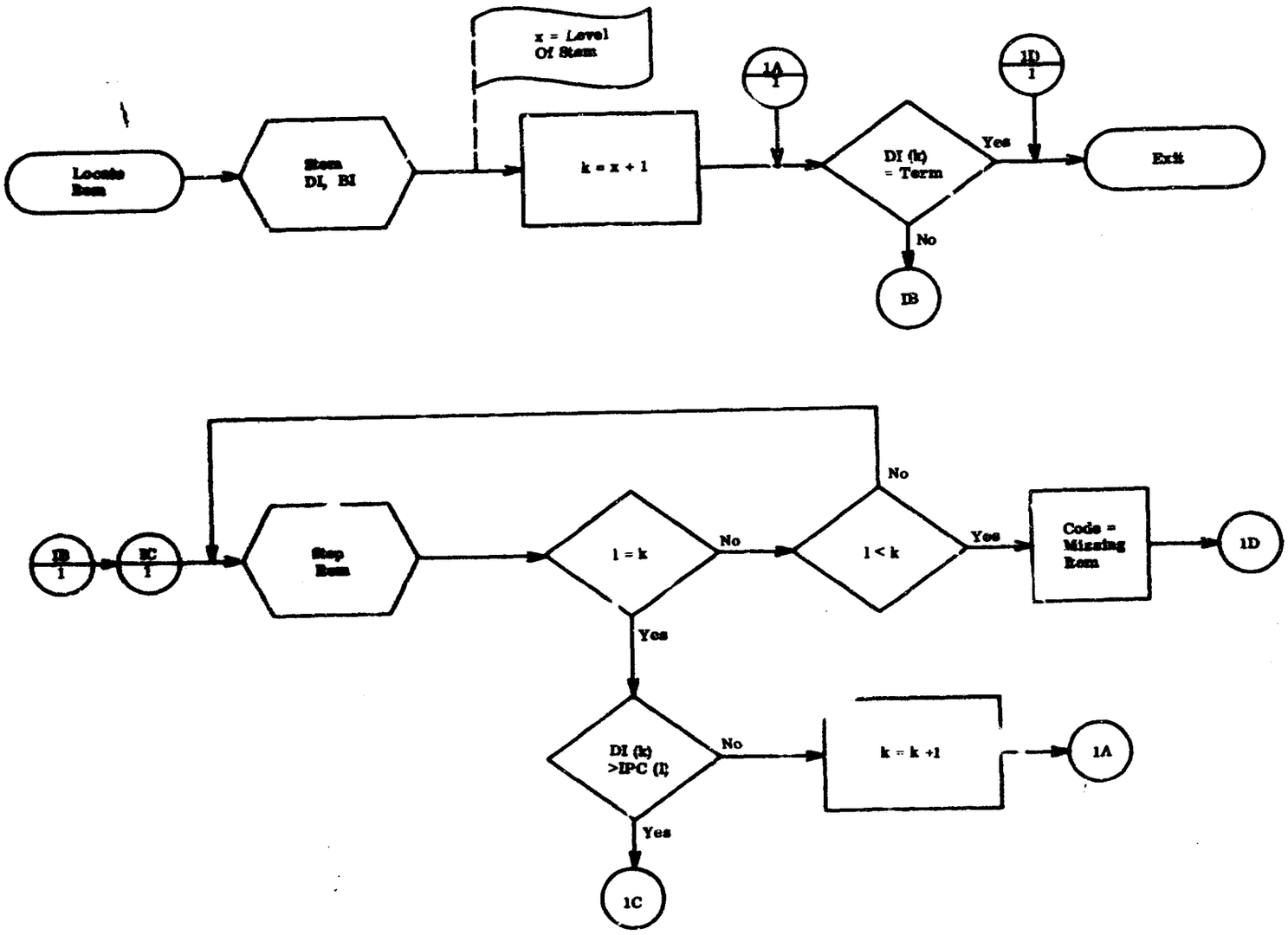
3.7.13.6 Jobs Used. None.

3.7.13.7 Methods of Operation. The following new symbols are used in the flowchart:

(1) BI. This represents the identifier (IPC) of the Base Item, i.e., the item at which the system pointers are set when control is transferred to the Locate Item routine.

(2) k. This represents a level number. It is the leftmost level at which the IPC of the desired item differs from the IPC of the current item.

Locate Item begins by executing the Stem routine, which matches the current IPC against the IPC of the desired item. This produces X, which is the number of consecutive IPC digits (starting from the left) which match. Locate Item executes the Step Item routine repeatedly until the IPC's match on digit X + 1. Then Step Item is used until the current IPC matches the desired item on digit X + 2. This procedure continues until the terminate symbol is detected in the IPC of the desired item.



Locate Item
Sheet 1 of 1

3.7.14 Seek

3.7.14.1 Functional Description. This routine sets the data pointers and the control pointers to the item identified by the requestor.

3.7.14.2 Inputs. The inputs are:

(1) The Access Parameter Table.

(2) The identifier of the desired item. The IPC of the opened item is assumed to be the beginning of the identifier. The caller need only supply a suffix for this IPC. The suffix may consist of IPC digits, a relative item number, or a relative item number with a record number. If IPC digits are not submitted as the given identifier, the Seek service executes a translation routine before performing the search for the desired item.

3.7.14.3 Results. The results are the same as those for Locate Item, namely:

(1) The Access Parameter Table. All controls are set at the desired item.

(2) Missing Item Code. This is set if a non-existent item has been requested.

3.7.14.4 Directories Used. None.

3.7.14.5 Services Used. Name Segment, Fetch Segment, Define Segment, and Locate Item.

3.7.14.6 Jobs Used. None.

3.7.14.7 Method of Operation. The following symbols are used in the flowchart:

(1) DI. This is the IPC of the desired item.

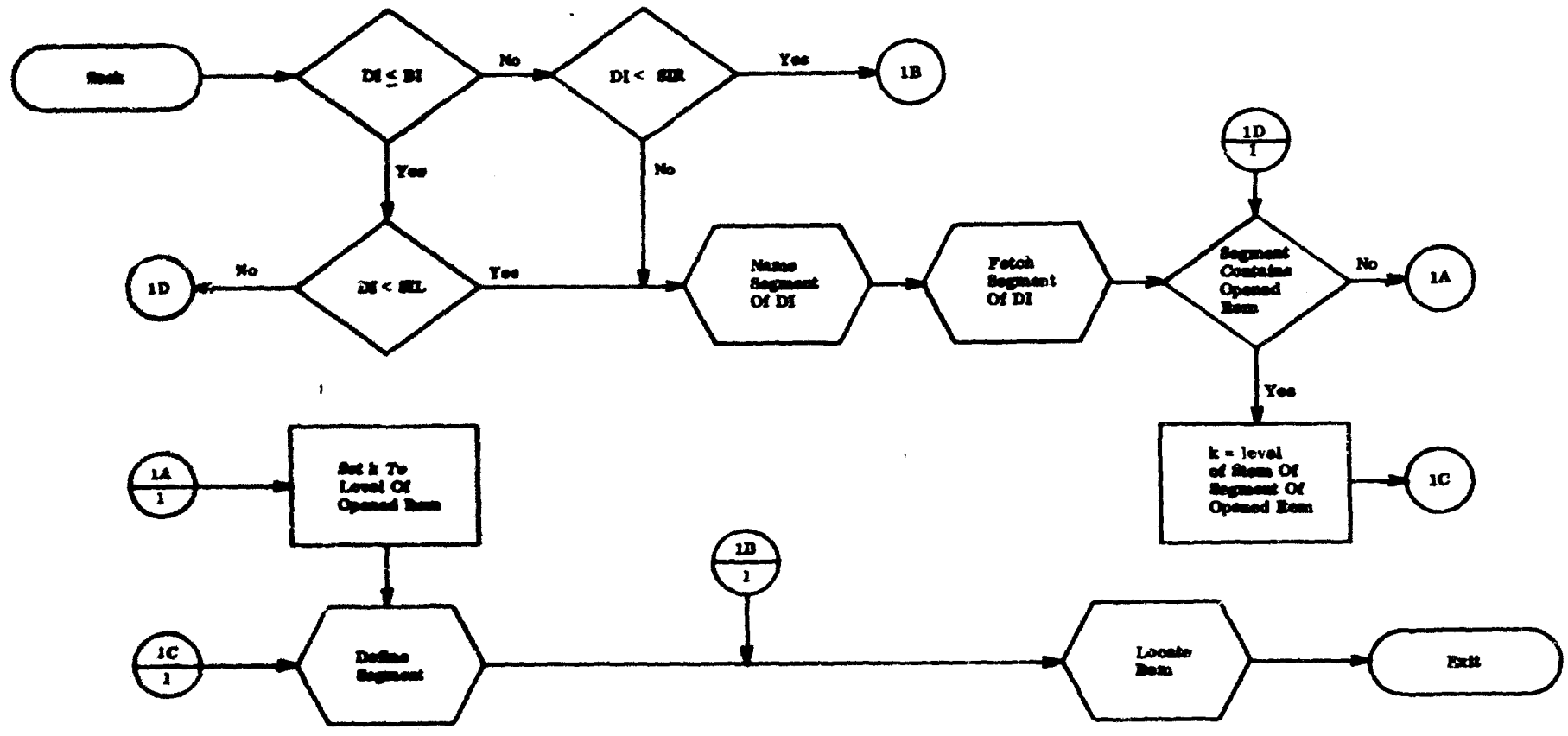
(2) BI. This is the IPC of the current item at the time of the request, i.e., the Base Item.

(3) k. This is a level number which is an input to Define Segment.

The Seek routine performs three tests to determine where the desired item is in relation to the current item. If the desired item falls below the current item in

the current segment, only a Locate Item operation is required. If the desired item is in the current segment, but above the current item, Define Segment is used to initialize the system controls to the beginning of the segment. Then, Locate Item is executed.

If the desired item is not in the current segment, Name Segment and Fetch Segment are executed to obtain the required segment. Then Define Segment and Locate Item are employed to set the system controls to the desired item.



Seek
Sheet 1 of 1

3.7.15 Seek with Copy

3.7.15.1 Functional Description. This routine is used during an update operation to move a string of items from an input segment into an output segment and to cause segments to be stored when they are full.

3.7.15.2 Inputs. The inputs are:

(1) The identifier of the desired item. This input has the same nature as in a Seek request. See 3.7.14. The Seek with Copy routine copies input data up to but excluding this item.

(2) Two Access Parameter Tables. System control information is needed both for the input item and for the output item.

3.7.15.3 Results. The results are:

(1) New Data Segment(s).

(2) System Pointers. The pointers for the input item are set at the requested item. The output indexes are the same as the input indexes, but the record numbers in the output item may differ from those of the input.

3.7.15.4 Directories Used. None.

3.7.15.5 Services Used. Stem, Step Item, Step Lists, Move and Pack SX.

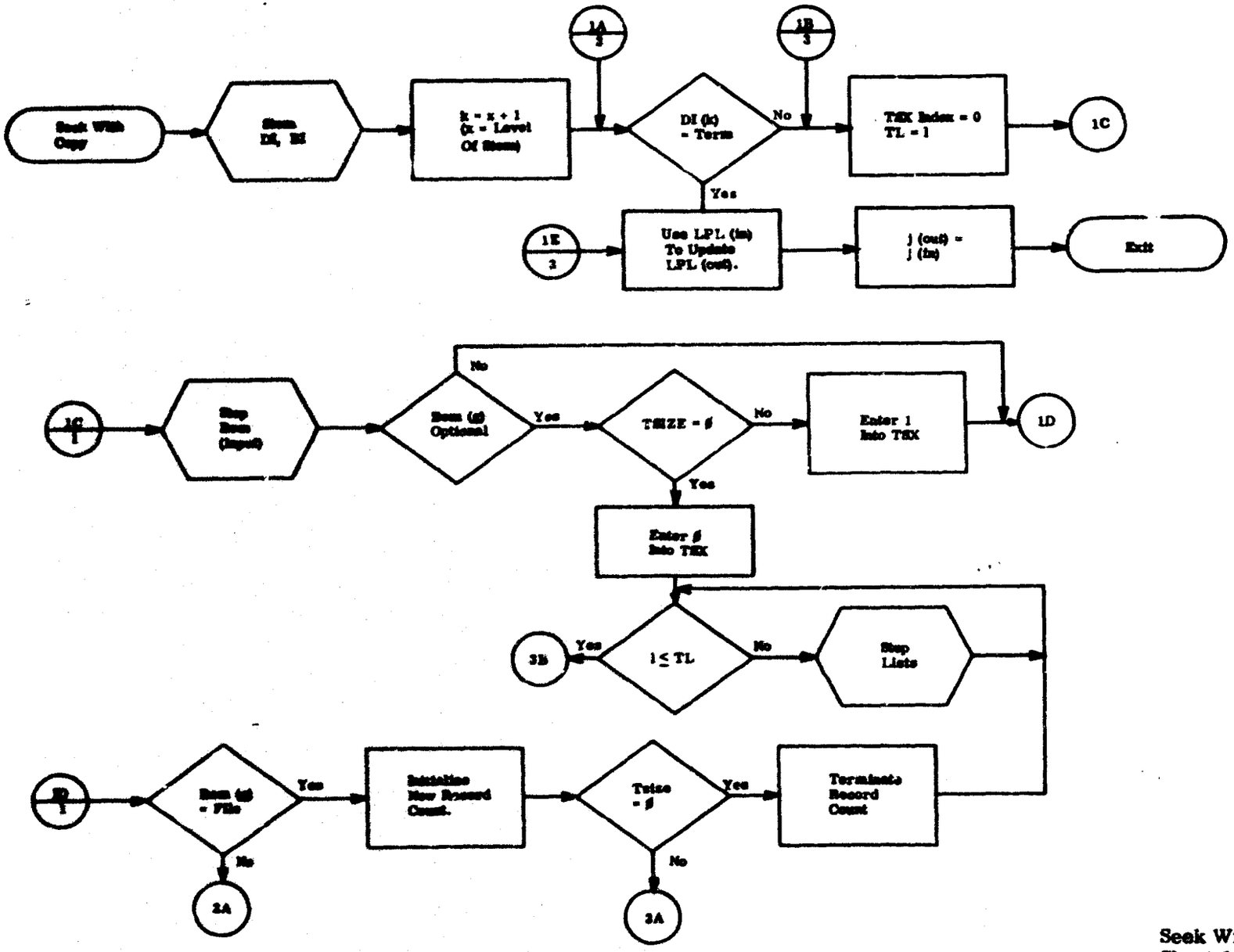
3.7.15.6 Jobs Used. None.

3.7.15.7 Method of Operation. The major control in this routine is similar to the control in Locate Item. The current IPC and the IPC of the desired item are stemmed. Then, after each item is processed, the IPC's are compared at the current level of difference (k). When the entire IPC of the desired item matches the current IPC, the routine exits.

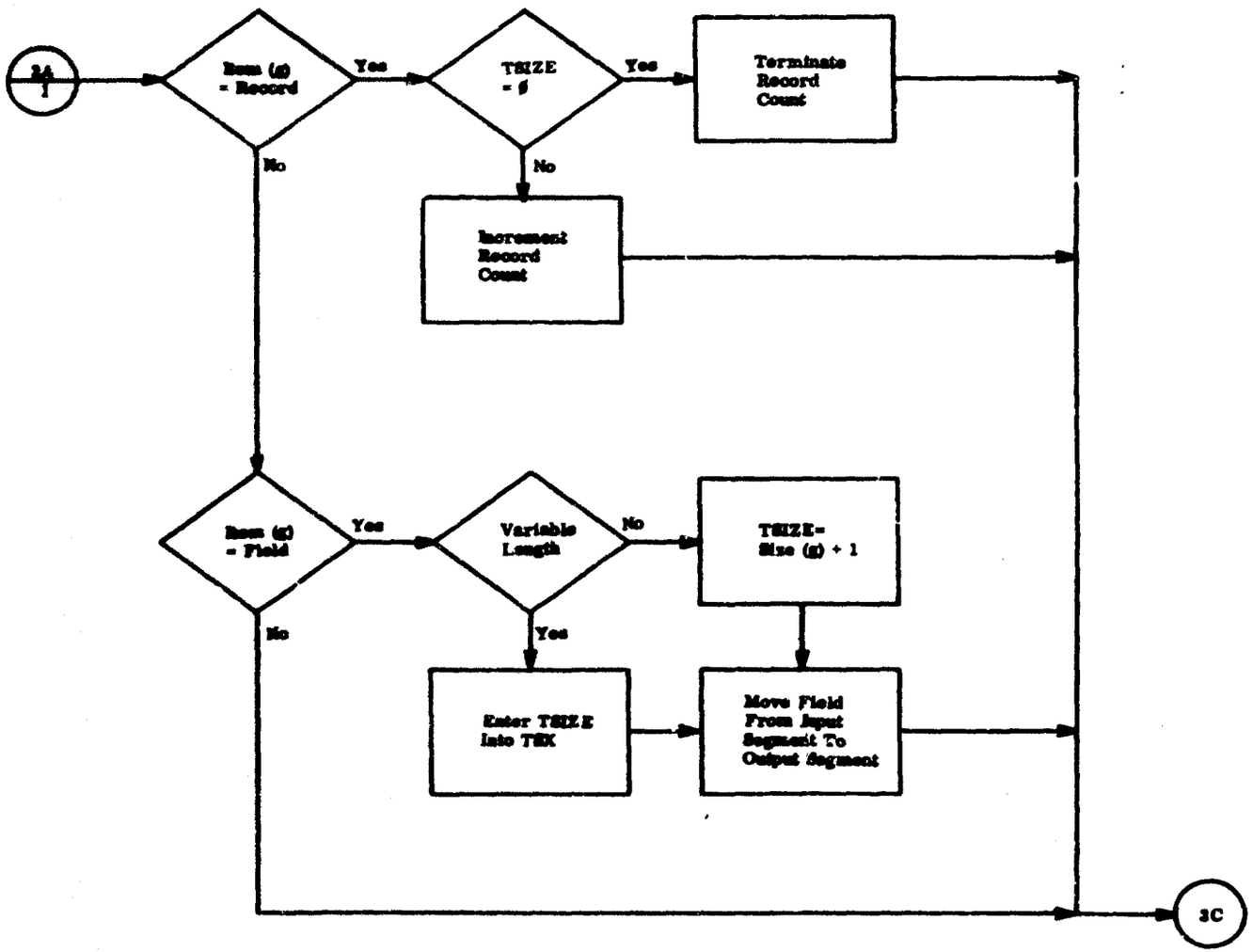
The processing for each input item begins with Step Item. This is followed by a test of the optional code. If an item is optional and missing, the segment index entry is set to zero, and the entire item is skipped. If the item is present, the routine branches to different processing paths based on item type. The processing of

fields is simpler than in a Write operation, because no special attention is required for null bits or null values. These elements are moved from input to output in the same manner as data items.

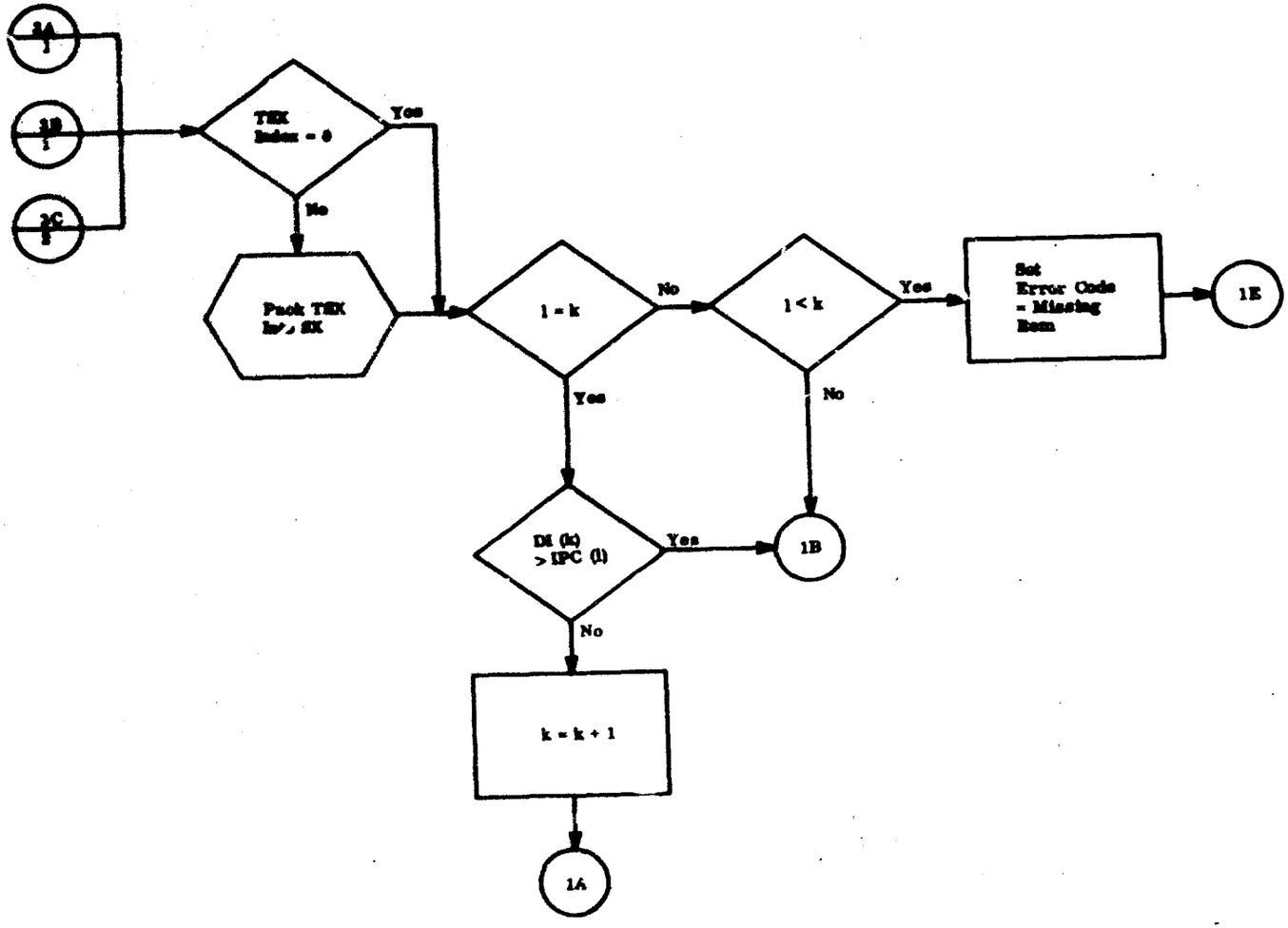
The actual storing of full segments is initiated during the Move routine and the Pack SX routine.



Seek With Copy
Sheet 1 of 3



Seek With Copy
Sheet 2 of 3



Seek With Copy
Sheet 3 of 3

SECTION IV. JOB SUPERVISOR

The DM-1 Job Supervisor is that portion of the system which directly responds to job requests. The Supervisor is responsible for creating and maintaining the records required to control the orderly execution of a job. In addition the Supervisor is responsible for transferring control to the various programs of a job in the appropriate sequence. The Supervisor programs can be divided into two logical groups:

(1) The Request Processor, which responds to the job request and determines which programs must be run.

(2) The Job Manager, which supervises the flow of program control throughout the execution of the job.

4.1 DYNAMIC TASK LIST

Each job in the DM-1 system is made up of one or more tasks. A task is composed of a program with its input items and output items. When a given job is requested to be run, the system prepares a list of the tasks which constitute that job. This list, the Dynamic Task List, defines the sequence of programs to be executed and the data upon which they are to operate. All of the functions of the Job Supervisor relate in some way to this list.

The structure of the task list is shown in Figure 4-1. The list contains one record for each task in a job. The record begins with three fields containing the type, identification, and length of the task program. The DM-1 system uses this information to control the loading of the program. The rest of the task list record consists of two binding lists, the Input List and the Output List. Each task input or output data item is represented by a record on one of these lists. The record contains three fields which relate the formal name used by the programmer to the system identifier (IPC) of the data to be processed by this job.

Three kinds of tasks can appear in the Dynamic Task List:

(1) Basic Tasks,

(2) Library Tasks, and

(3) Implied Tasks.

The basic tasks are the tasks which are part of the Job Supervisor itself. The library tasks are the tasks which constitute the job as it was described and entered into the Job Library. These are the principal processing tasks of the job. The implied tasks are tasks, such as Restructure, which preprocess input items to make them acceptable to the job. They are inserted into the Task List by the system if the job request indicates that preprocessing of the job input is required.

4.2 REQUEST PROCESSOR

The Request Processor is that portion of the Supervisor which builds most of the Dynamic Task List. It develops the task records for both the library tasks and the implied tasks. The method of handling the library tasks reflects a design decision to build as much of the Task List as possible at the time the job is described. This approach reserves until job-request time only that processing which is required to preserve flexibility in assigning job inputs and job outputs.

The principal inputs to the Request Processor (refer to Figure 4-2) are the request itself, the Static Task List and the Job Item List. The primary output is the updated Dynamic Task List. However, some requests also cause the Request Processor to write scratch items which are used during the execution of the job.

DYNAMIC TASK LIST, F

TYPE, B, 3

TASK ID, I, 12

NUMBER FLOATS, I, 3

INPUT LIST, F

FORMAL NAME, A, V

TYPE, B, 3

IPC, H, V

OUTPUT LIST, F

FORMAL NAME, A, V

TYPE, B, 3

IPC, H, V

Figure 4-1. Dynamic Task List



Figure 4-2. Overview of Request Processor

The two Input Lists are developed when the job is entered into the Job Library. The Static Task List (Figure 4-3) parallels closely the structure of the Dynamic Task List.

```
STATIC TASK LIST, F
    TYPE, B, 3
    TASK ID, I, 12
    NUMBER FLOATS, I, 3
    INPUT LIST, F
        FORMAL NAME, A, V
        CLASS, I, 3
        I/O R-NO, I, 15
    OUTPUT LIST, F
        FORMAL NAME, A, V
        CLASS, I, 3
        I/O R-NO, I, 15
```

Figure 4-3. Static Task List

The Request Processor converts records of the Static Task List into records of the Dynamic Task List by inserting an IPC into each record of the binding lists in place of the I/O R-No. To facilitate this operation, another list, the Job Item List, is developed by the DM-1 system when a job is entered into the library. The Job Item List contains

one record for every unique item required by a job. Within the Static Task List, each binding list record contains a reference to the Job Item List; this reference is the I/O R-number. If an item is used by more than one task, it appears only once in the Job Item List. Then, a reference to this record appears in the binding list of every task which uses this item. This arrangement speeds the conversion of the Static Task List records to Dynamic Task List records. Section V contains additional information on the job description procedure and on the creation of the Static Task List and the Job Item List.

This method of operation of the Request Processor points up a key function, namely the selection of the proper IPC for each record in the Job Item List. The Job Item List helps in the distribution of the IPC's to the binding lists. However, this is not possible until the Request Processor obtains the correct IPC for each item named in the Job Item List. This operation, and all of the request processing, is accomplished by three tasks:

(1) RQ Scan. This task edits and tests the external job request and stores the substance of the request in a data pool file (RQ List).

(2) Specify Item. This task produces an RQ-IPC List which contains an IPC or a null indicator for every job input and job output. In addition, Specify Item produces a record in the Implied Task List for each DM-1 utility task required by the job request.

(3) Update Dynamic Task List. This task inserts IPC's into the binding lists, and it inserts implied tasks and library tasks into the Dynamic Task List.

These tasks, which make up the Request Processor, are explained in greater detail at the end of this section.

4.3 JOB MANAGER

The Job Manager supervises the flow of program control throughout the execution of the job. The Job Manager operates at the beginning of each job, at the end of each job, and during each task-to-task transition. The Job Manager is principally composed of two tasks (RQ Bootstrap and RQ Terminate) and a core-resident routine (Task Terminate).

Figure 4-4 shows the flow of control in a DM-1 job. The user at a console initiates the job request. This causes the Executive program to load and transfer control to the first DM-1 program, the RQ Bootstrap. The RQ Bootstrap creates a new Dynamic Task List and inserts task records for the remaining tasks of the Job Supervisor. This use of the Task List, even for system programs, simplifies the flow of control in the DM-1 system. When this operation is completed, the RQ Bootstrap transfers control to the core-resident Task Terminate routine. Task Terminate uses the Executive program loader to load the next task program: then, control is transferred to this new program.

The three tasks following the RQ Bootstrap constitute the Request Processor (See Figure 4-4). Each task ends by transferring control to Task Terminate, which loads and executes the next program. By the time the third task of the Request Processor (Update DTL) is completed, the implied tasks and library tasks have been inserted into the Task List. Therefore, at the end of Update DTL, Task Terminate calls in the program for the first implied task. The sequence of a task program followed by Task Terminate is repeated until control is transferred to the last task on the Dynamic Task List, RQ Terminate. RQ Terminate deletes all information which was required only during the running of the job. Then, control is transferred to the executive program.

A more detailed description of the programs of the Job Manager is presented at the end of this section.

4.4 JOB EXTENSION

Up to this point, the Job Supervisor has been described in reference to the processing of job requests received from a console. The DM-1 system can also process requests generated by a running task program. While this facility adds substantial power to the system, it requires modifications to the Supervisor which processes console jobs rather than a separate Job Supervisor. The processing of Job Extensions requires the addition of a core-resident routine to the Job Manager and the addition of a task to the Request Processor.

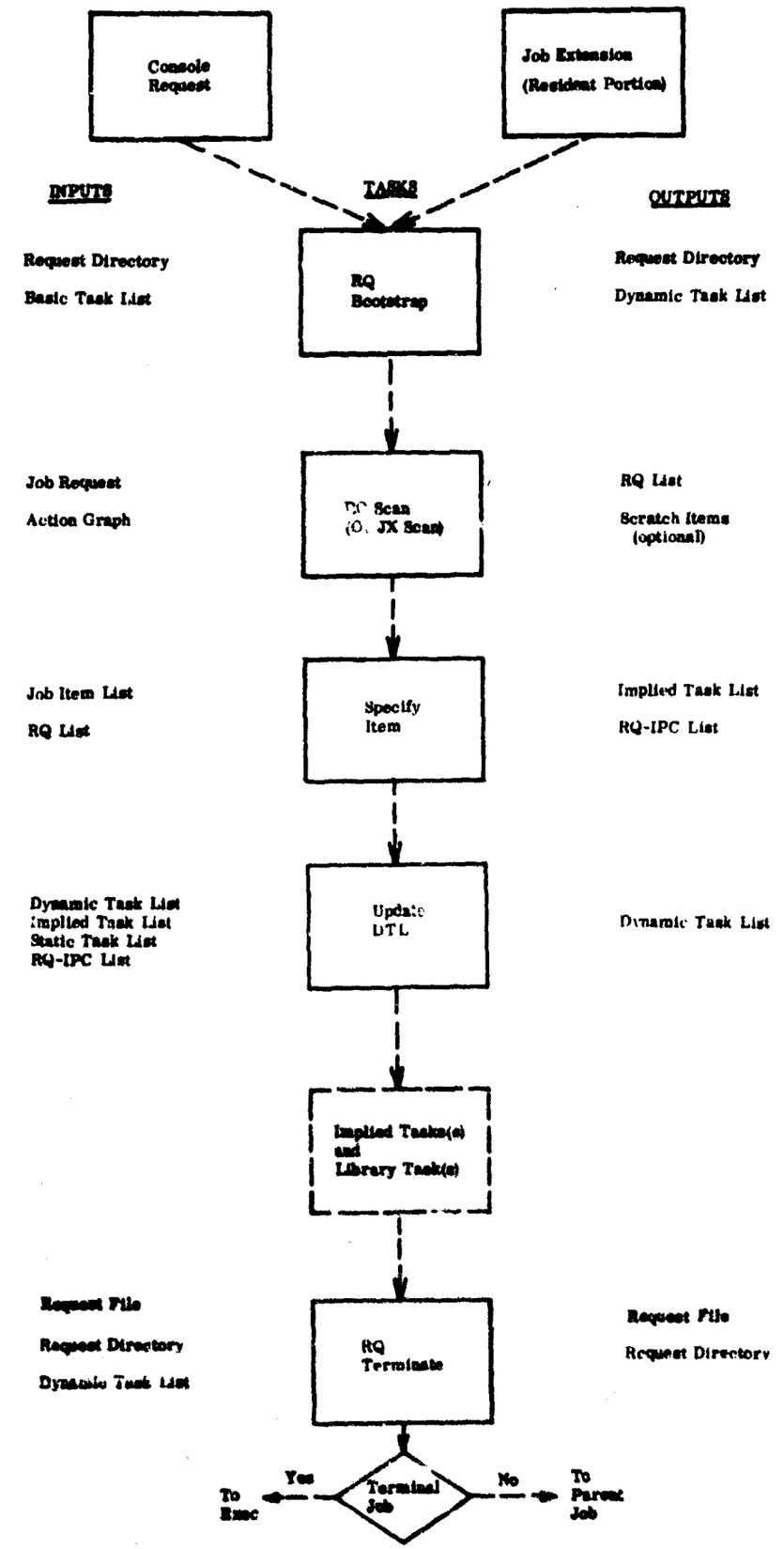


Figure 4-4. Flow of Task in a Job

The Job Extension causes a new Dynamic Task List to be generated. This list is essentially the same as the one described for console requests. It contains tasks for processing the request, implied tasks library tasks, and RQ Terminate. The entire task of the Job Extension is executed (before control) is returned to the parent job.

If the originating task executes a Job Extension as a subroutine, control is returned to this task program at the completion of the Job Extension. However, the task program may indicate in the Job Extension request that control should not be returned to this task. This means that the parent task, and perhaps the entire parent job, is logically complete before the Job Extension begins.

The DM-1 system handles these variations in a straightforward manner. At the end of the Job Extension, control is always transferred to the parent job. If the request so indicates, control is returned to the task which originated the request. Otherwise, control is returned to the beginning of the task which follows the originating task. If the parent job was logically complete, this following task will be the RQ Terminate for the parent job.

The originating task initiates a Job Extension by transferring control to a resident portion of the Job Manager, called the JX Processor. This routine writes all the information which is necessary for continuing the originating job. Then, it writes the Job Extension request and calls in the RQ Bootstrap. The remainder of the job flow is the same as for a console request, with one significant exception. A special task, JX SCAN, is used in place of RQ SCAN to do the initial processing of the request message. Then, standard processing is resumed until the end of RQ Terminate. Here, control is not transferred to the Executive program, but to the parent job.

4.5 TEMPORARY ITEMS REQUIRED BY JOB SUPERVISOR

Several data items are used by the Job Supervisor during the execution of a job. These items fall into four categories:

(1) Reserved Core Items,

(2) Request File,

(3) Request Directory, and

(4) Scratch Items.

4.5.1 Reserved Core Items

A few items need to be maintained by the Job Manager in a portion of core which is reserved throughout all of the tasks of a job. The principal items kept here are:

(1) The current job record number in the Request File,

(2) The current task record number in the Dynamic Task List, and,

(3) For Job Extensions, the record number of the parent job in the Request File.

4.5.2 Request File

This file contains one complete record for every job which is active, i.e., partially processed. Within the record is the external information that the Job Supervisor must maintain during the execution of a job (see Figure 4-5). The use of these items is explained in conjunction with the flow charts included in this section; however, their significance can be summarized as follows:

(1) Parent Task.

(2) Dump Data. These two items are needed to return to the originating task at the end of a Job Extension.

(3) RQ List. This list contains the substance of a job request after the initial editing by RQ Scan or JX Scan.

(4) Implied Task List. This list is temporary storage for the task records of DM-1 utility tasks required by the job.

(5) RQ-IPC List. This list is temporary storage for the IPC's before they are inserted into the binding lists.

(6) Dynamic Task List. This defines the sequence of programs to be executed and the data items upon which they are to operate.

(7) JX Request. This is temporary storage for Job Extension information before the RQ Bootstrap begins the Job Extension.

```
RQ FILE, F
    PARENT TASK, H, V
    *DUMP DATA, S, 2
        DUMP ID, O, 4
        RETURN ADDRESS, O, 4
    *RQ LIST, S, 2
        JOB NAME, A, V
        RQ ITEMS, F
            CODE, O, 1
            NAME, A, V
    IMPLIED TASK LIST, F
        TYPE, B, 3
        TASK ID, I, 12
        NUMBER FLOATS, I, 3
        INPUT LIST, F
            FORMAT NAME, A, V
            TYPE, B, 3
            IPC, H, V
        OUTPUT LIST, F
            FORMAL NAME, A, V
            TYPE, B, 3
            IPC, H, V
    RQ-IPC LIST
        JOB NAME, A, V
        JOB R-NUMBER, H, V
        IPC FILE, F
            IPC, H, V
    DYNAMIC TASK LIST, F
        TYPE, B, 3
        TASK ID, I, 12
        NUMBER FLOATS, I, 3
        INPUT LIST, F
            FORMAL NAME, A, V
            TYPE, B, 3
            IPC, H, V
        OUTPUT LIST, F
            FORMAL NAME, A, V
            TYPE, B, 3
            IPC, H, V
    *JX REQUEST, S, 3
        DUMP DATA, S, 2
            DUMP ID, O, 4
            RETURN ADDRESS, O, 4
        PARENT TASK, H, V
        REQUEST, A, V
```

Figure 4-5. Request File

4.5.3 Request Directory

This is a simple file which contains one record for every record in the Request File. The purpose of the directory is to prevent the record numbers of the Request File from changing each time a job terminates. RQ Terminate, instead of deleting a record of the Request File, simply deletes all of the data from the record. Then, an indicator which declares this record to be available is stored in the corresponding record of the Request Directory. The indicator is changed when the Request Record is assigned to a new job by the RQ Bootstrap.

4.5.4 Scratch Items

The tasks of a job may produce several intermediate items which are not needed after the job is completed. The number and structure of such items can vary considerably from one job to another. Since all such items are deleted at the end of a job, the DM-1 System groups all of the scratch items associated with a given job. One primary node in the scratch area is assigned to each job, and each scratch item required by the job is subsumed under that node. For simplicity, these primary nodes are assigned in parallel with the records of the Request File. The first node is reserved for the first Request Record, the second node is associated with the second record, etc. This arrangement facilitates not only the deletion at job termination time, but also the identification of scratch items during the job.

4.6 DETAILED DESCRIPTION OF JOB SUPERVISOR PROGRAMS

Each of the following paragraphs describes a Job Supervisor program and is followed by the appropriate flow chart or charts.

4.6.1 RQ Bootstrap

4.6.1.1 Functional Description. This task creates a task list containing the standard tasks required for every job. This includes the tasks for processing the request message and the task for terminating the request.

4.6.1.2 Inputs. The inputs are:

(1) Basic Task List, and

(2) Job Extension Indicator.

The Job Extension Indicator is set by the Job Manager, after it has called on the Executive program to load the RQ Bootstrap. It is not set if the bootstrap has been loaded in response to a console request.

4.6.1.3 Results. The results are:

(1) Dynamic Task List,

(2) Request R-number (this is saved in an area of core which is not available to user tasks), and

(3) Task R-number (this field is also in reserved core; it is set to zero by the RQ Bootstrap).

4.6.1.4 Directories Used. Request Directory.

4.6.1.5 Services Used.

(1) Open for Input.

(2) Close for Input.

(3) Open for Output.

(4) Close for Output.

(5) Open for Update.

(6) Close for Update.

(7) Read.

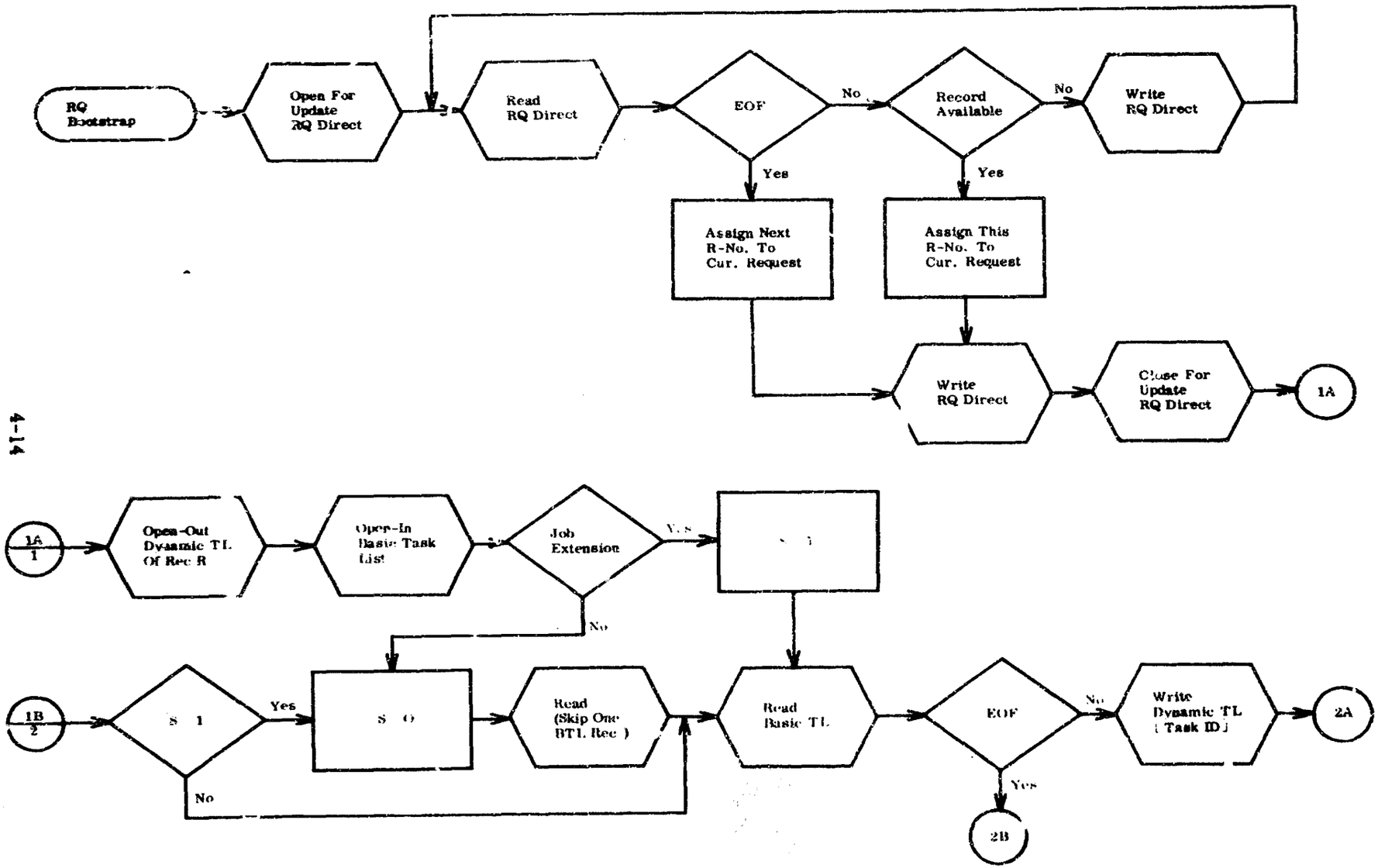
(8) Write.

4.6.1.6 Jobs Used. None.

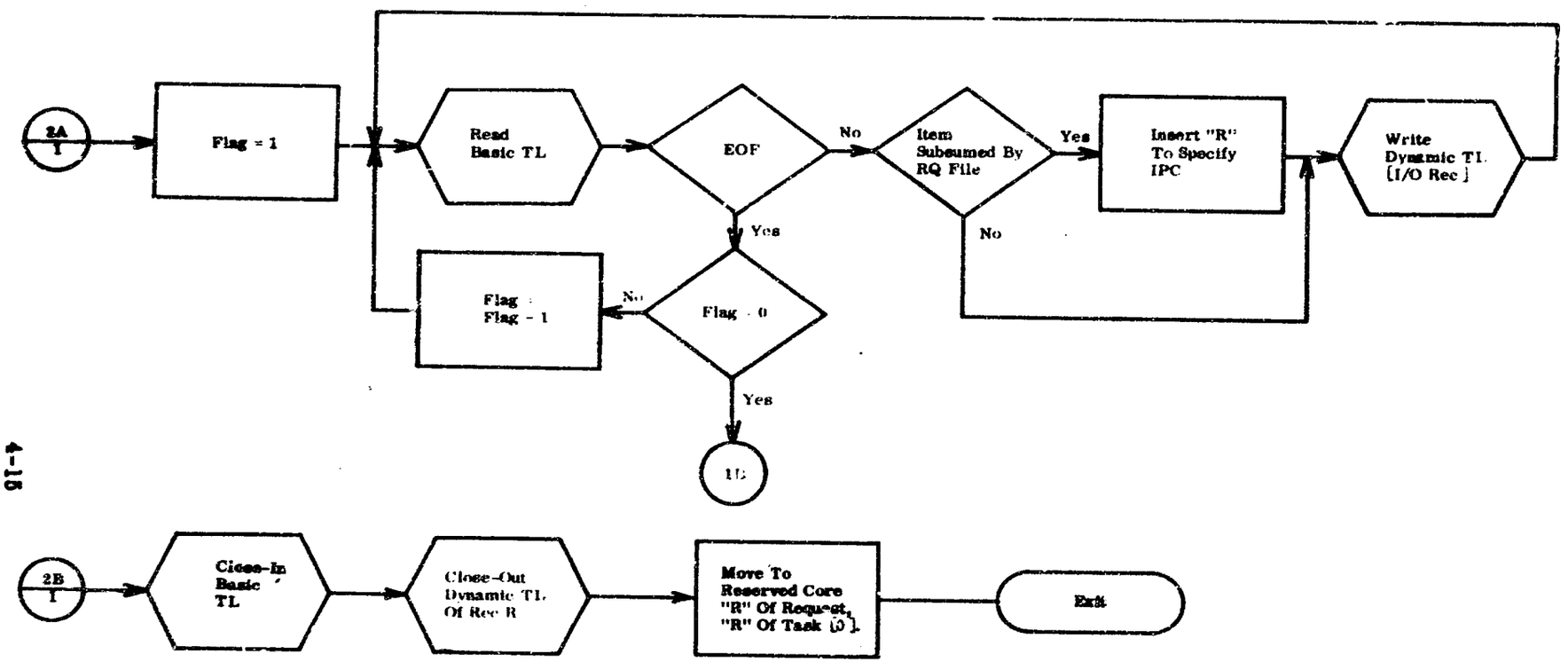
4.6.1.7 Method of Operation. The RQ Bootstrap searches the Request Directory for the lowest record number in the RQ File which is not in use. This number is assigned to the new request, and the directory is updated to reflect this assignment. This RQ record is either an old record from which all the data has been deleted or a new record at the end of the RQ file. The Bootstrap opens the Dynamic Task List which is subsumed in the assigned record of the RQ file. Then, the standard tasks, which are defined in the Basic Task List, are copied into the Dynamic Task List.

If the job is a Job Extension, a special scan task (JX Scan) is placed in the Dynamic Task List instead of the standard scan task (RQ Scan). JOX Scan is the first task in the Basic Task List, and RQ Scan is second. If the request is not a Job Extension, the JX Scan is skipped and RQ Scan is written into the Task List.

The data binding information in the Dynamic Task List is divided into two lists, an Input List and an Output List. A binary flag is used to assure that both lists are copied for each task. While the binding information is being copied, a check is made to identify the items which are subsumed in the RQ File. The IPC's of these items need the R-number of the record in the RQ File which was assigned to this job. This R-number is inserted before the IPC's are written. When the last task of the Basic Task List has been copied, the Bootstrap moves this R-number to reserved core and sets the Task R-number to zero. Then, control is transferred to Task Terminate.



RQ Bootstrap
Sheet 1 of 2



RQ Bootstrap
Sheet 2 of 2

4.6.2 Task Terminate

4.6.2.1 Functional Description. This task controls the loading of the next task of the current Task List and the transfer of control to this task.

4.6.2.2 Inputs. The inputs are:

(1) R-number of Request Record,

(2) R-number of Task, and

(3) Dynamic Task List.

4.6.2.3 Results. The results are:

(1) The new task program is in core, and

(2) The R-number of the task is updated.

4.6.2.4 Directories Used. None.

4.6.2.5 Services Used. Executive Program Loader.

4.6.2.6 Jobs Used. None.

4.6.2.7 Method of Operation. Task Terminate uses the Request R-number and the Task R-number from reserved core to retrieve the Dynamic Task List record for the next task. The Task ID from this record is inserted into a Load Request, and the request is issued to the Executive program. When the loading is completed, Task Terminate transfers control to the first instruction of the new task.



Task Terminate
Sheet 1 of 1

4.6.3 RQ Scan

4.6.3.1 Functional Description. The RQ Scan task edits and tests an external job request and stores the substance of the request in a data pool file.

4.6.3.2 Inputs. The input is: the job request.

4.6.3.3 Results. The results are:

(1) The RQ List, and

(2) Scratch Items, if required.

4.6.3.4 Directories Used. Action-Graph File.

4.6.3.5 Services Used.

(1) Input Scan Routine (INSCAN).

(2) Insert Data.

(3) Executive Read Console.

(4) Open for Output.

(5) Close for Output.

(6) Write.

(7) Assign Item.

4.6.3.6 Jobs Used. None.

4.6.3.7 Method of Operation. The RQ Scan task begins by inserting a null value in the parent field of the RQ record. This identifies the request as a console request rather than a Job Extension. Then the task issues a call to the Executive program, which reads the entire request into core. The job name is extracted from the request. The task uses this name to select the proper action-graph for editing the request. Then, the job name is written into the RQ List. The remainder of the flow chart represents the logic that is implemented by means of the selected action-graph and the INSCAN subroutine. INSCAN is described in Section VIII, but the operations for which RQ Scan uses the subroutine are described in the following paragraphs.

Following the job name in the request are the job inputs and outputs. These can appear in the following forms.

(1) A literal.

(2) A Restructure specification.

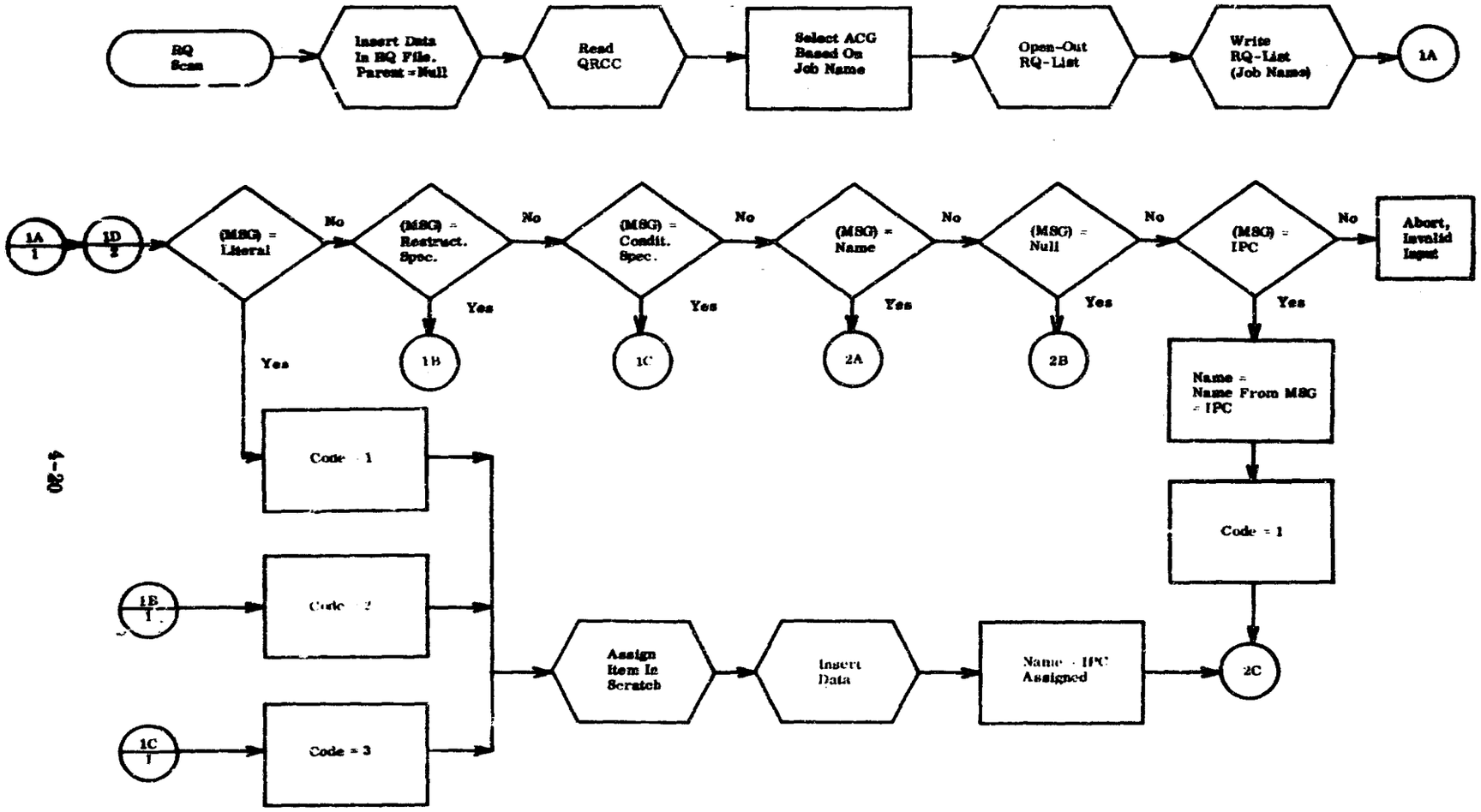
(3) A Conditional Select specification.

(4) The name of an item which is already defined in the Term Ending Table.

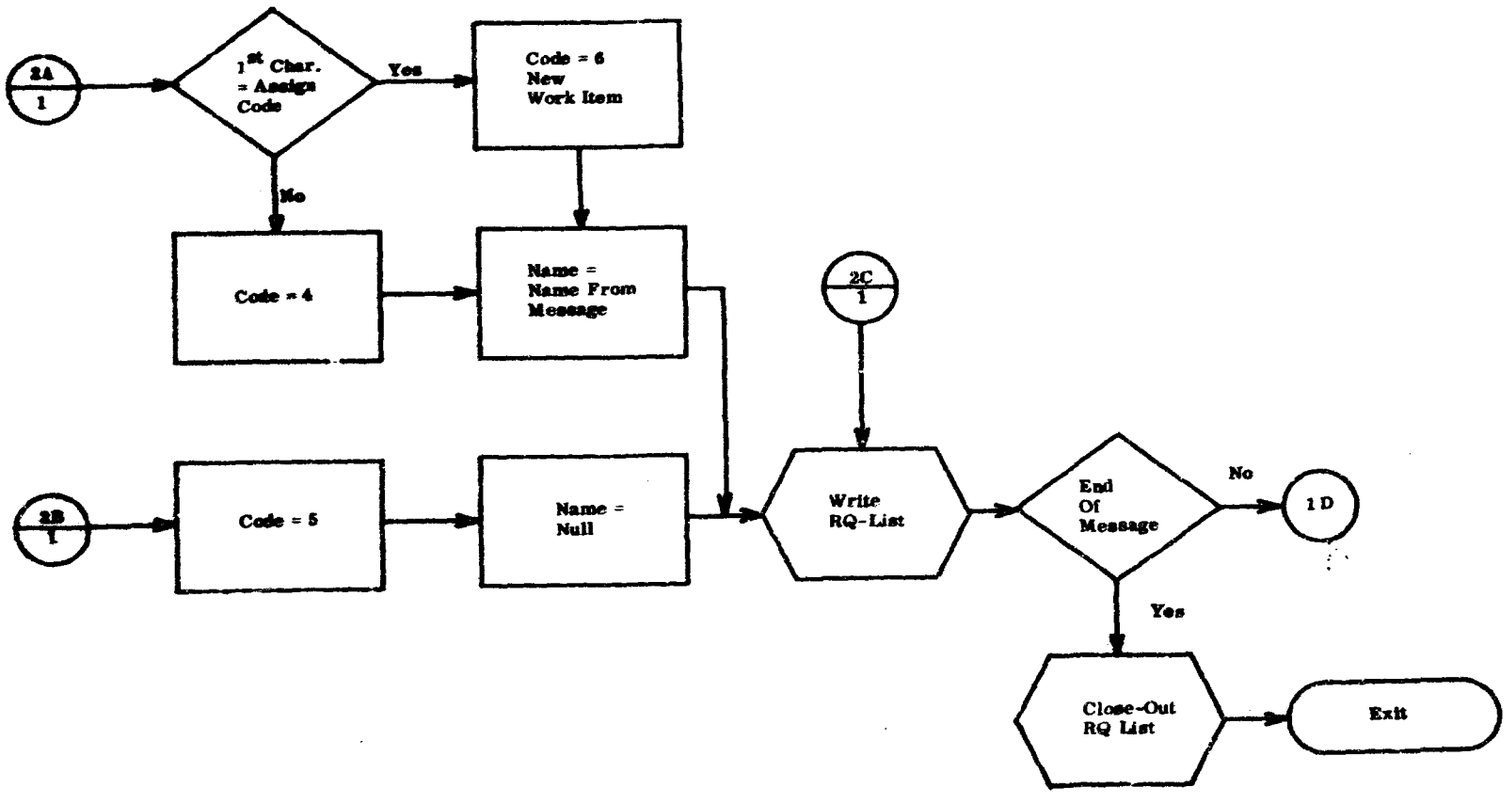
(5) A null indicator, defining that the item is not to be used in this request.

(6) The name of a new item to be defined in accordance with the program description.

During the editing process, one item is written into the RQ List for each job input and job output. The item consists of a code, which specifies one of the above six categories, and a Name Field. The latter field may contain several different values. If the request item is a literal or a specification, it is written into the scratch area, and its IPC is stored in the Name field. If the request item is a name or an IPC, this is copied into the name field. If it is a null indicator, a null field is written. When the end-of-the-request message is reached, the RQ List is closed, and control is transferred to Task Terminate.



RQ-Scan
Sheet 1 of 2



RQ-Scan
Sheet 2 of 2

4.6.4 Specify Item

4.6.4.1 Functional Description. This task produces an RQ-IPC List which contains an IPC or a null indicator for every job input and job output. In addition, Specify Item produces a record in the Implied Task List for each Restructure or Conditional Select required by this job request.

4.6.4.2 Inputs. The inputs are:

(1) The RQ List, and

(2) The Job Item List.

4.6.4.3 Results. The results are:

(1) The RQ-IPC List, and

(2) The Implied Task List.

4.6.4.4 Directories Used.

(1) Job Name List.

(2) Term Encoding Table.

4.6.4.5 Services Used

(1) Open for Input.

(2) Close for Input.

(3) Open for Output.

(4) Close for Output.

(5) Read.

(6) Write.

(7) Retrieve.

(8) Assign Item.

(9) Assign IPC.

(10) Translate Term Name.

4.6.4.6 Jobs Used. None.

4.6.4.7 Method of Operation. Specify Item begins by reading the name of the requested job from the RQ List. This name is used to retrieve the associated record from the Job Name List, which is an ordered file. The Job Name Record contains a record number which points to the Job Item List of the requested job. The Specify Item Task is controlled primarily by this Job Item List. This list contains an entry for each unique data item required by the job. Specify Item must provide one IPC or null indicator for each Job Item entry. The IPC is obtained by different methods, depending on the class to which the Job Item belongs. The item may be:

(1) A Job Input-Output.

(2) An Intermediate Input-Output (scratch).

(3) An Internal Item.

(4) A Literal.

(5) A Null Indicator.

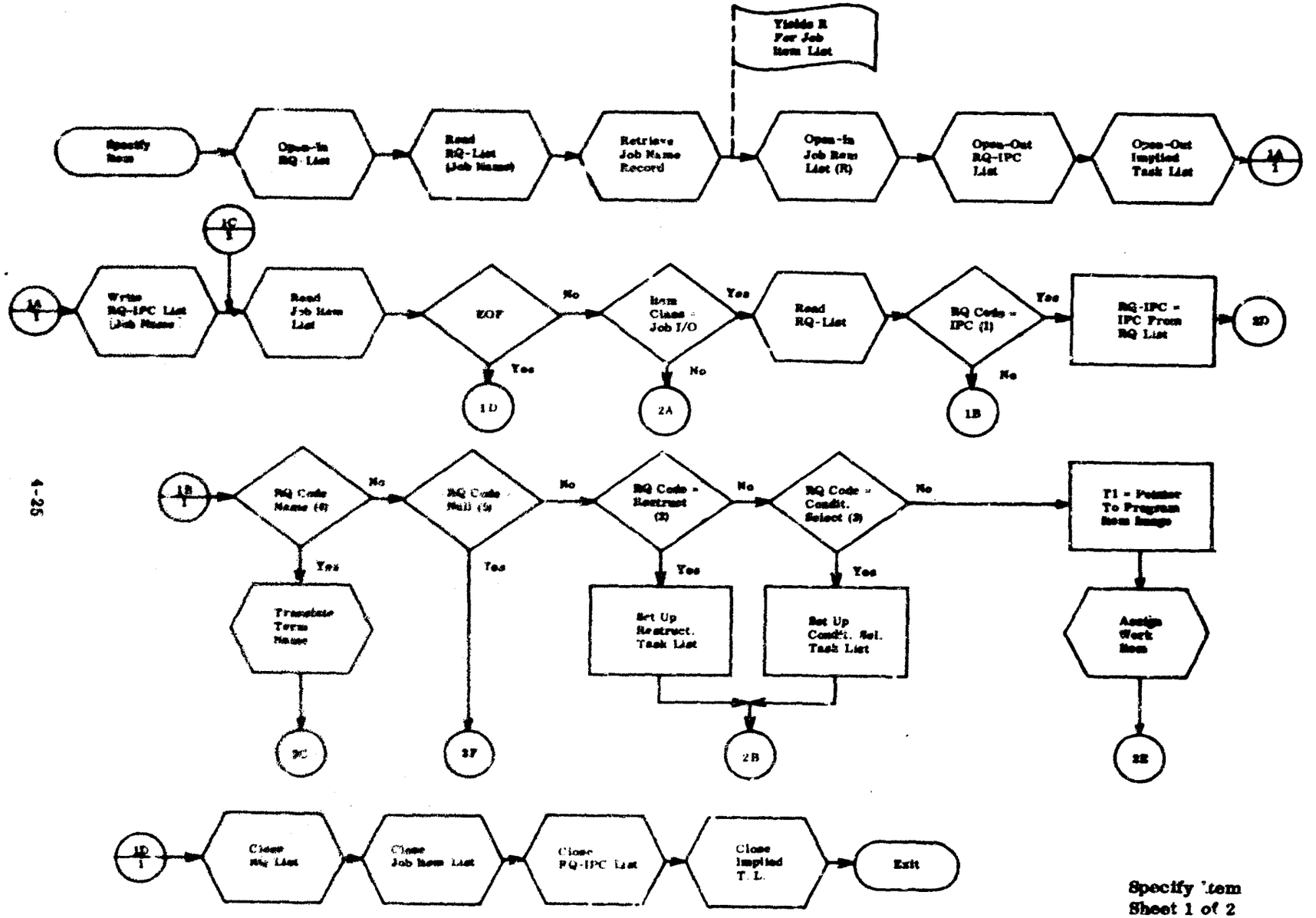
These classes are described in Paragraph 5.2

The Job Input-Output items require the most complex processing. These are the items which must be specified in the job request, and they can be specified in several ways. For each Job Input-Output item, Specify Item reads the RQ List and writes an IPC into the RQ-IPC List. Depending on the code in the RQ List, the IPC is obtained by one of the following methods:

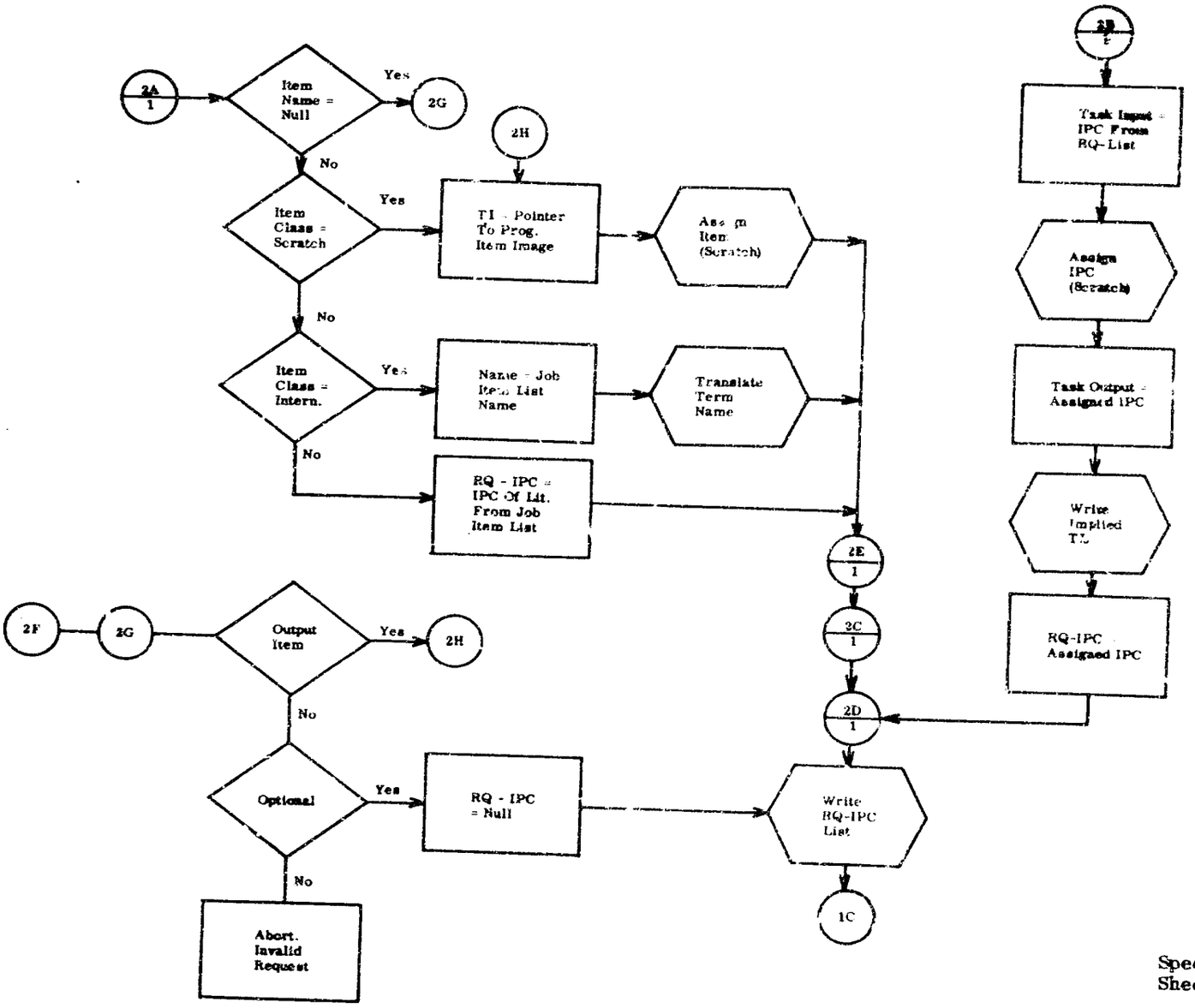
Code	Method
1	The IPC is taken directly from the RQ List entry.
2, 3	The IPC is obtained by assigning a Scratch Item for an implied task. This procedure is described later.
4	The IPC is obtained by translating the term name contained in the RQ List.
5	No IPC is needed. A null field is written into the RQ-IPC List.
6	The IPC is obtained by defining a job output in the Work Area and using its IPC.

The items that require an implied task are those for which the request contains a Restructure specification or a Conditional Select specification. For each such item, Specify Item creates a task description. The task identification is a program constant. The task input is the IPC of the specification which was stored in the Scratch Area during the editing of the request. The task output is a Scratch Item assigned by Specify Item. The IPC of this item is written both as the output of the implied task and as the next IPC in the RQ-IPC List.

The processing is less complex for the Job Item List entries that do not belong to the Job Input-Output class. For the intermediate items, the item is assigned to the Scratch Area, and its IPC is written into the RQ-IPC List. For null inputs, a null field is written. For null outputs, a Scratch Area is assigned in case the user program is not prepared to bypass the writing of the item. Finally, for literals, the IPC of the literal is moved from the Job Item List to the RQ-IPC List. When the end of the Job Item List is reached, control is transferred to Task Terminate.



Specify Item
Sheet 1 of 2



Specify Item
Sheet 2 of 2

4.6.5 Update DTL

4.6.5.1 Functional Description. This task completes the Task List for the requested job.

4.6.5.2 Inputs. The inputs are:

(1) The Dynamic Task List,

(2) The Implied Task List,

(3) The Static Task List, and

(4) The RQ-IPC List.

4.6.5.3 Results. The result is: the Dynamic Task List is updated.

4.6.5.4 Directories Used. None.

4.6.5.5 Services Used

(1) Open for Update.

(2) Close for Update.

(3) Open for Input.

(4) Close for Input.

(5) Seek with Copy.

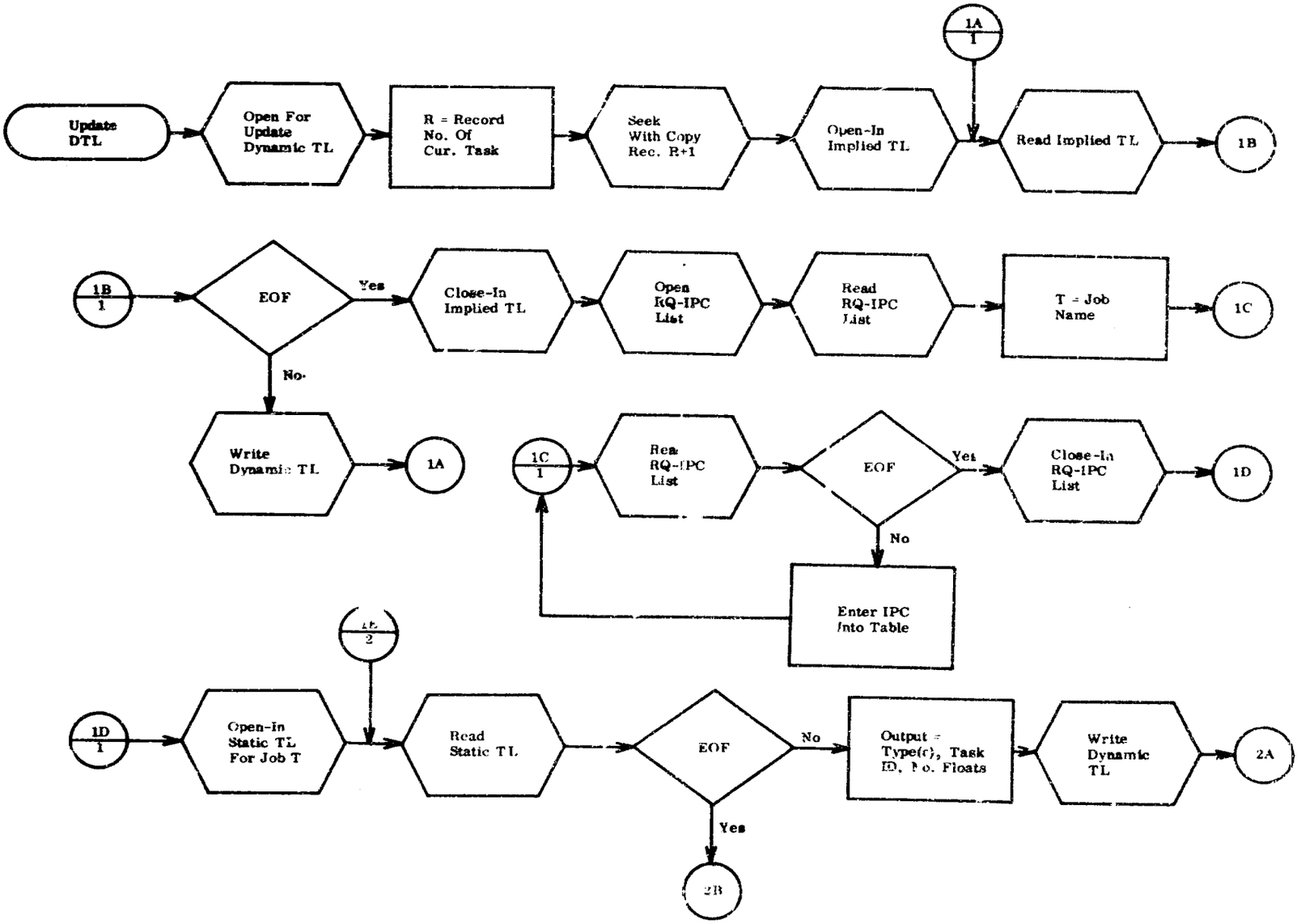
(6) Read.

(7) Write.

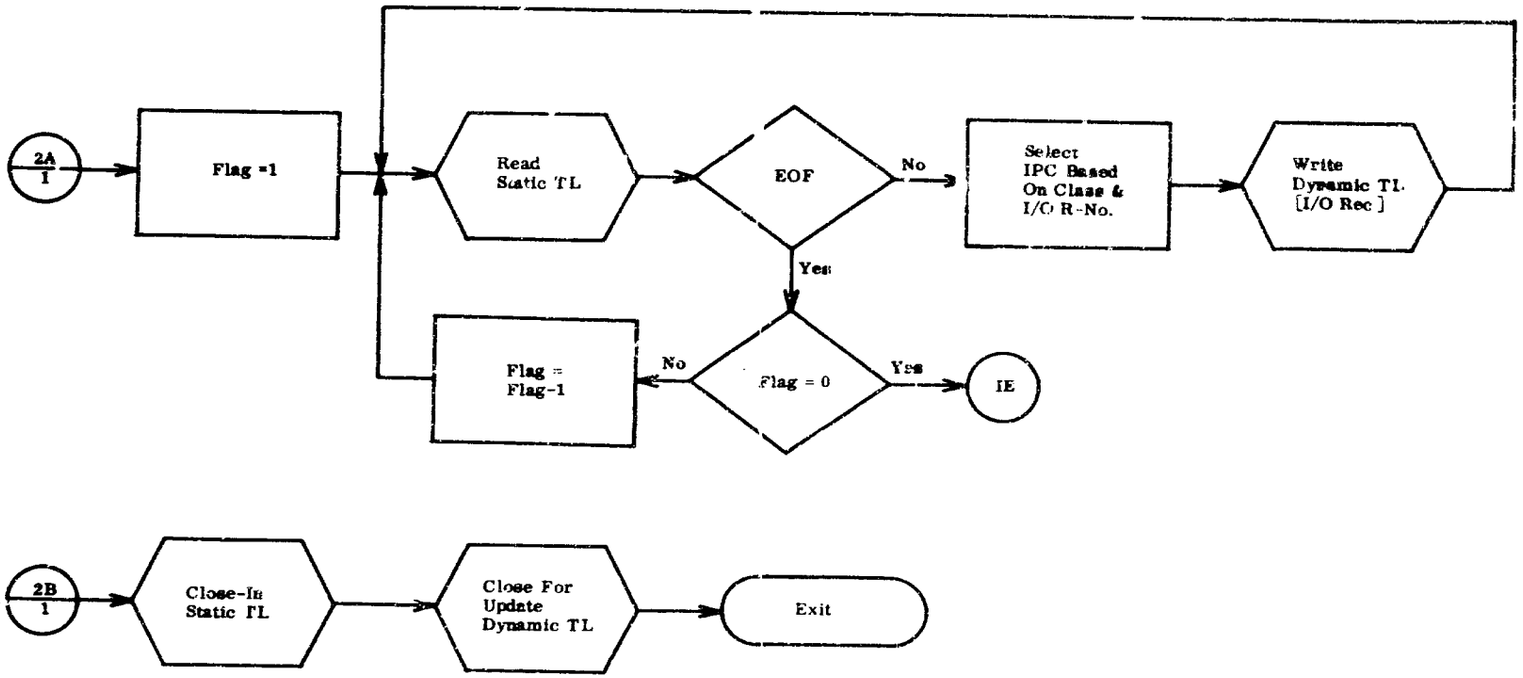
4.6.5.6 Jobs Used. None.

4.6.5.7 Method of Operation. Update DTL completes the Task List for the requested job. First, the implied tasks are written into the list immediately following the record for Update DTL itself. Next, the entire RQ-IPC List is read. The job name is saved, and a table of IPC's is constructed.

Then, the Static Task List for the requested job is opened. As each Static Task List record is read, the header fields (Task ID, etc.) are copied into the Dynamic Task List. A binary flag is used to control the reading of the two binding lists, the Inputs List and the Outputs List. Each record of these lists contains a formal name, a type, and a pointer. The name and type are copied into the Dynamic Task List. The pointer is used to select an IPC from the IPC table, and this IPC is written into the Dynamic Task List. This simple operation for inserting IPC's into the binding lists is possible, because much of the binding work is done at the time a job description is defined. When the end of file is reached in the Static Task List, control is transferred to Task Terminate. Task Terminate loads the first task of the requested job: this may be an implied task or a static task. Control is transferred to the first instruction of this task.



Update DTL
Sheet 1 of 2



Update DTL
Sheet 2 of 2

4.6.6 RQ Terminate

4.6.6.1 Functional Description. This task deletes all information which is required only during the running of a job. Then, control is transferred either to a parent job or to the Executive program.

4.6.6.2 Inputs. The inputs are:

(1) The Request Directory,

(2) The RQ File, and

(3) The Dynamic Task List.

4.6.6.3 Results. The results are:

(1) The Request Directory (updated), and

(2) The RQ File (updated).

4.6.6.4 Directories Used. None.

4.6.6.5 Services Used.

(1) Open for Input.

(2) Close for Input.

(3) Open for Output.

(4) Close for Output.

(5) Open for Update.

(6) Close for Update.

(7) Read.

(8) Write.

(9) Delete.

(10) Executive Program Loader.

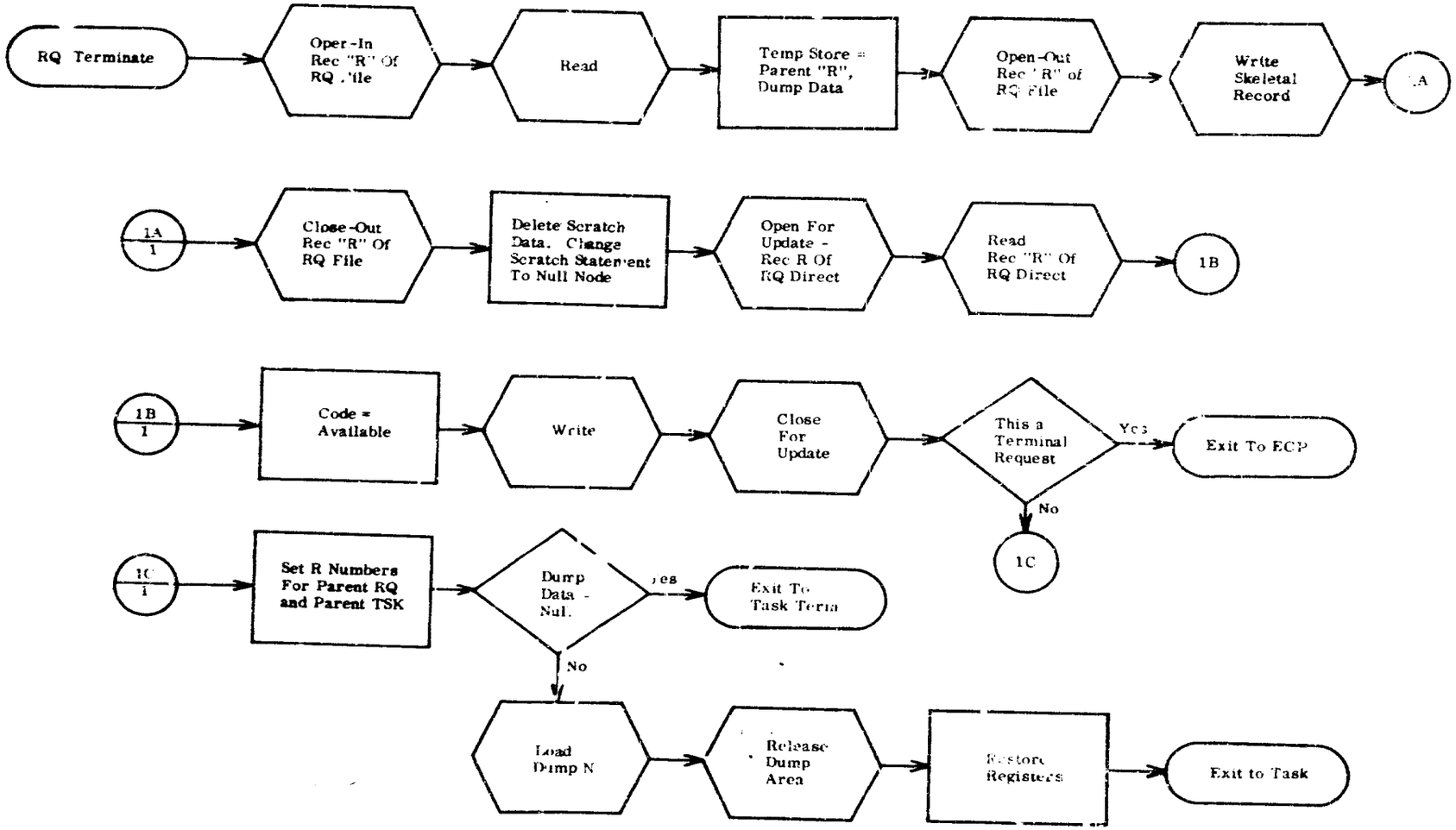
4.6.6.6 Jobs Used. None.

4.6.6.7 Method of Operation. The RQ Terminate task begins by opening the record of the Request File which was assigned to this job. The parent field and the Dump Data statement are read and saved for use at the end of this task. Then, all of the data in this RQ record is deleted, but a skeletal record is retained so that the record numbers in the rest of the file do not change. To develop this skeletal record, the parent field is set to null, the subsumed files are written with zero records, and the statements (which are all optional) are declared to be missing. The RQ Directory is updated to reflect the fact that this record is now available.

RQ Terminate is also concerned with the scratch items which were used by the completed job. All scratch items associated with a given job request are subsumed under a single scratch statement which is reserved for use with this record of the Request File. Scratch statement number 1 subsumes the scratch items needed by request number 1, etc. See Paragraph 4.5.4. RQ Terminate deletes all the data subsumed by the scratch statement associated with the job request which is being terminated. Then all the definitions for these scratch items are deleted, and the scratch statement is converted to a null node.

Next, the parent field, which is in temporary storage, is tested. If the field is null, this means a console request has been satisfied. Control is transferred to the Executive program. If the parent field is present, the completed job is a Job Extension, and control must be returned to the parent job. The parent field contains the R-value of the parent task, which consists of a record number for the Request File and a record number for the Dynamic Task List. RQ Terminate moves these two R-numbers into the control fields in permanent core, which the Job Supervisor uses to keep track of the current task.

If the Dump Data, which is in temporary storage, is null, control is transferred to Task Terminate. Task Terminate loads the task following the task which generated the job extension. If the Dump Data is present, the parent task is loaded and control is returned to this task.



RQ Terminate
Sheet 1 of 1

4.6.7 JX Processor (Resident Portion)

4.6.7.1 Functional Description. This routine prepares for the processing of the Job Extension Request and for the resumption of the parent job after the Job Extension (JX) has been completed.

4.6.7.2 Inputs. The inputs are:

(1) The Job Extension Request,

(2) Return Address, or Terminate Flag, and

(3) R-Value of Current Task (in reserved core).

4.6.7.3 Results. The results are:

(1) A Job Extension Request statement in the RQ Record of the current jobs (see Paragraph 4.5.2 and Figure 4-5), and

(2) A core dump, if required.

4.6.7.4 Directories Used. None.

4.6.7.5 Services Used.

(1) Executive Program Dump.

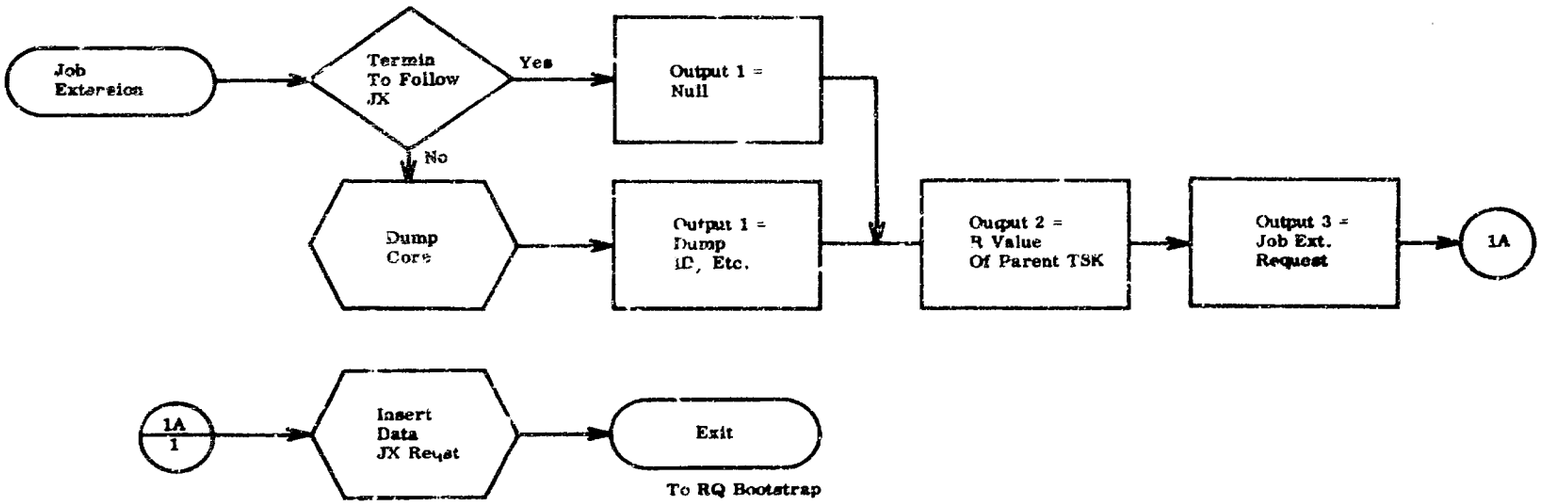
(2) Insert-Data.

4.6.7.6 Jobs Used. None.

4.6.7.7 Method of Operation. The JX Processor must prepare for the continuation of the parent job, that is, the job which initiated the Job Extension (JX). This job may be continued in the task which issued the JX (called the parent task) or at the beginning of the following task. In some cases, this following task may be RQ Terminate. For such a case, the parent job is essentially completed before the extension, but this does not cause special processing.

If the parent task is to be resumed, the JX Processor calls on the Executive program to dump the core occupied by the task. The identifier for this dump and the address to be used for continuing the task are both written into the Request Record of

the parent job. Figure 4-5 shows the JX Request Statement at the end of the Re Record. If the parent task is not to be resumed, no data is dumped, and the D statement is declared to be absent. In all cases, the R-value of the parent tas written. This contains an R-number for the Request File and an R-number for Dynamic Task List. The JX Processor writes the JX request into the Request and control is transferred to the RQ Bootstrap.



JX Processor
-Resident Portion
Sheet 1 of 1

4.6.8 JX Scan

4.6.8.1 Functional Description. The JX Scan task edits and tests a Job Extension request and stores the substance of the request in a data pool file.

4.6.8.2 Inputs. The inputs are:

(1) The R-Number of the parent job (in reserved core), and

(2) The job extension request.

4.6.8.3 Results. The results are:

(1) The RQ List, and

(2) Scratch Items, if required.

4.6.8.4 Directories Used. None.

4.6.8.5 Services Used.

(1) Open for Output.

(2) Close for Output.

(3) Retrieve Item.

(4) Write.

(5) Assign Item.

(6) Insert Data.

4.6.8.6 Jobs Used. None.

4.6.8.7 Method of Operation. The JX Scan task begins by retrieving the JX Request Statement from the Request Record of the parent job. From this statement, the JX Scan copies the parent field and the Dump Data statement into the beginning of the Request Record which has been assigned to the Job Extension. The job name is extracted from the JX Request and written into the RQ List. The remainder of the request specifies the job inputs and job outputs. These can appear in the following forms:

(1) A Literal.

(2) A Restructure specification.

(3) A Conditional Select specification.

(4) The name of an item which is already defined in the Term Encoding Table.

(5) A null indicator, defining that the item is not to be used in this request.

(6) A scratch suffix, which identifies a scratch item associated with the parent job.

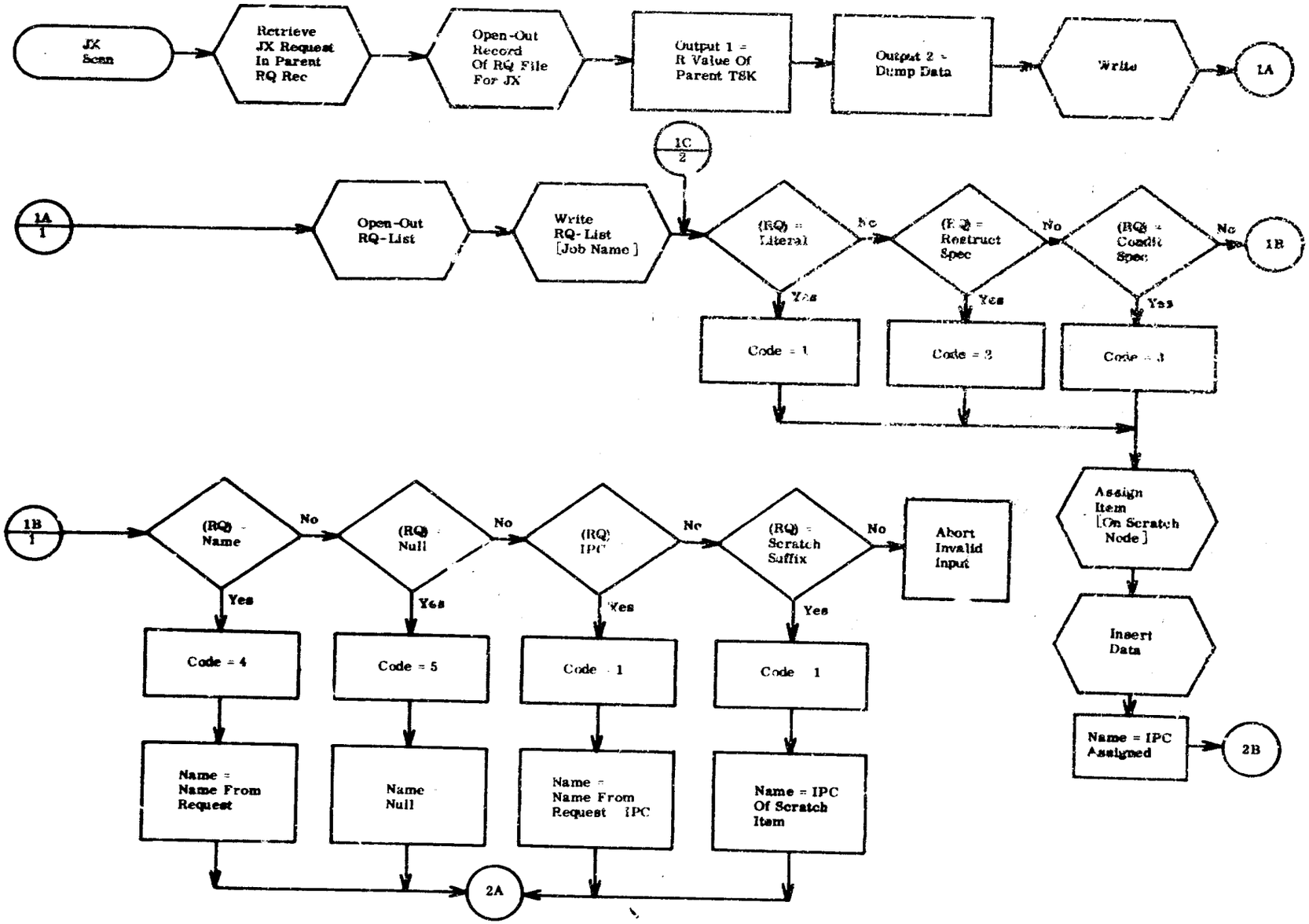
These options for specifying input-output items are very similar to the options available in a console request. An exception occurs in specifying the outputs of a Job Extension. The DM-1 system does not assign new work areas for the output of a Job Extension. However, the requestor is free to have the Job Extension output stored in one of three places:

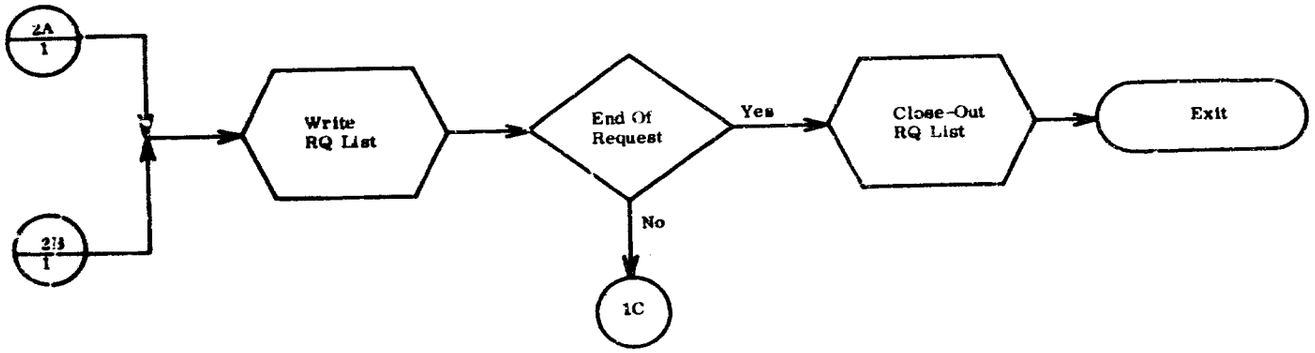
(1) A work item which was assigned to the parent job (requestor supplies names),

(2) A scratch item which was assigned to the parent job (requestor supplies scratch suffix), or

(3) A scratch item of the job extension (requestor supplies null indicator).

JX Scan writes one item into the RQ List for each job input and job output in the request. The item consists of a Name field and a code which reflects the kind of value contained in the Name field. If the request item is a literal, a restructure specification, or a Conditional Select specification, the item is written into the scratch area of the Job Extension. For these three categories, the IPC of the scratch item is written in the Name field. If the request item is a name, this name is copied into the Name field. If the request item is an IPC or a scratch suffix, an IPC is put into the Name field. For each null indicator in the request, a Name field is set to the null state. When the entire JX request has been scanned in this manner, the RQ List is closed, and control is transferred to Task Terminate. Task Terminate loads the Specify Item Task and the processing of the JX Request continues.


JX Scan
Retrieve JX Request In Parent RQ Rec
Open-Out Record Of RQ File For JX
Output 1 = R Value Of Parent TSK
Output 2 = Dump Data
Write
1A
Open-Out RQ-List
Write RQ-List [Job Name]
1C
2
(RQ) = Literal
No
Yes
Code = 1
(RQ) = Restruct Spec
Code = 2
(RQ) = Condit Spec
Code = 3
1B
1B
1
(RQ) = Name
(RQ) = Null
(RQ) = IPC
(RQ) = Scratch Suffix
Abort Invalid Input
Code = 4
Code = 5
Code = 1
Code = 1
Name = Name From Request
Name = Null
Name = Name From Request IPC
Name = IPC Of Scratch Item
2A
Assign Item [On Scratch Node]
Insert Data
Name = IPC Assigned
2B



JX Scan
Sheet 2 of 2

SECTION V. JOB LIBRARY MAINTENANCE JOBS

The DM-1 library is maintained by four system jobs which provide for the entry of programs, the description of jobs, the deletion of programs and jobs, and the display of job descriptions. These jobs are described in the following paragraphs.

5.1 PROGRAM ENTRY

A program comes under the control of the DM-1 system when it is entered into the library through the Program Entry job. Programs are compiled independently of the system and their object code is stored under the control of the operating system. They are entered into the DM-1 system through a program specification which includes the following elements:

(1) Program Name
(2) Program Inputs (formal)
(3) Program Outputs (formal)
(4) Program Executive Control Description

The formal input-output parameters are described by naming them and giving an item definition for the fixed parameters. After the Program Entry job has processed the program specification, the program becomes a job in the DM-1 library. It may be called for execution by a job-run request issued by a user at a console or by a job-extension request issued by a running program. It may also be used as a component in a job description.

The job request image for the Program Entry job is as follows:

Job Request: PROGRAM-ENTRY (program name), (program inputs), (program outputs), (program ECD).

5.1.1 Functional Description

Programs are compiled independently of the Data Management System. Nevertheless, they may be entered into the system via the Program Entry job.

To do this, a formal description of the program, which consists of the following items, must be given:

(1) Program name,

(2) Program inputs and their descriptions,

(3) Program outputs and their descriptions,

(4) Program ECD (Executive Control Description).

The program name is the name by which the program is known to the Data Management System, and not necessarily the identifier by which it is known to the Executive Control Program. Program inputs and outputs are specified formally with the name by which they are called within the program. At any subsequent time, a particular input or output may be bound to a specific item within the data pool whenever that item is identical in structure to the formally specified input or output. Thus, in addition to a name, a formal input or output specification may contain an item definition to which any bound input or output item must conform. The program ECD is an identifier by which the Executive Control Program can recognize the program. It is necessary whenever control is passed to the executive in order to run the program.

Whenever a program is entered into the system, it automatically becomes a one-task job. Subsequently, all references to the program are made to this job. In this manner, the program may either be run directly as a job or be used as a job component of a still larger job.

In all cases, programs are maintained within the system through appropriate entries in the system directories. These include:

(1) Program Statement,

(2) Program Description List,

(3) Job Statement,

(4) Job Description List.

The program name is entered into both the Program and Job Name Lists of the Program and Job Statements respectively. The input-output item definitions are translated and entered into the Program Description List. All input-output names along with the program ECD are entered into the Job Description List. In addition, references to the input-output item definitions are established.

The major items of the Program Statement include the Program Null List, the Program List R-No., and the Program Name List. All missing R-numbers of the Program Description List are listed within the Program Null List. In this manner, new programs are entered into vacant slots of the description list. A last R-number is provided if there are no missing R-numbers. The Program Name List is an alphabetically ordered list of all program names within the system with a reference to the corresponding entry of the Program Description List.

The major items of the Program Description List include a Program Binding List for each input-output Parameter of a program. This consists of an item list and term list entry to each input-output parameter.

The Job Statement and Job Description List are described subsequently within the Job Description job (Paragraph 5.2).

5.1.2 Inputs

(1) PROGRAM NAME, A, V

(2) PROGRAM INPUTS, F
INPUT, A, V

(3) PROGRAM OUTPUTS, F
OUTPUT, A, V

(4) PROGRAM ECD, A, V

5.1.3 Results

No outputs for the job are specified. The appropriate entries to the directories constitute the results.

5.1.4 Directories Used

(1) Program Statement.

(2) Program Description List.

(3) Job Statement.

(4) Job Description List

5.1.5 Services Used

(1) Open for Reading.

(2) Close for Reading.

(3) Read.

(4) Open for Writing.

(5) Close for Writing.

(6) Write.

(7) Open for Updating.

(8) Close for Updating.

(9) Seek.

(10) Replace.

(11) Insert.

(12) Delete.

(13) Retrieve Item.

5.1.6 Jobs Used

(1) Unit.

(2) Transaction.

5.1.7 Method of Operation

The Program Entry job is known to the system by the following Job Description:

Job Name: PROGRAM-ENTRY

Job Inputs: program name, program inputs, program outputs, program ECD.

Job Outputs: none specified.

Job Components:

(1) NAME: program name, JOB STATEMENT;
program R-No. field.

(2) NAME: program name, PROGRAM STATEMENT;
program auxiliary R-No. field.

(3) PRIMARY-ENTRY: JOB DESCRIPTION LIST, program R-No. field,
program name, program inputs, program outputs,
program auxiliary R-No. field, program ECD;
program item image list.

(4) AUXILIARY ENTRY: PROGRAM DESCRIPTION LIST,
program auxiliary R-No. field,
program item image list.

The internal Job Description may be represented graphically as shown in Figure 5-1. There are four tasks in the Program Entry Job. They perform the following functions:

(1) Name. This task enters the program name into the Job Name List of the job description library. This name will be used as the job name for the program.

(2) Name. The task Name is used again. This time, the parameters are such that it enters the program name into the Program Name List of the program description library.

(3) Primary Entry. This task takes the two record numbers created by the two preceding tasks and updates the Job Description List. It uses the Unit job in a Job Extension.

(4) Auxiliary Entry. This task takes the record number created by task (2) and the output of task (3) and updates the Program Description List. It uses the Translation job in a job extension.

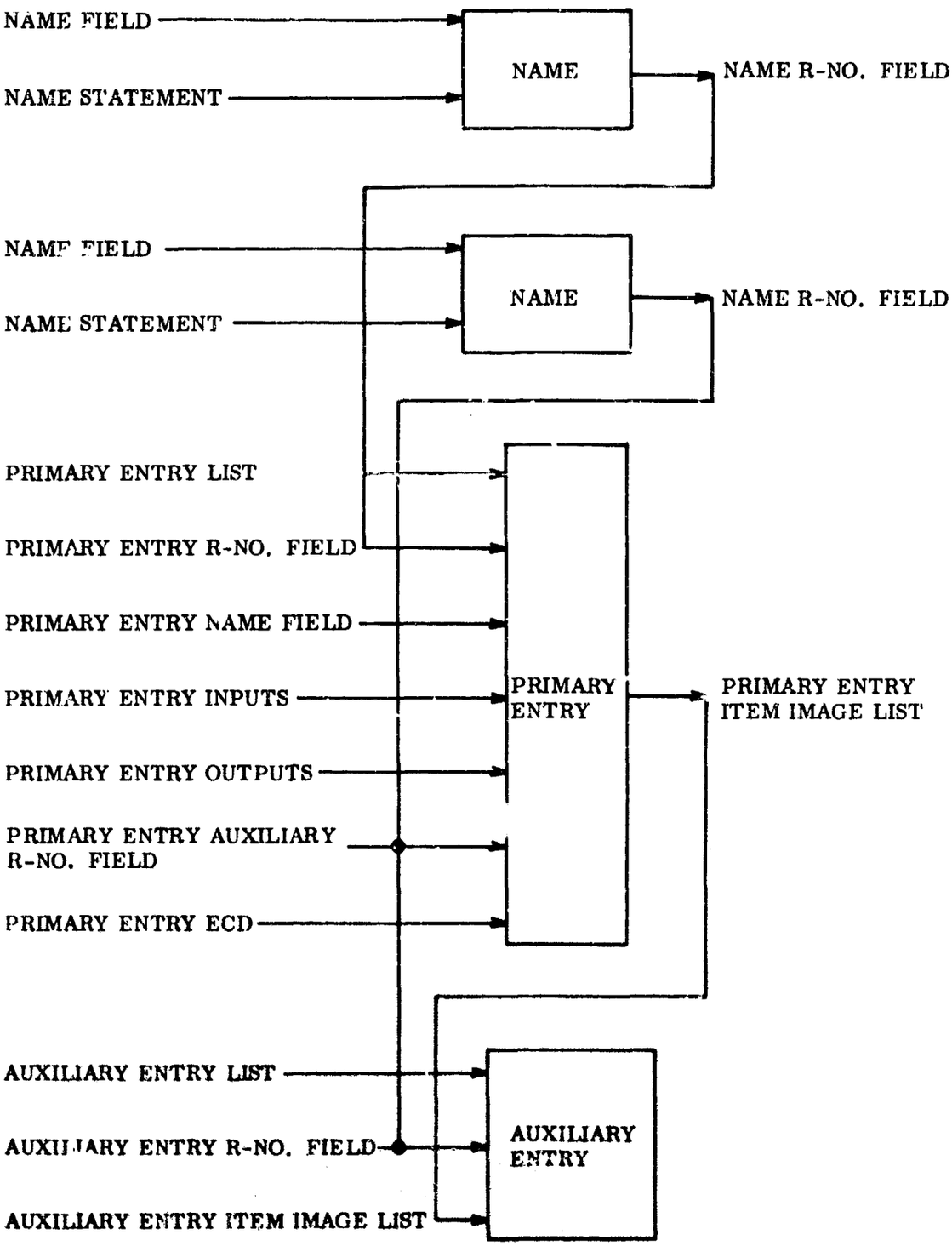


Figure 5-1. The Program Entry Job

5.1.7.1 Name

Job Request: NAME (name field), (name statement);
(name R-No. field).

5.1.7.1.1 Functional Description. The Name Job updates the (name statement) the (name field) and writes the (name R-No. field).

5.1.7.1.2 Inputs

(1) NAME FIELD, A, V

(2) NAME STATEMENT, S, 3

NULL LIST, F

NULL R-NO., I, 18

LAST R-NO., I, 18
NAME LIST, F, ORDERED (1)

NAME, A, V
NAME R-NO., I, 18

5.1.7.1.3 Results. NAME R-NO. FIELD, I, 18

5.1.7.1.4 Directories Used. No directories are used unless externally bound.

5.1.7.1.5 Services Used

(1) Open for Writing.

(2) Close for Writing.

(3) Write.

(4) Open for Updating.

(5) Close for Updating.

(6) Seek.

(7) Read.

(8) Replace.

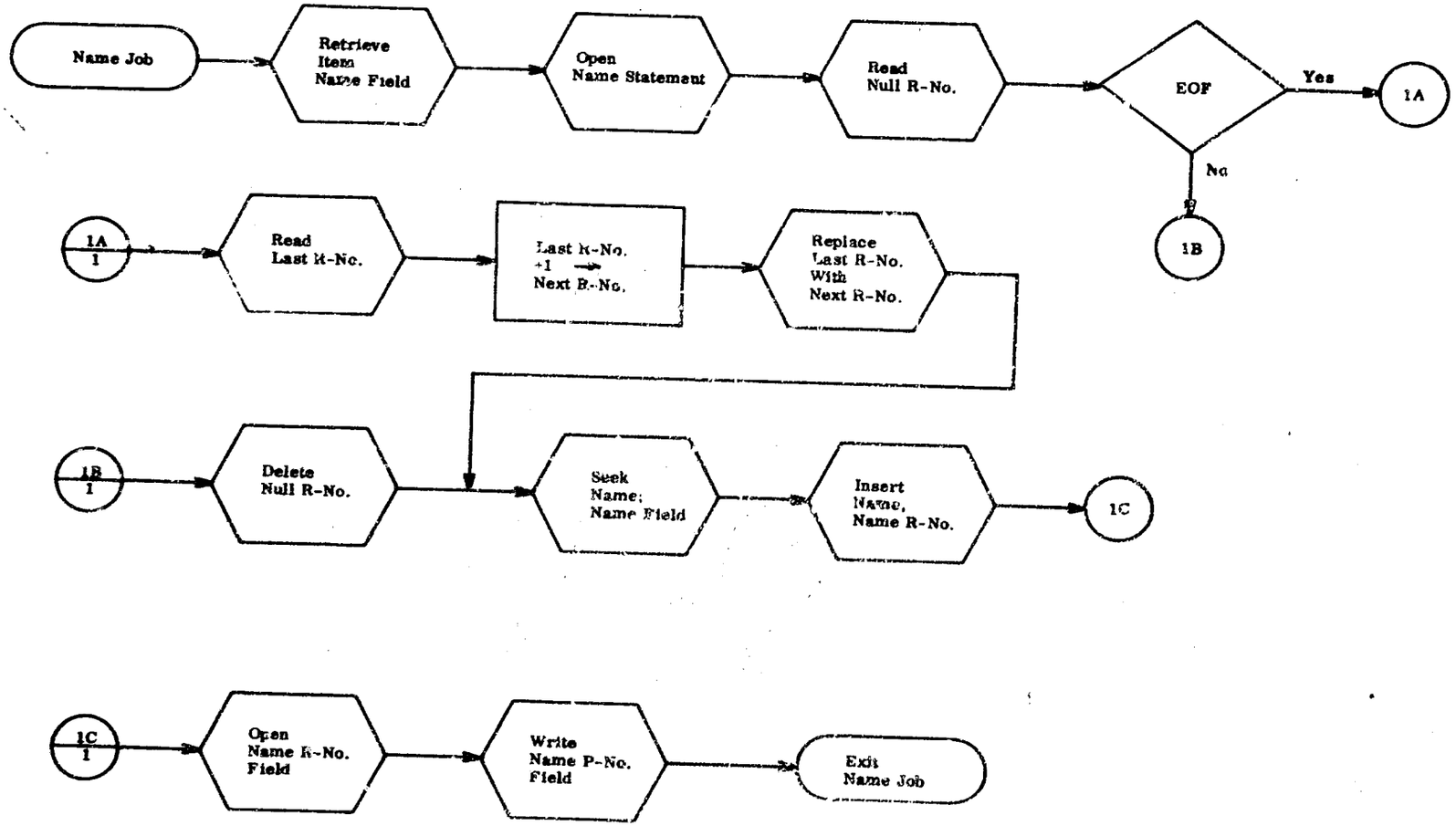
(9) Insert.

(10) Delete.

(11) Retrieve Item.

5.1.7.1.6 Jobs Used. No job extensions are used.

5.1.7.1.7 Method of Operation. The Null List is searched for an R-number (R-No.) and updated. If no R-No. is available, the Last R-No. is taken and updated. The Name and the corresponding Name R-No. are inserted into the Name List. In addition the Name R-No. is written



Name Job
Sheet 1 of 1

5.1.7.2 Primary Entry

Job Request: PRIMARY-ENTRY (primary entry list), (primary entry R-No. field),
(primary entry name field), (primary entry inputs),
(primary entry outputs),
(primary entry auxiliary R-No. field),
(primary entry ECD);
(primary entry item image list).

5.1.7.2.1 Functional Description. With the Job Inputs, the Primary Entry Job updates the Primary Entry List and writes the Primary Entry Item Image List.

5.1.7.2.2 Inputs

(1) PRIMARY ENTRY LIST, F

ID, A, V
ITEM LIST, F

CLASS, I, 3
I/O LIST, F

TYPE, B, 3
I/O NAME, A, V
R-VALUE, H, V

STATIC TASK LIST, F

TYPE, B, 3
TASK ID, I, 12
NO. FLOATS, I, 3
INPUT LIST, F

FORMAL NAME, A, V
CLASS, I, 3
I/O R-NO., I, 15

COMPONENT LIST, F

COMPONENT NAME, A, V
COMPONENT R-NO., I, 18
COMPONENT I/O LIST, F

TYPE, B, 3
I/O NAME, A, V
CLASS, I, 3
I/O R-NO., I, 15

USAGE LIST, F

NAME, A, V

(2) PRIMARY ENTRY R-NO. FIELD, I, 18

(3) PRIMARY ENTRY NAME FIELD, A, V

(4) PRIMARY ENTRY INPUTS, F
INPUT, A, V

(5) PRIMARY ENTRY OUTPUTS, F
OUTPUT, A, V

(6) PRIMARY ENTRY AUXILIARY R-NO. FIELD, I, 18

(7) PRIMARY ENTRY ECD, A, V

5.1.7.2.3 Results

PRIMARY ENTRY ITEM IMAGE LIST, F

ITEM LITERAL LIST, F

ITEM LITERAL, A, V

5.1.7.2.4 Directories Used. No directories are used unless externally bound.

5.1.7.2.5 Services Used

(1) Open for Reading.

(2) Close for Reading.

(3) Read.

(4) Open for Writing.

(5) Close for Writing.

(6) Write.

(7) Open for Updating.

(8) Close for Updating.

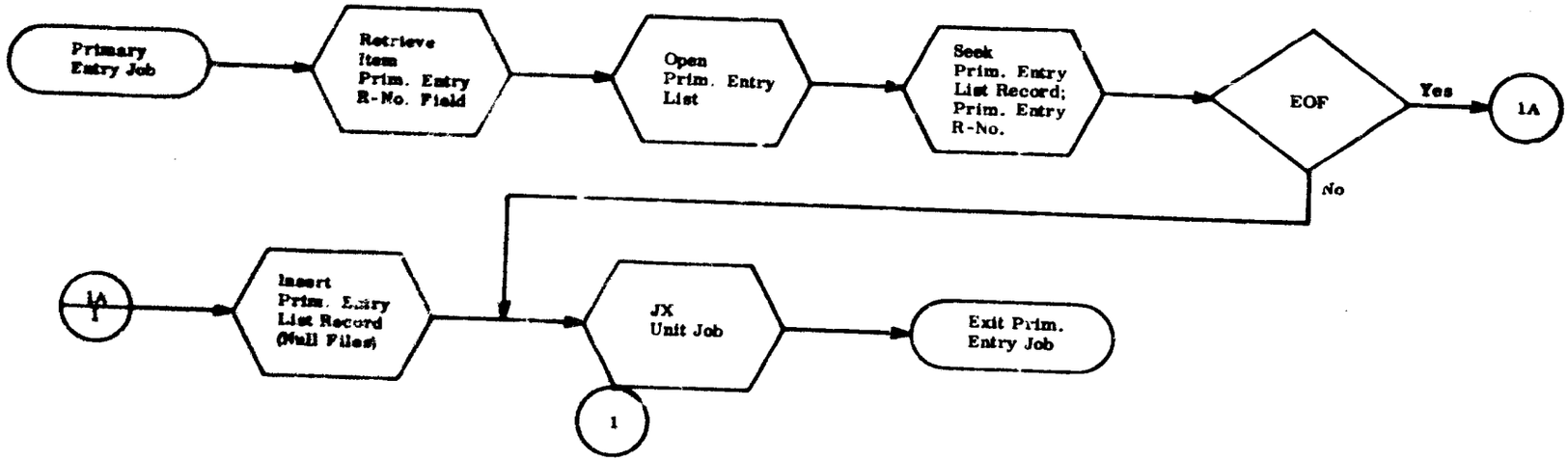
(9) Seek.

(10) Insert.

(11) Retrieve Item.

5.1.7.2.6 Jobs Used: Unit.

5.1.7.2.7 Method of Operation. If the Primary Entry List, modified by the Primary Entry R-No. Field, indicates EOF, a primary entry list record is inserted. In both cases, the Unit Job is then requested as a Job Extension.



NOTE (1)

Prim. Entry Aux. R-No. Field
Prim. Entry List; 2; Prim. Entry R-No.
Prim. Entry List; 8; Prim. Entry R-No.
Prim. Entry List; 20; Prim. Entry R-No.
Prim. Entry Item Image List

Primary Entry Job
Sheet 1 of 1

5.1.7.3 Auxiliary Entry

Job Request: AUXILIARY-ENTRY (auxiliary entry list),
(auxiliary entry R-No. field),
(auxiliary entry item image list).

5.1.7.3.1 Functional Description. The Auxiliary Entry job translates the item image to an internal form and updates the Auxiliary Entry List.

5.1.7.3.2 Inputs

(1) AUXILIARY ENTRY LIST, F

BINDING LIST, F

ITEM LIST, F

RESERVED, B, 10
SRL - ACCESS, ∅, 1
SRL - MODIFICATION, ∅, 1
RESERVED, B, 2
ITEM TYPE, B, 6
OPTION CODE, B, 1
ITEM SIZE, I, 11

TERM LIST, F

TERM NAME, A, V
UNIT, B, 6
RESERVED, I, 18

(2) AUXILIARY ENTRY R-NO. FIELD, I, 18

(3) AUXILIARY ENTRY ITEM IMAGE LIST, F

ITEM LITERAL LIST, F

TERM LITERAL, A, V

5.1.7.3.3 Results. No outputs for the job are specified. The appropriate entries to the Auxiliary Entry List constitute the results.

5.1.7.3.4 Directories Used. No directories are used unless externally bound.

5.1.7.3.5 Services Used

(1) Open for Reading.

(2) Close for Reading.

(3) Read.

(4) Open for Writing.

(5) Close for Writing.

(6) Write.

(7) Open for Updating.

(8) Close for Updating.

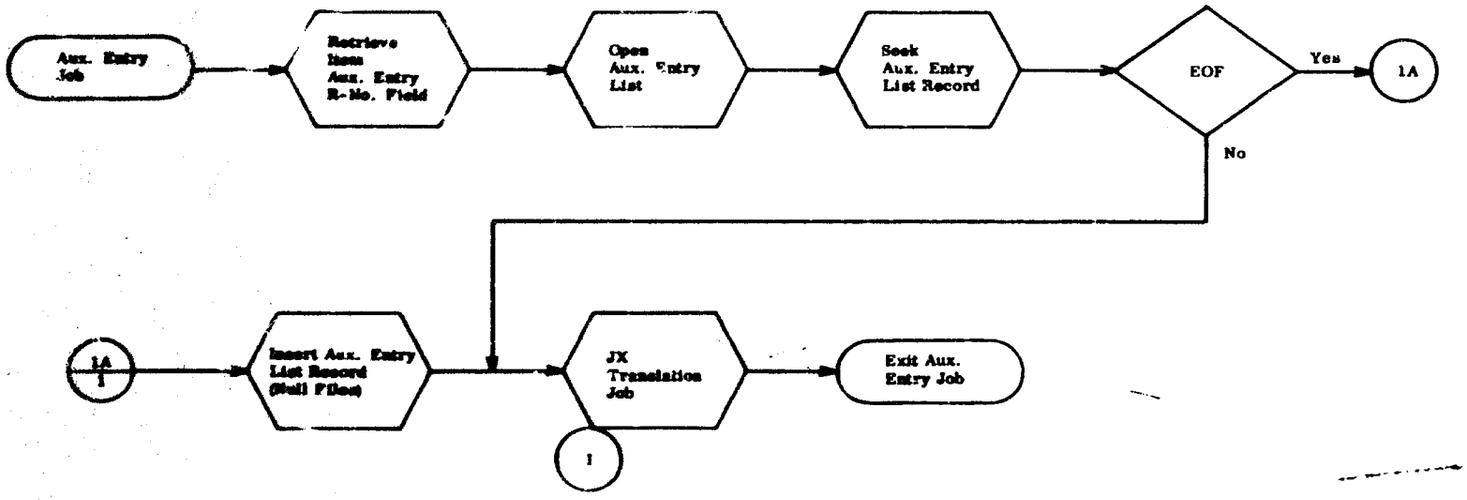
(9) Seek.

(10) Insert.

(11) Retrieve Item

5.1.7.3.6 Jobs Used. Translation.

5.1.7.3.7 Method of Operation. If the Auxiliary Entry List, modified by the Auxiliary Entry R-No. Field, indicates EOF, an auxiliary entry list record is inserted. In both cases, the Translation job is then requested as a Job Extension.



Note 1 Aux. Entry Item Image List
Aux. Entry List; 1; Aux. Entry R-No.

Aux. Entry Job
Sheet 1 of 1

5.1.7.4 Unit

Job Request: UNIT (unit name field), (unit inputs), (unit outputs),
(unit R-No. field), (unit ECD);
(unit component list), (unit item image list),
(unit item list), (unit task list).

5.1.7.4.1 Functional Description. A one-task entry identical to that of the Job Description List is created in three steps by the Unit job. This consists of three files, each of which is written by one of three sequential component jobs. The files are:

(1) Unit Component List,

(2) Unit Item List,

(3) Unit Task List.

In addition, the first component generates the Unit Item Image List for the purpose of subsequent translation.

5.1.7.4.2 Inputs

(1) UNIT FIELD NAME, A, V

(2) UNIT INPUTS, F

INPUT, A, V

(3) UNIT OUTPUTS, F

OUTPUT, A, V

(4) UNIT R-NO. FIELD, I, 18

(5) UNIT ECD, A, V

5.1.7.4.3 Results

(1) UNIT COMPONENT LIST, F

COMPONENT NAME, A, V
COMPONENT R-NO., I, 18
COMPONENT I/O LIST, F

TYPE, B, 3
I/O NAME, A, V
CLASS, I, 3
I/O R-NO., I, 15

(2) UNIT ITEM IMAGE LIST, F

ITEM LITERAL LIST, F

ITEM LITERAL, A, V

(3) UNIT ITEM LIST, F

CLASS, I, 3
I/O LIST, F

TYPE, B, 3
I/O NAME, A, V
R-VALUE, H, V

(4) UNIT TASK LIST, F

TYPE, B, -
TASK ID, I, 12
NO. FLOATS, I, 3
INPUT LIST, F

FORMAL NAME, A, V
CLASS, I, 3
I/O R-NO., I, 15

5.1.7.4.4 Directories Used. No directories are used unless externally bound.

5.1.7.4.5 Services Used.

(1) Open for Reading.

(2) Close for Reading.

(3) Read.

(4) Open for Writing.

(5) Close for Writing.

(6) Write.

(7) Retrieve Item.

5.1.7.4.6 Job Used. No Job Extensions are used.

5.1.7.4.7 Method of Operation. The Unit job is known to the system by the [illegible] Job Description:

Job Name: UNIT

Job Inputs: unit name field, unit inputs, unit outputs, unit R-No. field, unit ECD.

Job Outputs: unit component list, unit item image list, unit item list, unit task list.

Job Components:

(1) UNIT 1: unit name field, unit inputs, unit outputs, unit component list, unit item image list.

(2) UNIT 2: unit component list, unit R-No. field, unit item list.

(3) UNIT 3: unit ECD, unit component list, unit task list

The internal Job Description may be represented graphically as shown Figure 5-2. Unit 1 creates a component list and an item image. From this list, Unit 2 generates an item list and Unit 3 generates a task list.

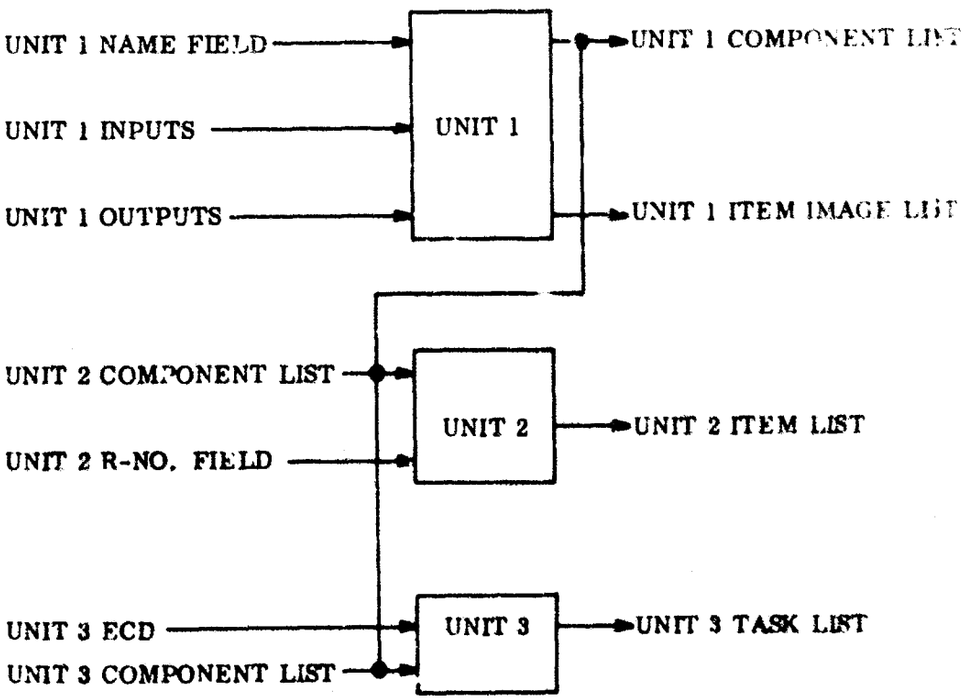


Figure 5-2. Unit Job, Internal Job Description

5.1.7.4.7.1 Unit 1

Job Request: UNIT-1 (unit 1 name field), (unit 1 inputs),
(unit 1 outputs);
(unit 1 component list), (unit 1 item image list).

5.1.7.4.7.1.1 Functional Description. The Unit 1 Job creates a one-task component list and an item image list for that task.

5.1.7.4.7.1.2 Inputs

(1) UNIT 1 NAME FIELD, A, V

(2) UNIT 1 INPUTS, F

INPUT, A, V

(3) UNIT 1 OUTPUTS, F

OUTPUT, A, V

5.1.7.4.7.1.3 Results

(1) UNIT 1 COMPONENT LIST, F

COMPONENT NAME, A, V
COMPONENT R-NO., I, 18
COMPONENT I/O LIST, F

TYPE, B, 3
I/O NAME, A, V
CLASS, I, 3
I/O R-NO., I, 15

(2) UNIT 1 ITEM IMAGE LIST, F

ITEM LITERAL LIST, F

ITEM LITERAL, A, V

5.1.7.4.7.1.4 Directories Used. No directories are used unless externally bound.

5.1.7.4.7.1.5 Services Used

(1) Open for Reading.

(2) Close for Reading.

(3) Read.

(4) Open for Writing.

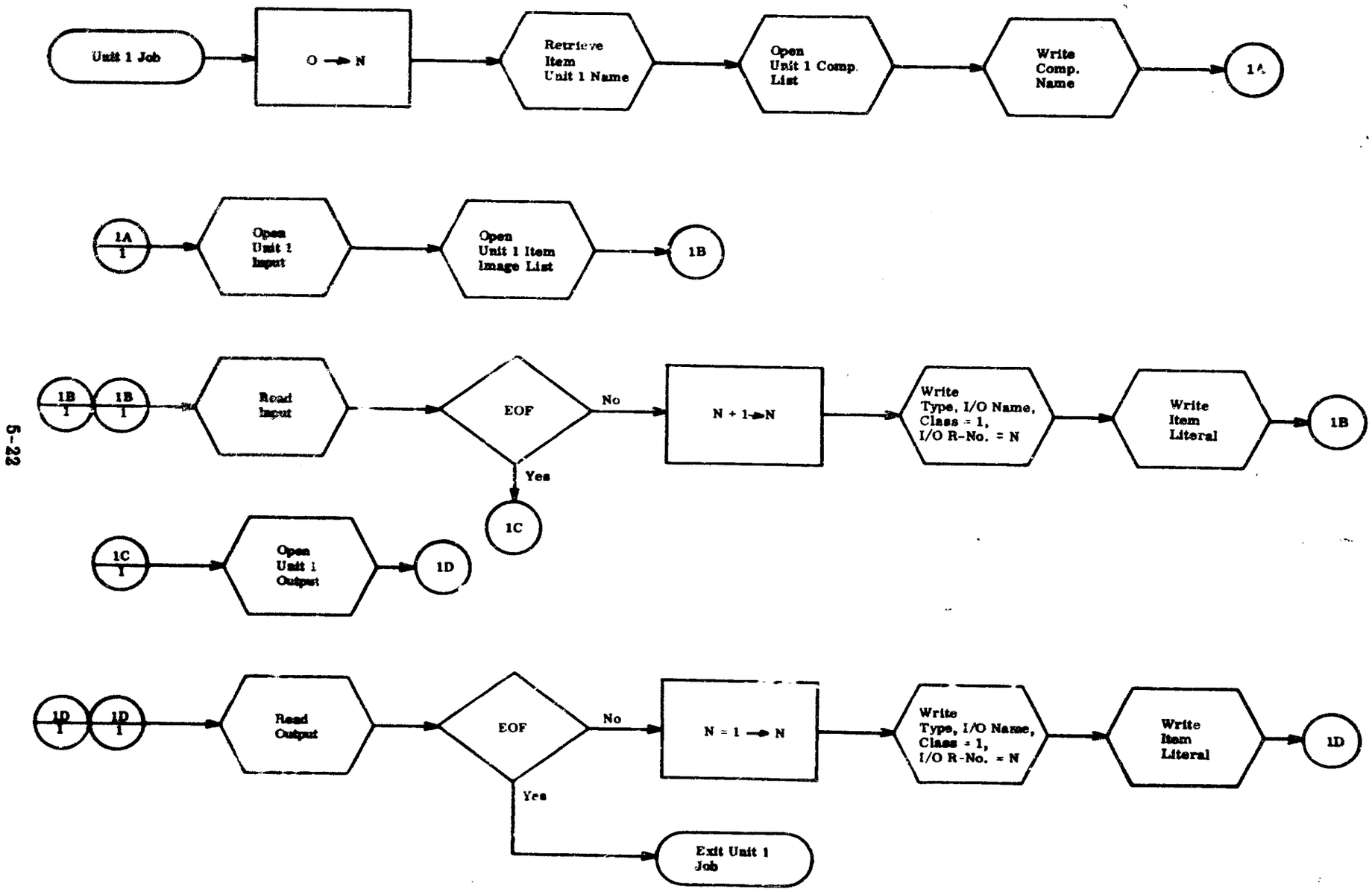
(5) Close for Writing.

(6) Write.

(7) Retrieve Item.

5.1.7.4.7.1.6 Jobs Used. No Job Extensions are used.

5.1.7.4.7.1.7 Method of Operation. The index N is initialized to zero and is used to indicate the N^{th} input-output of the Component I/O List. Upon retrieval of the job inputs, the job outputs are formed and written.



Unit 1 Job
Sheet 1 of 1

5.1.7.4.7.2 Unit 2

Job Request: UNIT-2 (unit 2 component list), (unit 2 R-No. field), (unit 2 item list)

5.1.7.4.7.2.1 Functional Description. The Unit 2 job generates an item list from a one-task component list.

5.1.7.4.7.2.2 Inputs

(1) UNIT 2 COMPONENT LIST, F

COMPONENT NAME, A, V
COMPONENT R-NO., I, 18
COMPONENT I/O LIST, F

TYPE, B, 3
I/O NAME, A, V
CLASS, I, 3
I/O R-NO., I, 15

(2) UNIT 2 R-NO. FIELD, I, 18

5.1.7.4.7.2.3 Results

CLASS, I, 3
I/O LIST, F

TYPE, B, 3
I/O NAME, A, V
R-VALUE, H, V

5.1.7.4.7.2.4 Directories Used. No directories are used unless externally bound.

5.1.7.4.7.2.5 Services Used

(1) Open for Reading.

(2) Close for Reading

(3) Read.

(4) Open for Writing.

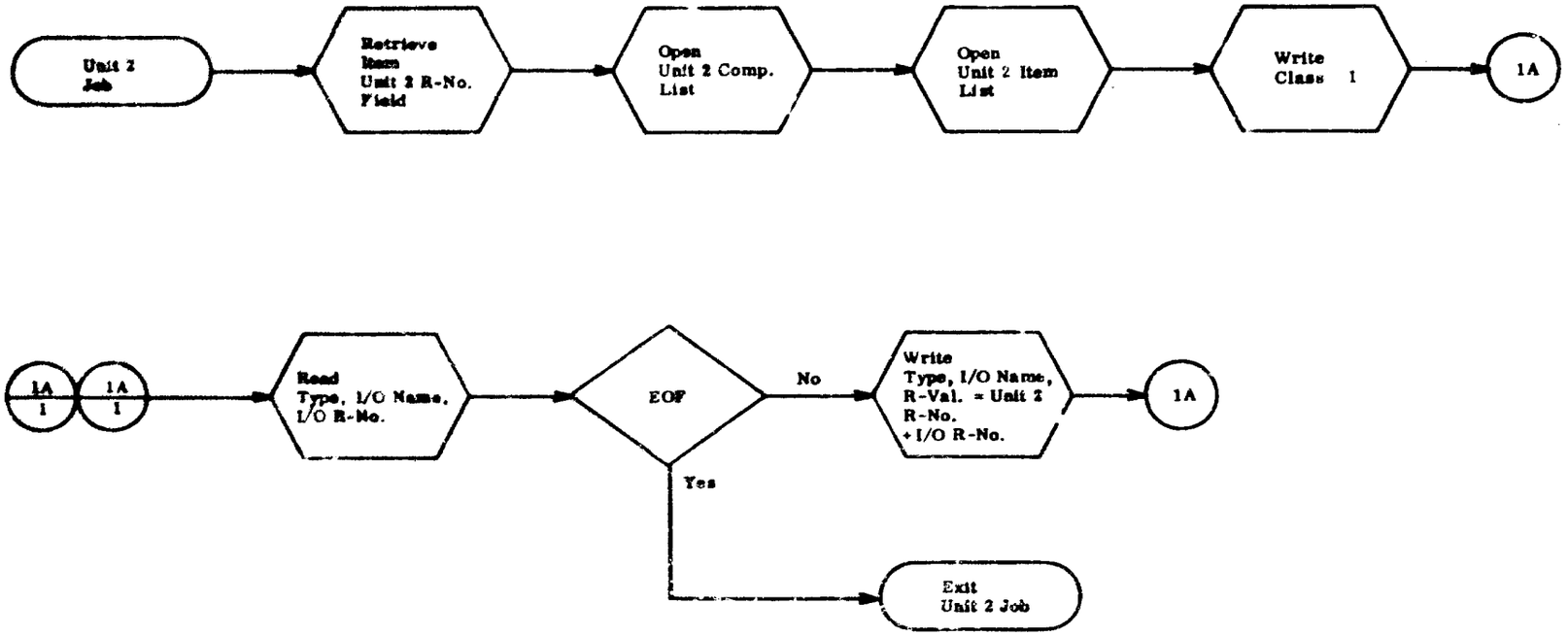
(5) Close for Writing.

(6) Write.

(7) Retrieve Item.

5.1.7.4.7.2.6 Jobs Used. No Job Extensions are used.

5.1.7.4.7.2.7 Method of Operation. A one-task component list is read, and, from this, an item list is formed and written. The R-VALUE field is formed by compounding the I/O R-No. field with the Unit 2 R-No. Field.



CI-25

Unit 2 Job
Sheet 1 of 1

5.1.7.4.7.3 Unit 3

Job Request: UNIT-3 (unit 3 ECD), (unit 3 component list); (unit 3 task list).

5.1.7.4.7.3.1 Functional Description. The Unit 3 Job generates a task list from a One-Task Component List.

5.1.7.4.7.3.2 Inputs

(1) UNIT 3 ECD, A, V

(2) UNIT 3 COMPONENT LIST, F

COMPONENT NAME, A, V
COMPONENT R-NO., I, 18
COMPONENT I/O LIST, F

TYPE, B, 3
I/O NAME, A, V
CLASS, I, 3
I/O R-NO., I, 15

5.1.7.4.7.3.3 Results

TYPE, B, 3
TASK ID, I, 12
NO. FLOATS, I, 3
INPUT LIST, F

FORMAL NAME, A, V
CLASS, I, 3
I/O R-NO., I, 15

5.1.7.4.7.3.4 Directories Used. No directories are used unless externally bound.

5.1.7.4.7.3.5 Services Used

(1) Open for Reading.

(2) Close for Reading.

(3) Read.

(4) Open for Writing.

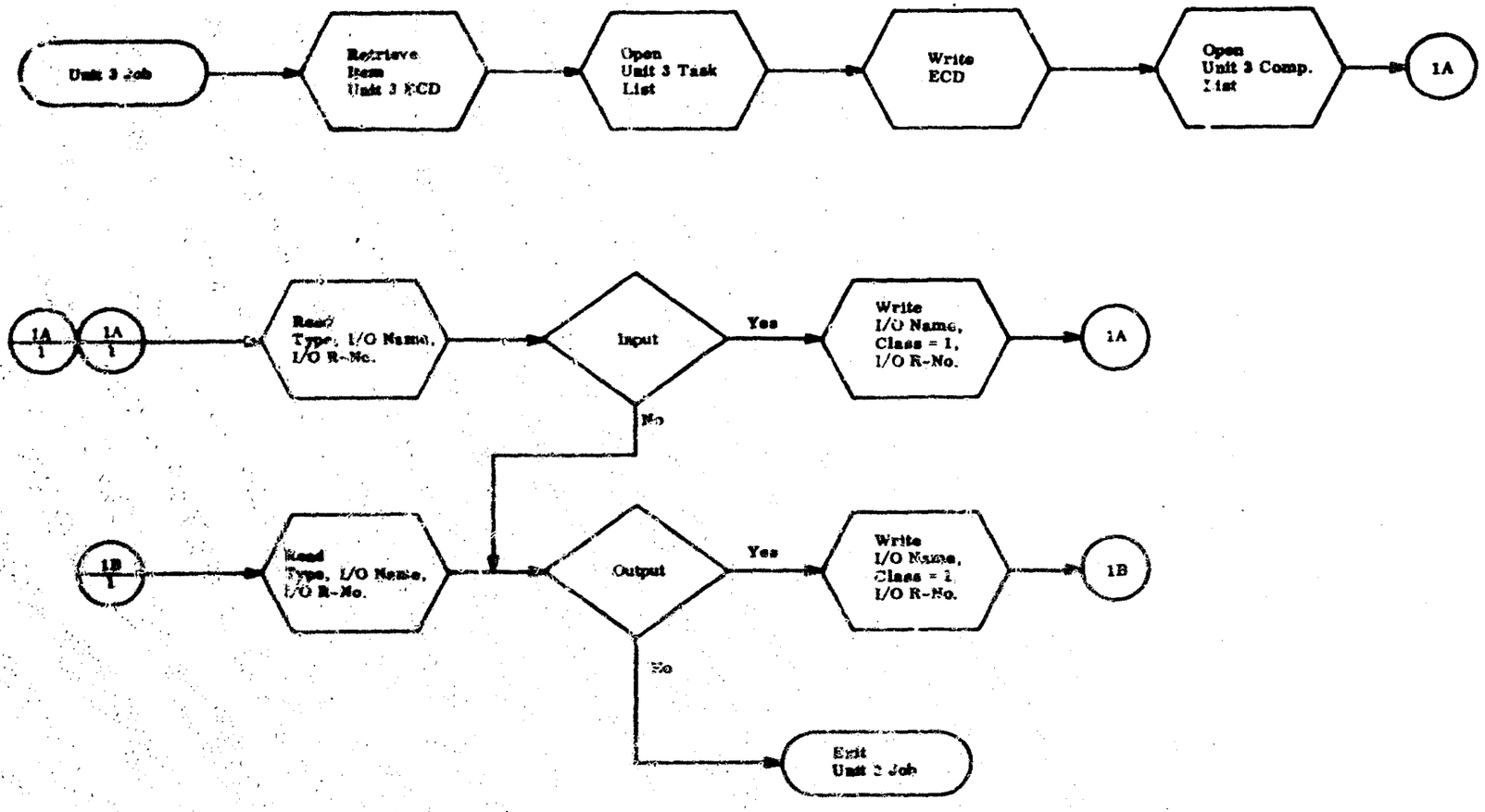
(5) Close for Writing

(6) Write.

(7) Retrieve Item.

5.1.7.4.7.3.6 Jobs Used. No Job Extensions are used.

5.1.7.4.7.3.7 Method of Operation. A one-task component list is read. From this a task list with an ECD heading is formed and written.



Unit 3 Job
Sheet 1 of 1

5.1.7.5 Translation

Job Request: TRANSLATION (translation item image list);
(translation binding list).

5.1.7.5.1 Functional Description. The Translation Job translates an item image list to a binding list.

5.1.7.5.2 Inputs

TRANSLATION ITEM IMAGE LIST, F

ITEM LITERAL LIST, F

ITEM LITERAL, A, V

5.1.7.5.3 Results

TRANSLATION BINDING LIST, F

ITEM LIST, F

RESERVED, B, 10
SRL - ACCESS, ø. 1
SRL - MODIFICATION, ø, 1
RESERVED, B, 2
ITEM TYPE, B, 6
OPTION CODE, B, 1
ITEM SIZE, I, 11

TERM LIST, F

TERM NAME, A, V
UNITS, B, 6
RESERVED, I, 18

5.1 7.5.4 Directories Used. No directories are used unless externally bound.

5.1.7.5.5 Services Used

(1) Open for Reading.

(2) Close for Reading.

(3) Read.

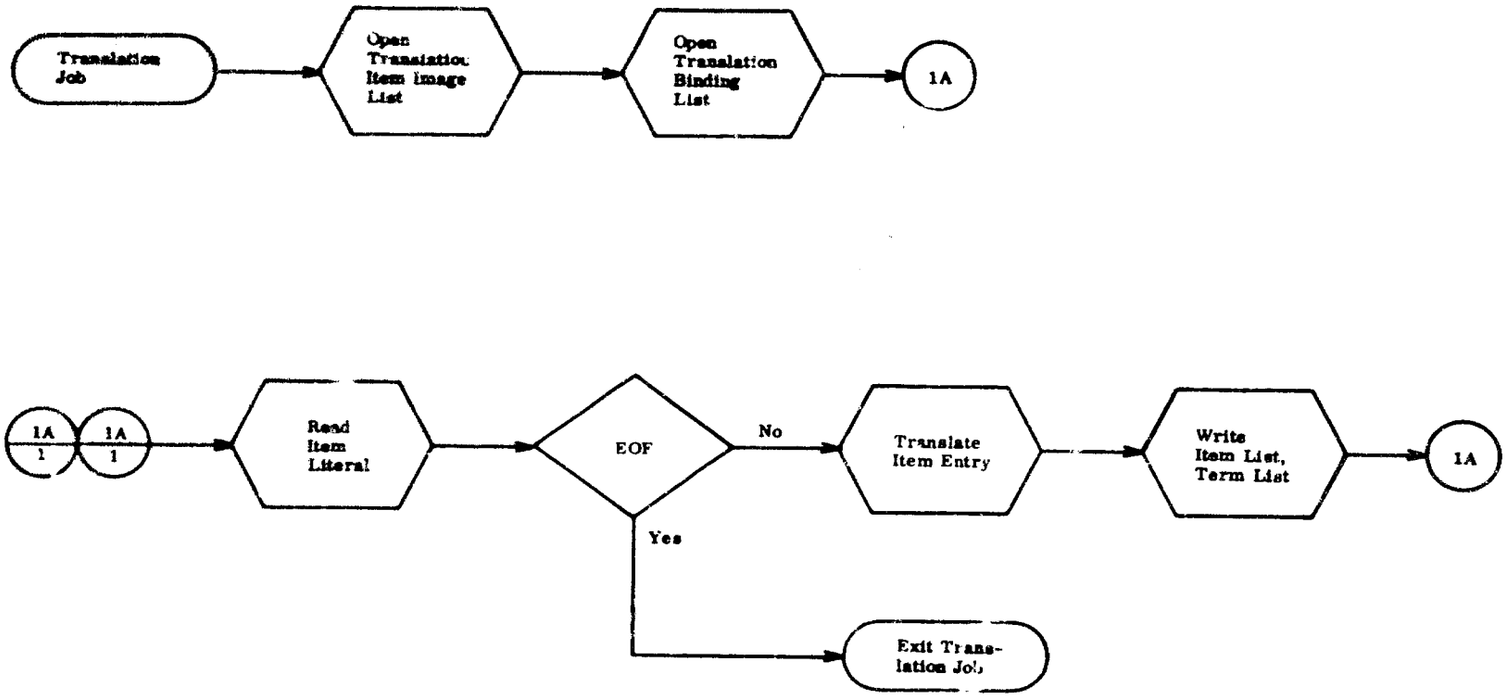
(4) Open for Writing.

(5) Close for Writing.

(6) Write.

5.1.7.5.6 Jobs Used. No Job Extensions are used.

5.1.7.5.7 Method of Operation. An item image list is read, translated to internal form, and written as a binding list.


Translation Job
Open Translation Item Image List
Open Translation Binding List
1A
1A 1
1A 1
Read Item Literal
EOF
No
Translate Item Entry
Write Item List, Term List
1A
Yes
Exit Translation Job

5.2 JOB DESCRIPTION

A job description defines a new job as a sequence of existing jobs. The components in the sequence are jobs from the DM-1 library. They are in the library because they were defined by a previous job description which was processed by the Job Description job or they were entered by a program specification which was processed by the Program Entry job.

To describe a job, the user names the job and its input-output parameters, identifies each component job, and binds the indirect input-output parameters of each component job. The job description includes the following elements:

(1) Job Name

(2) Job Inputs (indirect)

(3) Job Outputs (indirect)

(4) Job Component List

The job name is the name through which the new job will be called for execution. The job inputs and outputs are a series of parameter names for the bindable, indirect, input-output parameters for the new job. The job components list contains the name and binding specification for each job.

The job inputs and outputs are user-assigned names for component inputs and outputs which are not to be made specific by the job description. They are dummy names which are used in the components list to show the relationship between the inputs and outputs of the new job and the inputs and outputs of the component jobs. If the job description makes all component inputs and outputs specific, there are no job inputs or outputs.

The components list contains an entry for each component job in the sequence in which they are to be executed in the new job. It gives the name of the component and binds each of the component's indirect input-output parameters.

Each component input parameter is bound by the assignment of one of the following:

(1) An external parameter. This is a literal value to be used as the value for the component input parameter.

(2) A permanent parameter. This is the name of an item in the data base. The binding specification may include a condition clause and a reformat clause to direct the system to select a subset of the named item and to interpret the selected subset in a format which differs from its format in the data base. The resulting item is to be used as the input to the component each time the new job is run.

(3) An inter-job parameter. This is the name of an item in the work area. The binding specification is the same as that for a permanent parameter.

(4) An indirect parameter. The component input parameter is itself an indirect parameter which must be bound whenever the component is called as a job or used in a job description. The first three parameter types which may be bound to the component input parameter make it specific; i.e., they specify an item in the data pool as the source of the input data. (A literal is an item in the data pool when the job is executed.) However, the new job might be more flexible if the binding for some of the component inputs can be deferred until the job is executed. This is accomplished by binding the component input parameter to an indirect input parameter of the new job. The indirect parameter is assigned a name in the list of job inputs and this name is used to bind the inputs of some of the components. When the new job is executed, its indirect input parameter is made specific by the user. The associated component inputs are made specific at the same time.

(5) A direct parameter. This is a name used to identify an output of a previous component in the component list. It is not specific because there is no node in the data pool corresponding to it. When the new job is executed, the system will assign a node in the scratch area (an intra-job parameter) to accept the output of the earlier component so that it may be used as the input to later components. The output of the earlier component may also be an output of the job. In this case, the name of the job output is used, and the system uses the node bound to the job output as the source of the input data for components whose inputs were bound this way. A condition clause and a reformat clause may be used to specify a subset of the source item and a change in its structure.

Each component output parameter is bound by the assignment of one of the following:

(1) A permanent parameter. This is the name of an item in the data base and a condition, if necessary. It defines the unique node in the data base which is to receive the output item from the component.

(2) An inter-job parameter. This is the name of an item in the work area and a condition, if necessary. It defines a unique node in the work area which is to receive the output from the component.

(3) An indirect parameter. When the binding specification for a component output parameter is to be deferred until the new job is executed, the component output parameter is bound to an output of the new job. The user assigns a name in the job output list and uses this name to bind the component output.

(4) A direct parameter. This is a name used to specify that the component output is to be used as an input to components which occur later in the components list.

The choices for binding the indirect input-output parameters may be summarized as follows:

(1) Specific data-base and work-area items may be bound to some of the input and output parameters of the components.

(2) Some of the components inputs are connected to outputs of previous components by assigning a name to the output and using that name for the inputs.

(3) With either (1) or (2), a condition clause and a reformat clause may be used to specify a subset of the source item and a change in its structure when the source item is used as an input. A condition may be used with (1) to define a unique node if the named item is embedded in a file.

(4) The specific binding of component input and output parameters may be deferred until the new job is executed by assigning a name in the job input or job output list and associating the component input or output with that name.

The job request image for the Job Description job is as follows:

Job Request: JOB DESCRIPTION (job name), (job inputs), (job outputs), (job components).

5.2.1 Functional Description

In general, jobs consist of many component jobs, ordered into a task list. Such jobs are entered into the system via the Job Description job. It is assumed that all specified component jobs exist within the system at the time of this job description.

A description of the job must be given, which consists of the following items:

(1) Job name,

(2) Job inputs,

(3) Job outputs,

(4) Job components.

The job name is the name by which the job is known to the system. Job inputs and job outputs consist of the sum total of all job component inputs and outputs which the user desires as generalized input-output parameters of the new job. All remaining job component inputs and outputs must be bound within the job or to a specific item of the data pool. As such, all bound input-output parameters are no longer considered as job inputs or job outputs, but are known only internally. Job components consist of any number of previously defined jobs and are specified in the desired order of execution and possibly bound through input-output parameters. Each job component is identified by its job name and some input-output parameter for each job input and job output.

Four clases of input-output parameters are possible. These are:

(1) Job input-output,

(2) Intermediate input-output (Intra-Job),

(3) Internal input-output (TET),

(4) Literal input.

Input-output parameters are discussed extensively in Volume I, Section VII.

Jobs are maintained within the system through appropriate entries in the system directories. These include:

(1) Job Statement,

(2) Job Description List.

The job name is entered into the Job Name List of the Job Statement. The job inputs, job outputs and job components are identified and entered into the Job Description List.

The major items of the Job Statement include the Job Null List, the Job Last R-No., and the Job Name List. All missing R-numbers of the Job Description List are listed within the Job Null List. In this manner, new jobs cause entries into vacant slots of the description list. A last R-number is provided should missing numbers be unavailable. The Job Name List is an alphabetically ordered list of all job names within the system along with a reference to the corresponding entry of the Job Description List.

The major items of the Job Description List include a job item list, a static task list, a job components list, and a usage list for every job within the system. A job item list is a list of all inputs and outputs to all tasks of a job grouped into the four classes of input-output parameters. A static task list is a skeletal copy of the identical list of the Request Processor where the input-output IPC's are replaced by references to input-output items of the corresponding job item list. A job component list is a list of all job components as they were entered into the system to form a particular job. Finally, a usage list is a list of job names, identifying all those jobs in which a particular job is used as a component.

5.2.2 Inputs

(1) JOB NAME, A, V

(2) JOB INPUTS, F
INPUT, A, V

(3) JOB OUTPUTS, F
OUTPUT, A, V

(4) JOB COMPONENTS, F
COMPONENT, A, V

5.2.3 Results

No outputs for this job are specified. The appropriate entries to the directories constitute the results.

5.2.4 Directories Used

(1) Job Statement.

(2) Job Description List.

5.2.5 Services Used

(1) Open for Reading.

(2) Close for Reading.

(3) Seek.

(4) Read.

(5) Open for Writing.

(6) Close for Writing.

(7) Write.

(8) Open for Updating.

(9) Close for Updating.

(10) Replace.

(11) Insert.

(12) Delete.

(13) Retrieve Item.

(14) Insert Data.

5.2.6 Jobs Used

Binding

5.2.7 Method of Operation

The Job Description Job is known to the system by the following Job Description:

Job Name: JOB-DESCRIPTION

Job Inputs: job name, job inputs, job outputs, job components.

Job Outputs: none specified.

Job Components:

(1) NAME: job name, JOB STATEMENT;
job R-No. Field.

(2) ENTRY: JOB DESCRIPTION LIST, job R-No. field,
job inputs, job outputs, job components,
JOB STATEMENT.

(3) CATALOG: job name, JOB DESCRIPTION LIST,
job R-No. field.

The internal Job Description may be represented graphically as shown in Figure 5-3.

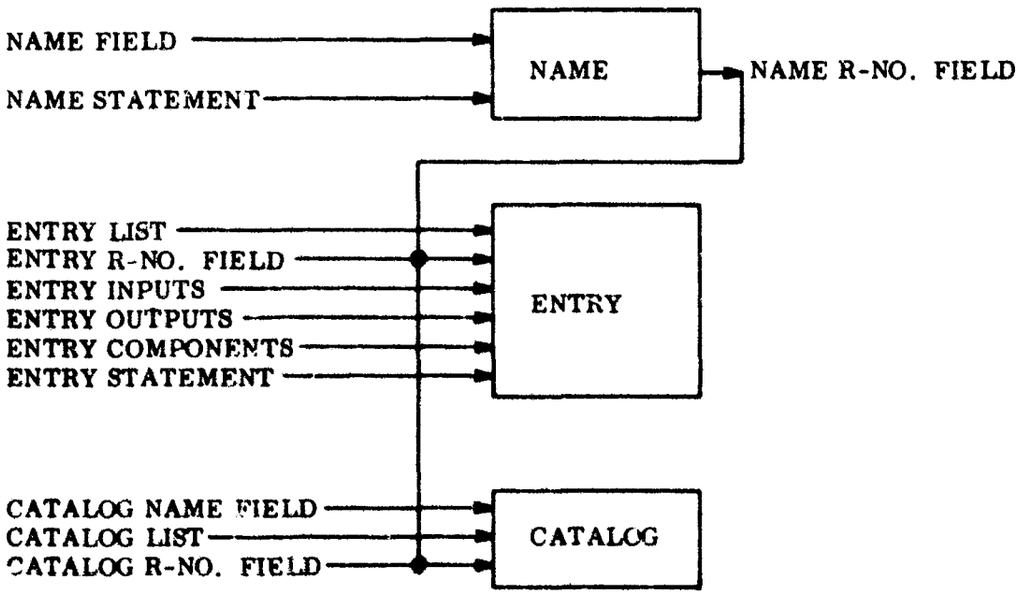


Figure 5-3. Job Description Job, Internal Job Description

The job name is entered into the Job Name List of the Job Statement and the corresponding R-number is written. With the job inputs (the job name is no longer necessary) and this R-number, the Entry job updates the Job Description List. With the job name and the R-number, the Catalog job updates the appropriate usage lists of the Job Description List.

5.2.7.1 Entry

Job Request: ENTRY (entry list), (entry R-No. field), (entry inputs), (entry outputs), (entry components), (entry statement).

5.2.7.1.1 Functional Description. With the job inputs the Entry job updates the Entry List.

5.2.7.1.2 Inputs

(1) ENTRY LIST, F

ID, A, V
ITEM LIST, F

CLASS, I, 3
I/O LIST, F

TYPE, B, 3
I/O NAME, A, V
R-VALUE, H, V

STATIC TASK LIST, F

TYPE, B, 3
TASK ID, I, 12
NO. FLOATS, I, 3
INPUT LIST, F

FORMAL NAME, A, V
CLASS, I, 3
I/O R-NO., I, 15

COMPONENT LIST, F

COMPONENT NAME, A, V
COMPONENT R-NO., I, 18
COMPONENT I/O LIST, F

TYPE, B, 3
I/O NAME, A, V
CLASS, I, 3
I/O R-NO., I, 15

USAGE LIST, F

NAME, A, V

(2) ENTRY R-NO. FIELD, I, 18

(3) ENTRY INPUTS, F
INPUT, A, V

(4) ENTRY OUTPUTS, F
OUTPUT, A, V

(5) ENTRY COMPONENTS, F
COMPONENT, A, V

(6) ENTRY STATEMENT, S, 3
NULL LIST, F
NULL R-NO., I, 18
LAST R-NO., I, 18
NAME LIST, F, ORDERED (1)
NAME, A, V
NAME R-NO., I, 18

5.2.7.1.3 Results. No outputs for the job are specified. The appropriate entries to the Entry List constitute the results.

5.2.7.1.4 Directories Used. No directories are used unless externally bound.

5.2.7.1.5 Services Used

(1) Open for Reading.

(2) Close for Reading.

(3) Seek.

(4) Read.

(5) Open for Writing.

(6) Close for Writing.

(7) Write.

(8) Open for Updating.

(9) Close for Updating.

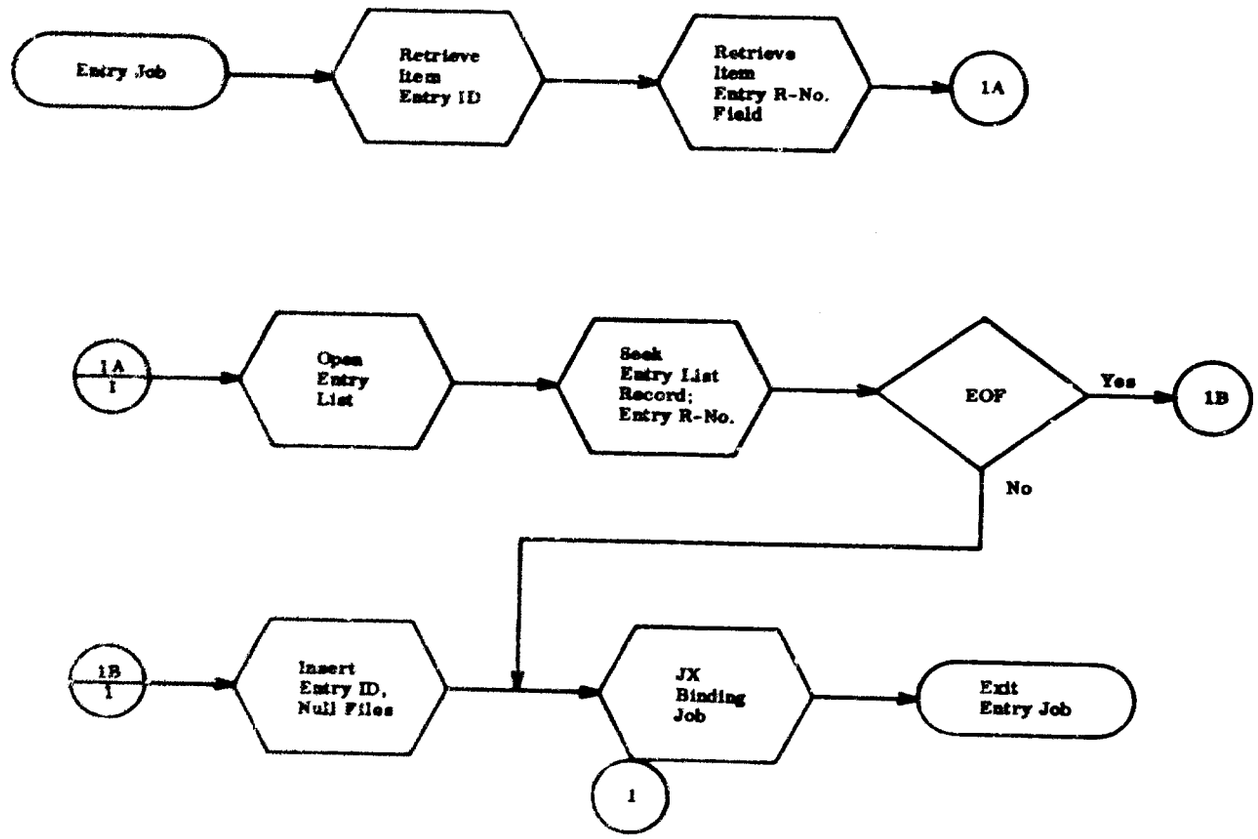
(10) Replace.

(11) Insert.

(12) Retrieve Item.

5.2.7.1.6 Jobs Used. Binding.

5.2.7.1.7 Method of Operation. If the Entry List, modified by the Entry R-No. Field, indicates EOF, an entry list record is inserted. In both cases the Binding job is then requested as a Job Extension.



Note (1) Entry Statement
Entry List
Entry List; 20; Entry R-No.
Entry List; 2; Entry R-No.
Entry List; 8; Entry R-No.
Entry List, 20, Entry R-No.

5.2.7.2 Catalog

Job Request: CATALOG (catalog name field), (catalog list), (catalog R-No. field).

5.2.7.2.1 Functional Description. With the Catalog Name Field and the Catalog R-No. Field, the Catalog job updates the appropriate usage lists of the Catalog List.

5.2.7.2.2 Inputs

(1) CATALOG NAME FIELD, A, V

(2) CATALOG LIST, F

ID, A, V
ITEM LIST, F

CLASS, I, 3
I/O LIST, F

TYPE, B, 3
I/O NAME, A, V
R-VALUE, H, V

STATIC TASK LIST, F

TYPE, B, 3
TASK ID, I, 12
NO. FLOATS, I, 3
INPUT LIST, F

FORMAL NAME, A, V
CLASS, I, 3
I/O R-NO., I, 15

COMPONENT LIST, F

COMPONENT NAME, A, V
COMPONENT R-NO I, 18
COMPONENT I/O LIST, F

TYPE, B, 3
I/O NAME, A, V
CLASS, I, 3
I/O R-NO., I, 15

USAGE LIST, F

NAME, A, V

(3) CATALOG R-NO. FIELD, I, 18

5.2.7.2.3 Results. No outputs for the job are specified. The appropriate entries to the Catalog List constitute the results.

5.2.7.2.4 Directories Used. No directories are used unless externally bound.

5.2.7.2.5 Services Used

(1) Open for Reading.

(2) Close for Reading.

(3) Seek.

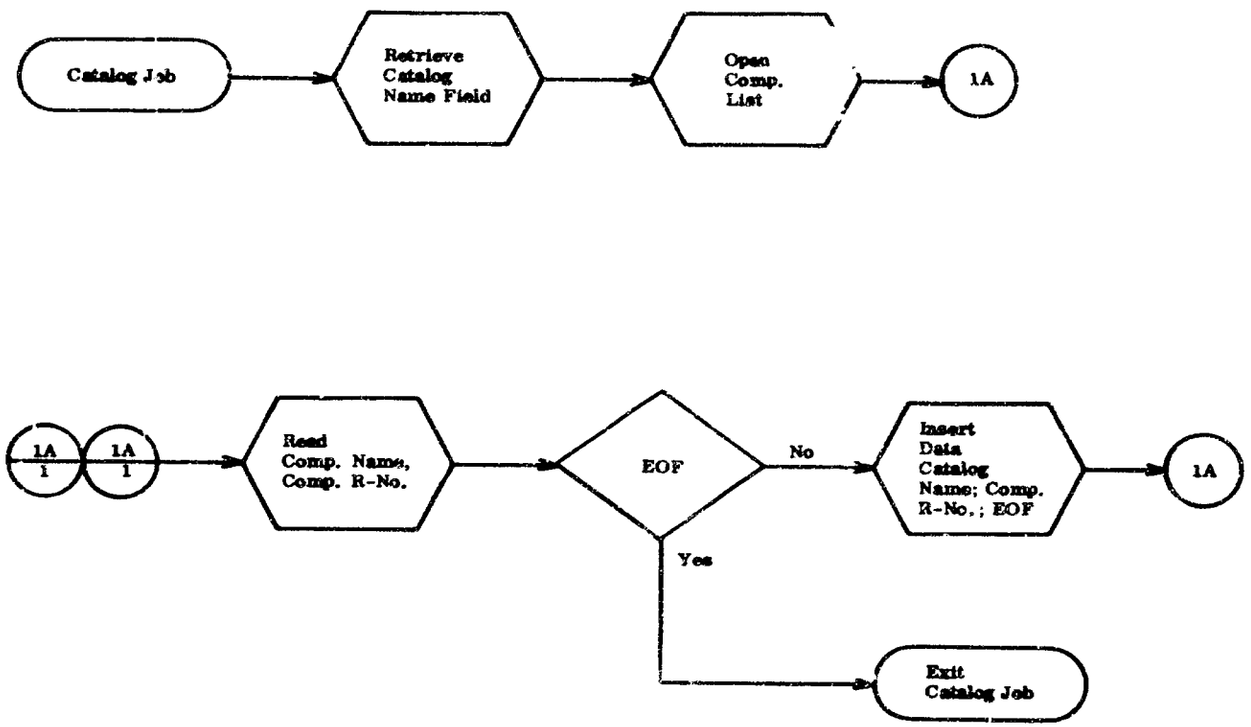
(4) Read.

(5) Retrieve Item.

(6) Insert Data.

5.2.7.2.6 Jobs Used. No Job Extensions are used.

5.2.7.2.7 Method of Operation. The Component List of the Catalog List, modified by the Catalog R-No. Field, is identified. Each Component R-No. is read and the Catalog Name Field is added to the Usage List, modified by this R-number.



Catalog Job
Sheet 1 of 1

5.2.7.3 Binding

Job Request: BINDING (bind inputs-opt.), (bind outputs-opt.),
(bind components), (bind statement), (bind list);
(bind component list), (bind item list), (bind task list).

5.2.7.3.1 Functional Description. An entry identical to that of the Job Description List is created in three steps by the Binding job. This consists of three files, each of which is written by one of three sequential component jobs. The files are:

(1) Bind Component List,

(2) Bind Item List,

(3) Bind Task List.

The Binding job relates all input-output parameters to four general classes. These are:

(1) Indirect Input-Output,

(2) Direct Input-Output,

(3) Internal Input-Output,

(4) External Input.

Input-output parameters are discussed extensively in Volume I, Section VII.

5.2.7.3.2 Inputs

(1) BIND INPUTS, F
INPUT, A, V

(2) BIND OUTPUTS, F
OUTPUT, A, V

(3) BIND COMPONENTS, F
COMPONENT, A, V

(4) BIND STATEMENTS, S, 3
NULL LIST, F
NULL R-NO., I, 18
LAST R-NO., I, 18
NAME LIST, F, CONDITIONED (2)
NAME, A, V
NAME R-NO., I, 18

(5) BIND LIST, F

ID, A, V
ITEM LIST, F

CLASS, I, 3
I/O LIST, F

TYPE, B, 3
I/O NAME, A, V
R-VALUE, H, V

STATIC TASK LIST, F

TYPE, B, 3
TASK ID, I, 12
NO. FLOATS, I, 3
INPUT LIST, F

FORMAL NAME, A, V
CLASS, I, 3
I/O R-NO., I, 15

COMPONENT LIST, F

COMPONENT NAME, A, V
COMPONENT R-NO., I, 18
COMPONENT I/O LIST, F

TYPE, B, 3
I/O NAME, A, V
CLASS, I, 3
I/O R-NO., I, 15

USAGE LIST, F

NAME, A, V

5.2.7.3.3 Results

(1) BIND COMPONENT LIST, F

COMPONENT NAME, A, V
COMPONENT R-NO., I, 18
COMPONENT I/O LIST, F

TYPE, B, 3
I/O NAME, A, V
CLASS, I, 3
I/O R-NO., I, 15

(2) BIND ITEM LIST, F

CLASS, I, 3
I/O LIST, F

TYPE, B, 3
I/O NAME, A, V
R-VALUE, H, V

(3) BIND TASK LIST, F

TYPE, B, 3
TASK ID, I, 12
NO. FLOATS, I, 3
INPUT LIST, F

FORMAL NAME, A, V
CLASS, I, 3
I/O R-NO., I, 15

5.2.7.3.4 Directories Used. No directories are used unless externally bound.

5.2.7.3.5 Services Used

(1) Open for Reading.

(2) Close for Reading.

(3) Seek.

(4) Read.

(5) Open for Writing.

(6) Close for Writing.

(7) Write.

(8) Open for Updating.

(9) Close for Updating.

(10) Replace.

5.2.7.3.6 Jobs Used. No Job Extensions are used.

5.2.7.3.7 Method of Operation. The Binding job is known to the system by the following Job Description:

Job Name: BINDING

Job Inputs: bind inputs (optional), bind outputs (optional), bind components, bind statement, bind list.

Job Outputs: bind component list, bind item list, bind task list.

The internal Job Description may be represented graphically as shown in Figure 5-4. Bind 1 creates the Component List. From this, Bind 2 generates the Item List, and Bind 3 generates the Task List.

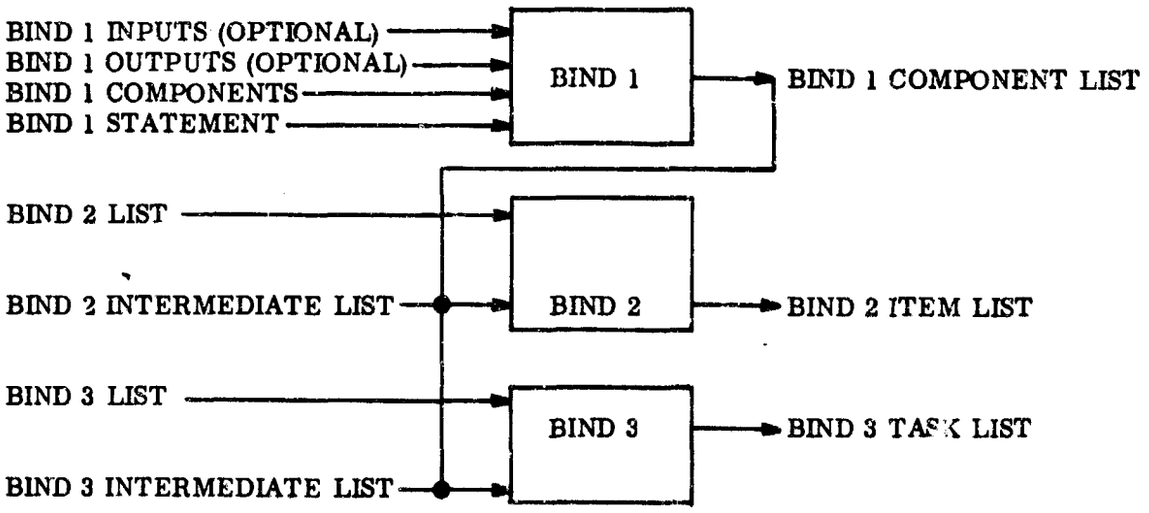


Figure 5-4. Binding Job, Internal Job Description

5.2.7.3.7.1 Bind 1

Job Request: BIND-1 (bind 1 inputs), (bind 1 outputs), (bind 1 components), (bind 1 statement); (bind 1 component list).

5.2.7.3.7.1.1 Functional Description. The Bind 1 job creates a component list.

5.2.7.3.7.1.2 Inputs

(1) BIND 1 INPUTS, F
INPUT, A, V

(2) BIND 1 OUTPUTS, F
OUTPUT, A, V

(3) BIND 1 COMPONENTS, F
COMPONENT, A, V

(4) BIND 1 STATEMENT, S, 3
NULL LIST, F
NULL R-NO., I, 18
LAST R-NO., I, 18
NAME LIST, F, ORDERED (1)
NAME, A, V
NAME R-NO., I, 18

5.2.7.3.7.1.3 Results

BIND 1 COMPONENT LIST, F

COMPONENT NAME, A, V
COMPONENT R-NO., I, 18
COMPONENT I/O LIST, F

TYPE, B, 3
I/O NAME, A, V
CLASS, I, 3
I/O R-NO., I, 15

5.2.7.3.7.1.4 Directories Used. No directories are used unless externally bound.

5.2.7.3.7.1.5 Services Used.

(1) Open for Reading.

(2) Close for Reading.

(3) Seek.

(4) Read.

(5) Open for Writing.

(6) Close for Writing.

(7) Write.

5.2.7.3.7.1.6 Jobs Used. No Job Extensions are used.

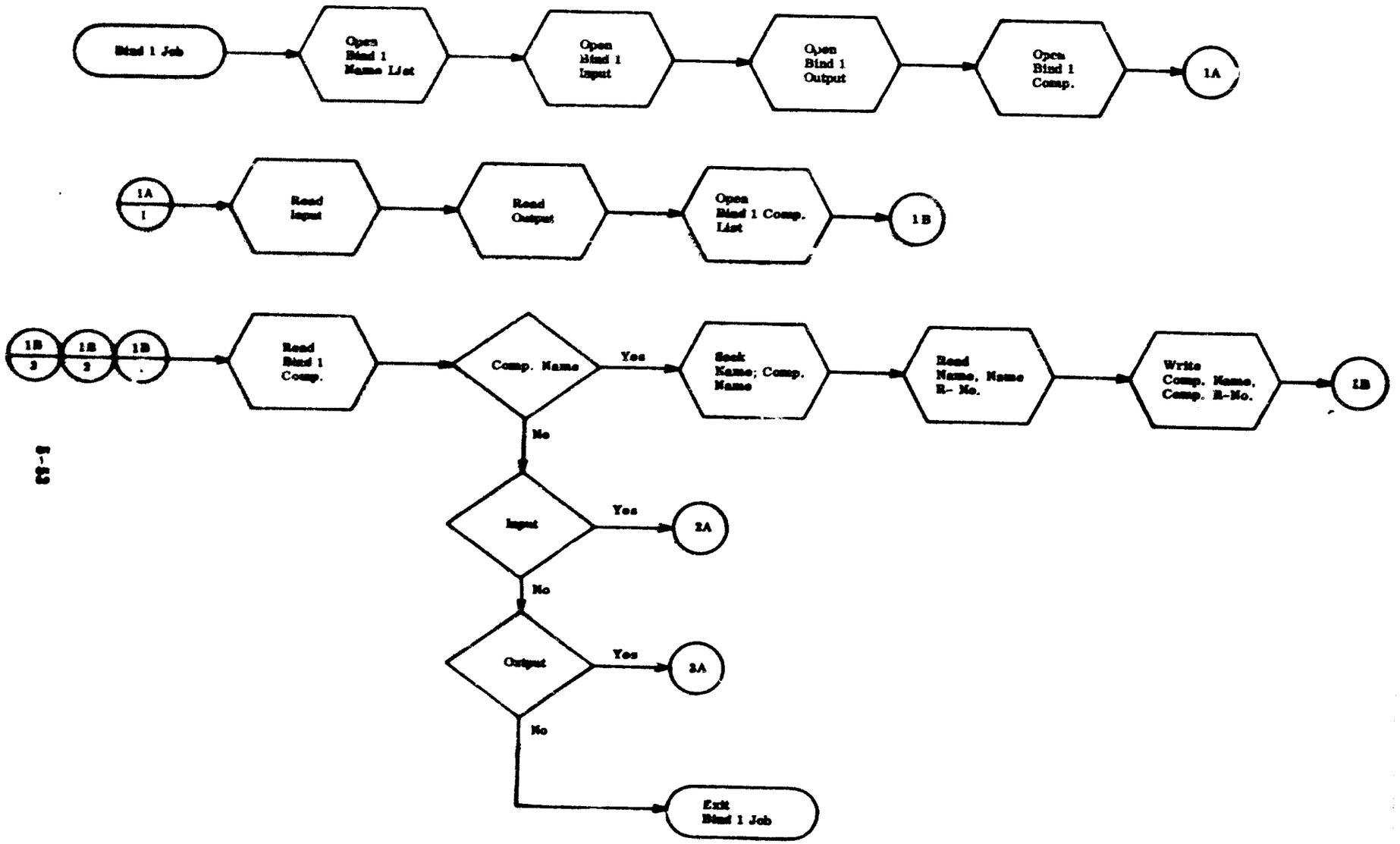
5.2.7.3.7.1.7 Method of Operation. The Component List is formed and written in the order in which the job components are read. With each component, inputs precede outputs. In both cases, the following identifications are made:

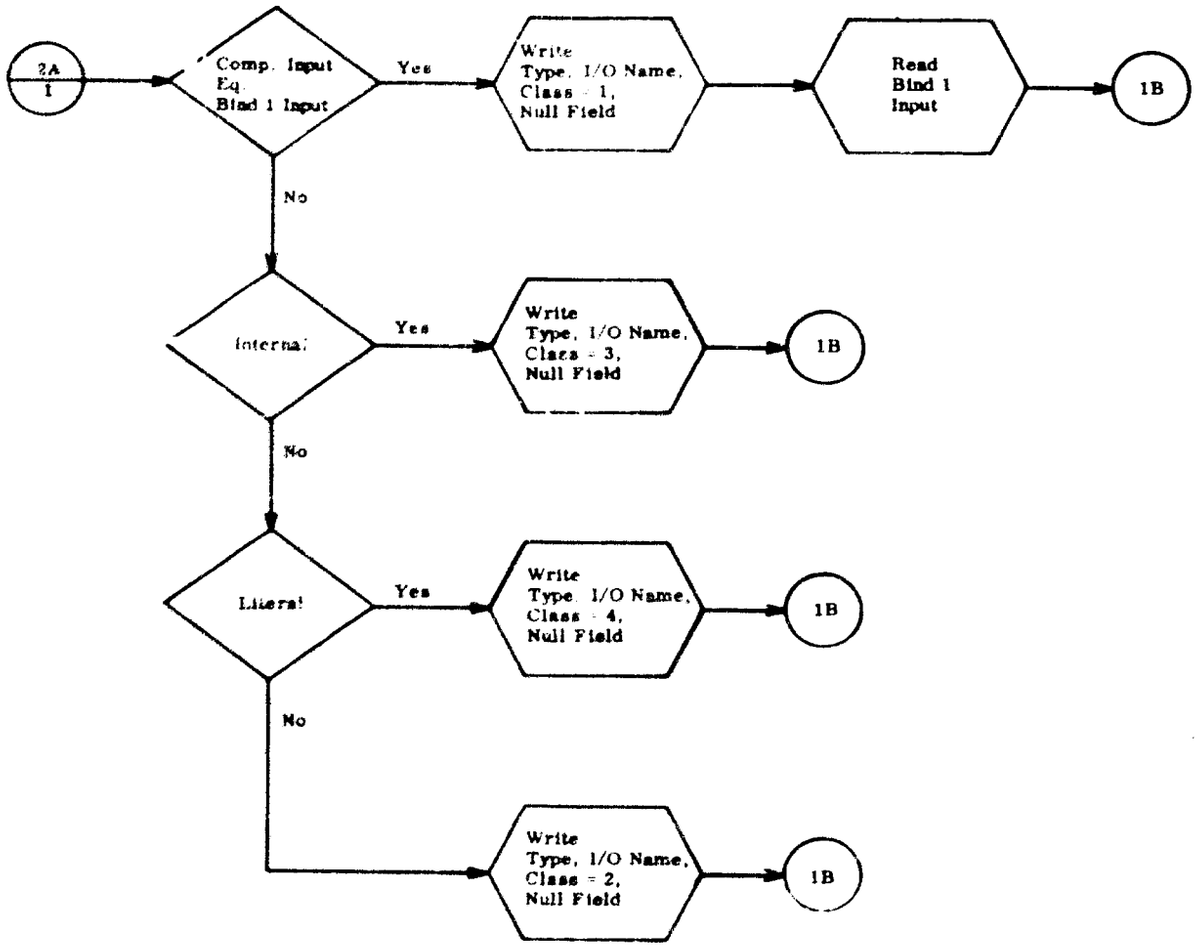
(1) If component input-output equals any Bind 1 input-output, Class = 1 (Indirect).

(2) If component input-output is marked (TET), Class = 3 (Internal).

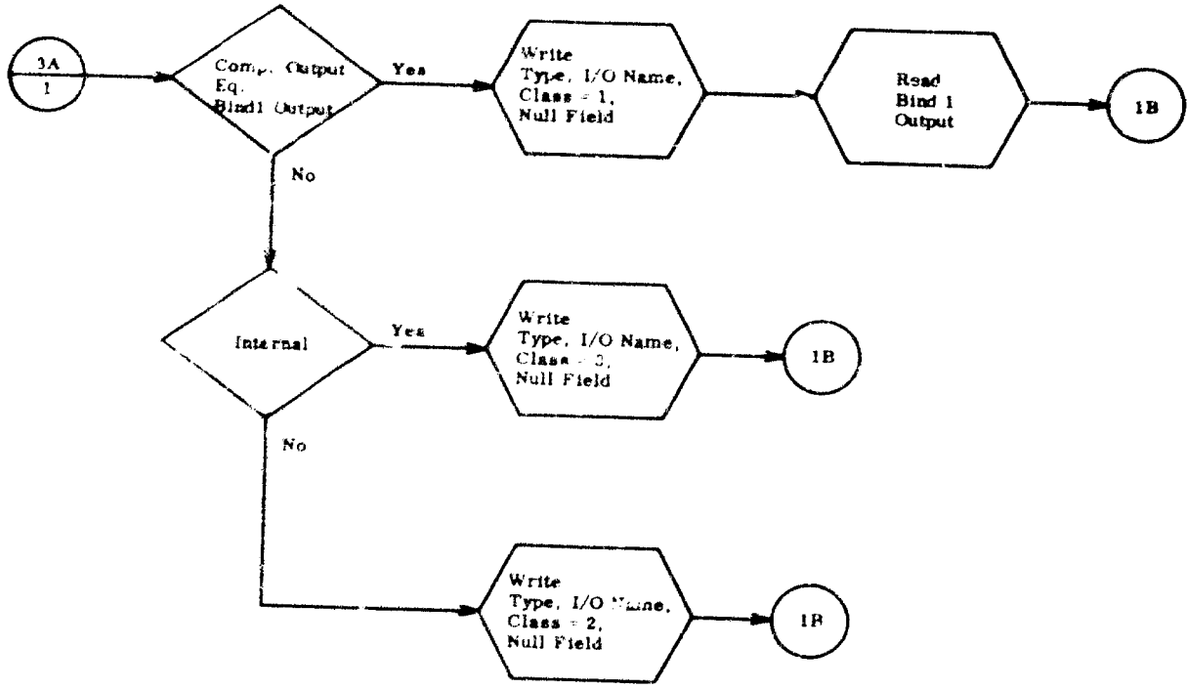
(3) If component input is external, Class = 4 (External).

(4) In all other cases, Class = 2 (Direct).


Bind 1 Job
Open Bind 1 Name List
Open Bind 1 Input
Open Bind 1 Output
Open Bind 1 Comp.
1A
1A / 1
Read Input
Read Output
Open Bind 1 Comp. List
1B
1B / 2
1B / 2
1B / .
Read Bind 1 Comp.
Comp. Name
Yes
Seek Name; Comp. Name
Read Name, Name R- No.
Write Comp. Name, Comp. R-No.
1B
No
Input
Yes
2A
No
Output
Yes
2A
No
Exit Bind 1 Job



Bind 1 Job
Sheet 2 of 3


3A
1
Comp. Output Eq. Bind1 Output
Yes
Write Type, I/O Name, Class = 1, Null Field
Read Bind 1 Output
1B
No
Internal
Yes
Write Type, I/O Name, Class = 3, Null Field
1B
No
Write Type, I/O Name, Class = 2, Null Field
1B

5.2.7.3.7.2 Bind 2

Job Request: BIND-2 (bind 2 list), (bind 2 intermediate list), (bind 2 item list).

5.2.7.3.7.2.1 Functional Description. The Bind 2 job generates an item list.

5.2.7.3.7.2.2 Inputs

(1) BIND 2 LIST, F

ID, A, V
ITEM LIST, F

CLASS, I, 3
I/O LIST, F

TYPE, B, 3
I/O NAME, A, V
R-VALUE, H, V

STATIC TASK LIST, F

TYPE, B, 3
TASK ID, I, 12
NO. FLOATS, I, 3
INPUT LIST, F

FORMAL NAME, A, V
CLASS, I, 3
I/O R-NO., I, 15

COMPONENT LIST, F

COMPONENT NAME, A, V
COMPONENT R-NO., I, 18
COMPONENT I/O LIST, F

TYPE, B, 3
I/O NAME, A, V
CLASS, I, 3
I/O R-NO., I, 15

USAGE LIST, F

NAME, A, V

(2) BIND 2 INTERMEDIATE LIST, F

COMPONENT NAME, A, V
COMPONENT R-NO., I, 18
COMPONENT I/O LIST, F

TYPE, B, 3
I/O NAME, A, V
CLASS, I, 3
I/O R-NO., I, 15

5.2.7.3.7.2.3 Results

BIND 2 ITEM LIST, F

CLASS, I, 3
I/O LIST, F

TYPE, B, 3
I/O NAME, A, V
R-VALUE, H, V

5.2.7.3.7.2.4 Directories Used. No directories are used unless externally bound.

5.2.7.3.7.2.5 Services Used

(1) Open for Reading.

(2) Close for Reading.

(3) Seek.

(4) Read.

(5) Open for Writing.

(6) Close for Writing.

(7) Write.

(8) Open for Updating.

(9) Close for Updating.

(10) Replace.

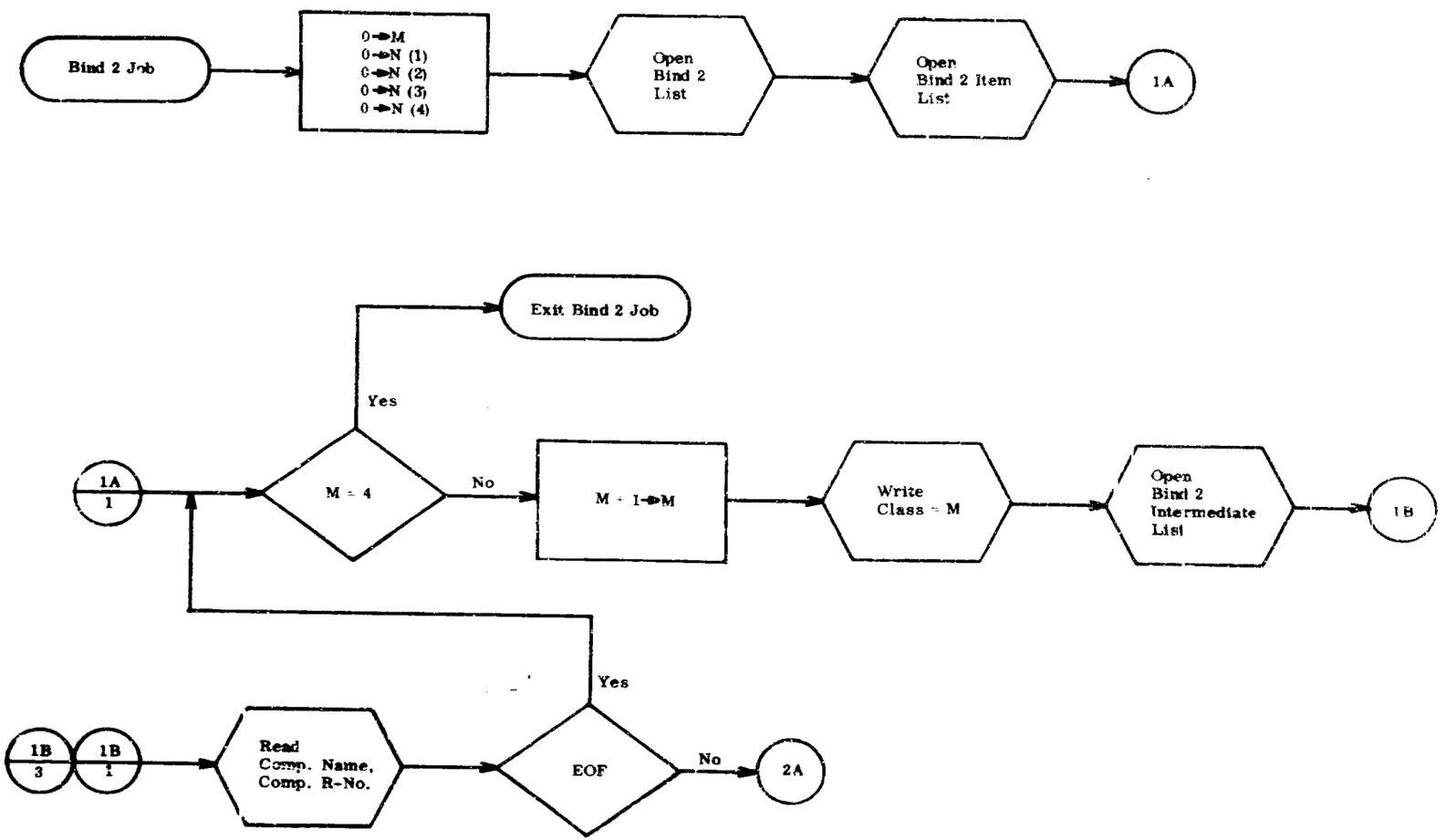
5.2.7.3.7.2.6 Jobs Used. No Job Extensions are used.

5.2.7.3.7.2.7 Method of Operation. Index M is initialized to zero. It indicates the M^{th} class. Indices N(1), N(2), N(3), and N(4) are likewise initialized to zero. They specify the N^{th} input-output parameter of one of four classes. Thus, N(M) specifies the N^{th} input-output parameter of the M^{th} class.

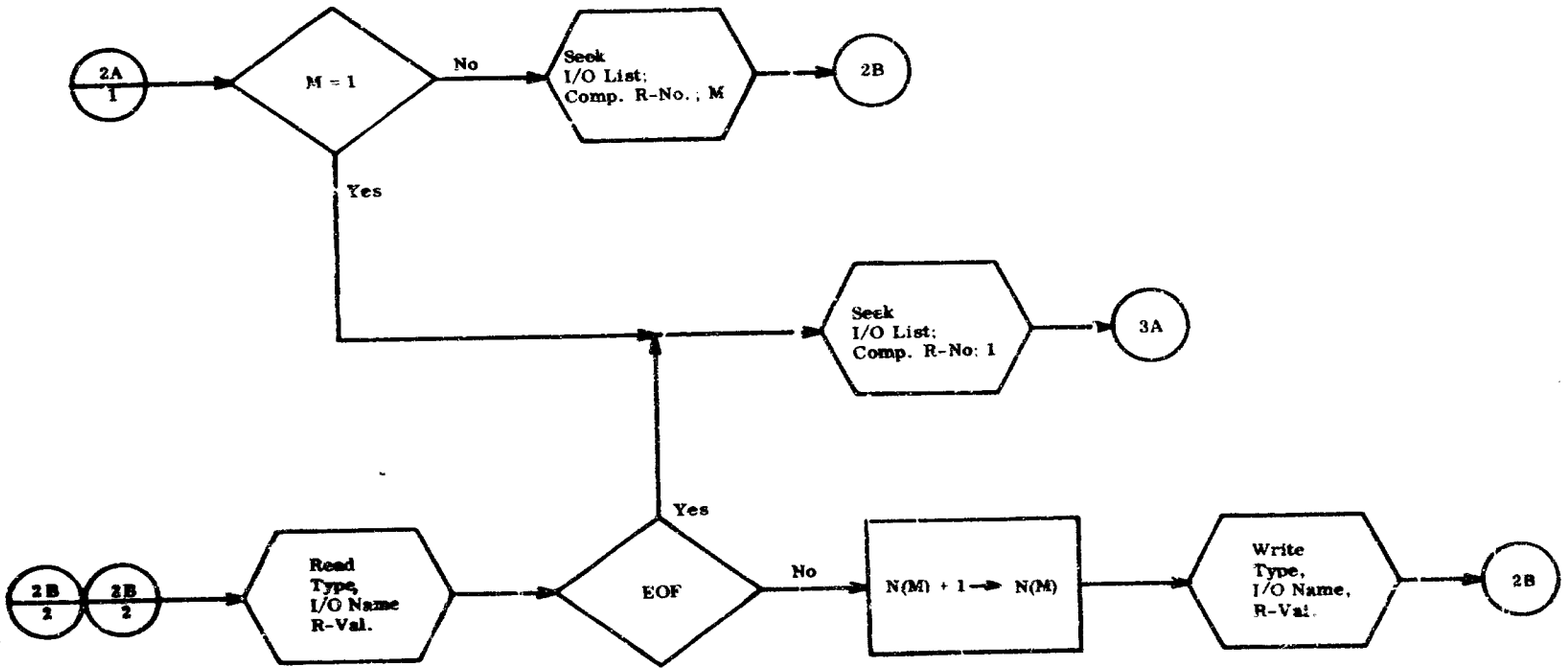
The entire Intermediate List is read for each M^{th} class. The following actions are taken for each component of this list.

(1) Assemble all previous input-output parameters of the M^{th} class from the item list of the component and in each case increment N(M).

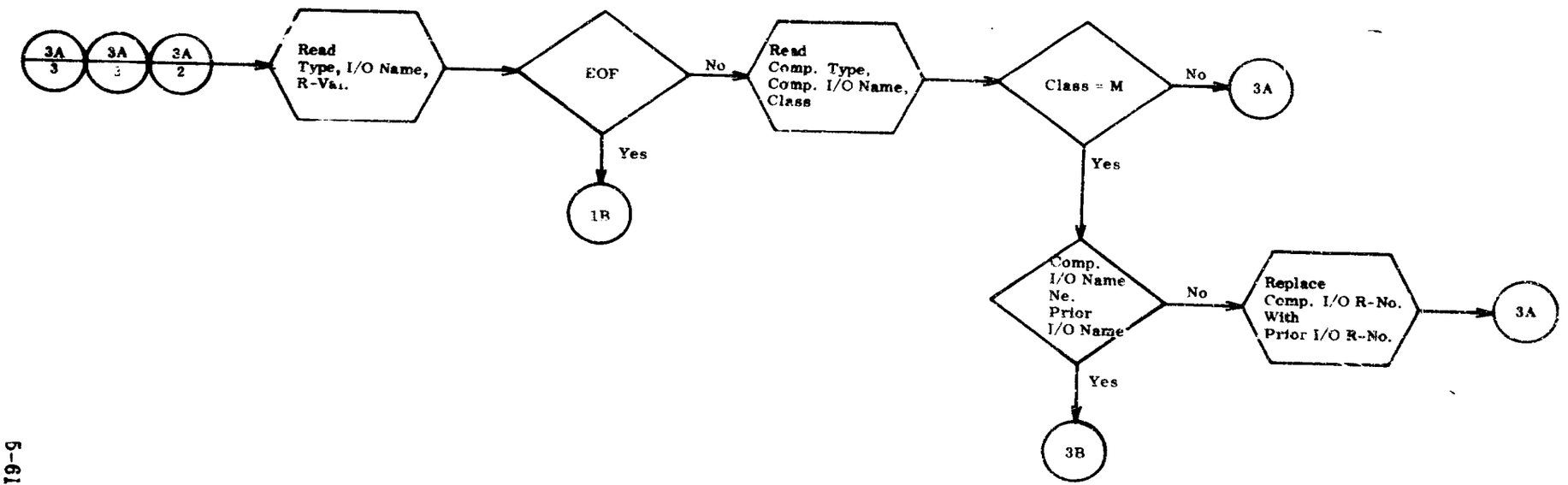
(2) Add all unique input-output parameters of the M^{th} class from the Component I/O List of the Intermediate List Record and in each case increment N(M).

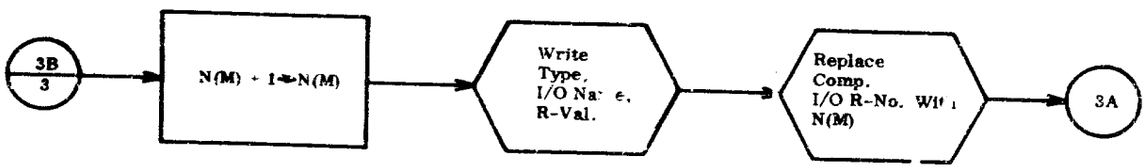


Bind 2 Job
Sheet 1 of 3



Bind 2 Job
Sheet 2 of 3





Bind 2 Job
Sheet 3 of 3

5.2.7.3.7.3 Bind 3

Job Request: BIND-3 (bind 3 list), (bind 3 intermediate list); (bind 3 task list).

5.2.7.3.7.3.1 The Bind 3 job creates a component list.

5.2.7.3.7.3.2 Inputs

(1) BIND 3 LIST, F

ID, A, V
ITEM LIST, F

CLASS, I, 3
I/O LIST, F

TYPE, B, 3
I/O NAME, A, V
R-VALUE, H, V

STATIC TASK LIST, F

TYPE, B, 3
TASK ID. I, 12
NO. FLOATS, I, 3
INPUT LIST, F

FORMAL NAME, A, V
CLASS, I, 3
I/O R-NO., I, 15

COMPONENT LIST, F

COMPONENT NAME, A, V
COMPONENT R-NO., I, 18
COMPONENT I/O LIST, F

TYPE, B, 3
I/O NAME, A, V
CLASS, I, 3
I/O R-NO., I, 15

USAGE LIST, F

NAME, A, V

(2) BIND 3 INTERMEDIATE LIST, F

COMPONENT NAME, A, V
COMPONENT R-NO., I, 18
COMPONENT I/O LIST, F

TYPE, B, 3
I/O NAME, A, V
CLASS, I 3
I/O R-NO., I, 15

5.2.7.3.7.3.3 Results

BIND 3 TASK LIST, F

TYPE, B, 3
TASK ID, I. 12
NO. FLOATS, I, 3
INPUT LIST, F

FORMAL NAME, A, V
CLASS, I, 3
I/O R-NO., I, 15

5.2.7.3.7.3.4 Directories Used. No directories are used unless externally bound.

5.2.7.3.7.3.5 Services Used

(1) Open for Reading.

(2) Close for Reading.

(3) Seek.

(4) Read.

(5) Open for Writing.

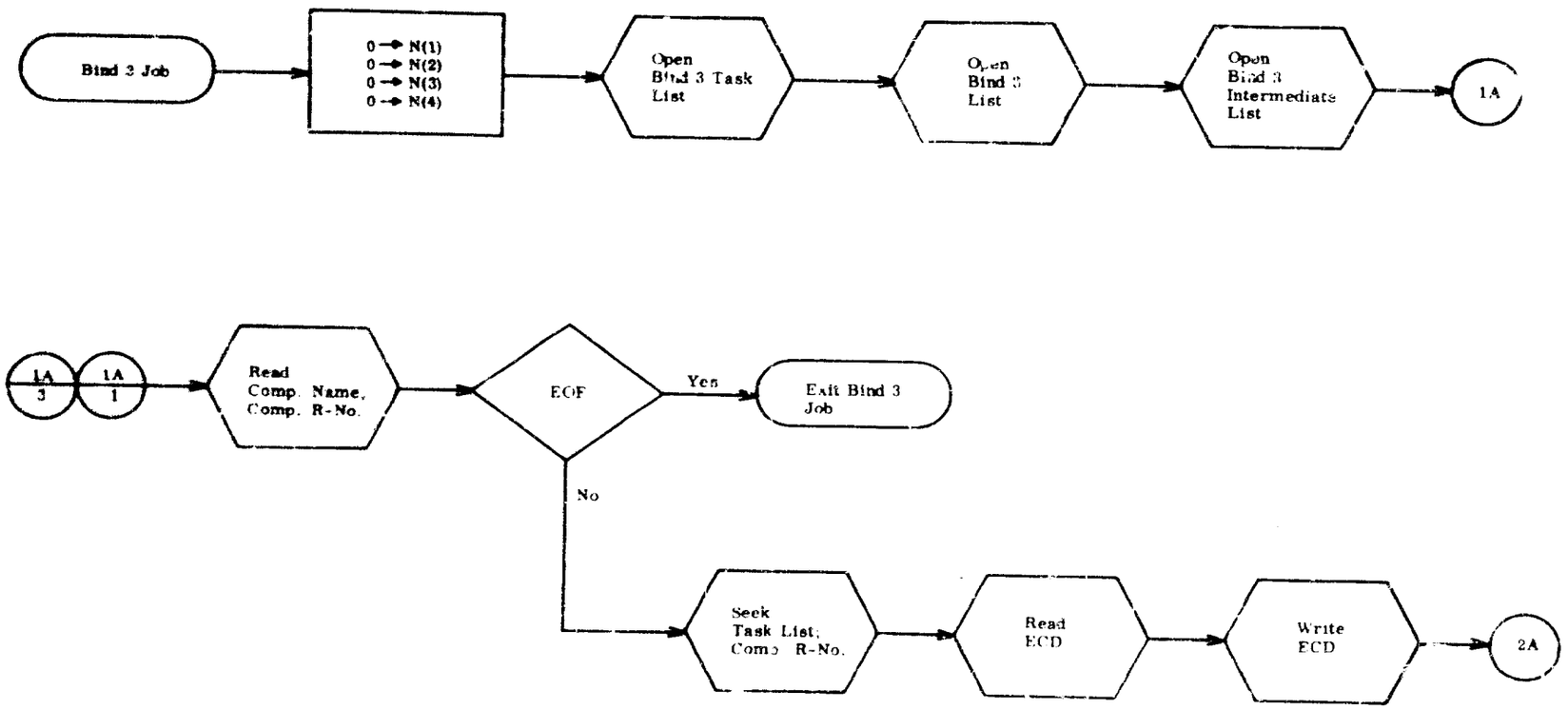
(6) Close for Writing.

(7) Write.

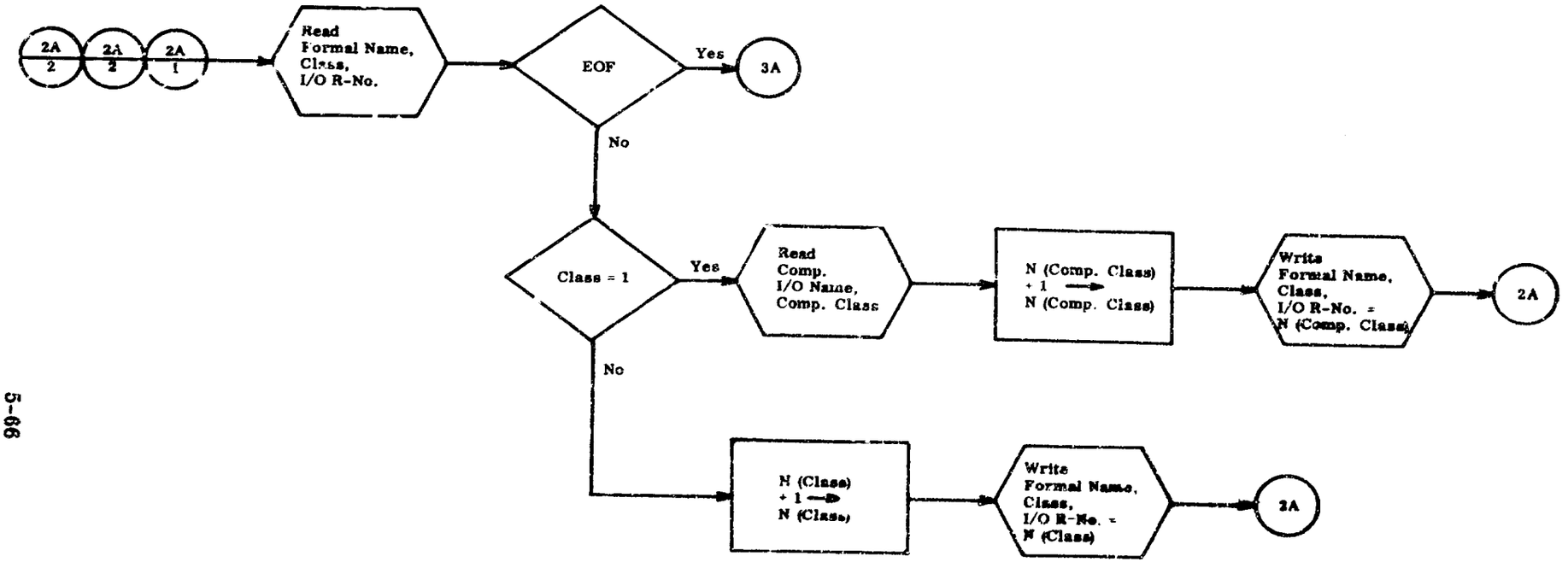
5.2.7.3.7.3.6 Jobs Used. No Job Extensions are used.

5.2.7.3.7.3.7 Method of Operation. Indices N(1), N(2), N(3), and N(4) are initialized to zero. They specify the N^{th} input-output parameter of one of four classes.

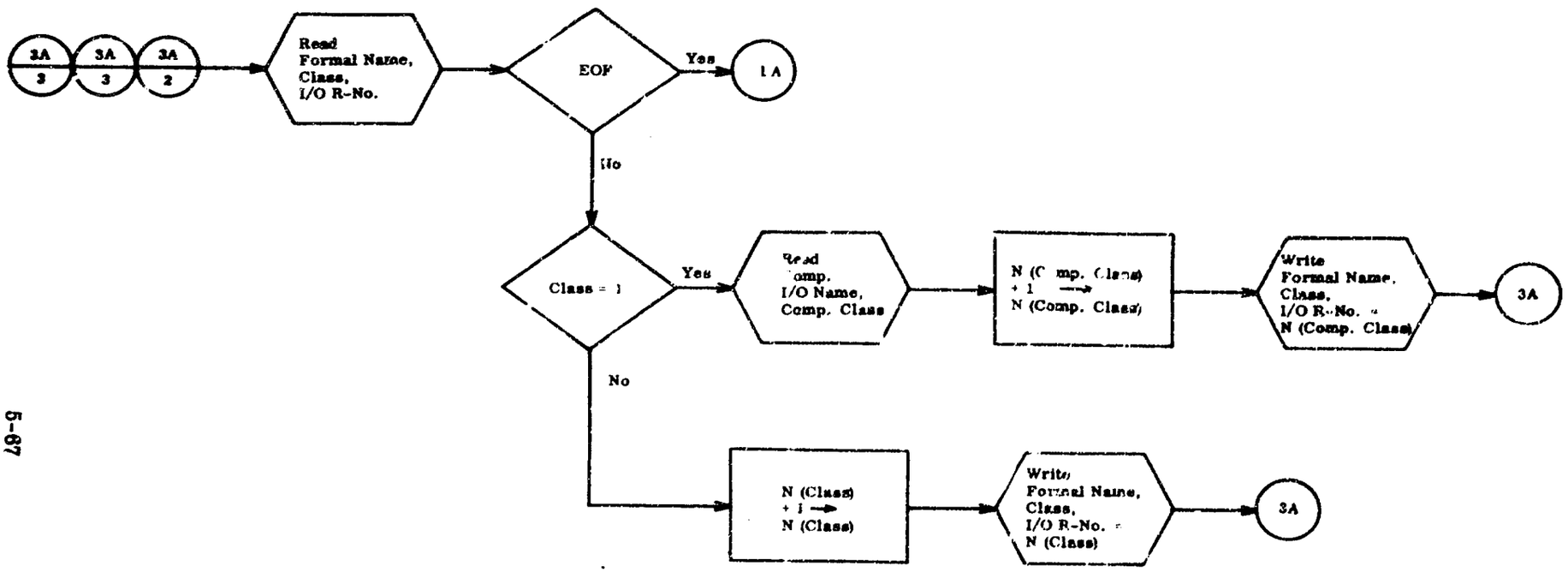
The task lists of all components of the Intermediate List are assembled into one task list. Each sequential input-output parameter is referenced to a hypothetical corresponding item list and index N (Component Class) is incremented.



Bind 3 Job
Sheet 1 of 3



Bind 3 Job
Sheet 2 of 3


3A
3
3A
3
3A
2
Read
Formal Name,
Class,
I/O R-No.
EOF
Yes
1 A
No
Class = 1
Yes
Read
Comp.
I/O Name,
Comp. Class
N (Comp. Class)
+ 1 →
N (Comp. Class)
Write
Formal Name,
Class,
I/O R-No. =
N (Comp. Class)
3A
No
N (Class)
+ 1 →
N (Class)
Write
Formal Name,
Class,
I/O R-No. =
N (Class)
3A

5.3 JOB AND PROGRAM DELETION

Jobs and, therefore, programs may be deleted from the job description library by the Job Deletion job. The deletion may be accompanied by a display of the job description so that the user may scan the usage list to determine which higher level jobs are affected by the deletion. The display may not be required because the deletion is frequently made to accommodate the entry of an updated version of the job. If an updated version involves no changes in the job's indirect input-output parameters, the change can have no effect on higher level jobs which use the changed job as a component.

The job request image for the Job Deletion job is as follows:

Job Request: JOB DELETION (job name).

5.3.1 Functional Description

Job Descriptions may be deleted from the system by the Job Deletion job. Should the job be a terminal or one-task job, the corresponding Program Description would also be deleted.

5.3.2 Inputs

JOB NAME, A, V

5.3.3 Results

No outputs for the job are specified. The deletions to the directories constitute the results.

5.3.4 Directories Used

(1) Program Statement.

(2) Program Description List.

(3) Job Statement.

(4) Job Description List.

5.3.5 Services Used

(1) Open for Writing.

(2) Close for Writing.

(3) Write.

(4) Open for Updating.

(5) Close for Updating.

(6) Seek.

(7) Read.

(8) Replace.

(9) Delete.

(10) Retrieve Item.

(11) Insert Data.

5.3.6 Jobs Used

Auxiliary Deletion.

5.3.7 Method of Operation

The Job Deletion job is entered into the system by the following Job Description:

Job Name: JOB-DELETION

Job Inputs: job name

Job Outputs: none specified

Job Components:

(1) DELETE-NAME: job name, JOB STATEMENT;
job R-No. field.

(2) DELETE: JOB DESCRIPTION LIST: job R-No. field,
job name, PROGRAM STATEMENT,
PROGRAM DESCRIPTION LIST.

The internal Job Description may be represented graphically as shown in Figure 5-5.

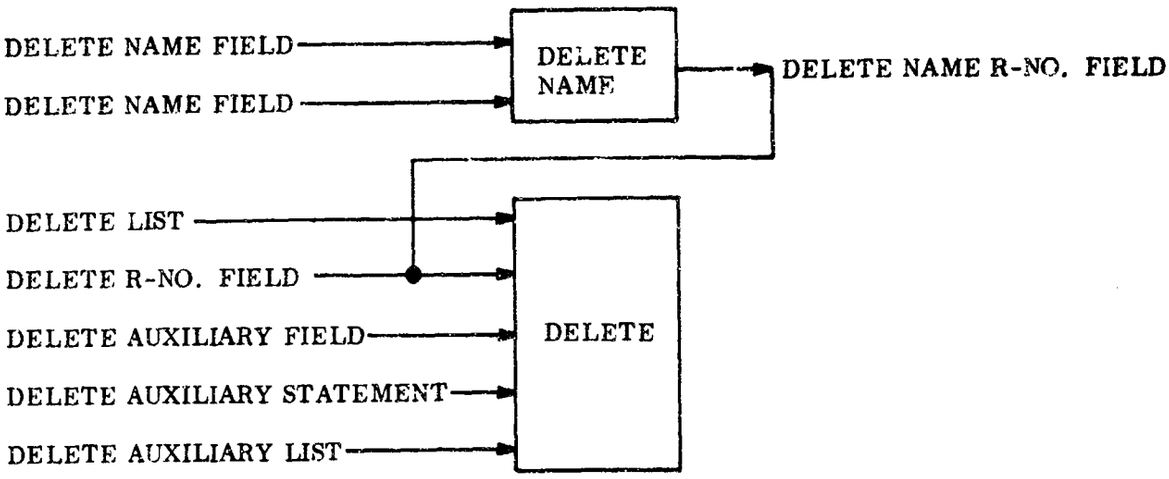


Figure 5-5. Job Deletion Job, Internal Job Description

Delete Name deletes the job name from the Job Name List of the Job Statement. Delete deletes the specified record from the Job Description List and, if the job is a one-task job, it requests a Job Extension to the Auxiliary Deletion Job.

5.3.7.1 Delete Name

Job Request: DELETE-NAME (delete name field), (delete name statement); (delete name R-No. field).

5.3.7.1.1 Functional Description. This job updates the Delete Name Statement by deleting Delete Name Field from the Name List. It likewise writes the corresponding R-number field.

5.3.7.1.2 Inputs

(1) DELETE NAME FIELD, A, V

(2) DELETE NAME STATEMENT, S, 3

NULL LIST, F

NULL R-NO., I, 18

LAST R-NO., I, 18
NAME LIST, F, ORDERED (1)

NAME, A, V
NAME R-NO., I, 18

5.3.7.1.3 Outputs

DELETE NAME R-NO. FIELD, I, 18

5.3.7.1.4 Directories Used. No directories are used unless externally bound.

5.3.7.1.5 Services Used

(1) Open for Writing.

(2) Close for Writing.

(3) Write.

(4) Open for Updating.

(5) Close for Updating.

(6) Seek.

(7) Read.

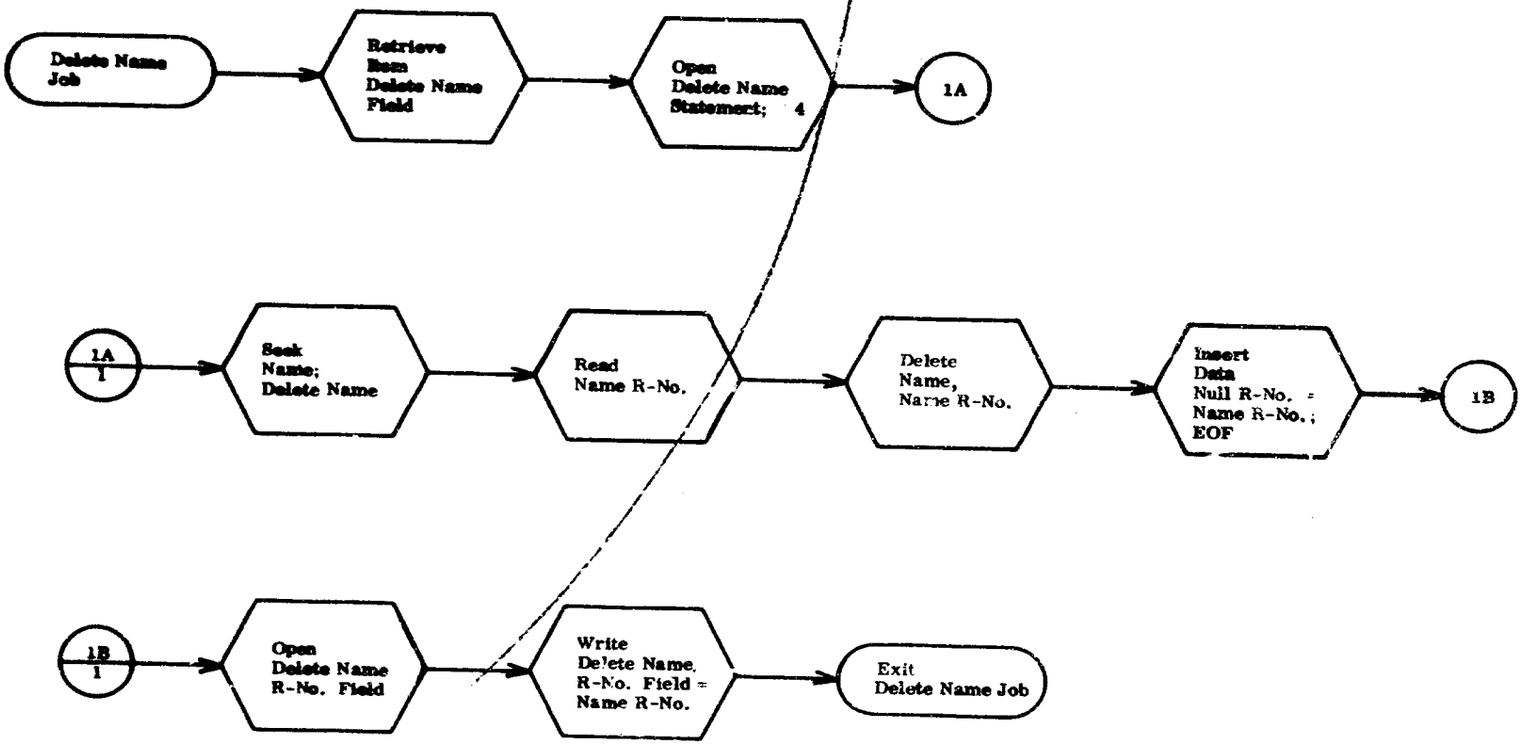
(8) Delete.

(9) Retrieve Item.

(10) Insert Data.

5.3.7.1.6 Job Used. No Job Extensions are used.

5.3.7.1.7 Method of Operation. After the Delete Name Field is read, this name is deleted from the Name List of the Delete Name Statement. The corresponding R-number is inserted into the Null List and is also written.



Delete Name Job
Sheet 1 of 1

5.3.7.2 Delete

Job Request: DELETE (delete list), (delete R-No. field),
(delete auxiliary field), (delete auxiliary statement),
(delete auxiliary list).

5.3.7.2.1 Functional Description. This job updates the Delete List by deleting the specified record from the Delete List. Should this record constitute a terminal item, the corresponding record of the Delete Auxiliary List would also be deleted.

5.3.7.2.2 Inputs

(1) DELETE LIST, F

ID, A, V
ITEM LIST, F

CLASS, I, 3
I/O LIST, F

TYPE, B, 3
I/O NAME, A, V
R-VALUE, H, V

STATIC TASK LIST, F

TYPE, B, 3
TASK ID, I, 12
NO. FLOATS, I, 3
INPUT LIST, F

FORMAL NAME, A, V
CLASS, I, 3
I/O R-NO., I, 15

COMPONENT LIST, F

COMPONENT NAME, A, V
COMPONENT R-NO., I, 18
COMPONENT I/O LIST, F

TYPE, B, 3
I/O NAME, A, V
CLASS, I, 3
I/O R-NO., I, 15

USAGE LIST, F

NAME, A, V

(2) DELETE R-NO. FIELD, I, 18

(3) DELETE AUXILIARY FIELD, A, V

(4) DELETE AUXILIARY STATEMENT, S, 3

NULL LIST, F

NULL R-NO., I, 18

LAST LIST, F, ORDERED (1)

NAME, A, V
NAME R-NO., I, 18

(5) DELETE AUXILIARY LIST, F

BINDING LIST, F

ITEM LIST, F

RESERVED, B, 10
SRL - ACCESS, ∅, 1
SRL - MODIFICATION, ∅, 1
RESERVED, B, 2
ITEM TYPE, B, 6
OPTION CODE, B, 1
ITEM SIZE, I, 11

TERM LIST, F

TERM NAME, A, V
UNITS, B, 6
RESERVED, I, 18

5.3.7.2.3 Results. No outputs for the job are specified. The deletions to the Delete List and the Delete Auxiliary List constitute the results.

5.3.7.2.4 Directories Used. No directories are used unless externally bound.

5.3.7.2.5 Services Used

(1) Open for Writing.

(2) Close for Writing.

(3) Write.

(4) Open for Updating.

(5) Close for Updating.

(6) Seek.

(7) Read.

(8) Replace.

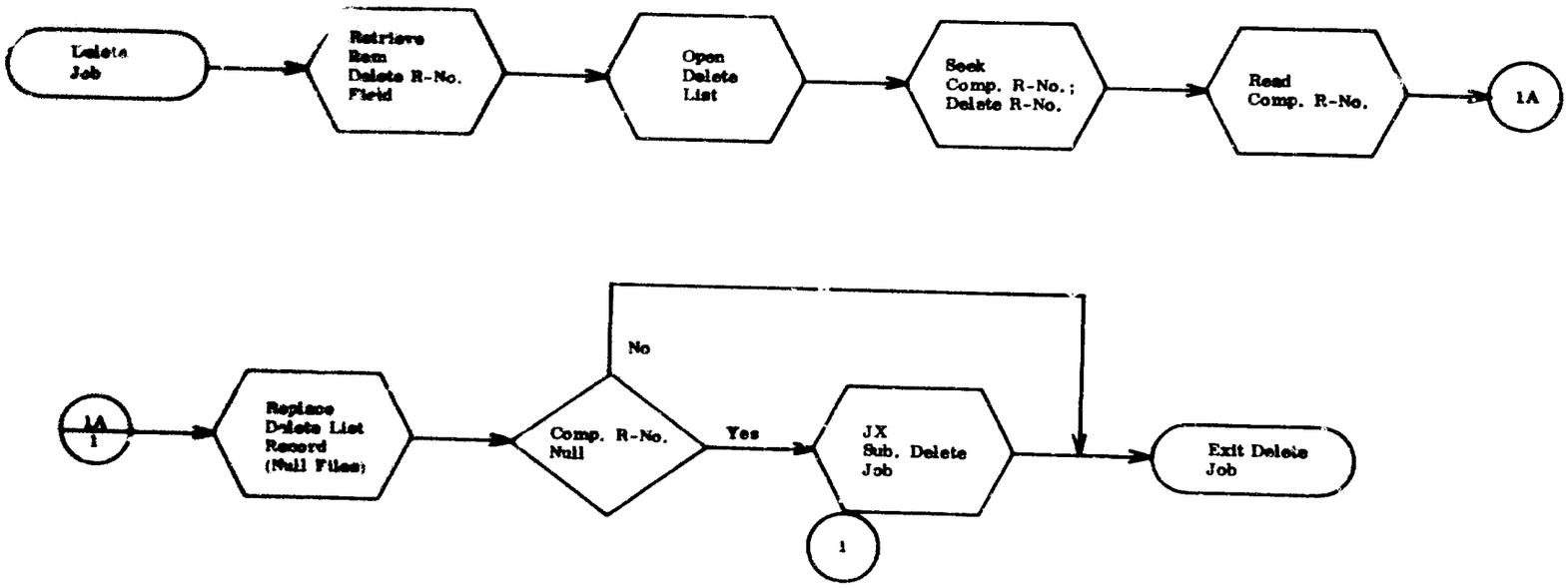
(9) Delete.

(10) Retrieve Item.

(11) Insert Data.

5.3.7.2.6 Jobs Used. Auxiliary Deletion.

5.3.7.2.7 Method of Operation. The subsumed files of the specified record of Delete List are replaced by null items. If this record constitutes a terminal item, a Job Extension to the Auxiliary Deletion job is requested.



Note 1 Delete Aux. List
Delete Statement
Delete Field

5.3.7.3 Auxiliary Deletion

Job Request: AUXILIARY-DELETION (auxiliary deletion field),
(auxiliary deletion statement),
(auxiliary deletion list)

5.3.7.3.1 Functional Description. This job updates both the Auxiliary Deletion Statement and the Auxiliary Deletion List by deleting the Auxiliary Deletion Field from the Name List and deleting the specified record from the Auxiliary Deletion List.

5.3.7.3.2 Inputs

(1) AUXILIARY DELETION FIELD, A, V

(2) AUXILIARY DELETION STATEMENT, S, 3

NULL LIST, F

NULL R-NO., I, 18

LAST R-NO., I, 18
NAME LIST, F, ORDERED (1)

NAME, A, V
NAME R-NO., I, 18

(3) AUXILIARY DELETION LIST, F

BINDING LIST, F

ITEM LIST, F

RESERVED, B, 10
SRL - ACCESS, Ø, 1
SRL - MODIFICATION, Ø, 1
RESERVED, B, 2
ITEM TYPE, B, 6
OPTION CODE, B, 1
ITEM SIZE, I, 11

TERM LIST, F

TERM NAME, A, V
UNITS, B, 6
RESERVED, I, 18

5.3.7.3.3 Results. No outputs for the job are specified. The deletions from the Auxiliary Deletion Statement and the Auxiliary Deletion List constitute the results.

5.3.7.3.4 Directories Used. No directories are used unless externally bound.

5.3.7.3.5 Services Used

(1) Open for Writing.

(2) Close for Writing.

(3) Write.

(4) Open for Updating.

(5) Close for Updating.

(6) Seek.

(7) Read.

(8) Replace.

(9) Delete.

(10) Retrieve Item.

(11) Insert Data.

5.3.7.3.6 Jobs Used. No Job Extensions are used.

5.3.7.3.7 Method of Operation. The Auxiliary Deletion job is entered into the system by the following Job Description:

Job Name: AUXILIARY-DELETION

Job Inputs: auxiliary deletion field, auxiliary deletion statement, auxiliary deletion list.

Job Outputs: none specified.

Job Components:

(1) DELETE-NAME: auxiliary deletion field, auxiliary deletion statement; auxiliary deletion R-No. field.

(2) AUXILIARY-DELETE: auxiliary deletion list, auxiliary deletion R-No. field.

The internal Job Description may be represented graphically as shown in Figure 5-6.

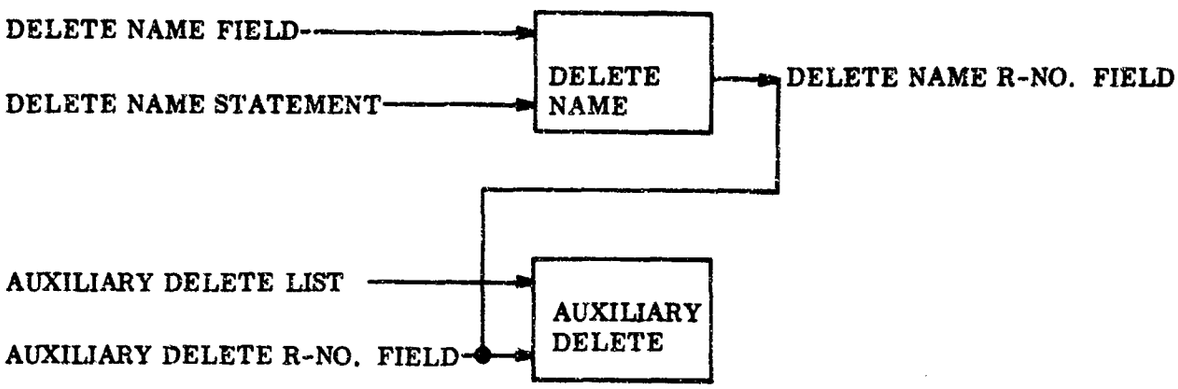


Figure 5-6. Auxiliary Deletion Job, Internal Job Description

Delete Name deletes Delete Name Field from the Name List of the Delete Name Statement. Auxiliary Delete deletes the specified record from the Auxiliary Delete List.

5.3.7.3.7.1 Auxiliary Delete

Job Request: AUXILIARY-DELETE (auxiliary delete list),
(auxiliary delete R-No. field).

5.3.7.3.7.1.1 Functional Description. This job updates the Auxiliary Delete List by deleting the specified record from that list.

5.3.7.3.7.1.2 Inputs

(1) AUXILIARY DELETE LIST, F

BINDING LIST, F

ITEM LIST, F

RESERVED, B, 10
SRL - ACCESS, ø, 1
SRL - MODIFICATION, ø, 1
RESERVED, B, 2
ITEM TYPE, B, 6
OPTION CODE, B, 1
ITEM SIZE, I, 11

TERM LIST, F

TERM NAME, A, V
UNITS, B, 6
RESERVED, I, 18

(2) AUXILIARY DELETE R-NO., I, 18

5.3.7.3.7.1.3 Results. No outputs for this job are specified. The deletions to the Auxiliary Delete List constitute the results.

5.3.7.3.7.1.4 Directories Used. No directories are used unless externally bound.

5.3.7.3.7.1.5 Services Used

(1) Open for Updating.

(2) Close for Updating.

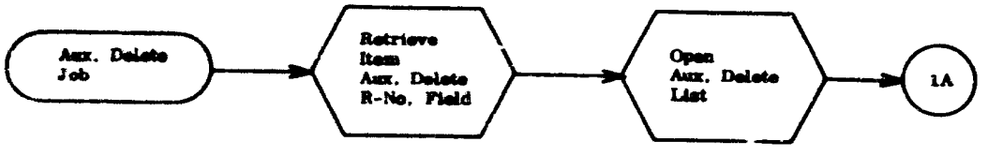
(3) Seek.

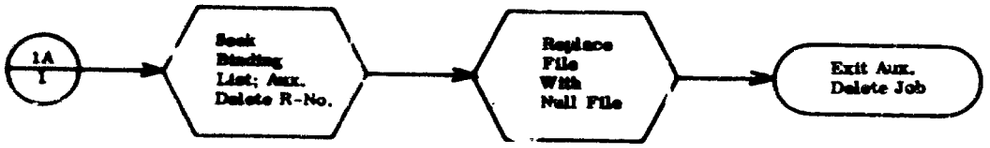
(4) Replace.

(5) Retrieve Item.

5.3.7.3.7.1.6 Jobs Used. No Job Extensions are used.

5.3.7.3.7.1.7 Method of Operation. The subsumed file of the specified record of the Auxiliary Delete List is replaced by a null item.





Aux. Delete Job
Sheet 1 of 1

5.4 DISPLAY JOB DESCRIPTION

The job description of any job in the library may be displayed through use of the Display Job Description job. The entire description or a part of it may be displayed. This gives the user the means of uncovering the descriptive information he might need to execute the job or to use it as a component in another job.

The job request image for the Display Job Description job is as follows:

Job Request: DISPLAY-JOB-DESCRIPTION (display job description name),
(display job description format).

5.4.1 Functional Description

Any Job Description of any job within the system may be displayed either in part or in its entirety. The optional format sentence provides the user with flexibility in his specification of display characteristics. This format specification is identical to that of the Display job (Section VII, Utility Jobs).

5.4.2 Inputs

(1) DISPLAY JOB DESCRIPTION NAME, A, V

(2) DISPLAY JOB DESCRIPTION FORMAT, A, V

5.4.3 Results

No outputs for this job are specified. The display constitutes the results.

5.4.4 Directories Used

(1) Job Statement.

(2) Job Description List.

5.4.5 Services Used

(1) Open for Reading.

(2) Close for Reading.

(3) Seek.

(4) Read.

(5) Open for Writing.

(6) Close for Writing.

(7) Write.

(8) Retrieve Item.

5.4.6 Jobs Used

Display (Section VII, Utility Jobs)

5.4.7 Method of Operation

The Display Job Description job is entered into the system by the following one-component Job Description:

Job Name: DISPLAY-JOB-DESCRIPTION

Job Inputs: display job description name, display job description format.

Job Outputs: none specified.

Job Components: DISPLAY JD: display jd name,
JOB STATEMENT; JOB DESCRIPTION LIST,
display jd format; display jd statement.

The job name is read and the specified record of the Job Description List is initialized. Each job input and job output is read and written. From this and the corresponding component list, the following item is formed and written:

```
DISPLAY J. D. STATEMENT, S, 5

    NAME, A, V
    ID, A, V
    INPUT LIST, F

        I/O NAME, A, V

    OUTPUT LIST, F

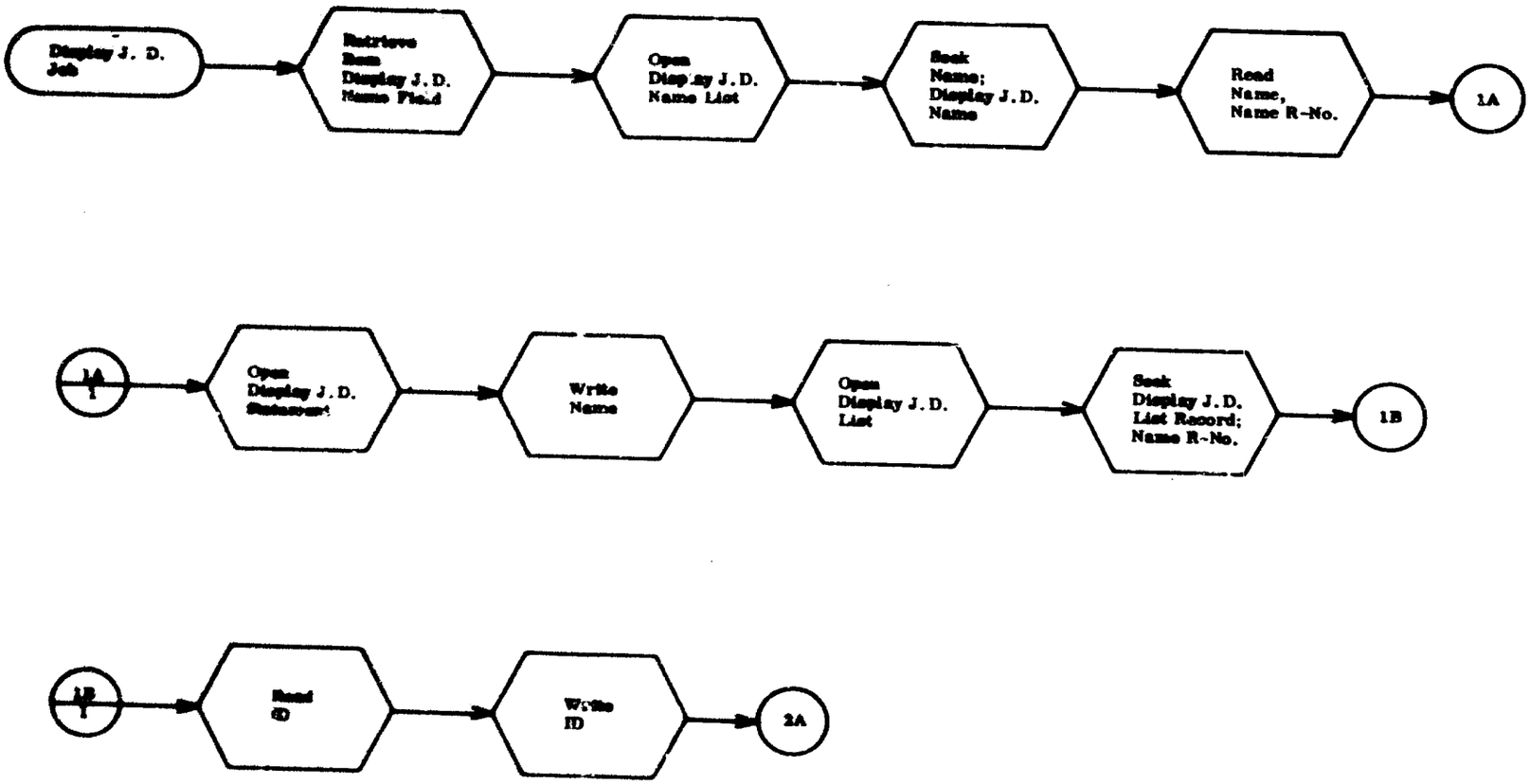
        I/O NAME, A, V

    COMPONENT LIST, F
```

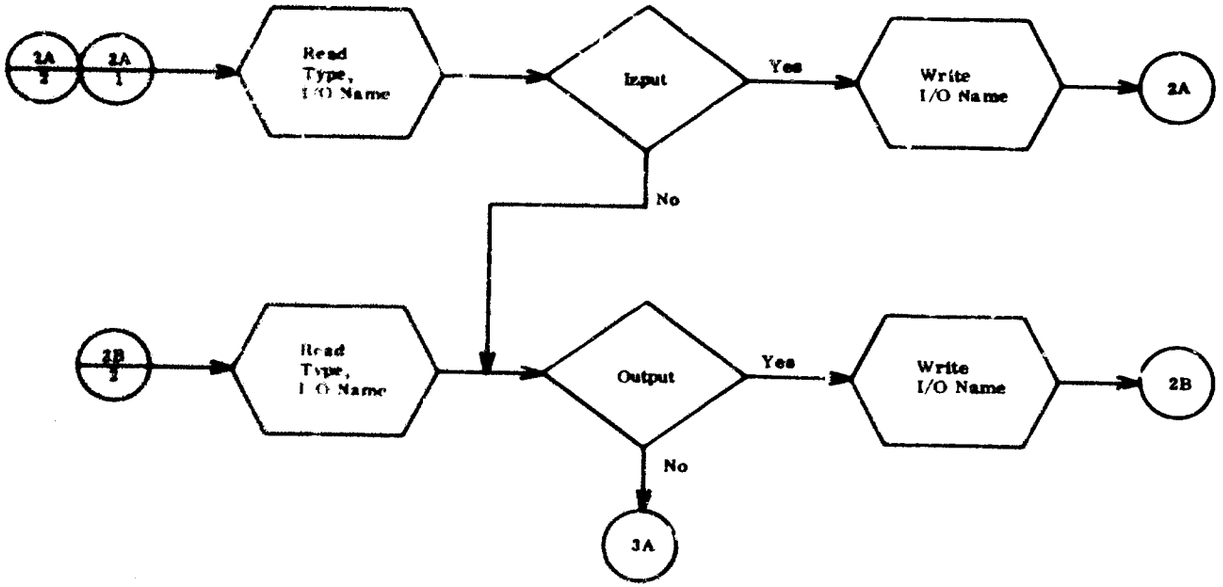
COMPONENT NAME, A, V
COMPONENT I/O LIST, F

TYPE SYMBOL, A, V
CLASS NAME, A, V
I/O NAME, A, V

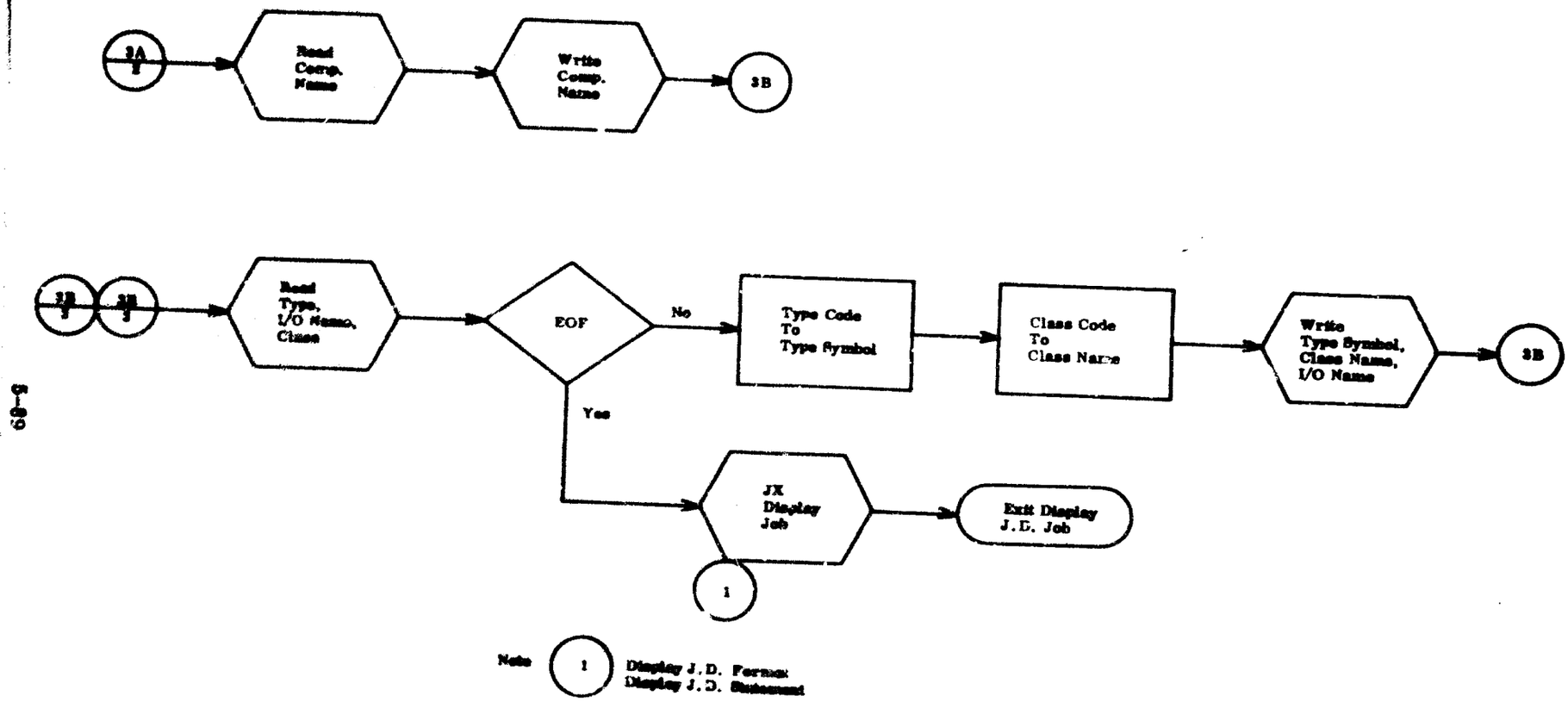
A Job Extension is requested to execute the Display job with the above item.



Display J. D. Job
Sheet 1 of 3



Display J. D. Job
Sheet 2 of 3


3A
Read Comp. Name
Write Comp. Name
3B
3B
3B
Read Type, I/O Name, Class
EOF
No
Type Code To Type Symbol
Class Code To Class Name
Write Type Symbol, Class Name, I/O Name
3B
Yes
JX Display Job
Exit Display J.D. Job
1
Note
1
Display J.D. Format
Display J.D. Statement

SECTION VI. DATA POOL MAINTENANCE JOBS

Data Pool Maintenance is defined as the support required to keep the data and directories in a state of efficiency or validity. Figure 6-1 shows the entire data pool as a statement subsuming four items.

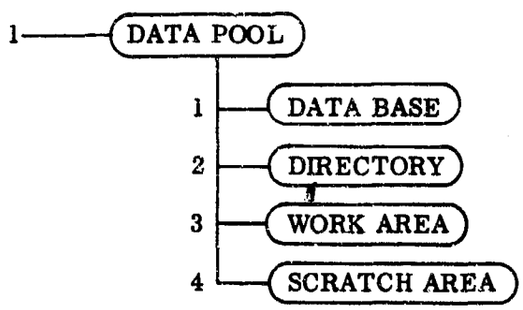


Figure 6-1. The Data Pool

Maintenance of the Data Base and Directory is the responsibility of the Data Administrator. The tools provided for his use are a set of system jobs collectively known as Data Pool Maintenance jobs.

Data stored in the Scratch Area is destroyed when a job request has been fully executed. Data Pool Maintenance jobs do not operate on this data unless a user job is defined to include a system Data Pool Maintenance job as a task within the user job.

Data stored within the Work Area can be maintained by system Data Pool Maintenance jobs or by user jobs.

The Data Pool Maintenance jobs to be described in this section are system jobs primarily concerned with Data Base data and the directories pertaining to Data Base data. The jobs are categorized as:

(1) Item Definition Manipulations (Paragraph 6.1),

(2) Data Manipulations (Paragraph 6.2),

(3) Indexing (Paragraph 6.3),

(4) Linkage (Paragraph 6.4),

(5) Other Maintenance Jobs (Paragraph 6.5).

6.1 ITEM DEFINITION MANIPULATION

The logical structure of the data pool is altered through item definition manipulations. The elements of the directory which describe the relationship of data items are altered when maintenance jobs of this class are executed.

Addition and deletion of definitions are provided for directly; modifications are accomplished through a deletion followed by an addition. Addition is accomplished with Define Item, and deletions are accomplished with Delete Definition, Delete Node, or Renovate Item. Delete Node is the inverse of Define Item. Delete Definition leaves the node in the structure as a null node, and Renovate Item squeezes out null nodes. The reason for deletion will indicate which job is the best one to use.

Item Definition manipulations are primarily concerned with maintenance of the Term Encoding Table, the Item List, and the Term List.

6.1.1 Define Item

Job Request: DEFINE-ITEM (node specification), (item image).

6.1.1.1 Functional Description. This job adds vocabulary and structure to the directories of AIMS in preparation for the entry of Reliability Central Data Base data.

EXAMPLE:

```
DEFINE-ITEM          TO PART FILE,
                     STRESS TEST', F
                         STRESS PARAMETER, A, V
                         TEST POINT, F
                             SEVERITY, E
                             TEST RESULTS, F
                                 NO. SAMPLES, I, 6
                                 TEST HRS., I, 4
                                 NO. FAILURES, I, 5
```

The Part file has been previously defined as:

```
PART FILE, F
    PART NAME, A, V
        .
        .
        .
    PART NUMBER, A, V
```

In the above example, the requestor wants to define a Stress Test file to be subsumed under the Part file as follows:

```
PART FILE, F
    PART NAME, A, V
        .
        .
        .
    PART NUMBER, A, V
STRESS TEST, F
    STRESS PARAMETER, I
    TEST POINT, F
```

SEVERITY, I, 6
TEST RESULTS, F

NO. SAMPLES, I, 6
TEST HRS., I, 4
NO. FAILURES, I, 2

When the Define Item job has run to completion, the Stress Test file and all of its subitems are fully defined to the system and data may now be added.

6.1.1.2 Inputs. The inputs to the Define Item job are the node specification and the item image.

(1) Node Specification. The node which is to be defined by the new item structure is specified by a phrase. To add the new structure to the end of an existing file or statement, the phrase

TO (item name)

is used. The item name must identify an existing file or statement. The phrase specifies that a new node is to be added to the end of the statement or record and the new structure is to be placed at that node. To place the new structure at a node which is currently null, the phrase

AT (item name)

is used. The item name must identify an existing file, statement, field or null node.

A term name used in any of the above phrases is the Term Encoding Table name of a previously defined item. If the TET has more than one ICC cross-referenced to the given term name, a qualifier must be included in the phrase. For example: TO PART FILE IN DIODE FILE.

(2) Item Image. The item image is an item definition in a form suitable for computer input. The item may be a single terminal item such as a field or null node, or a nonterminal item such as a file or a statement with its subitems.

If the item image includes a file or a statement with no subsumed items, the item name will be retained, but the item type will be changed to a null node.

6.1.1.3 Results. There are no outputs produced by this job. The directories are updated to include the new item definition.

6.1.1.4 Directories Used

(1) Term Encoding Table.

(2) Item List.

(3) Term List.

(4) Segment Name List.

6.1.1.5 Services Used

(1) Locate IL Entry.

(2) Open for Reading.

(3) Seek.

(4) Read.

(5) Close for Reading.

(6) Open for Writing.

(7) Write.

(8) Close for Writing.

(9) Open for Updating.

(10) Insert.

(11) Replace.

(12) Close for Updating.

(13) Retrieve Item.

(14) Term Name to ICC Translation.

6.1.1.6 Jobs Used

(1) Sort.

(2) Item Image Translation.

6.1.1.7 Method of Operation. A is the program's name (formal) for the alphanumeric string which includes the word TO, AT, or BEFORE followed by the term name of the node, with whatever qualifiers are necessary for uniqueness. B is the program's name (formal) for the item image after it has been translated into internal format and validated.

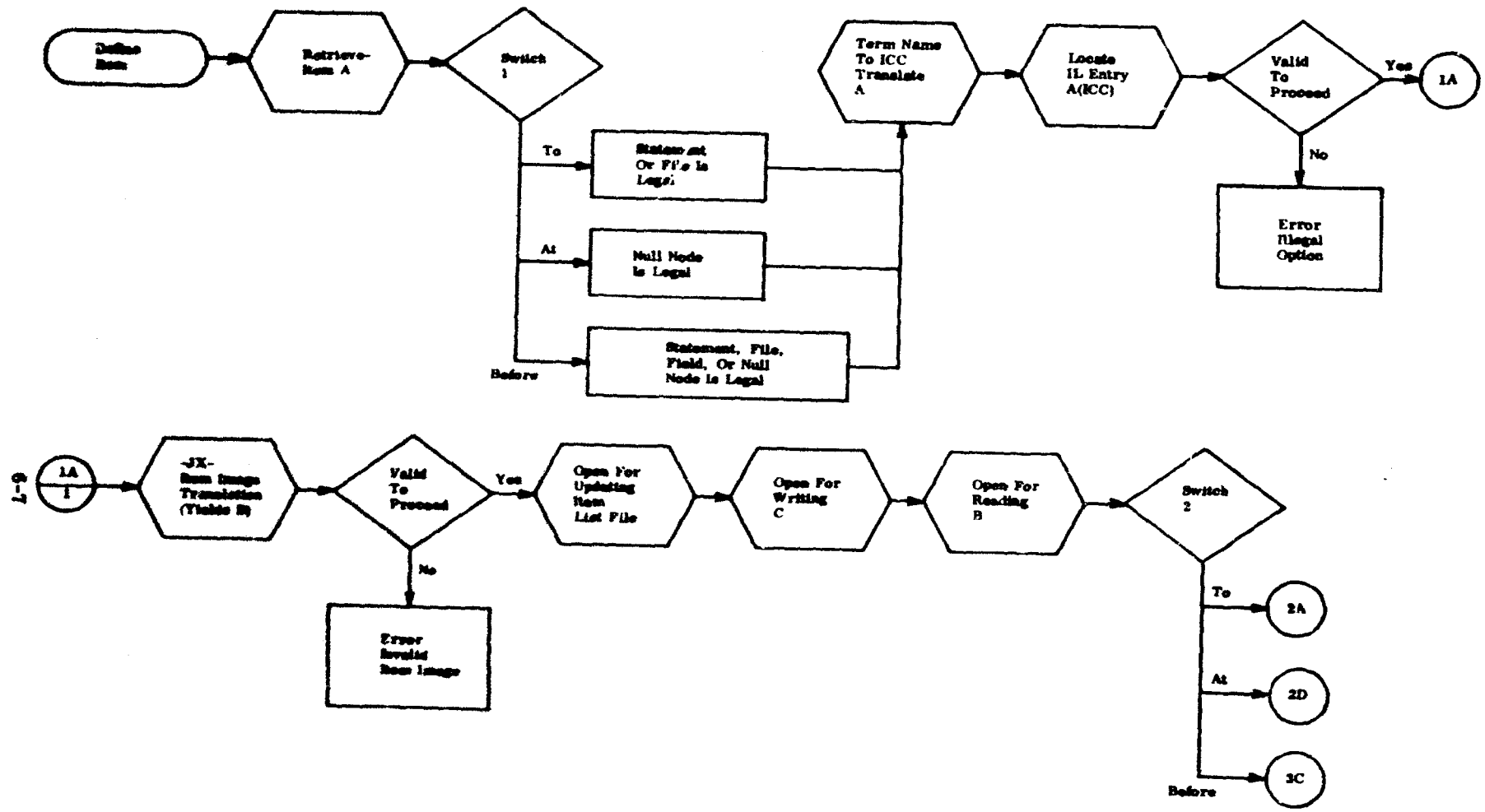
(1) Page 1 is concerned with the translation of the job inputs into internal coded form and the validation of these inputs. Then, preparation is made to read file B, update the Item List with the new items, and write a scratch file C, which contains the information necessary for subsequent Term List and Term Encoding Table updating.

(2) Connectors 2A, 2D, and 3C handle the Item List updating for the TO, AT, and BEFORE options. File C is read, and the new items are inserted into the Item List at the proper position. When EOF is reached on file C, all paths join at 3D and 3E.

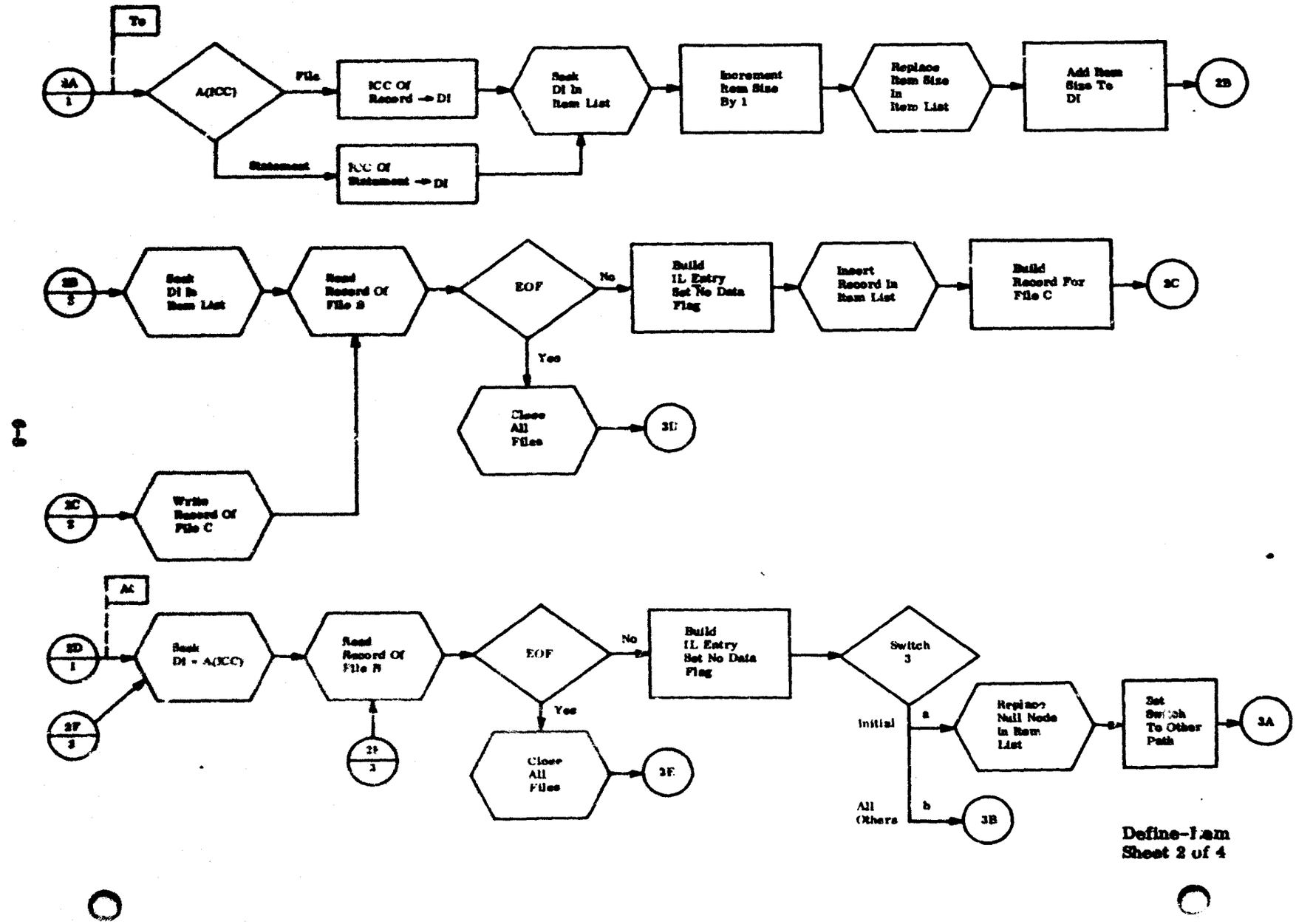
(3) File C is read again, and this time the Term List is updated with term names and units fields from the original item image. At EOF, control goes to connector 4A.

(4) At connector 4A, file C is sorted by term name. Then file C is read, and the Term Encoding Table is updated by inserting the records of file C into their proper positions in the TET. Records with blank term names are discarded in TET updating, but not in Term List updating. The linkage mechanism between the Item List and Term List requires a one-to-one correspondence between the records of these two files.

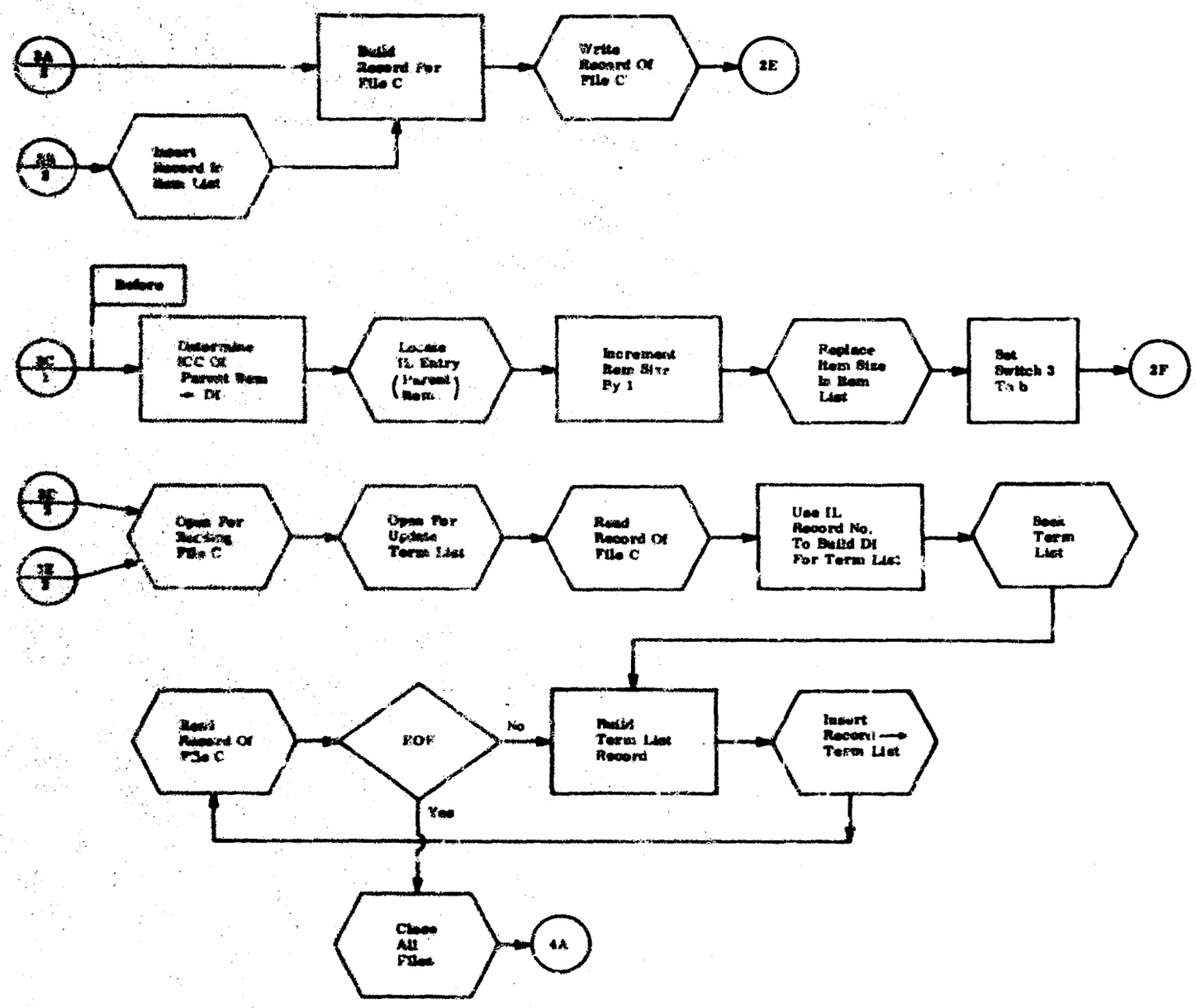
(5) At EOF on file C, the job is terminated.



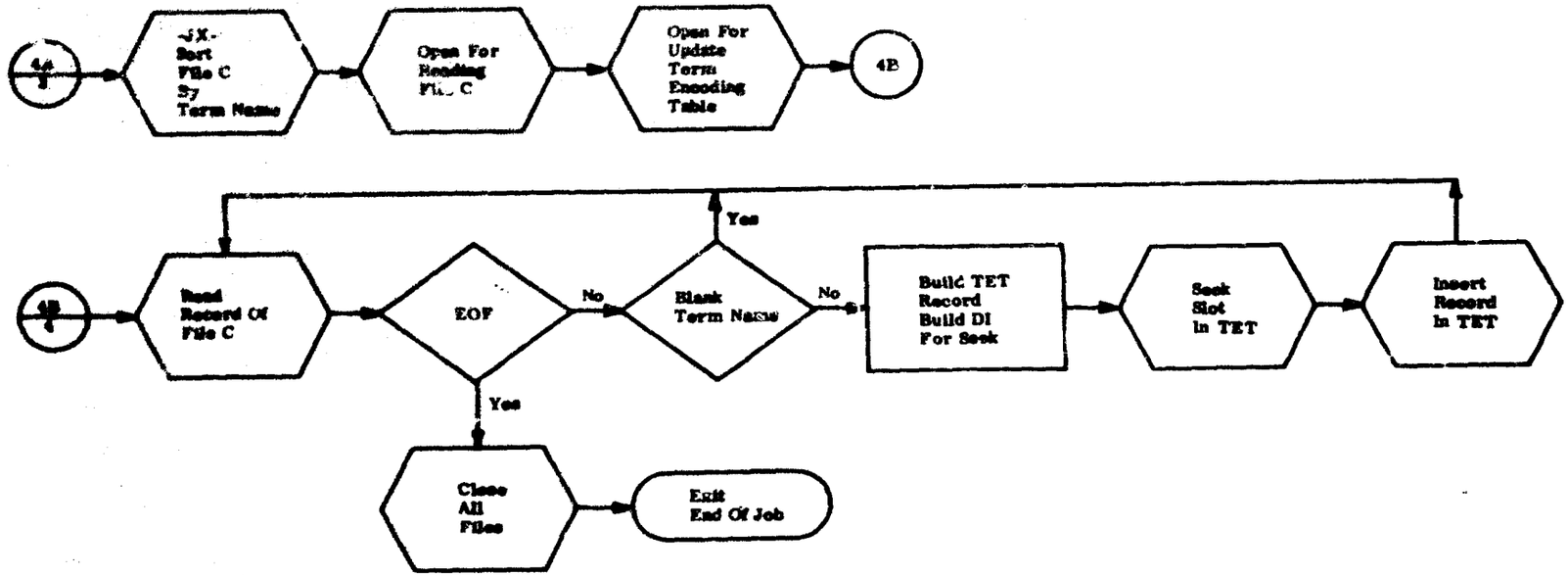
Define-Item
Sheet 1 of 4



Define-Item
Sheet 2 of 4



Define-Item
Sheet 3 of 4



Define--Item
Sheet 4 of 4

6.1.2 Delete Definition

Job Request: DELETE-DEFINITION (item name).

6.1.2.1 Functional Description. This job redefines the item names in (item name) as a null node. The name remains in the TET, but the Item List entry is converted to a null node. If (item name) is a statement or file (record is illegal), all subsumed nodes are removed from the Item List and TET.

If this job is executed at a time when the Data Base has data filed under this structure, the data will be deleted from the Data Base.

6.1.2.2 Inputs. The only input is an item name. This is a term name (qualified) for the item which is to be redefined as a null node.

6.1.2.3 Results. The directories are updated as a result of this job, but there are no job outputs.

6.1.2.4 Directories Used

(1) Term Encoding Table.

(2) Item List.

(3) Term List.

(4) Segment Name List.

6.1.2.5 Services Used

(1) Locate IL Entry.

(2) Open for Reading.

(3) Read.

(4) Close for Reading.

(5) Open for Writing.

(6) Write.

(7) Close for Writing.

(8) Open for Update.

(9) Seek.

(10) Replace.

(11) Delete.

(12) Close for Update.

(13) Retrieve Item.

(14) Term Name to ICC Translation.

6.1.2.6 Jobs Used.

(1) Data Delete.

(2) Sort.

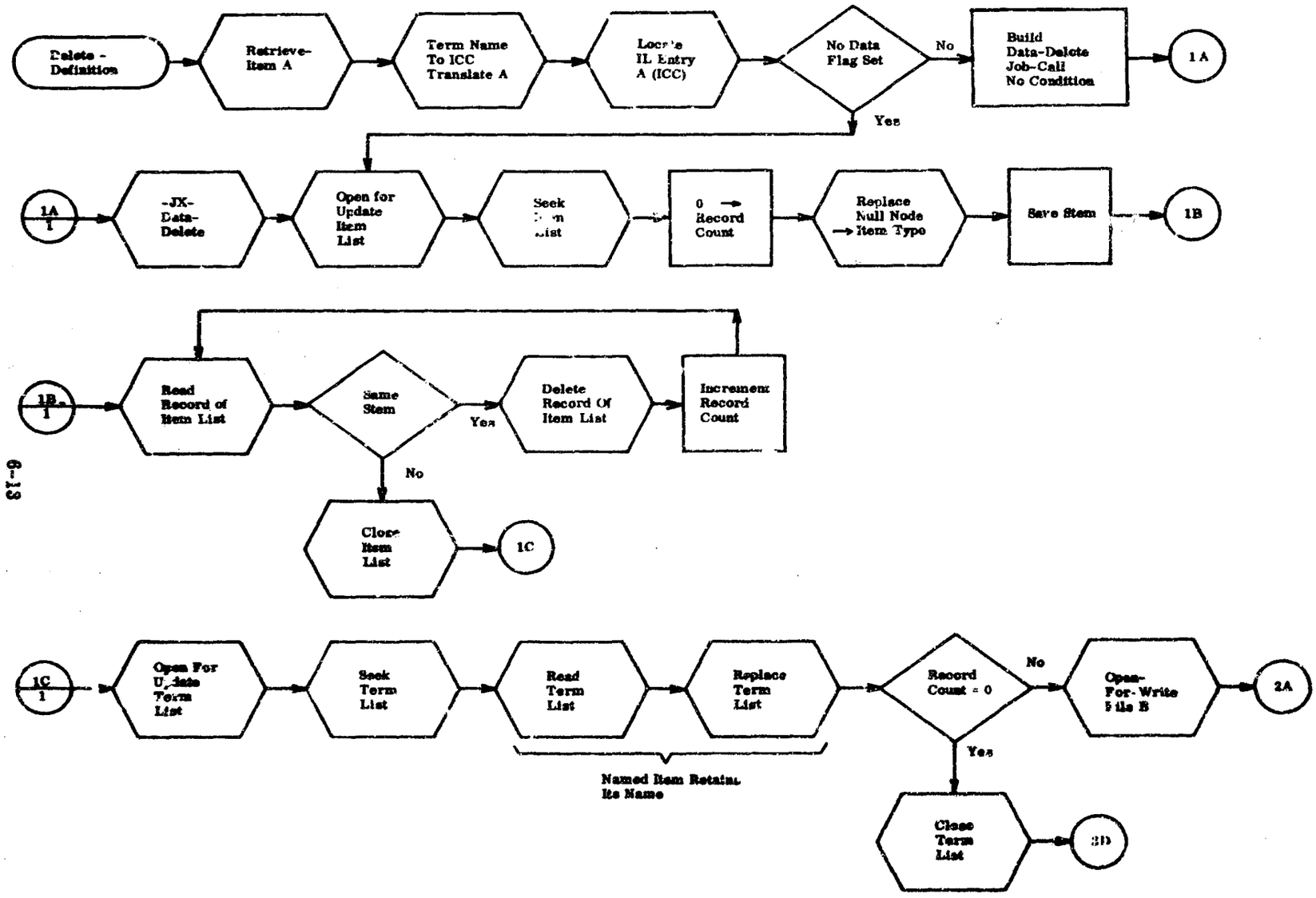
6.1.2.7 Method of Operation. A is the program's name (formal) for an alphanumeric string which contains the term name for the item whose definition is to be deleted.

(1) After translating the term name to an ICC, the Item List entry is retrieved. If the data exists, it is deleted. The record number of this IL entry is saved so that it can be used in deleting the proper records of the Item List.

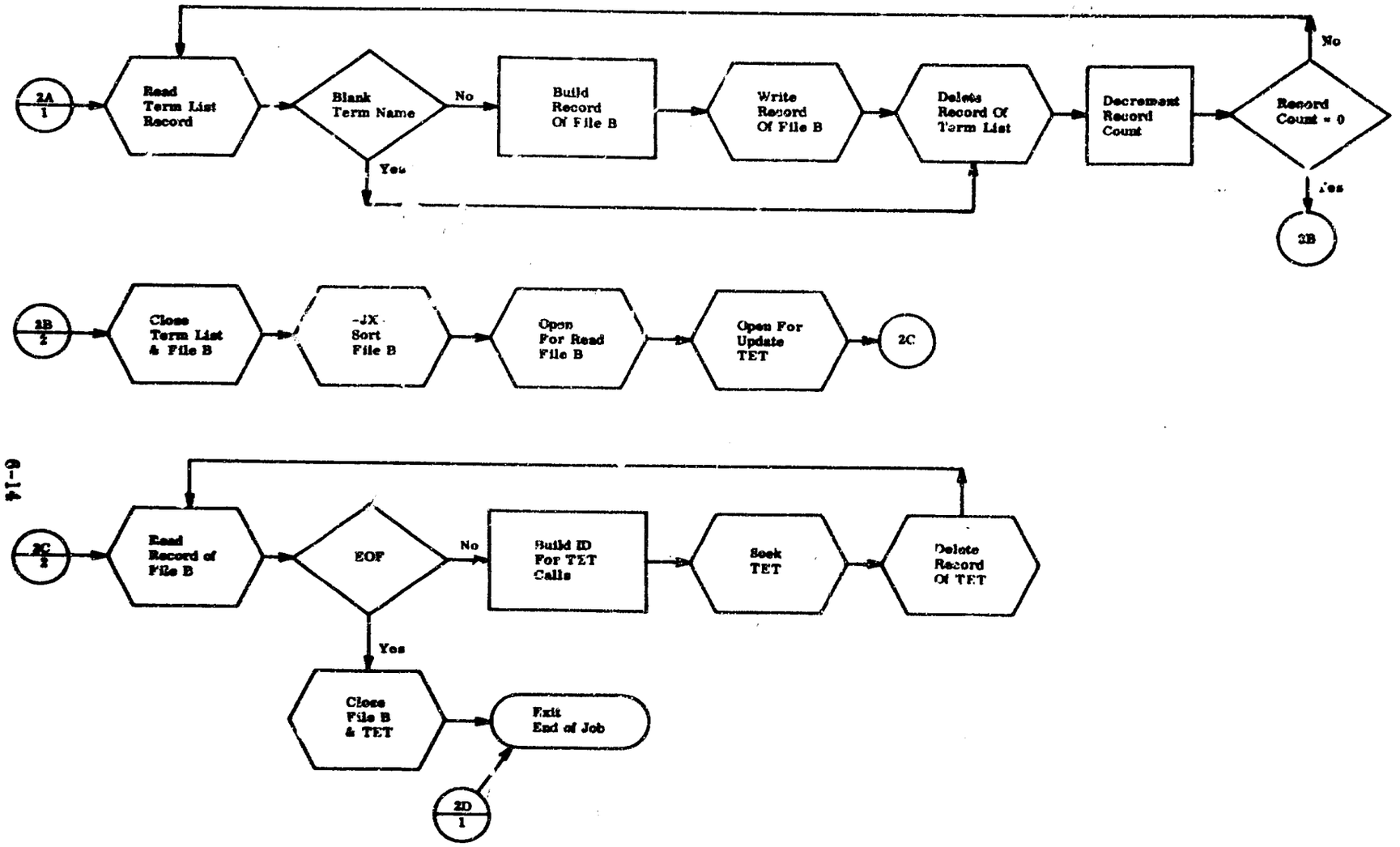
(2) Connector 1B performs the deletion of Item List records, counting as it goes, so that the same number of Term List records can be deleted.

(3) At connector 1C and 2A, the equivalent records are deleted from the Term List, and, for any subsumed items with nonblank term names, a scratch file B is written. Scratch file B includes a record for each term name which will have to be deleted from the Term Encoding Table (TET).

(4) File B is sorted by term name and is then used at connector 2C to control the deletion of records (or parts of records when a term name is equated to several ICC's) from the TET. When TET updating is complete, the job ends.



Delete-Definition
Sheet 1 of 2



Delete-Definition
Sheet 2 of 2

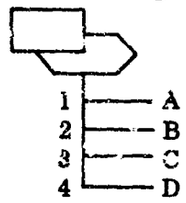
6.1.3 Delete Node

Job Request: DELETE-NODE (item name).

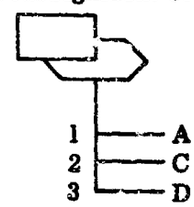
6.1.3.1 Functional Description. This job removes the item named in (item name) from the TET, Term List, and Item List. If (item name) is a statement or file (record is illegal), all subsumed nodes are also removed. If this removal creates a gap, the nodes beyond the point of removal will be reassigned to fill the gap. For example, for the Job Request:

DELETE-NODE (B).

the following structural diagrams illustrate the reassignment of nodes C and D.



BEFORE



AFTER

If Delete Node causes a record's number of nodes to be reduced to zero, the parent node (file) will be converted to a null node. If this job is executed at a time when the Data Base has data filed under this structure, the data will be deleted from the Data Base.

6.1.3.2 Inputs. The only input is an item name. This is a term name (qualified) for the item which is to be deleted.

6.1.3.3 Results. The directories are updated as a result of this job, but there are no job outputs.

6.1.3.4 Directories Used

(1) Term Encoding table.

(2) Item List.

(3) Term List.

(4) Linkage Table.

(5) Segment Name List.

6.1.3.5 Services Used

(1) Locate IL Entry.

(2) Open for Reading.

(3) Read.

(4) Close for Reading.

(5) Open for Writing.

(6) Write.

(7) Close for Writing.

(8) Open for Update.

(9) Seek.

(10) Replace.

(11) Delete.

(12) Close for Update.

(13) Retrieve Item.

(14) Replace Item.

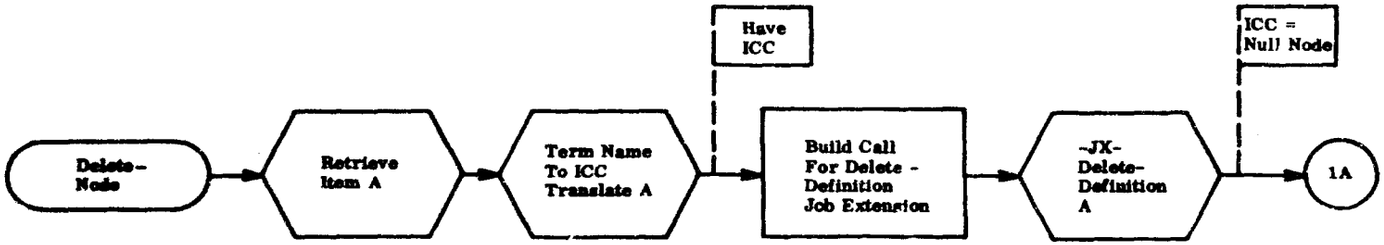
(15) Delete Data.

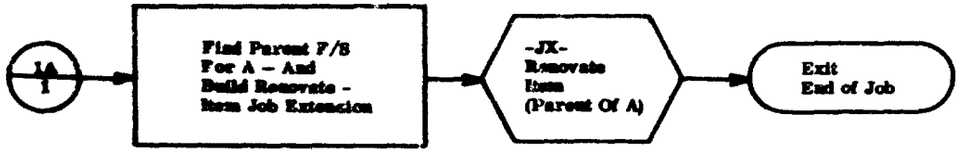
(16) Term Name to ICC Translation.

6.1.3.6 Jobs Used

(1) Data Delete.
(2) Sort.
(3) Delete Definition.
(4) Renovate Item.

6.1.3.7 Method of Operation. This job first calls on Delete Definition as a Job Extension to redefine the item named in (item name) as a null node. All subsumed nodes are removed through this process. Then, the direct parent item (a file or statement) is given to Renovate Item; through this Job Extension the null node is squeezed out and any nodes beyond the null node are reassigned to fill the gap. Then, the Delete Node job terminates.





Delete-Node
Sheet 1 of 1

6.1.4 Renovate Item

Job Request: RENOVATE-ITEM (item name).

6.1.4.1 Functional Description. This job removes all null nodes directly subsumed by the statement or file named in item name. For example, if nodes C and F are listed as null nodes in the Item List, Renovate Item will accomplish the following:

1	A	1	A
2	B	2	B
3	C	3	D
4	D	4	E
5	E		
6	F		
BEFORE		AFTER	

If Renovate Item reduces the number of nodes subsumed by a record to zero, the item name will be converted, in the Item List, to a null node. If this job is executed at a time when the Data Base has data filed under this structure, the data will be deleted from the Data Base.

6.1.4.2 Inputs. The only input is an item name, which is a term name (qualified) for the item which subsumes one or more null nodes.

6.1.4.3 Results. There are no job outputs; the directories are updated to reflect the structural change.

6.1.4.4 Directories Used

(1) Term Encoding Table.

(2) Item List.

(3) Term List.

(4) Linkage Table.

(5) Segment Name List.

6.1.4.5 Services Used

(1) Locate IL Entry.

(2) Open for Reading.

(3) Seek.

(4) Read.

(5) Close for Reading.

(6) Open for Writing.

(7) Write.

(8) Close for Writing.

(9) Open for Update.

(10) Replace.

(11) Delete.

(12) Close for Update.

(13) Retrieve Item.

(14) Replace Item.

(15) Delete Data.

(16) Term Name to ICC Translation.

6.1.4.6 Jobs Used

(1) Data Delete.

(2) Sort.

6.1.4.7 Method of Operation. A is the program's name (formal) for an alphanumeric string which contains the term name for the item whose subsumed nodes are to be renovated.

(1) After translating the term name to an ICC, the Item List entry is retrieved. Only files or statements are acceptable items for renovation. If the named item is a file, its record entry is substituted because the record entry contains the number of directly subsumed nodes.

(2) At connector 1A, the No Data Flag is tested. If data exists, it is deleted by calling on the Data Delete job as a Job Extension. While the parent item is still available, the record number of this item in the Item List file is stored along with the number of nodes currently subsumed.

(3) The loop at connector 1B looks at each subsumed item to detect which are null nodes and how many null nodes there are. If this process detects no null nodes, the job terminates because there is no work to be done.

(4) With the Item List record numbers for all null nodes available in a "first in, first out" list, the Item List file and the Term List file are opened for updating at connector 1C. Scratch files B and C are opened for writing. File B will get two kinds of records:

(a) A record for each term name which is to be deleted from the Term encoding Table.

(b) A record for each term name referencing an ICC which has to be modified in the Term Encoding Table.

File C will get records for all link items in the Item List beyond the first null node on the stem being renovated.

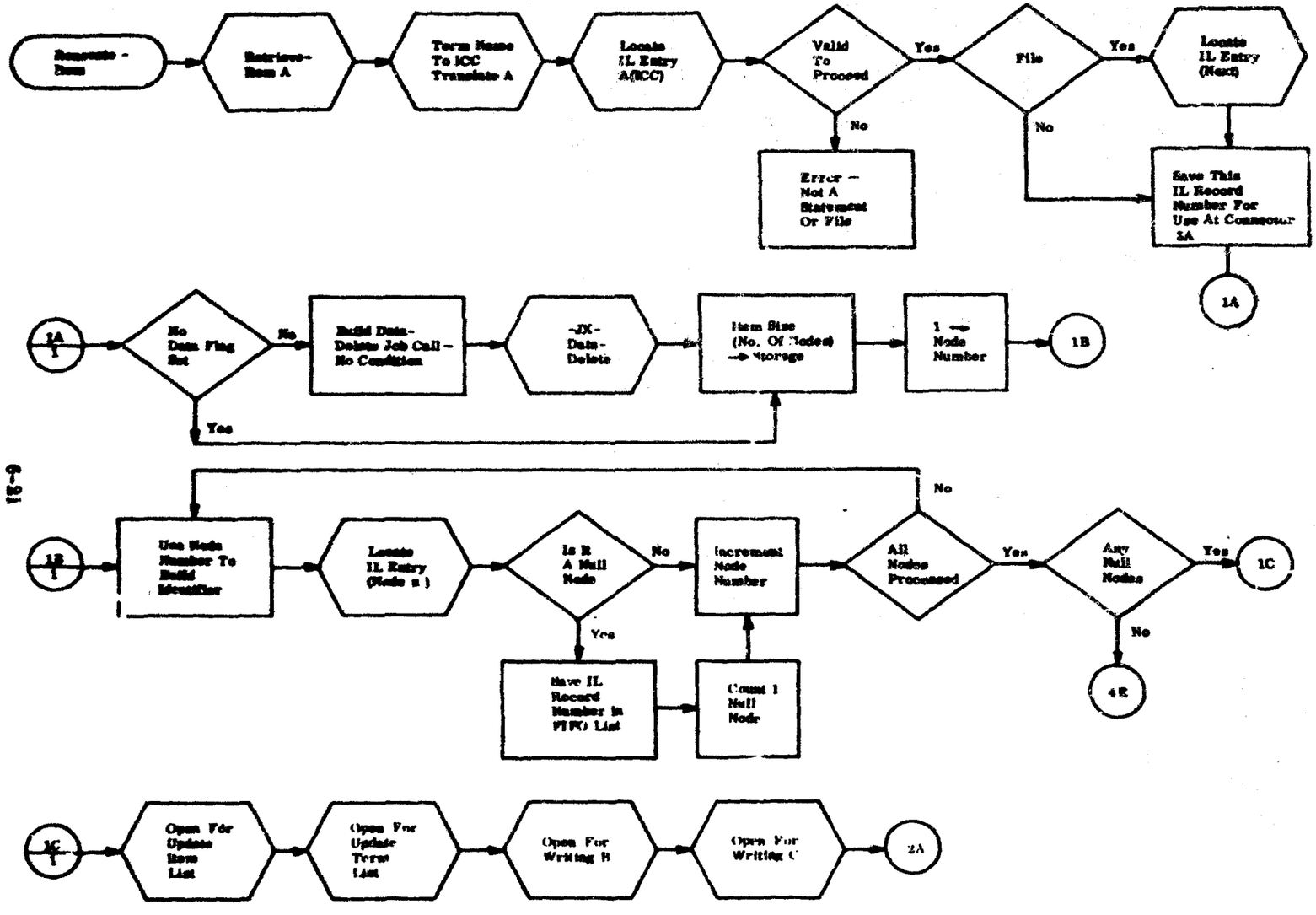
(5) Connectors 2A through 2D are concerned with updating the Item List and Term List by deleting the records representing the null nodes and writing records of files B and C.

(6) Connector 2F takes care of the following special problem: if the parent item is a statement, the item size may legally be decremented to zero. If the parent item is a record, and if the decrementing reduces the size to zero, this is illegal. The record item must be deleted and the file item converted to a null node.

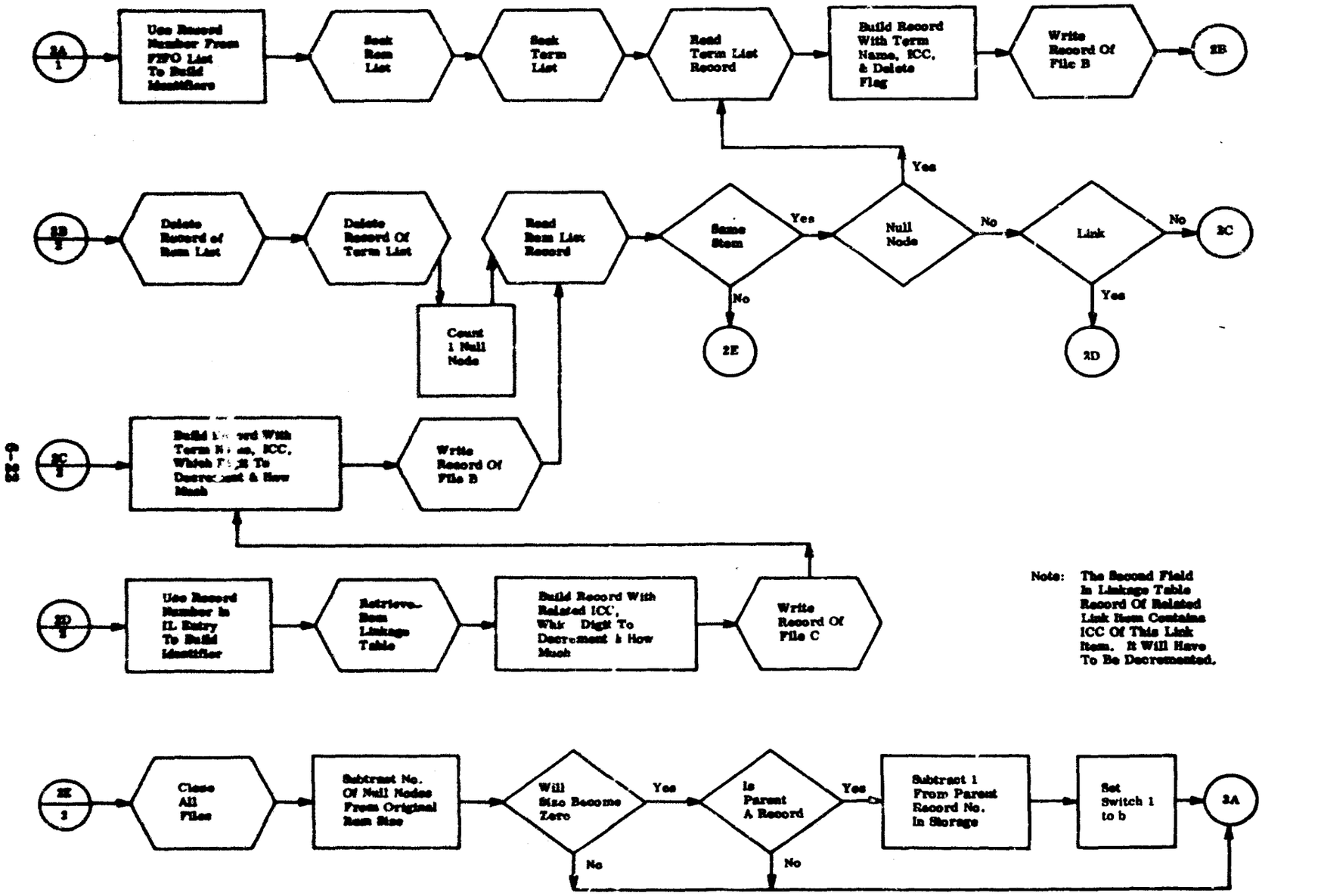
(7) Connector 3A takes care of decrementing the item size by the number of null nodes deleted.

(8) Scratch file B is sorted by term name at connector 3D and is then used to control a Term Encoding Table (TET) update which removes the names of null nodes from the TET and decrements all ICC's in the TET which are affected by the squeezing out of null nodes.

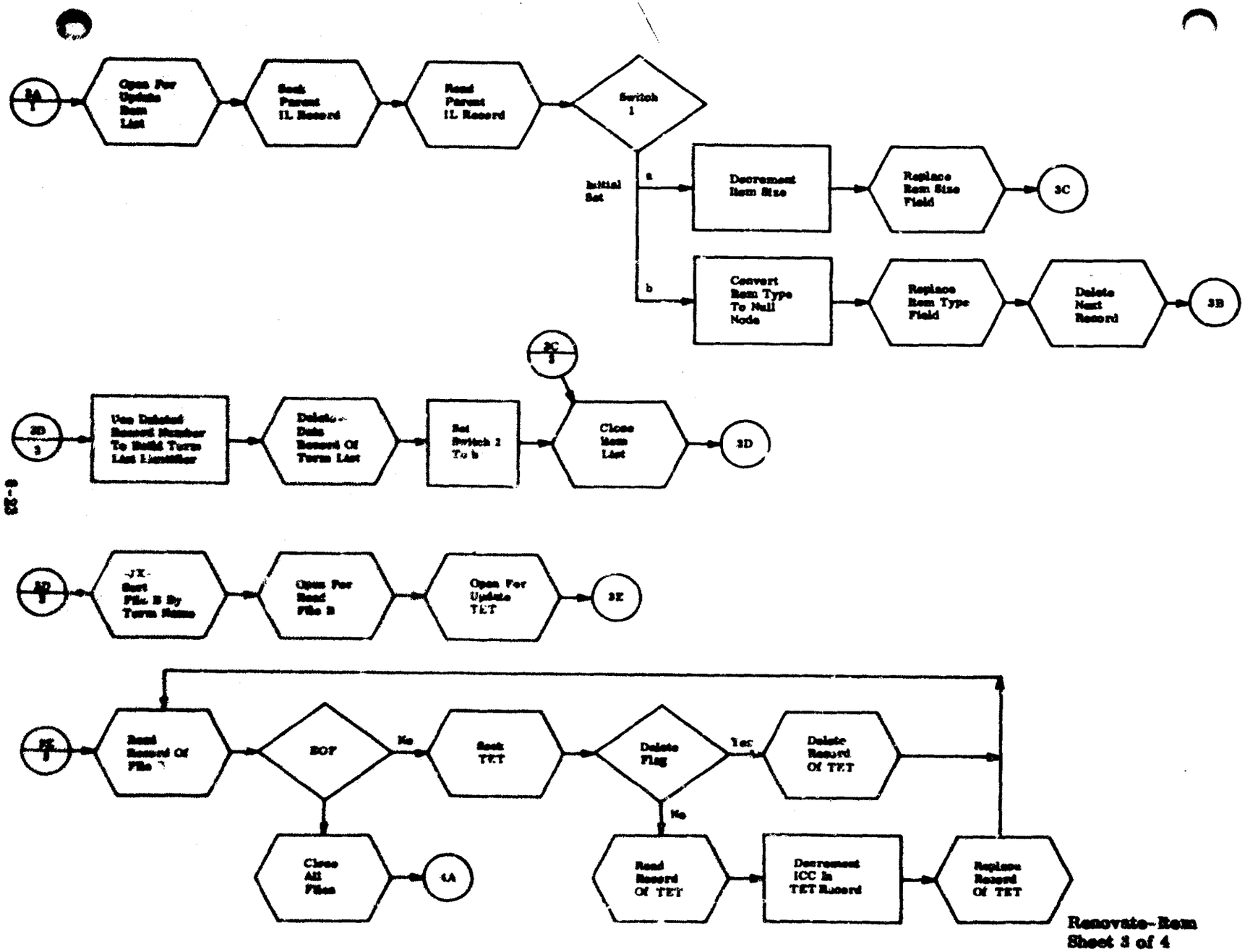
(9) Scratch file C is sorted by ICC at connector 4A and is then used to control an Item-List-to-Linkage-Table access so that the ICC's in the Linkage Table records at the other end of the links can be decremented properly. After this is accomplished, the job terminates.



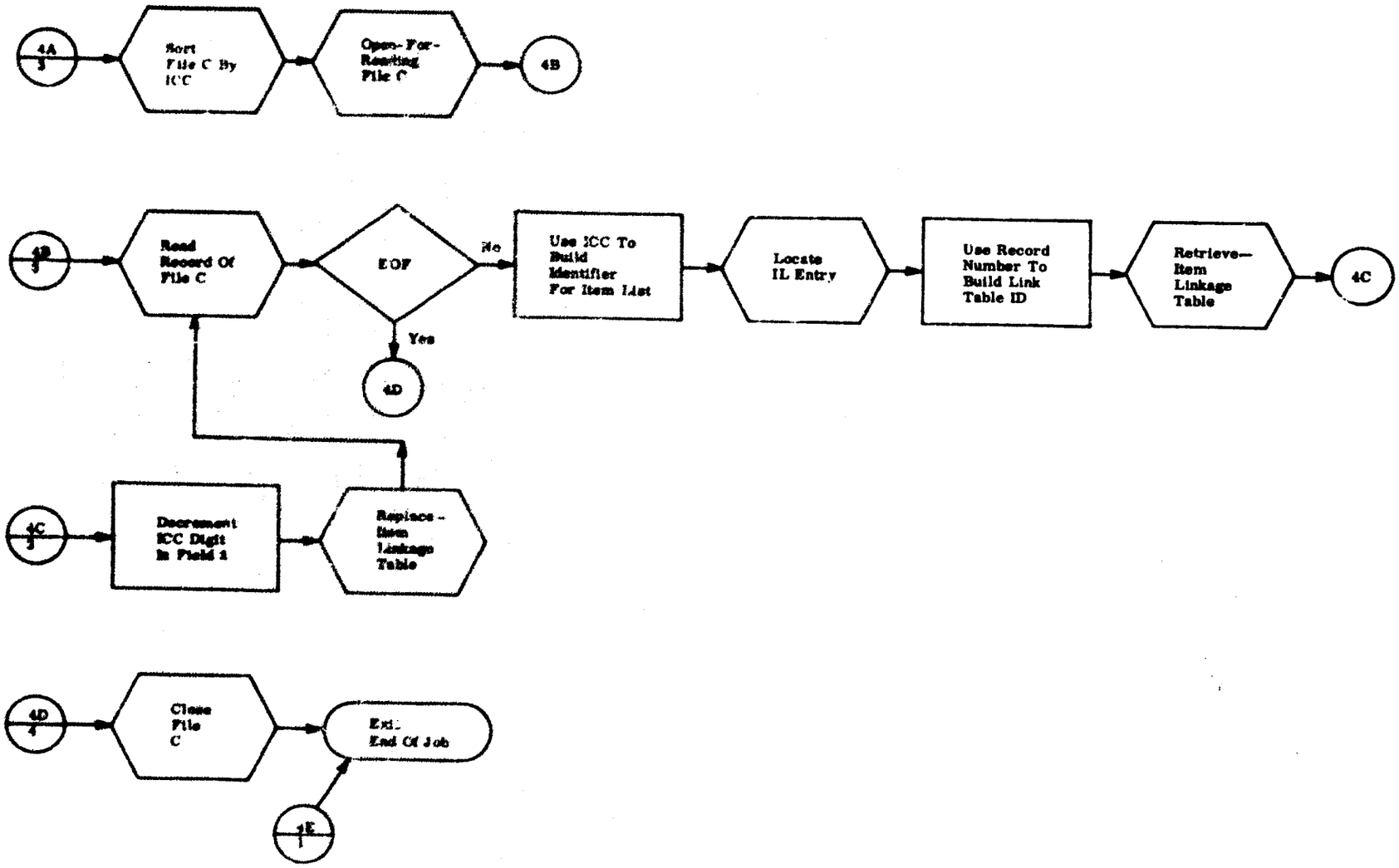
Renovate-Item
Sheet 1 of 4



Renovate-Item
Sheet 2 of 4



Renovate-Item
Sheet 3 of 4



Renovate-Item
Sheet 4 of 4

6.2 DATA MANIPULATION

Additions and deletions of Data Base data are the responsibility of the Data Administrator, and the system jobs described in this section are the tools provided for his use. Add Data and Delete Data are the basic jobs, with Replace Data combining a delete and add and Modify Data for simple standardized filed replacements.

Addition and deletion of records within a file present a maintenance problem if the file is ordered and/or indexed. Records of indexed files are deleted by marking their subsumed items as missing or null. This avoids the necessity of changing record numbers in the directory tables connected with indexing. After a file has had many records deleted by this technique, the file may be compressed by the Compress File job. This will make all the record number changes at once, and will reassign consecutive numbers.

For the ordered file, records must be added in sequence and be deleted by complete removal. If indexed fields are involved, the record number changing cannot be deferred for a subsequent compression. Update Data is a special job which allows a set of transactions to be applied to an ordered file through matching on key values.

6.2.1 Add Data

Job Request: ADD-DATA (source), (item), (condition).

6.2.1.1 Functional Description. This job adds the data named as (source) to the Data Base at the position specified by (item). If (item) does not represent a unique position, the (condition) clause is used to completely specify the position. If a conditional search does not yield a unique position, the assumption is that the source data is to be added at all positions which met the condition. If no condition is given, the assumption is that the source data is to be added at all positions. Except when adding ordered records to an ordered file, the source data must be a set of items which will occupy contiguous positions in the Data Base.

6.2.1.2 Inputs. The inputs to Add Data are:

(1) (source) - the term name for a data item which has been translated to internal form.

(2) (item) - the term name for the Data Base item. If this class of item is subsumed within the records of a file, the position at which the data is to be added should be specified by a condition.

(3) (condition) - a Boolean statement which specifies the position within the Data Base of the missing item which is to receive the source data.

6.2.1.3 Results. The data is added at the position specified. There are no job outputs.

6.2.1.4 Directories Used

(1) Term Encoding Table.

(2) Item List.

(3) Fields File.

(4) Shadow of Fields File.

(5) Segment Name List.

6.2.1.5 Services Used

(1) Locate IL Entry.

(2) Open for Reading.

(3) Read.

(4) Close for Reading.

(5) Open for Writing.

(6) Write.

(7) Close for Writing.

(8) Open for Update.

(9) Seek.

(10) Replace.

(11) Insert.

(12) Close for Update.

(13) Retrieve Item.

(14) Replace Item.

(15) Insert Data.

(16) Term Name to ICC Translation.

6.2.1.6 Jobs Used

(1) Conditional Search.

(2) Sort.

6.2.1.7 Method of Operation. A is the program's name (formal) for the Data Base item which is to receive the data. B is the program's name (formal) for the condition which will become the input to Conditional Search used as a Job Extension. C is the program's name (formal) for the list of IPC's produced by Conditional Search or by Data Add itself if Conditional Search is not executed. D is the program's name (formal) for the source item whose definition is the Directory and whose data is in the working area of the Data Pool.

(1) Page 1 is all preparation. A is examined first:

(a) If there are no R's in the ICC (i.e., a primary item), the ICC which is an IPC is written into file C.

(b) If there is one or more R's in the ICC and if B is present, Conditional Search is used as a Job Extension to build file C.

(c) If there is one or more R's in the ICC and if B is not present, an Explosion is executed. Explosion increments each R in turn and writes out IPC's until it reaches EOF. Its output is file C with IPC's for all occurrences of the given ICC.

All paths join after file C is written and the source item D is then translated into an ICC.

(2) At connector 2A, a test is made to discover if this data will be the first data to be filed at this node in the Data Pool structure. If the No Data Flag is set, the parent item in the structure is examined to make certain that the parent has data. If the parent item has data, then this addition of subsumed data is legal. Then, the Item Lists of source (D) and target (A) are compared to make sure that the Add is legal.

(3) At connector 2B, the source item is examined, and the program branches based on item type of the source item. All paths rejoin at connectors 7D-7H.

field ——→ connector 3A
nonordered file ——→ connector 3D
ordered file ——→ connector 4A
statement ——→ connector 5B

(4) For field addition, the single field named as source is retrieved by IPC and inserted into the Data Pool in each position listed in file C. For each insertion, if the field is indexed, a record containing ICC, Field value, and R-value is written into file E, which will later become the input to Fields file and Shadow of Fields file updating. When EOF is reached in file C, control goes to connector 7H.

(5) For nonordered file addition, the source file identified as file D is read one field at a time and inserted into the Data Pool at EOF in each target file listed in file C. For each insertion of an indexed field, a record is written into file E. When EOF is reached in file C, control goes to connector 7G.

(6) In ordered file addition, if a target file is missing the addition is equivalent to a nonordered file because the source file is assumed to be in order. If a target file has some records, the new records of file D must be inserted in order by key value, and this kind of insertion changes the record numbers of all records beyond the point of insertion. If indexed fields are involved, this will require changing the R-values in the RVIT files. To prepare for this, file F is opened for writing and an output buffer is established as follows:

R- VALUE OF FILE	OLD RECORD NUMBER	INCREMENT	ICC OF INDEXED FIELD

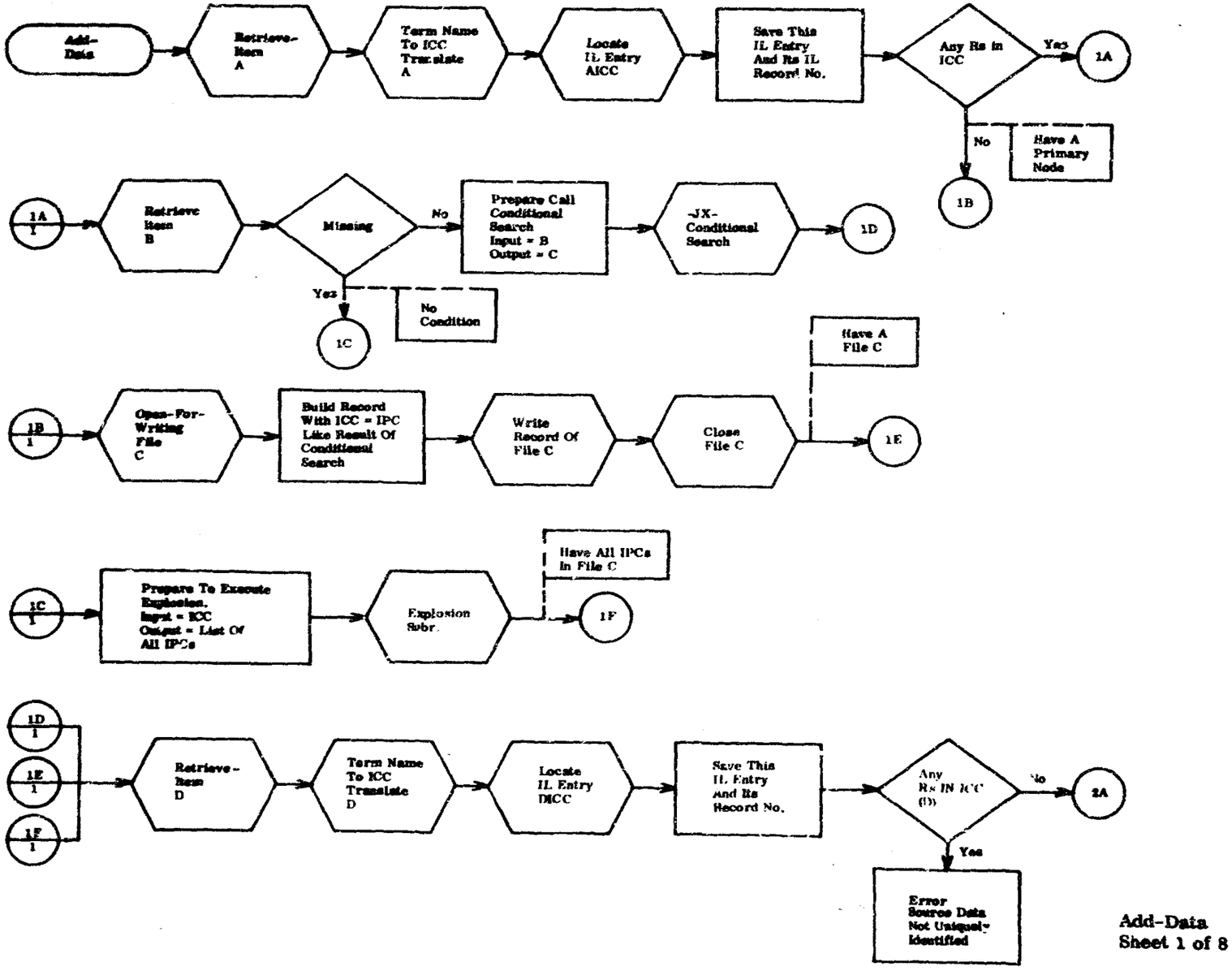
At connector 4D, a source record is read to get the key value. This key value is used as an identifier for Seek within the target file. The record number is stored in the output buffer along with an initial increment of one. Then, the source record is read one field at a time and inserted into the target. Each indexed field causes a record of file E to be written and the output buffer is written into file F for each indexed field in the record. At end of record, at connector 5A, the increment is increased by one and the Seek yields another old record number for the output buffer. Then the Read Source-Insert In Target process is repeated. At EOF of source, file C is read again to find the next target file which is to receive the same source data. When EOF is reached in file C, control goes to connector 6A. If no indexed fields were added, there were no records written into File F and control would go to connector 7E.

If file F has records, it is sorted by ICC, R-value of parent file, and old record number. Then, for each ICC in the records of file F, the following procedure is carried out to correct the R-values in the affected RVIT files. The ICC is used to access the FVT file in order to get a total record count of the FVT file and to discover how many RVIT files need to be scanned. Each RVIT file is read to discover the section which has R-values matching the R-value of the data file to which records have been added. Within the proper section of RVIT, the record numbers at which insertions were made (from file F) are used to discover which R-values need to be incremented in their least significant digit. The increment is added and the R-value is replaced in the RVIT file. When file F reaches EOF, all R-values have been incremented, and control goes to connector 7D.

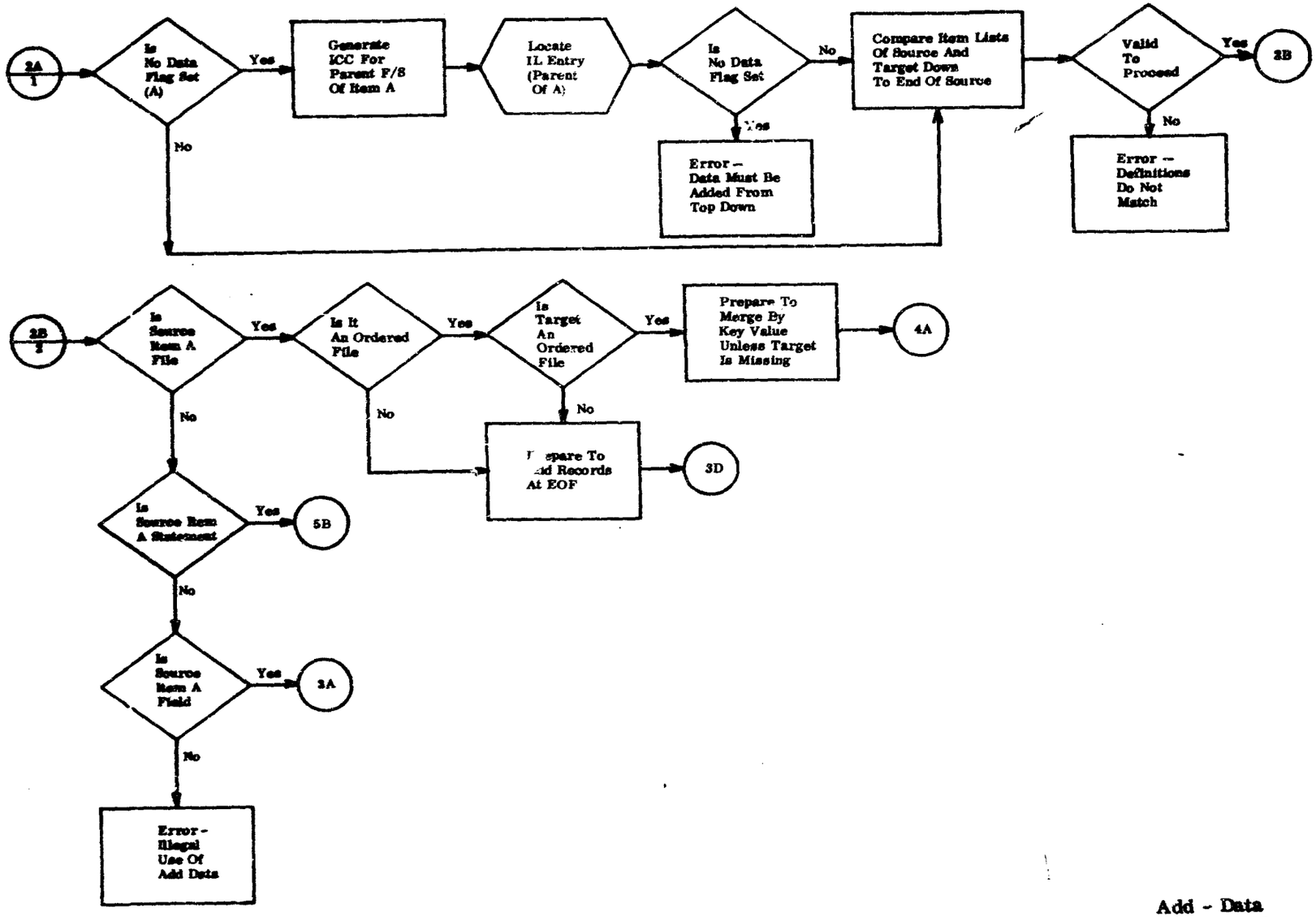
(7) For a statement addition, the source statement, D, is read one field at a time and inserted into the Data Pool in each target statement listed in file C. For each insertion of an indexed field a record is written into file E. At EOF of file C, control goes to connector 7F.

(8) All paths join at connectors 7D-7H. If there are no records in file E, no indexed fields were added, and the job terminates.

If indexed fields have been added, file E is sorted by ICC. Then, all records pertaining to a single ICC are written into file G which serves as input to task 2 of the Index job used as a Job Extension. When all ICC's within file E have been processed, the updating of the directories pertaining to indexing is complete, and Add Data terminates.

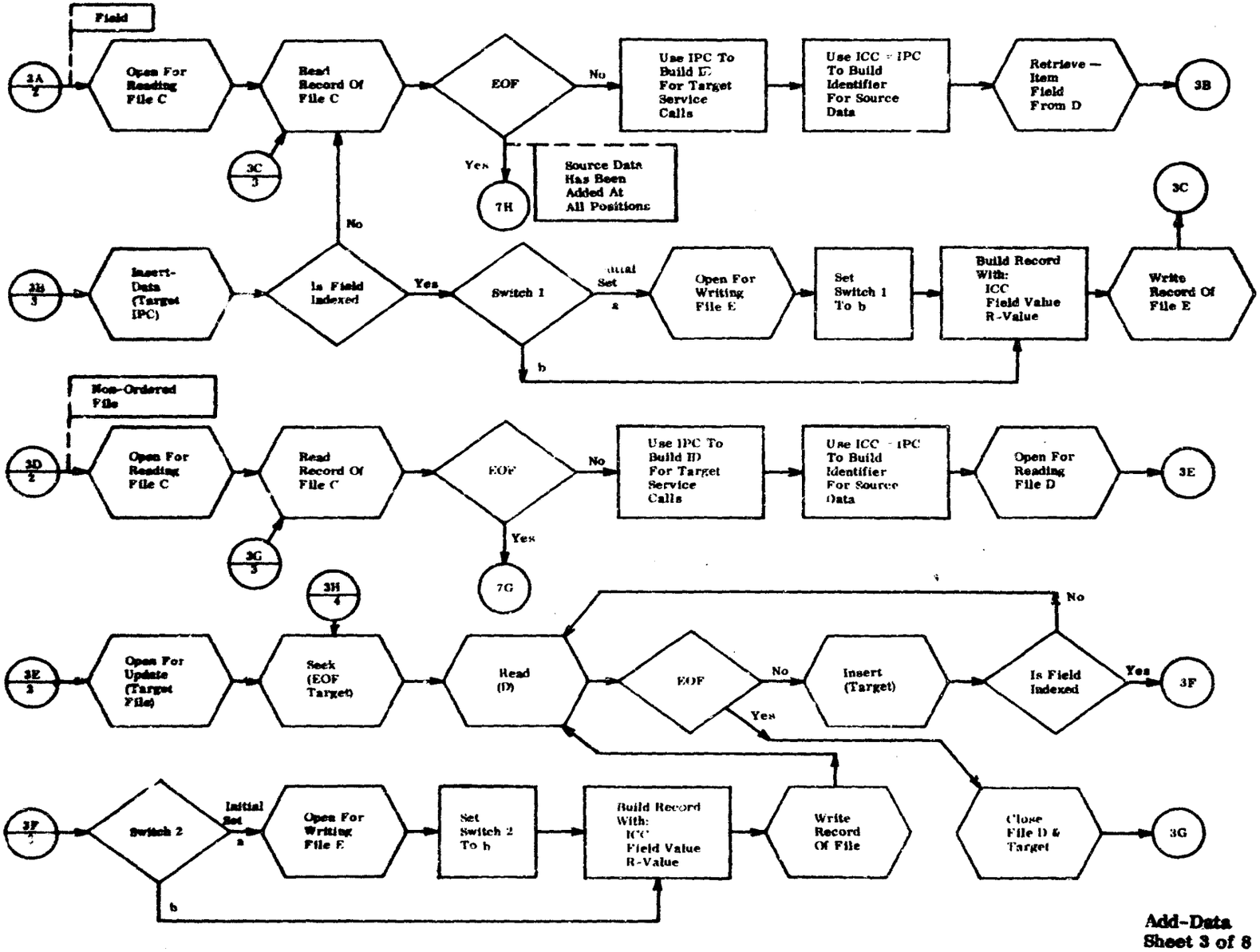


Add-Data
Sheet 1 of 8

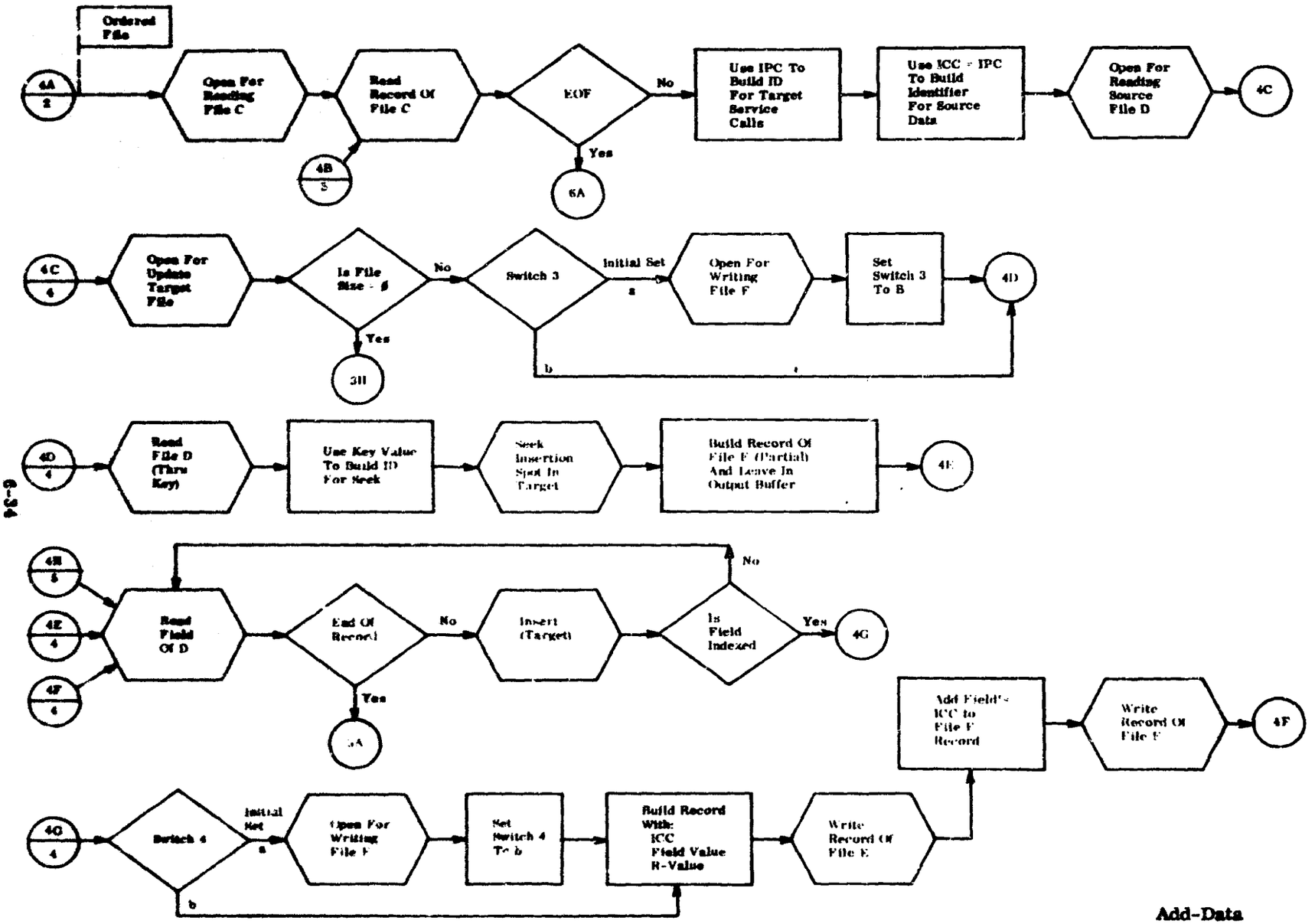


6-32

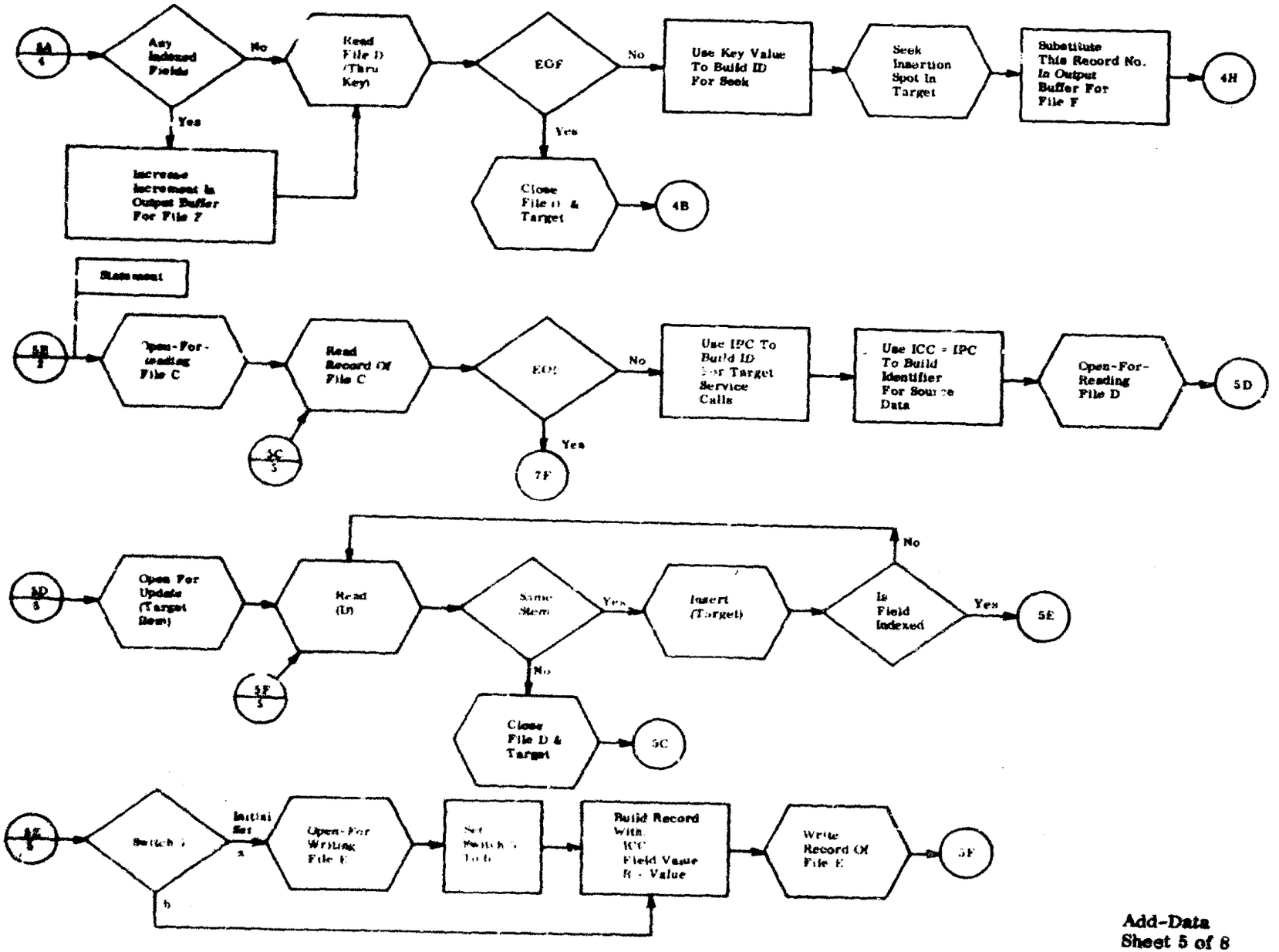
Add - Data
Sheet 2 of 8



Add-Data
Sheet 3 of 8

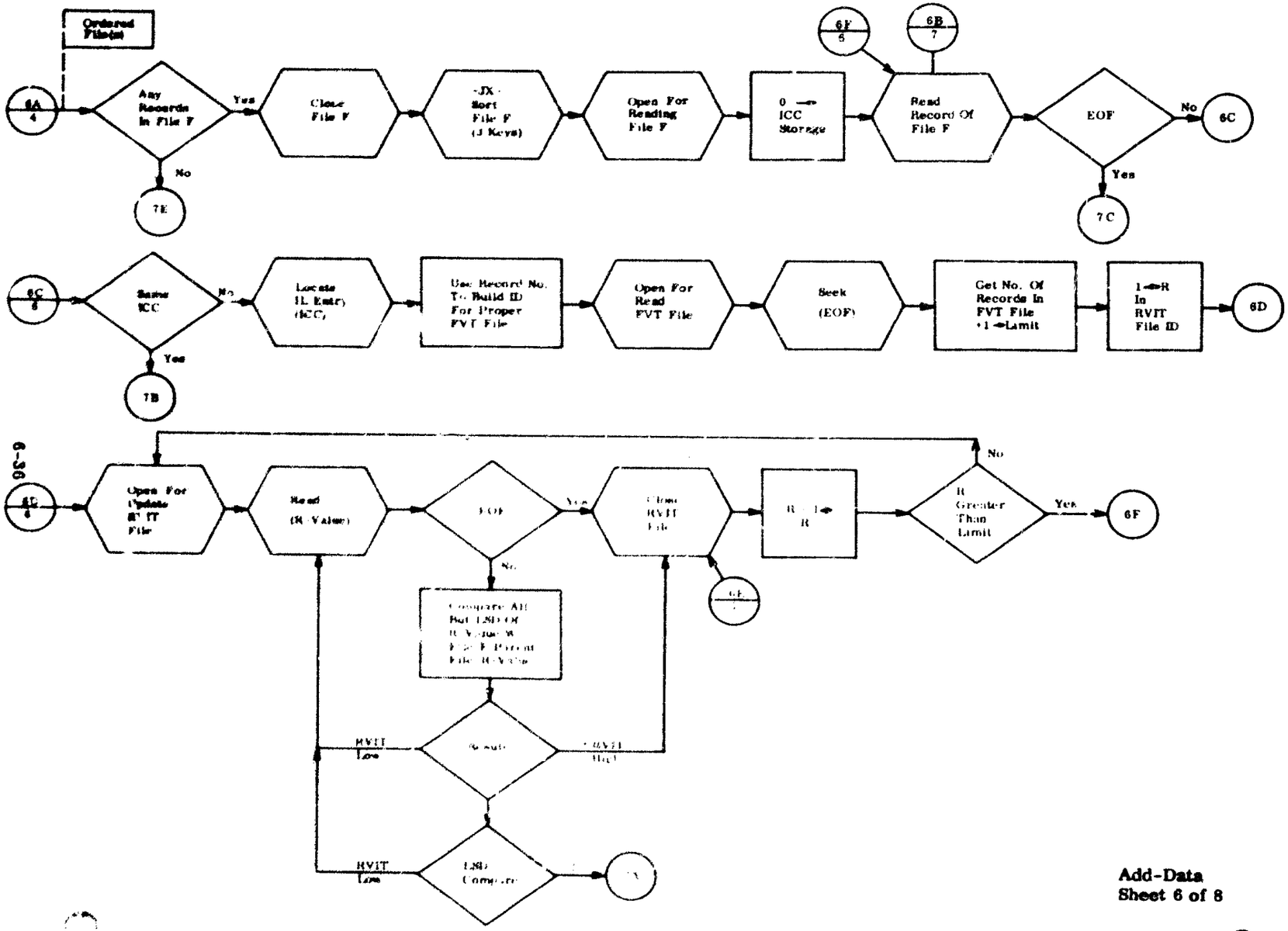


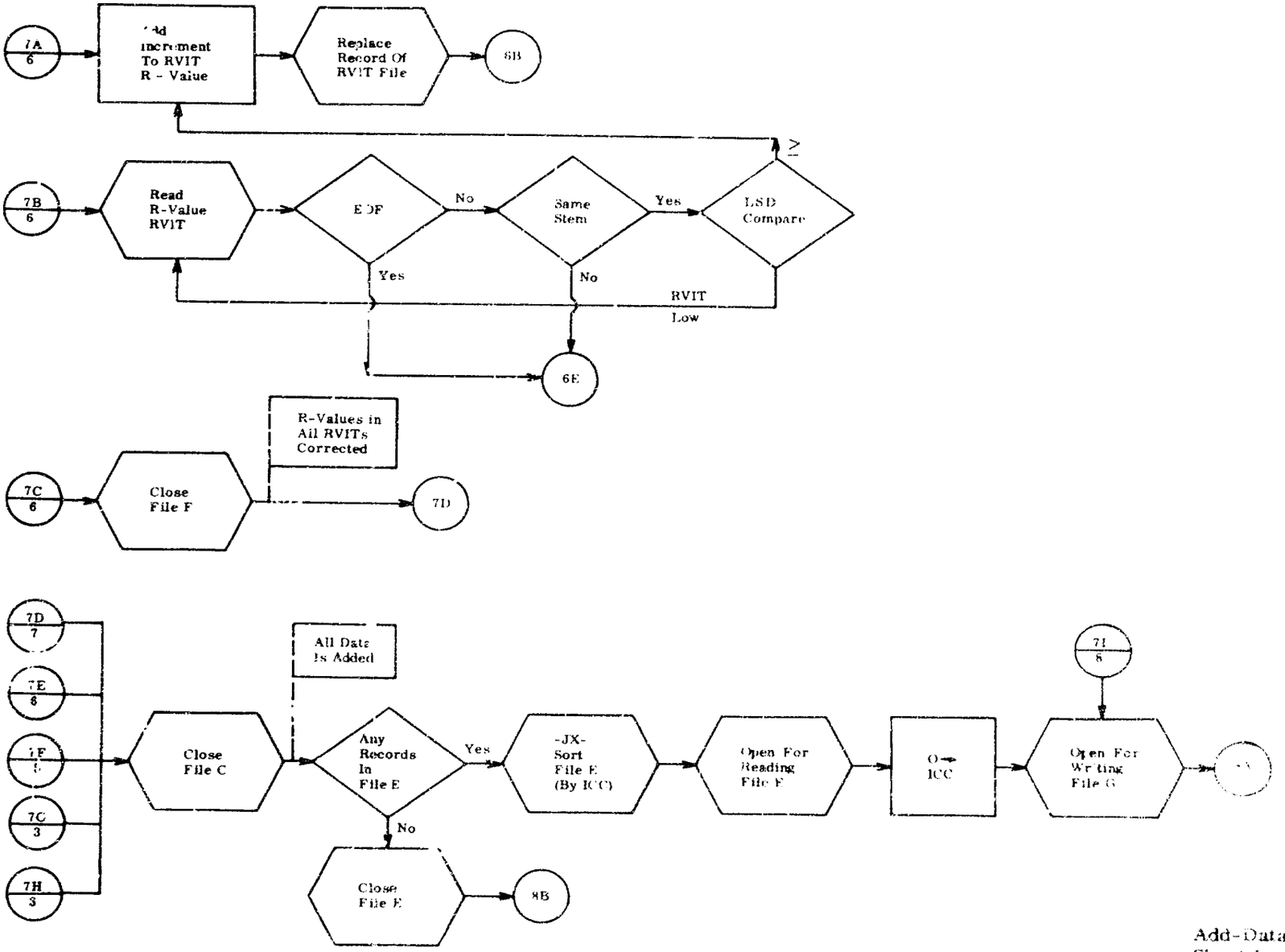
Add-Data
Sheet 4 of 8



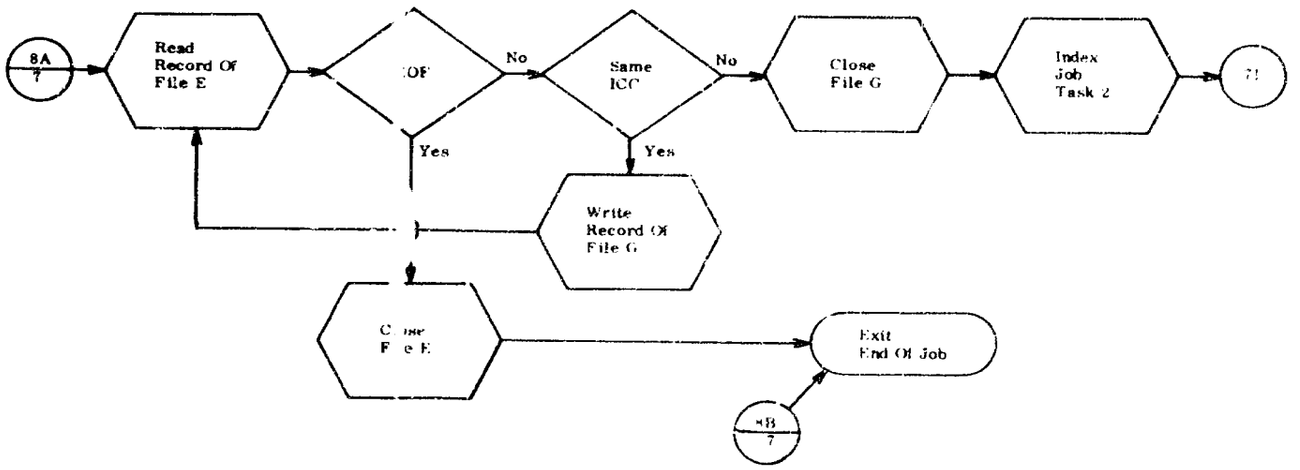
6-35

Add-Data
Sheet 5 of 8


Ordered File(s)
6A 4
Any Records In File F
Yes
No
7E
Close File F
-JX- Sort File F (3 Keys)
Open For Reading File F
0 → ICC Storage
6F 5
6B 7
Read Record Of File F
EOF
No
6C
Yes
7C
6C 8
Same ICC
No
Yes
7B
Locate (L Entr) (ICC)
Use Record No. To Build ID For Proper PVT File
Open For Read PVT File
Seek (EOF)
Get No. Of Records In PVT File +1 → Limit
1 → R In RVIT File ID
6D
6-96
6E 9
Open For Update RVIT File
Read (R Value)
EOF
Yes
Close RVIT File
R + 1 → R
R Greater Than Limit
Yes
No
6F
6E
RVIT Low
RVIT High
RVIT Low
Add-Data
Sheet 6 of 8



Add-Data
Sheet 1 of 3



Add-Data
Sheet 8 of 8

6.2.2 Replace Data

Job Request: REPLACE-DATA (source), (item), (condition).

6.2.2.1 Functional Description. This job substitutes the data named as (source) for the data which exists at the position specified by (item) and (condition). Replace Data is logically equivalent to Delete Data followed by Add Data.

If a conditional search does not yield a unique position, the assumption is that the source data is to replace the current data at all positions which met the condition. If no condition is given, the assumption is that the source data is to replace current data at all positions. The source data must be a set of items which will occupy contiguous positions in the Data Base and will completely replace the current contiguous data. See Update Data for selective or noncontiguous replacements of records within a file.

6.2.2.2 Inputs. The inputs to Replace Data are:

(1) (source) - The term name for a data item. The data represented by this name has been translated into internal segmented form and stored in the Data Pool.

(2) (item) - The term name for the Data Base item which is to receive the replacement. If the item is subsumed within the records of a file, the position at which the replacement is to occur should be specified by (condition).

(3) (condition) - a Boolean statement which specifies the position at which the replacement is to occur.

6.2.2.3 Results. There are no job outputs; the source data replaces the original data at the position specified.

6.2.2.4 Directories Used

(1) Term Encoding Table.

(2) Item List.

(3) Fields File.

(4) Shadow of Fields File.

(5) Segment Name List.

6.2.2.5 Services Used

(1) Locate IL Entry.

(2) Open for Reading.

(3) Read.

(4) Close for Reading.

(5) Open for Writing.

(6) Write.

(7) Close for Writing.

(8) Open for Update.

(9) Seek.

(10) Replace.

(11) Insert.

(12) Delete.

(13) Close for Update.

(14) Retrieve Item.

(15) Replace Item.

(16) Insert Data.

(17) Delete Data.

(18) Term Name to ICC Translation.

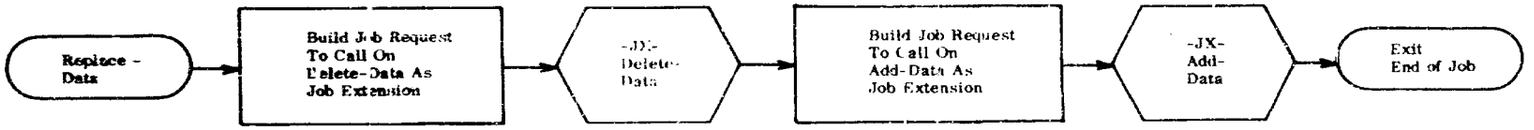
6.2.2.6 Jobs Used

(1) Conditional Search.

(2) Sort.

6.2.2.7 Method of Operation. For the initial implementation, this job will be accomplished by executing two other jobs as Job Extensions. First the Delete Data job will be used to mark the data as missing or null in the Data Base segments. Then the Add Data

job will be used to insert the source data at the position just recorded as missing or null. Subsequent versions of Replace Data can be made more efficient by analyzing the number of data segments to be affected by the replacement and choosing from a set of alternative procedures. More experience is needed on the rate of change, and kind of changes, before attempting to improve efficiency in this area.



Replace-Data
Sheet 1 of 1

6.2.3 Modify Data

Job Request: MODIFY-DATA (expression), (item), (condition).

6.2.3.1 Functional Description. This job is similar to Replace Data except that there is no source data. Instead of providing new data to replace the old, the expression clause gives an arithmetic operation to be performed on the old data. The results of the arithmetic operation then replace the old data (e.g., SALARY + 500) at the position(s) specified by (item) and (condition).

Modify-Data requires that (item) be defined as a field. If a conditional search does not yield a unique position, the assumption is that all positions which met the condition are to be modified. If no condition is given, the assumption is that all positions are to be modified.

6.2.3.2 Inputs. The job inputs are:

(1) (expression) - a specification for an arithmetic operation on two operands. The first operand is the term name for a field, the operator is an arithmetic operator (+ or -), and the second operand is a constant, (e.g., SALARY + 500).

(2) (item) - the term name for the Data Base field which is to be modified. Unless all occurrences are to receive the modification, a condition is needed.

(3) (condition) - a Boolean statement which specifies the position(s) at which the modification is to occur.

6.2.3.3 Results. There are no outputs except for the data which has been modified.

6.2.3.4 Directories Used

(1) Term Encoding Table.

(2) Item List.

(3) Segment Name List.

6.2.3.5 Services Used

(1) Locate IL Entry.

(2) Open for Reading.

(3) Read.

(4) Close for Reading.

(5) Open for Writing.

(6) Write.

(7) Close for Writing.

(8) Retrieve Item.

(9) Replace Item.

(10) Term Name to ICC Translation.

6.2.3.6 Jobs Used. Conditional Search.

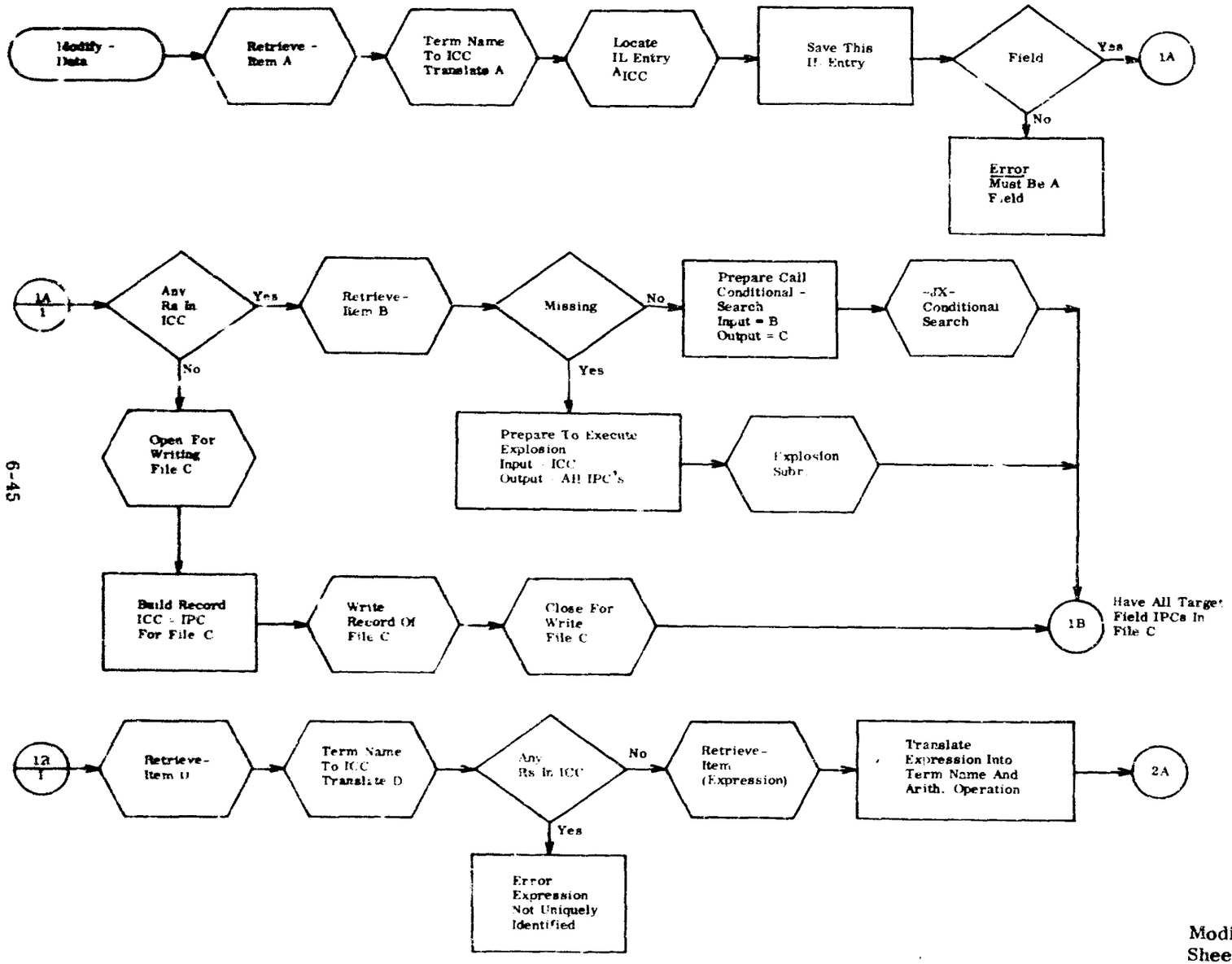
6.2.3.7 Method of Operation. A is the program's name (formal) for the Data Base item which is to receive data. B is the program's name (formal) for the condition which will become the input to Conditional Search used as a job extension. C is the program's name (formal) for the list of IPC's produced by Conditional Search or by the program itself if Conditional Search is not executed. D is the program's name (formal) for the expression.

(1) Page 1 is all preparation. A is examined first. It is retrieved, translated into an ICC, located in the Item List, and the item type is tested for field. If the ICC has one or more R's, B is retrieved. If B exists, the condition is given to Conditional Search for evaluation, and file C is written by Conditional Search to include the IPC's which satisfy the condition. If B is missing, an Explosion subroutine is executed to produce file C, which includes all IPC's for all occurrences of A. If the ICC of A has no R's, file C is produced with one record containing the ICC which is an IPC.

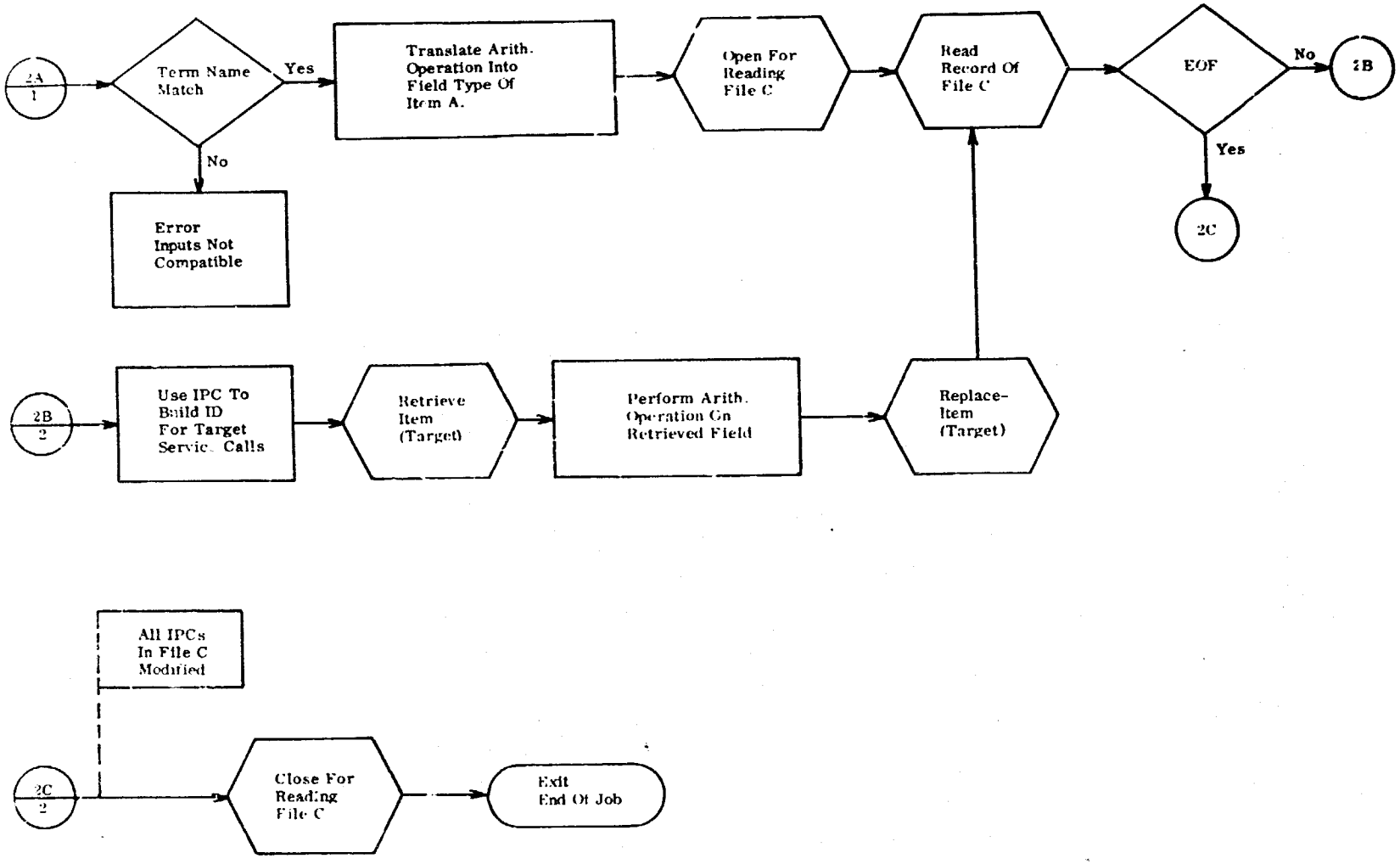
(2) After file C is created, the expression is retrieved at connector 1B. At connector 2A, the first operand of the expression, the term name, is compared with A. If they match, the second operand, the constant, is translated into the item type (decimal, octal, etc.) of the term.

(3) File C is read to get the target IPC and to build proper identifiers for the Retrieve Item and Replace Item service calls. Then, the field is retrieved from its position in the Data Base and the arithmetic operation is performed on it. The result is replaced in the Data Base at the same position. This completes the modification.

(4) Successive records are read from file C, and the same arithmetic operation is performed at each position listed in file C. When EOF is reached, file C is closed, and the Modify Data job terminates.



Modify-Data
Sheet 1 of 2



Modify-Data
Sheet 2 of 2

6.2.4 Update Data

Job Request: UPDATE-DATA (source), (item), (condition).

6.2.4.1 Functional Description. Replace Data is a one-to-one or one-to-many substitution of the source data for the current data. Replace Data is a one-to-one substitution if the position specified by (item) and (condition) is unique. Replace Data is a one-to-many substitution if Conditional Search does not yield a unique position.

Update Data is a many-to-many substitution specifically designed to provide for a batch or series of record replacements within an ordered file. If the source and target files are both ordered by the same key field, Update Data is a simple two-way merge, with a source record replacing a target record whenever the keys match. Nonmatching source records are ignored in Update Data. The Add Data job inserts new records in their proper positions within an ordered file.

6.2.4.2 Inputs. The inputs are:

(1) (source) - the term name for a data file which contains the transaction or change records.

(2) (item) - the term name for the Data Base file which is to be updated. If this file is subsumed, (condition) should be used to specify the position of the file.

(3) (condition) - a Boolean statement which specifies the position within the Data Base of the file named in (item).

6.2.4.3 Results. No job outputs except for the updated data itself.

6.2.4.4 Directories Used

(1) Term Encoding Table.

(2) Item List.

(3) Fields File.*

(4) Shadow of Fields File.*

(5) Segment Name List.

*Note: The method of operation described herein assumes that ordered files will not be indexed, so these directories are not affected. If the scope of Update Data is enlarged to include the updating of indexed field values, then these directories will be affected.

6.2.4.5 Services Used

(1) Locate IL Entry.

(2) Open for Reading.

(3) Read.

(4) Close for Reading.

(5) Open for Writing.

(6) Write.

(7) Close for Writing.

(8) Open for Update.

(9) Seek.

(10) Replace.

(11) Close for Update.

(12) Retrieve Item.

(13) Term Name to ICC Translation.

6.2.4.6 Jobs Used. Conditional Search.

6.2.4.7 Method of Operation. A is the program's name (formal) for the Data Base item which is to receive updated data. B is the program's name (formal) for the condition which will become the input to Conditional Search used as a job extension. C is the program's name (formal) for the list of IPC's produced by Conditional Search or by Update Data itself if Conditional Search is not executed. D is the program's name (formal) for the source file which contains the transaction or change records.

(1) A is retrieved and translated into an ICC. Then the Item List entry is retrieved and saved. If the ICC contains one or more R's, input B is retrieved. If a Boolean condition has been given, Conditional Search is executed to produce file C which will contain a list of all IPC's which satisfied the condition.

(2) If the target item's ICC contains one or more R's and input B is missing, an Explosion subroutine is executed to produce file C with all IPC's listed. This occurs at connector 1C.

(3) If the target item's ICC contains no R's (i.e., a primary node), control goes to connector 1B, where file C is produced with one record listing the ICC of A as an IPC.

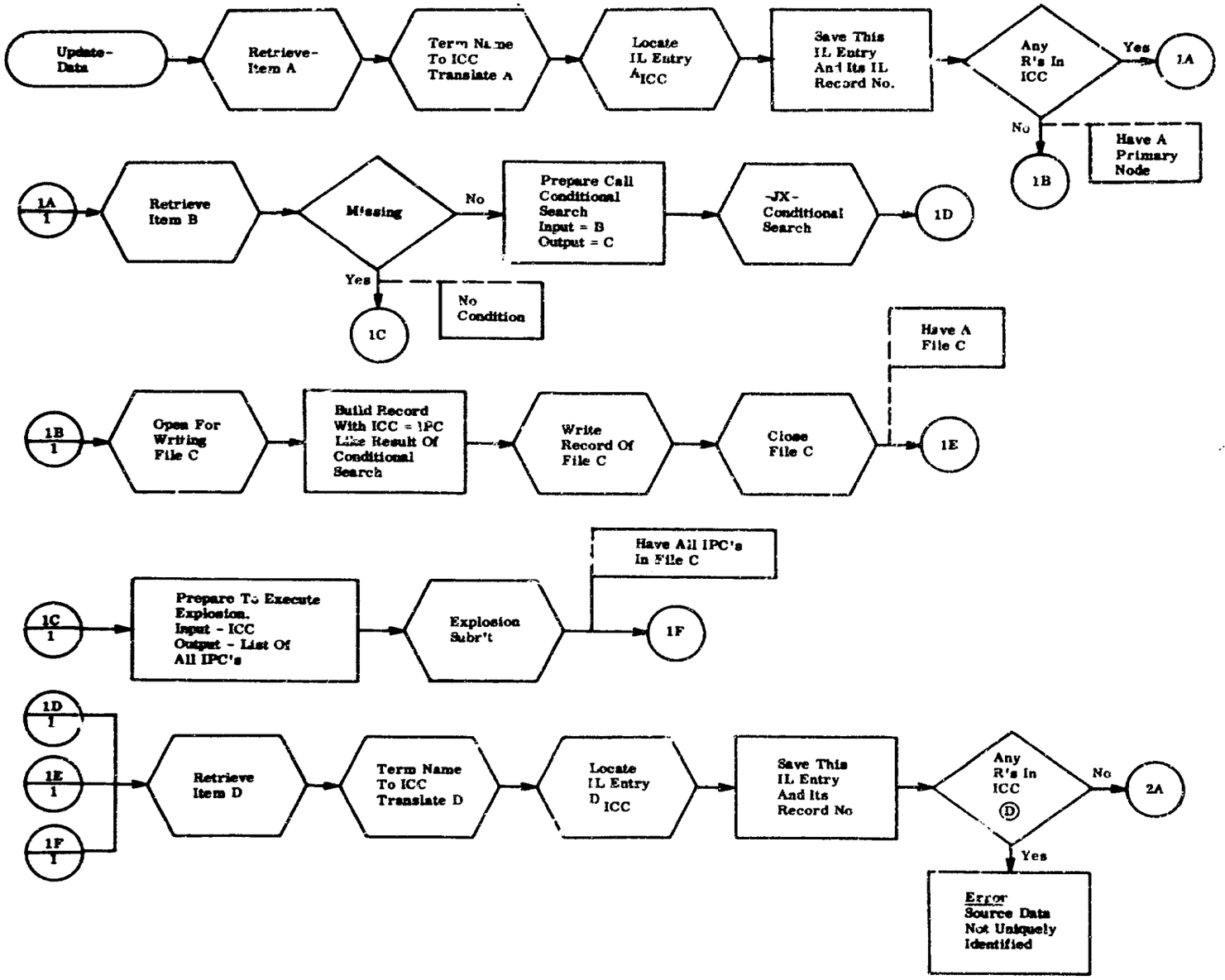
(4) File C now contains one or more IPC's locating the positions in the Data Base which are the targets for updating by the source data. At connectors 1D-1F, all paths join and the name of the source data is retrieved and translated into an ICC. If this ICC contains one or more R's, there is an error. Source data is expected to have a one-to-one relationship between name and IPC.

(5) At connector 2A, a series of validity checks is performed to assure that both source and target items are files and that both files have the same subsumed structures. Update Data operates on complete records only. Since updating will be accomplished through matching the values of key fields, both source and target files must be ordered by the same key field.

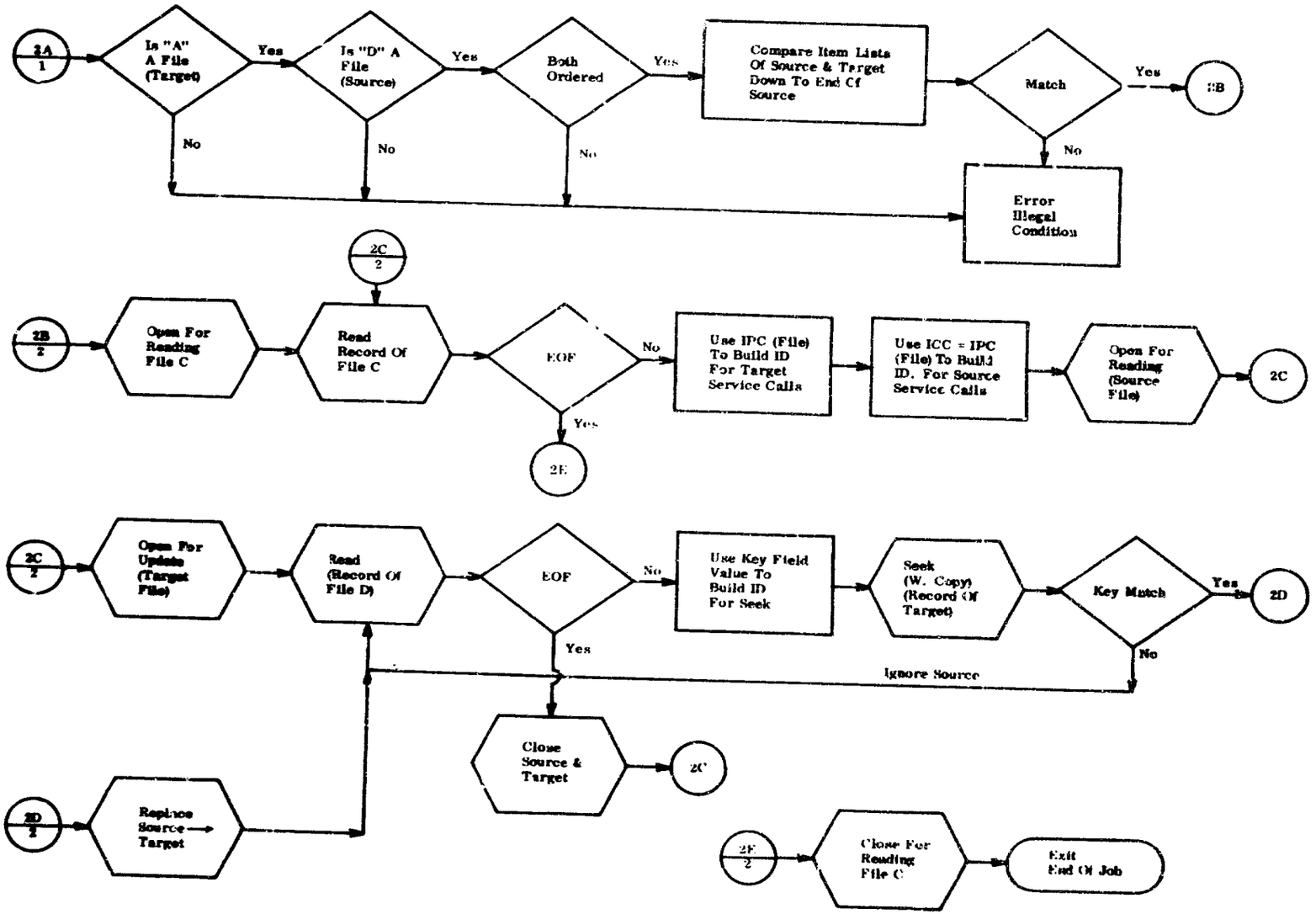
(6) The first target IPC is obtained from file C at connector 2B. Then, source file D is opened for reading and the target file is opened for updating. A record of source file D is read, and the value of its key field is used to build an identifier for positioning the target file. If the target file does not contain a record with a key field value matching the source record, the source record is skipped over and the next source record is read. When key values match, the source record replaces the target record in the target file.

(7) At EOF of source file D, both fields are closed, and control returns to connector 2C to repeat the process on the next target IPC.

(8) When file C finally reaches EOF, the Update Data job terminates.



Update-Data
Sheet 1 of 2



Update-Data
Sheet 2 of 2

6.2.5 Delete Data

Job Request: DELETE-DATA (item), (condition).

6.2.5.1 Functional Description. This job removes the data stored at the position specified by (item) and (condition) from the Data Pool. If the data named in (item) is primary (no R's in its ICC), the (condition) is not required because the ICC is equivalent to the position (IPC). If (condition) is used to specify a position, and if Conditional Search does not yield a unique position, the assumption is that the data is to be deleted from all positions which met the condition. If no condition is given, the assumption is that the data is to be deleted at all positions. This will not set the No Data Flag, because missing or null trace information is retained in the data segments.

6.2.5.2 Inputs. The job inputs are:

(1) (item) - the term name for the Data Base item. If this class of (item) is subsumed within the records of a file, the position at which the data is to be deleted should be specified by a condition.

(2) (condition) - a Boolean statement which specifies the position within the Data Base where the data deletion is to take place.

6.2.5.3 Results. The data is deleted.

6.2.5.4 Directories Used

(1) Term Encoding Table.

(2) Item List.

(3) Fields File.

(4) Shadow of Fields File.

(5) Segment Name List.

6.2.5.5 Services Used

(1) Locate IL Entry.

(2) Open for Reading.

(3) Read.

(4) Close for Reading.

(5) Open for Writing.

(6) Write.

(7) Close for Writing.

(8) Open for Update.

(9) Seek.

(10) Replace.

(11) Delete.

(12) Close for Update.

(13) Retrieve Item.

(14) Replace Item.

(15) Delete Data.

(16) Term Name to ICC Translation.

6.2.5.6 Jobs Used

(1) Conditional Search.

(2) Sort.

6.2.5.7 Method of Operation. A is the program's name (formal) for the data base item which is to be deleted. B is the program's name (formal) for the condition which will become the input to Conditional Search used as a Job Extension.

(1) A is retrieved, converted into an ICC, and the Item List entry for A is located. If there are no R's in the ICC (i.e., a primary item), the ICC which is an IPC is written into file C. If there is one or more R's in the ICC and if B is present, Conditional Search is used as a Job Extension to build file C. If there is one or more R's in the ICC and if B is not present, an Explosion is executed. Explosion increments each R in turn and writes out IPC's until it reaches EOF. Its output is file C with IPC's for all occurrences of a given ICC.

(2) The IPC's of file C are used one at a time to identify the position for deleting. If the No Data Flag is set in the Item List, the job terminates immediately.

(3) At connector 2A, the program branches, based on the item type of the item to be deleted.

(a) For a File. All subsumed items are examined in the Item List to discover if any fields are listed as indexed. If not, the file is marked as missing in the data segment, and the next IPC is retrieved from file C. If any subsumed fields are listed as indexed, the data itself is read and, for each indexed field in the data, the following record is written into scratch file D.

FILE D, F

FIELDS FILE RECORD NUMBER, I, 18
FIELD VALUE, B, V
R-VALUE, H, V

Then, the file is marked as missing in the data segment, and the next IPC is retrieved from file C.

(b) For a Statement. A statement is handled just like a file.

(c) For a Record. All subsumed items are examined in the Item List to discover if any fields are listed as indexed. If not, the record is deleted from the file and any following records are moved up to fill the gap. Then the next IPC is retrieved from file C. If any subsumed fields are listed as indexed, the data itself is read and a record is written into scratch file D for each indexed field. Then, each directly subsumed node is recorded as missing or null in the data. The record is not squeezed out of the file. The next IPC is retrieved from file C.

(d) For a Field. The field is retrieved from the data. If it is an indexed field, a record is written into scratch file D. The field is marked missing or null in the data. Then the next IPC is retrieved from file C.

(4) When EOF is reached in file C, control goes to connector 'E. If there has been no writing into scratch file D (i.e., no indexed fields involved), the Delete Data job terminates.

(5) If file D contains data, it is sorted by using the complete record as a key. This brings file D into the same order as the Fields file and Shadow of Fields file. For each different Fields File Record Number in file D, the following directory updating is performed, starting at connec 4F.

(6) The proper Fields File record is located in the Field file. Scratch file E is prepared for writing as follows:

FILE E, F

RVIT FILE IDENTIFIER, H, V
R-VALUE FILE, F

R-VALUE, H, V

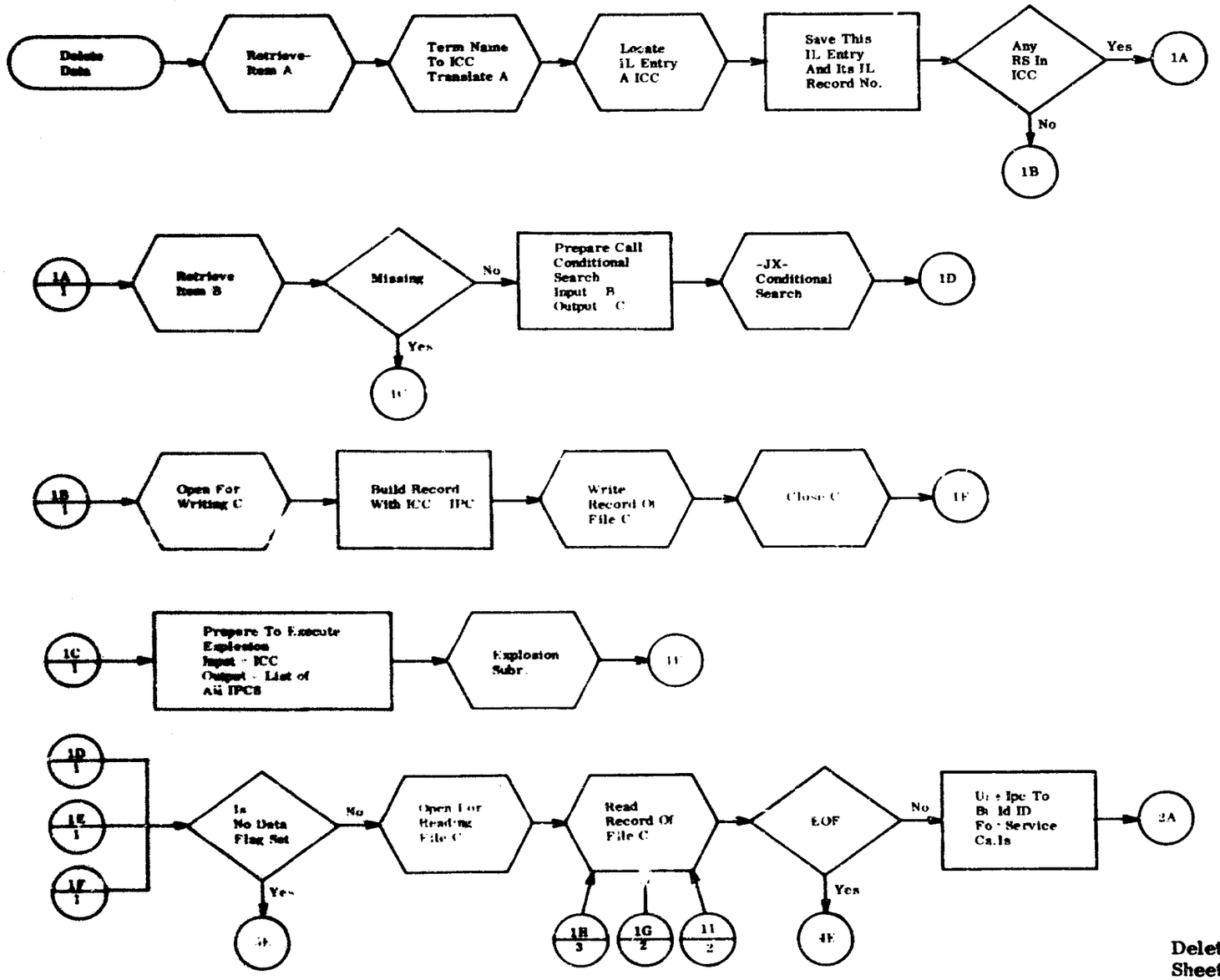
For each match between a field value in file D and a field value in the FVT file, a record is written in scratch file E. The records are counted as written. When the Fields File Record Number in file D changes, the Value/Range Occurrences field in FVT file is decremented.

(7) When EOF is reached in file D, Fields file updating is complete, and file E is used to control the updating of the Shadow of Fields file.

(8) For each different RVIT file identifier in file E, the following procedure is followed, starting at connector 6A.

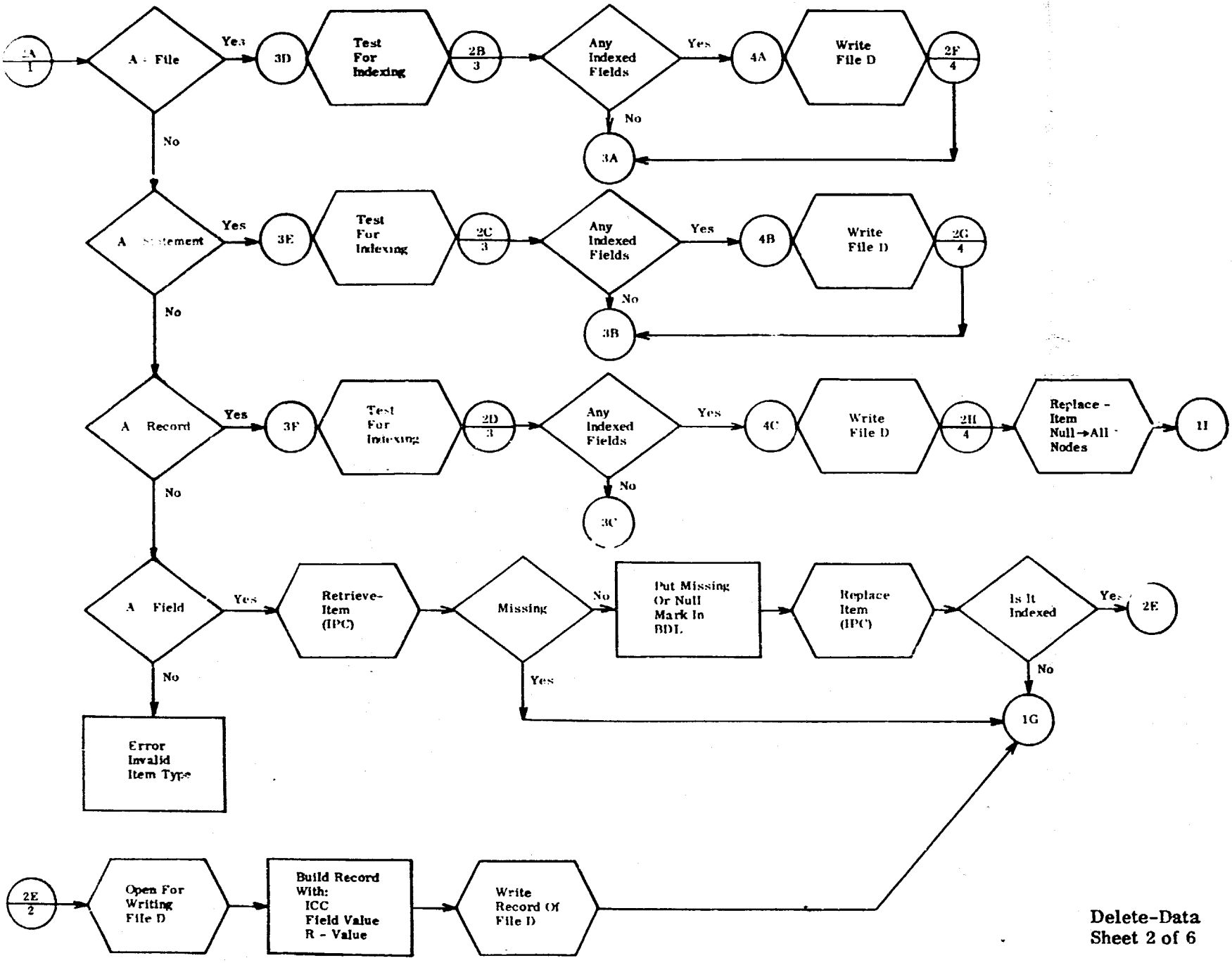
(a) The proper RVIT file is located by using the RVIT file identifier in file E. The records of the RVIT file are read, and each record matching an R-value in file E is deleted from the RVIT file.

(b) When all RVIT file identifiers in file E have been processed, and EOF is reached, Delete Data terminates.

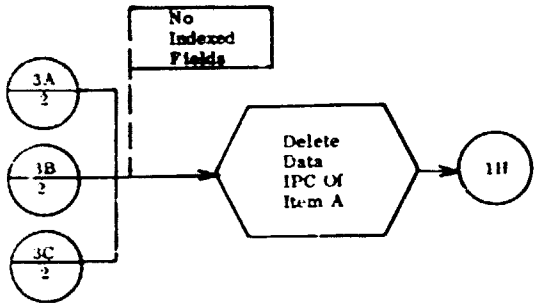


6-56

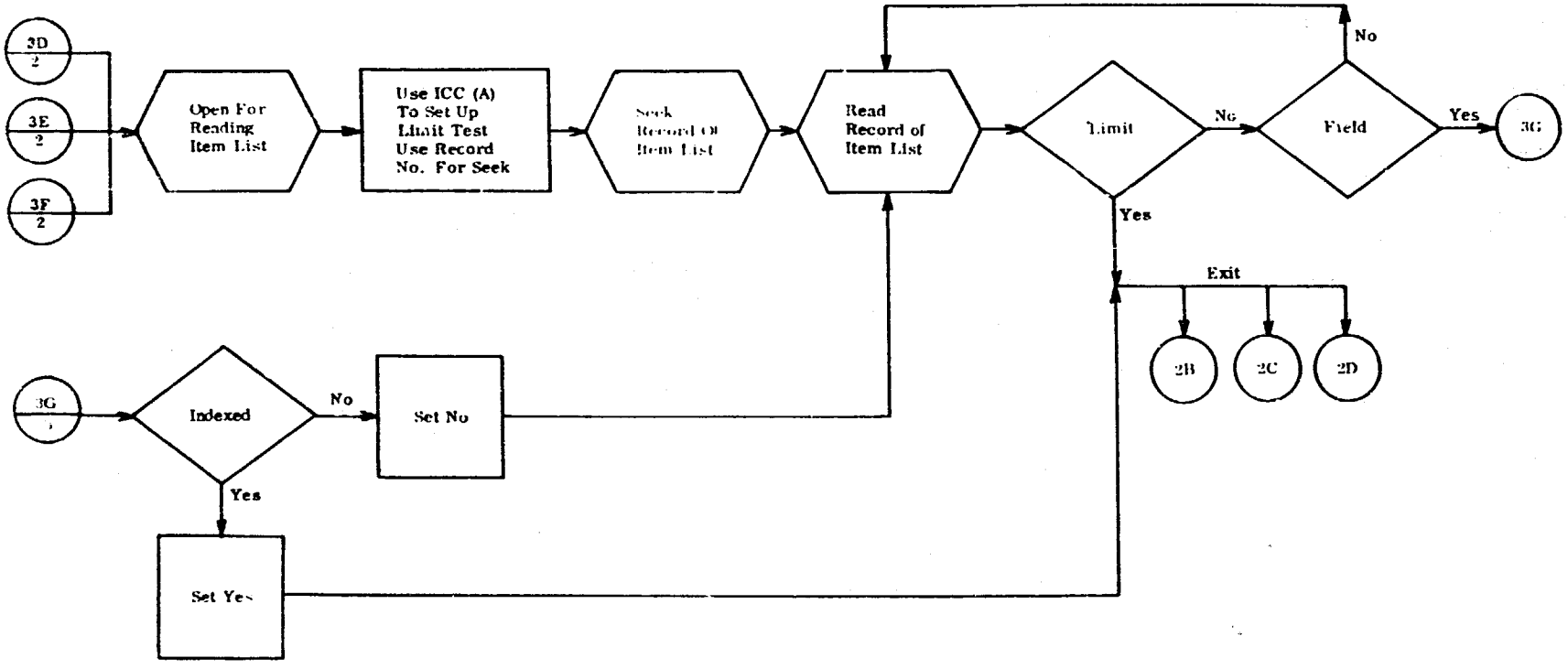
Delete-Data
Sheet 1 of 6



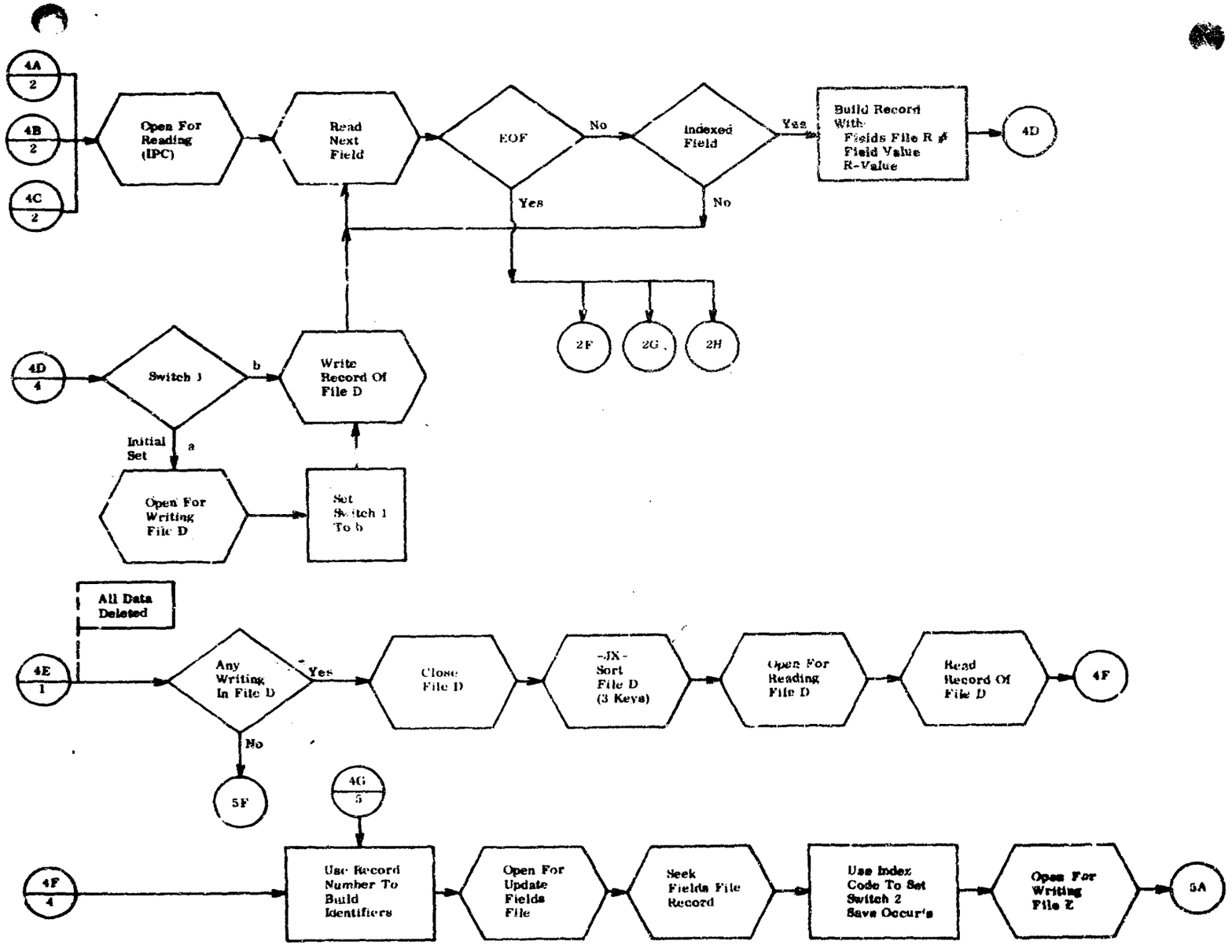
Delete-Data
Sheet 2 of 6



Subroutine To Test For Indexing

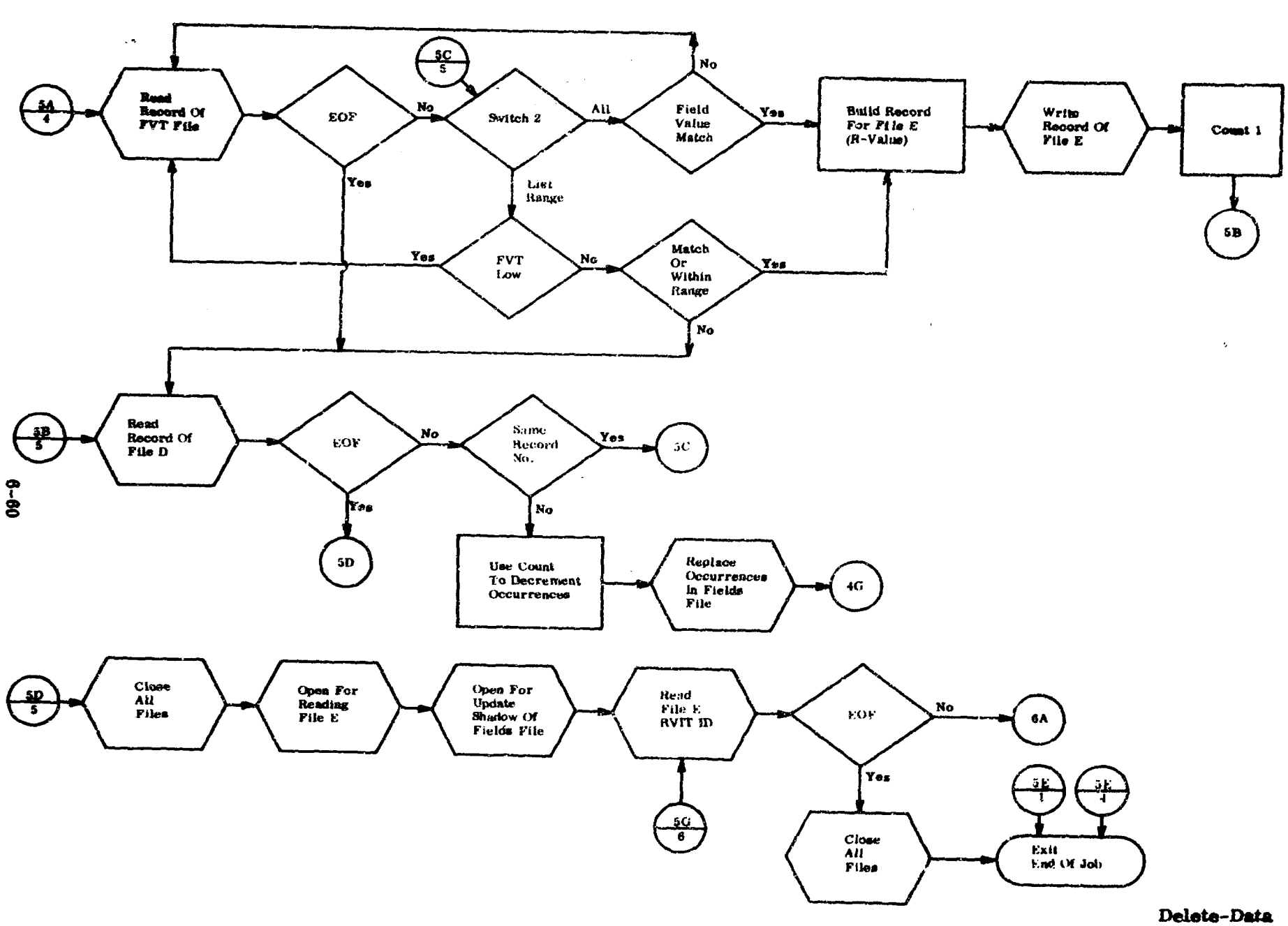


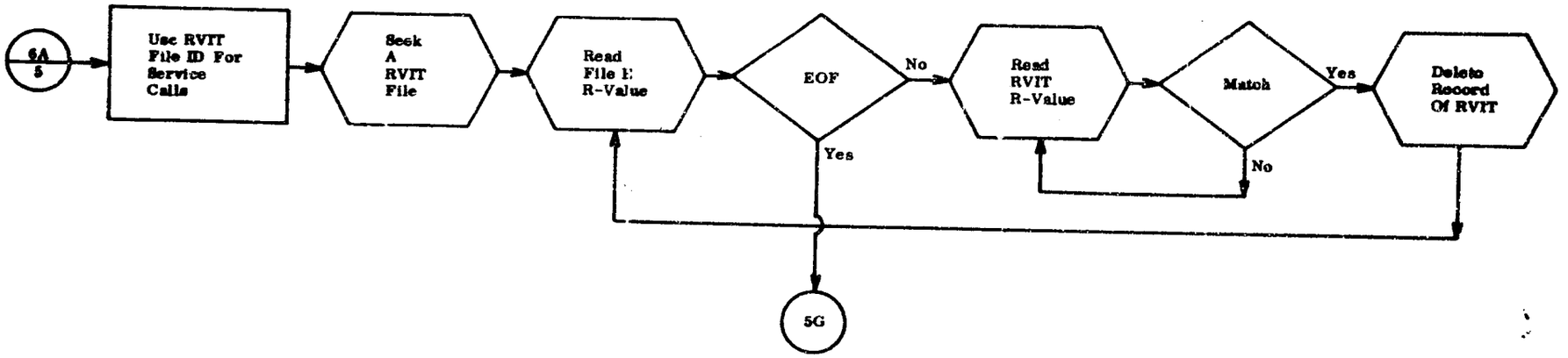
Delete-Data
Sheet 3 of 6



6-56

Delete-Data
Sheet 4 of 6


5A/4
Read Record Of FVT File
EOF
No
Yes
5C/5
Switch 2
All
List Range
Field Value Match
No
Yes
Build Record For File E (R-Value)
Write Record Of File E
Count 1
5B
FVT Low
Yes
No
Match Or Within Range
Yes
No
5B/5
Read Record Of File D
EOF
No
Same Record No.
Yes
3C
Yes
5D
No
Use Count To Decrement Occurrences
Replace Occurrences In Fields File
4G
5D/5
Close All Files
Open For Reading File E
Open For Update Shadow Of Fields File
Read File E RVIT ID
5G/6
EOF
No
6A
Yes
Close All Files
5K/1
5K/4
Exit End Of Job



Delete–Data
Sheet 6 of 6

6.2.6 Compress File

Job Request: COMPRESS-FILE (item), (condition).

6.2.6.1 Functional Description. When records are deleted from a file which has indexed fields, the records are not squeezed out because this causes a "ripple" effect in the directory table which stores R-values for indexed fields. Instead, the records are deleted by marking all directly subsumed nodes as missing or null in the data. This prevents the ripple effect at a small expense in storage space.

Compress File will squeeze out all records containing only missing or null data and will reassign record numbers as a contiguous set from 1 to n. Compress File will operate on ordered and/or nonindexed files, but is designed to complete the record deletion process on indexed files. If no condition is given, the assumption is that compression is to take place on all files of the class. If (condition) is used to specify a position, and if Conditional Search does not yield a unique position, the assumption is that compression is to take place at all positions that met the condition. Compress File operates on directly subsumed nodes; subsumed files are not compressed.

6.2.6.2 Inputs. The two job inputs are:

(1) (item) - The term name for the Data Base file. If this class of file is subsumed within the records of another file, the position at which file compression is to be accomplished should be specified by a condition.

(2) (condition) - a Boolean statement which specifies the position within the Data Base of the file to be compressed.

6.2.6.3 Results. The compressed file is the only result of this job.

6.2.6.4 Directories Used

(1) Term Encoding Table.

(2) Item List.

(3) Fields File.

(4) Shadow of Fields File.

(5) Segment Name List.

6.2.6.5 Services Used

(1) Locate IL Entry.

(2) Open for Reading.

(3) Read.

(4) Close for Reading.

(5) Open for Writing.

(6) Write.

(7) Close for Writing.

(8) Open for Update.

(9) Seek.

(10) Delete.

(11) Replace.

(12) Close for Update.

(13) Retrieve Item.

(14) Term Name to ICC Translate.

6.2.6.6 Jobs Used

(1) Conditional Search.

(2) Sort.

6.2.6.7 Method of Operation. A is the program's name (formal) for the file which is to be compressed. B is the program's name (formal) for the condition which will become the input to Conditional Search used as a Job Extension.

(1) A is retrieved, converted into an ICC, and the Item List entry for A is located.

(a) If there are no R's in the ICC (i.e., a primary file), the ICC which is an IPC is written into file C.

(b) If there is one or more R's in the ICC and if B is present, Conditional Search is used as a Job Extension to build file C.

(c) If there is one or more R's in the ICC and if B is not present, an Explosion is executed. Explosion increments each R in turn and writes out IPC's until it reaches EOF. Its output is file C with IPC's for all occurrences of a given ICC.

(2) The IPC's in file C will be used one at a time, and, for each IPC in file C, a file will be compressed. The return point in this outermost loop is to connector 2F. Before getting into this loop, scratch file D is written to include the ICC's of all subsumed indexed fields. This takes place at connector 2A.

(3) Starting at connector 2C, each directly subsumed node of record 1 is read (or the first part of a nonterminal item is read) to discover whether or not this node in this record is marked as missing or null in the data. As soon as any data is transmitted to the buffer as a result of a Read, the current record is discarded, and node 1 of the next record is read. When EOF is reached, this file is closed, the next IPC is read from file C, and control returns to connector 2F.

(4) Once all nodes of a record have been read and discovered to be missing or null, control goes to connector 2D.

(5) For each indexed field in an empty record, a record is written into scratch file E to include:

R-VALUE OF FILE	AMT OF DECREMENT	OLD RECORD NUMBER	ICC OF INDEXED FIELD

The number of records in scratch file D determine the number of records written into file E for each empty record. Then, the empty record is deleted from the file, and control returns to connector 2E to process the succeeding record.

(6) At connector 3C, after all IPC's in file C are exhausted, all files have been compressed, and the job terminates unless indexed fields have been involved and records have been written into scratch file E.

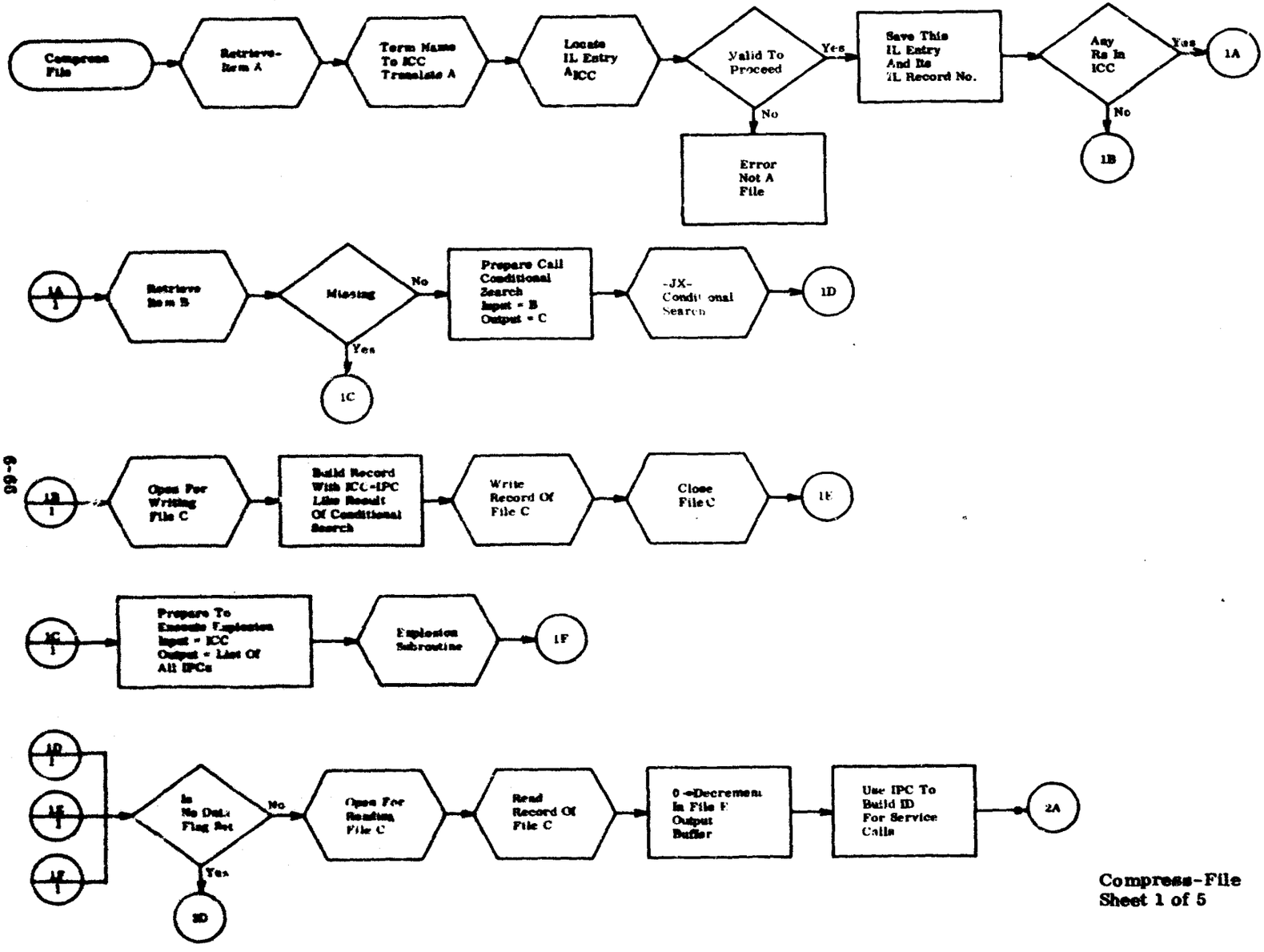
(7) File E is sorted at connector 4A by ICC, R-value of File, and Old Record Number. Each ICC in file E will lead to a separate FVT file, and from there to a set of RVIT files which contain the R-Values which must be decremented to account for the squeezed out records.

(8) For each ICC in file E, the item list is accessed to retrieve the record number needed to identify the FVT file. The number of records in the FVT file is determined because there is an RVIT file for each record in the FVT file.

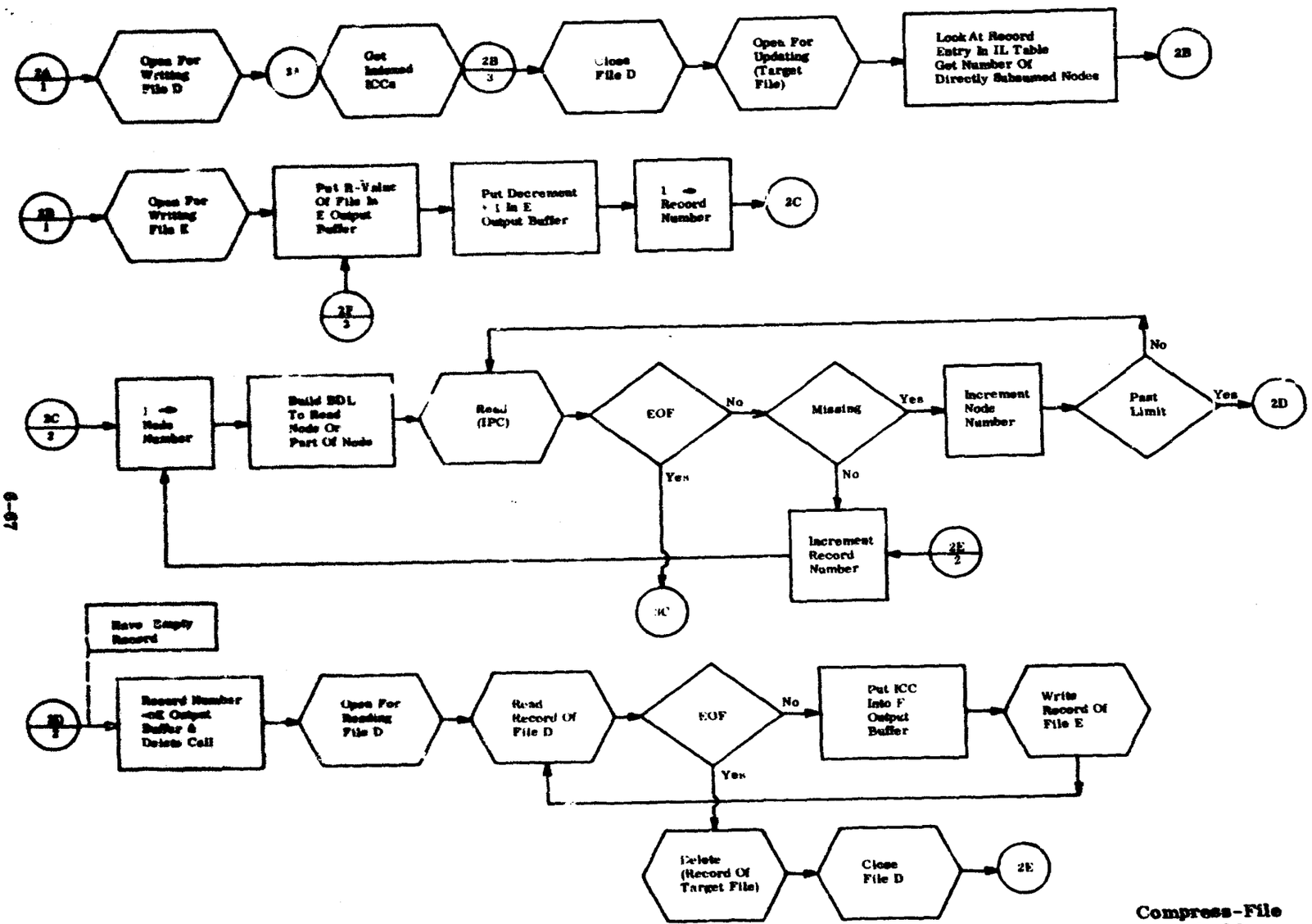
(9) At connector 4C, an RVIT file is scanned to find the section pertaining to the file which has been compressed. When the proper section is reached, any R-values beyond the squeezed-out records are decremented by the number of deleted records.

(10) When EOF is reached in an RVIT file, the next RVIT file is processed until all RVIT files pertaining to a single ICC are exhausted. Then, the next ICC from file E is selected as per step (8).

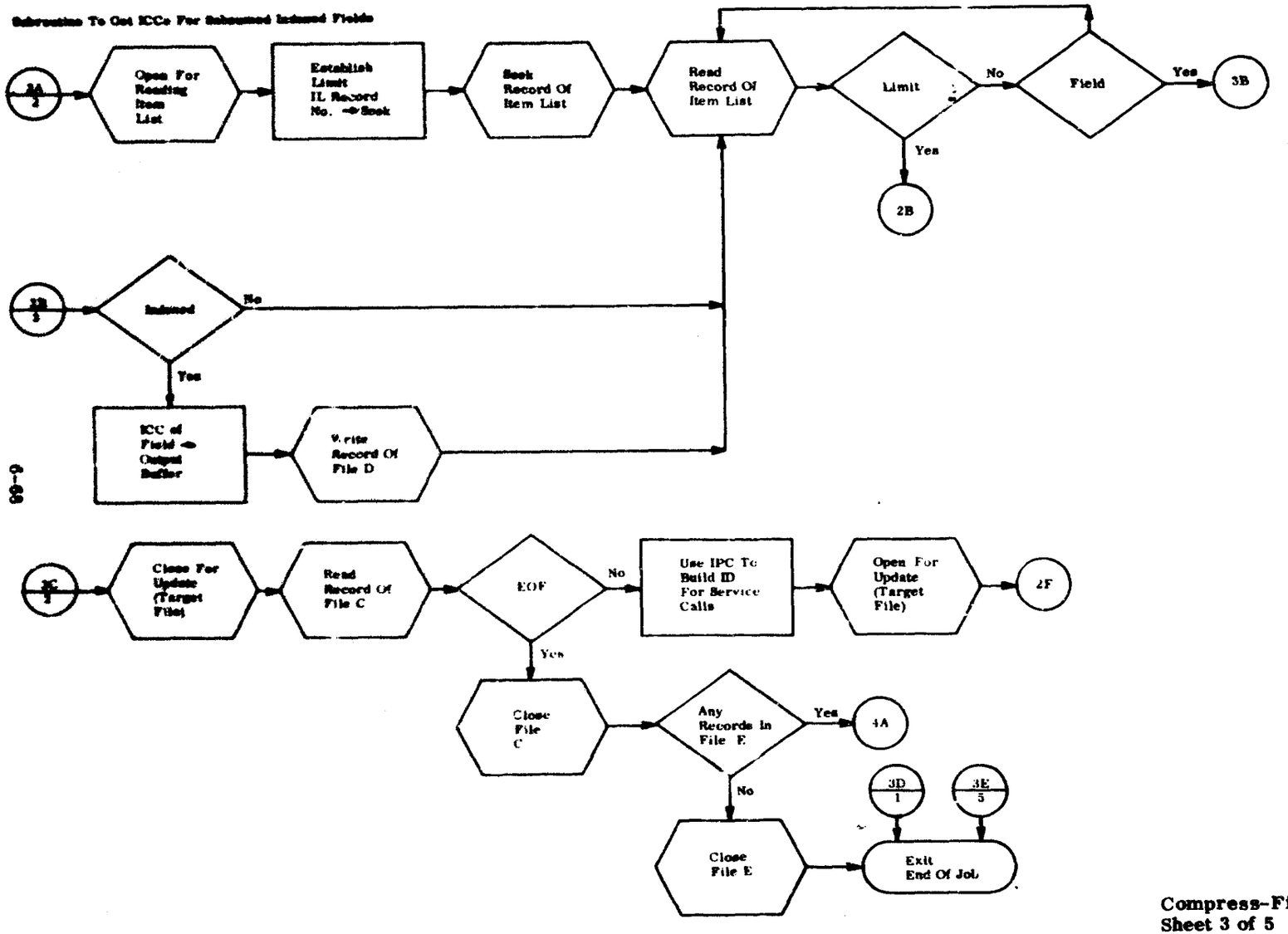
(11) When all ICC's in file E have been handled, all R-values in all RVIT's affected by all file compressions have been corrected, and Compress File terminates.



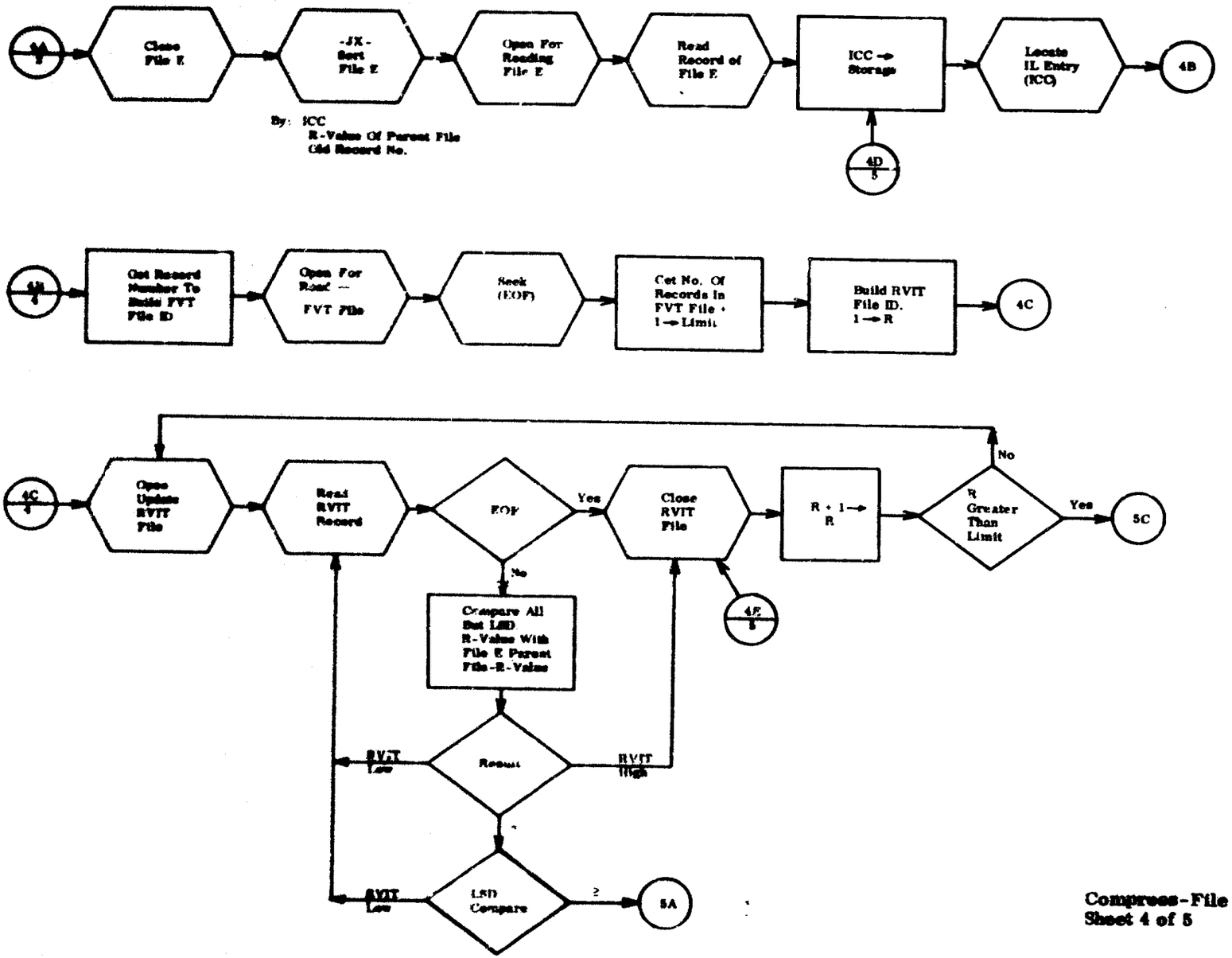
Compress-File
Sheet 1 of 5



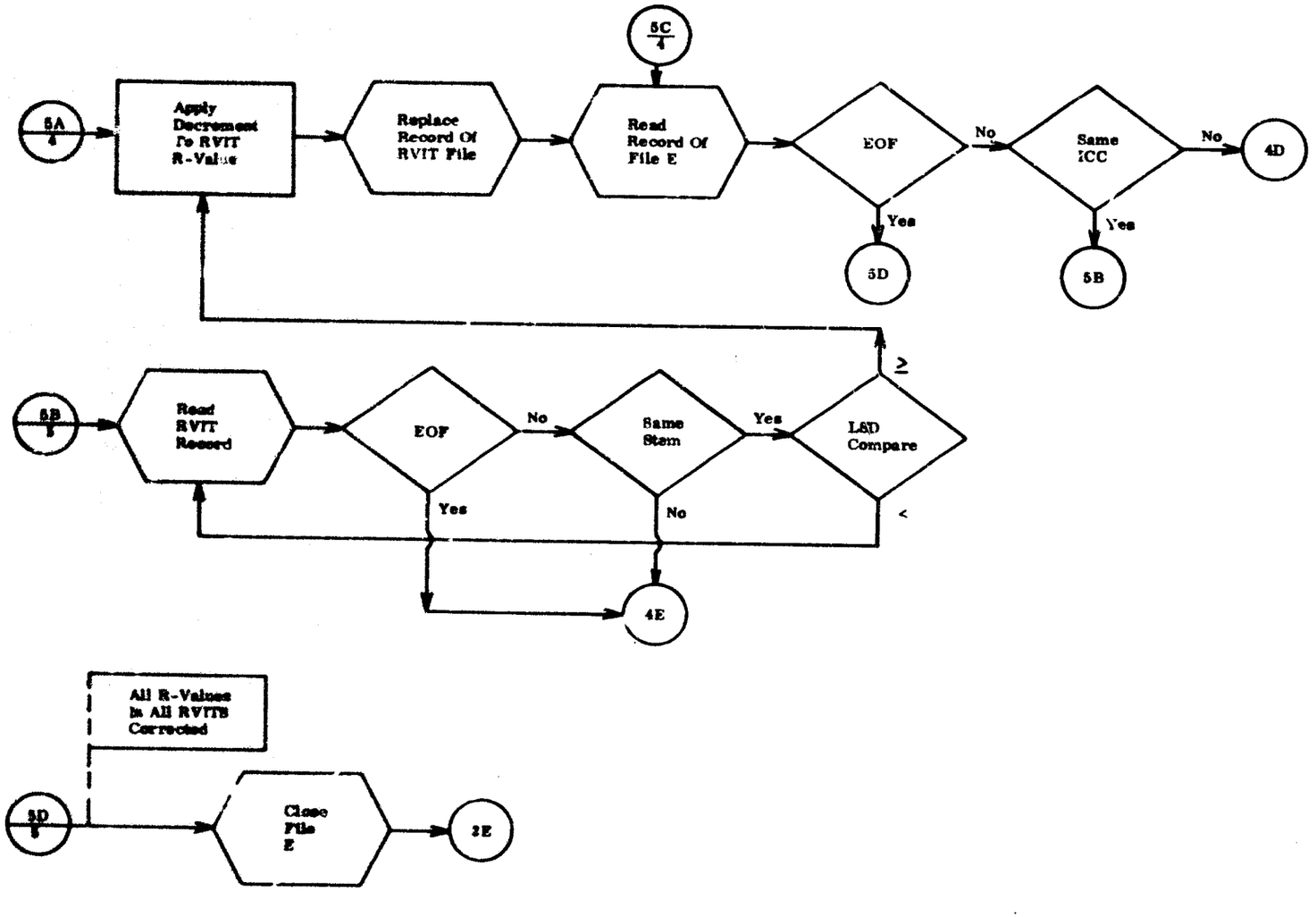
Compress-File
Sheet 2 of 5



Compress-File
Sheet 3 of 5



Compress-File
Sheet 4 of 5



6-70

Compress-File
Sheet 5 of 5

6.3 INDEXING

Indexing is the process of recording field values and record numbers in the directory so that retrieval requests can be satisfied without searching through unwanted data. Once a field is indexed, all subsequent data additions will cause automatic indexing by the same mode. Data deletions will cause automatic deletion of the record numbers cross-referenced to the field values of the deleted fields. An indexed field can be un-indexed through Remove Index if the usage for speedy retrieval does not justify the additional directory storage space and maintenance job running time. These jobs are primarily concerned with maintenance of the directory elements identified as Fields file and Shadow of Fields file.

6.3.1 Index

Job Request: INDEX (field), (mode), (list).

6.3.1.1 Functional Description. This job operates on fields within the Data Base or directories. It retrieves all occurrences of the field named in (field), cross-references the field values to the R-values, and stores this information in the directories. The method of grouping the field values is specified in (mode), and the (list) provides the groupings. The inverse of this job is Remove Index.

6.3.1.2 Inputs. The inputs to the Index job are (field), (mode), and (list). Field is the term name (qualified) for the data base field which is to be indexed. Mode is the word ALL, LIST, or RANGE. If ALL is specified, no third input is required because all field values will be listed in the index table. If LIST or RANGE is specified, the (list) input is a list of individual values or a list of ranges which represent the groupings of field values which are of interest.

6.3.1.3 Results. The index tables are updated as a result of the execution of this job.

6.3.1.4 Directories Used

(1) Term Encoding Table.

(2) Item List.

(3) Fields File.

(4) Shadow of Fields File.

(5) Segment Name List,

6.3.1.5 Services Used

(1) Locate IL Entry.

(2) Open for Reading.

(3) Seek.

(4) Read.

(5) Close for Reading.

(6) Open for Writing.

(7) Write.

(8) Close for Writing.

(9) Open for Update.

(10) Insert.

(11) Replace.

(12) Close for Update.

(13) Retrieve Item.

(14) Replace Item.

(15) Term Name to ICC Translation.

6.3.1.6 Jobs Used. Sort.

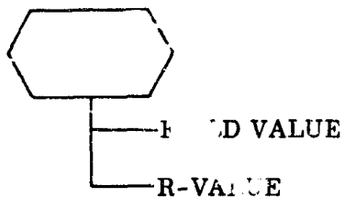
6.3.1.7 Method of Operation.

(1) Page 1 processes the three job inputs: (field), (mode), and (list). A is the program's name (formal) for the alphanumeric string which contains the term name for the field to be indexed, with whatever qualifiers are necessary for uniqueness. B is the program's name (formal) for the alphanumeric string which contains the mode of indexing; ALL, LIST or RANGE. C is the program's name (formal) for the optional list of values or ranges to be used for indexing modes LIST or RANGE. After translating the term name into an ICC,

the Item List is updated to include the new index code and the record number link to the Fields file. Then, for modes LIST or RANGE, file C is read and the Fields file Record is built to include the FVT file. For mode ALL, the Fields file Record has a missing FVT file.

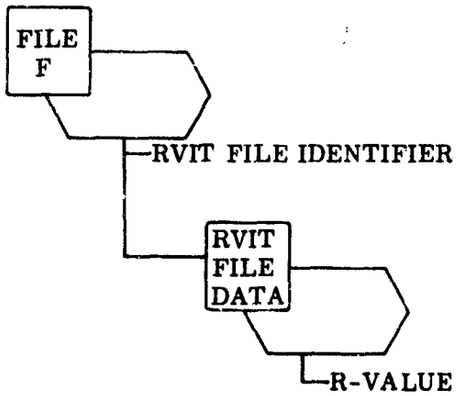
(2) At connector 2E for mode ALL, a missing Shadow of FVT file is inserted to correspond to the missing FVT file. At connector 2A for modes LIST and RANGE, the Shadow of FVT file gets a number of records equal to the number of records in the FVT file. Each of these records has a missing RVIT file.

(3) At connectors 2B and 2C, all paths join. Here the data is actually retrieved, and, for each occurrence of the field in the data, a record is built as follows:



These records are written into a scratch file D. Task 1 ends here, because the remaining work is part of another job (Add Data) as well as the Index job.

(4) Task 2 sorts scratch file D by field value (major) and R-value (minor), because both the FVT and RVIT are ordered files. E is the sorted file D, and F is a scratch file which will be written with the following structure:

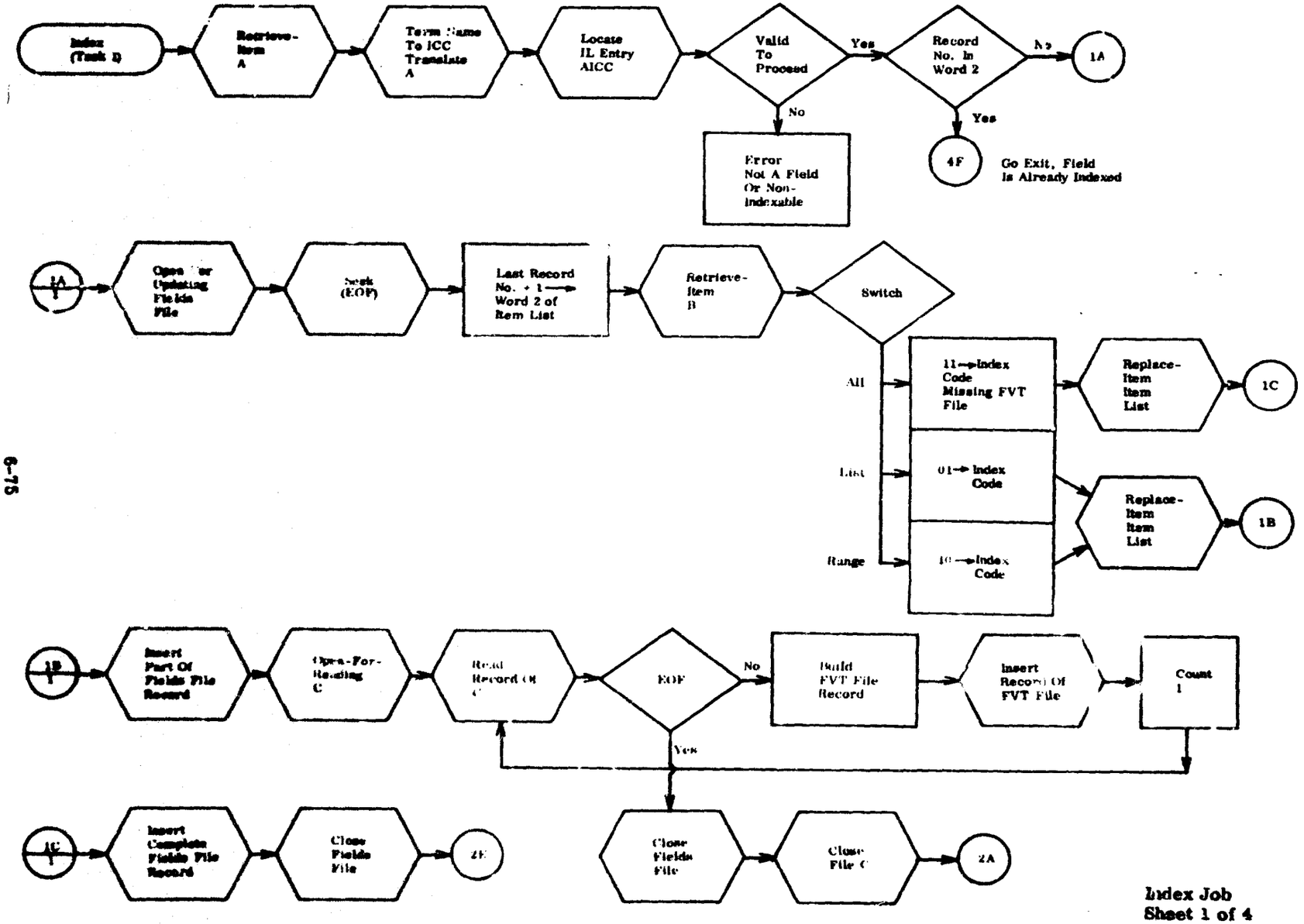


(5) Page 3 reads file E, inserts the field values in the FVT file, and writes the R-values into file F. This is for a newly indexed field, mode ALL, which has had a missing FVT file until this step.

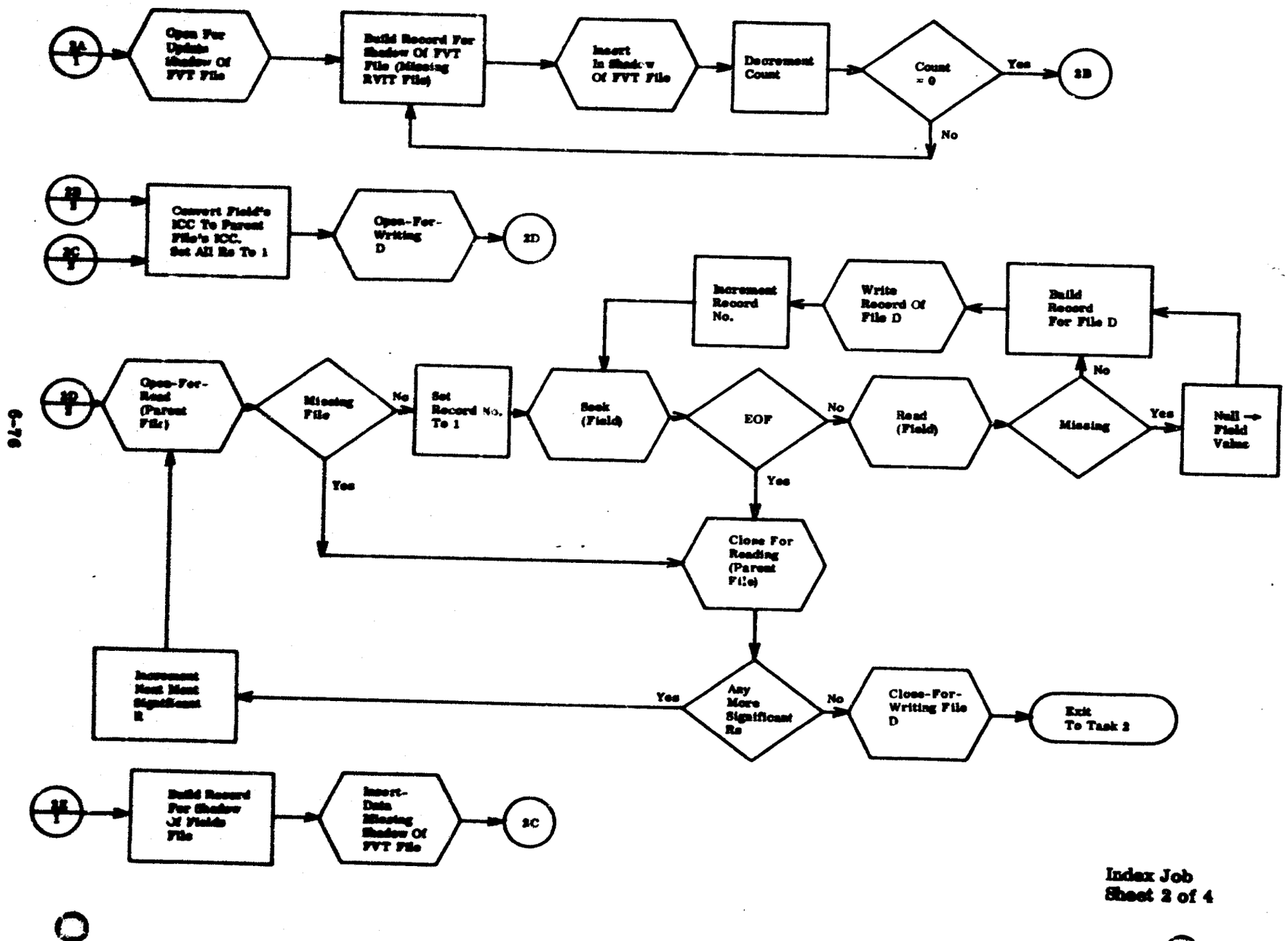
(6) Page 4 performs the same function for mode LIST/RANGE where each new field value has to be examined to see if it fits any of the LIST or RANGE entries in the FVT file. If it does not fit, it is discarded, but if it does fit, the R-value is written into file F.

(7) Connector K is common to all paths. Here the RVIT file is updated by inserting the R-values into the file in ascending order. Each field value in the FVT applies to a different RVIT file and the RVIT file identifier in file F is used to select the proper RVIT files.

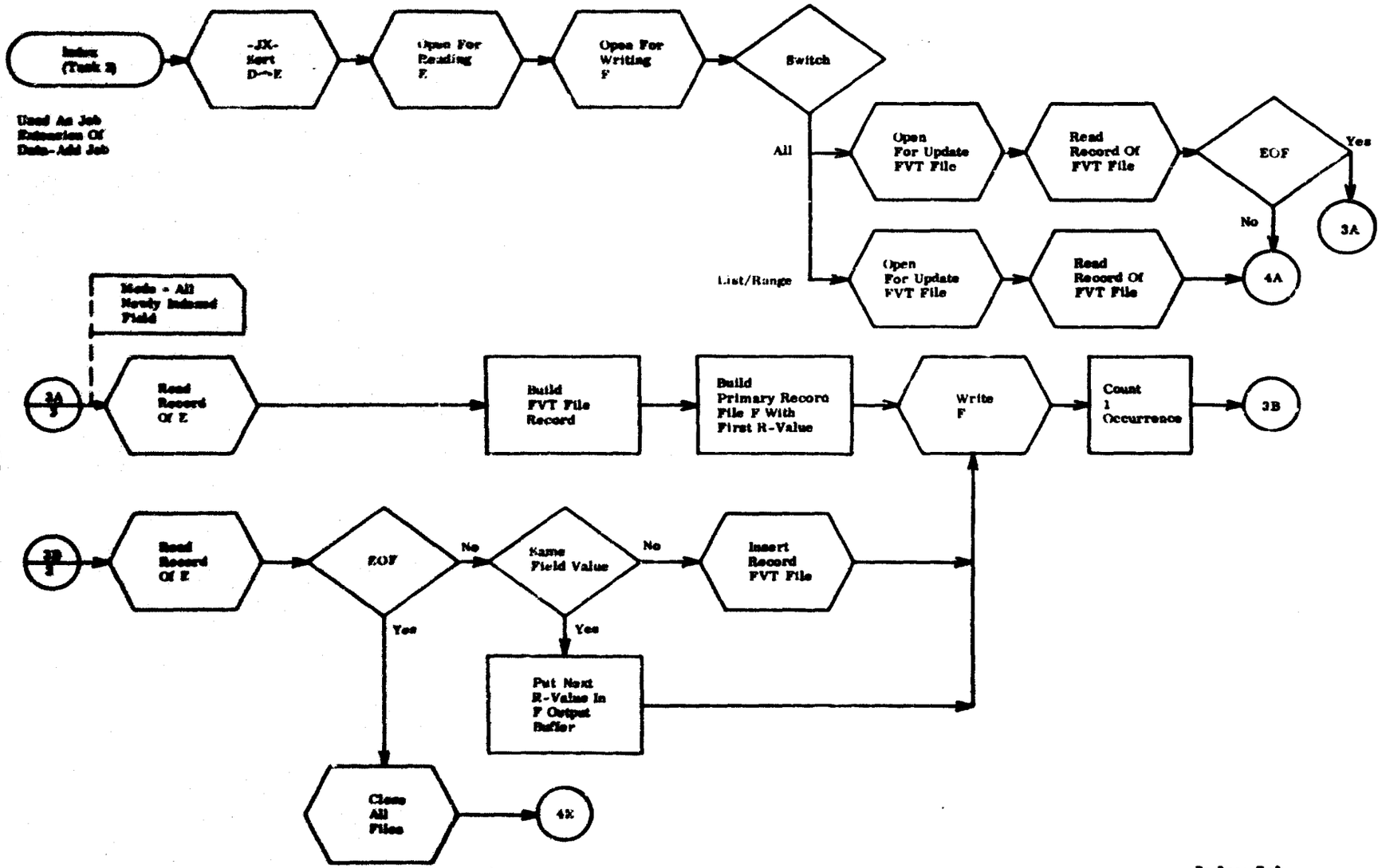
(8) On reaching EOF in file F, task 2 terminates and this is the end of the job.



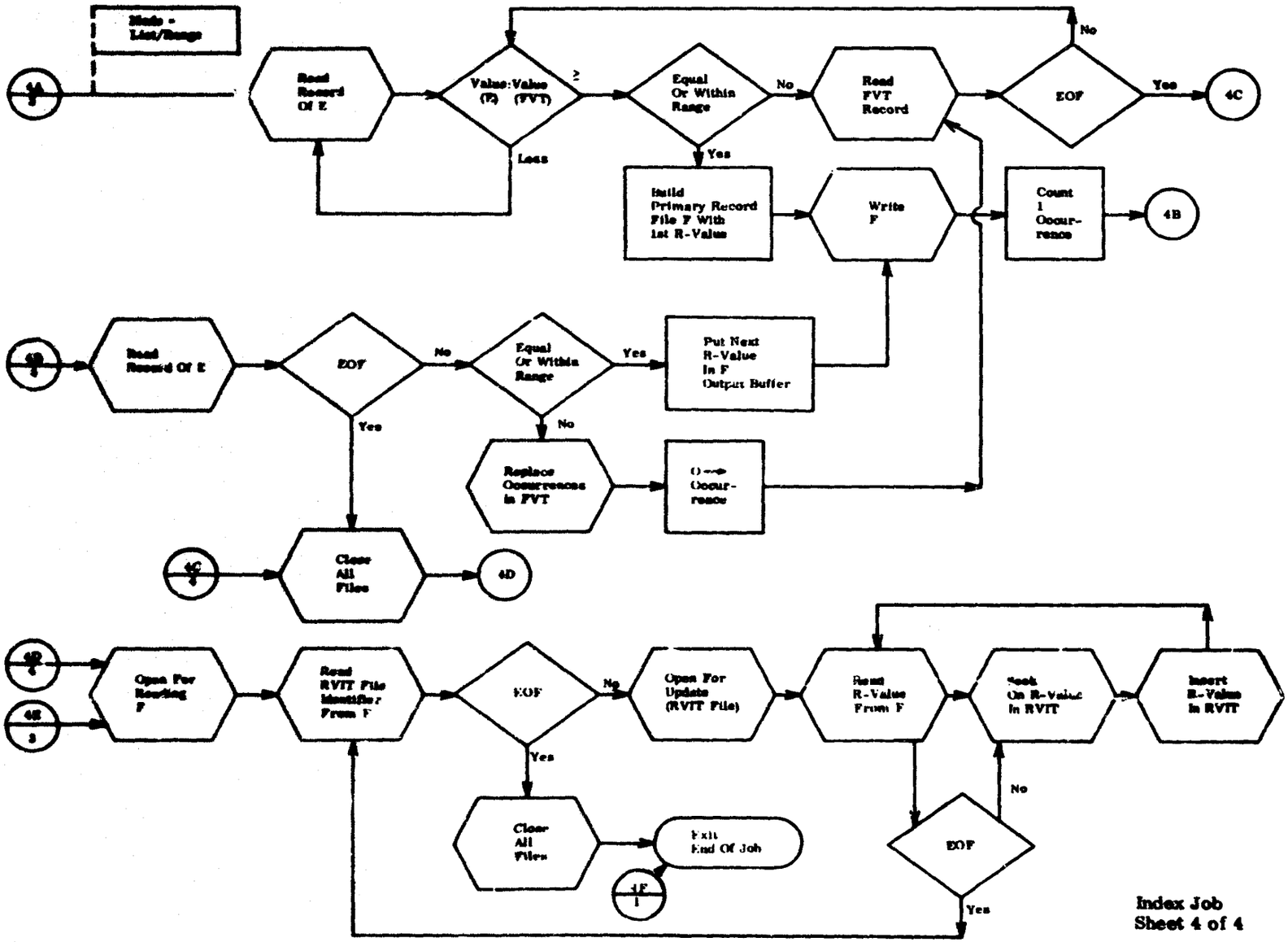
Index Job
Sheet 1 of 4



Index Job
Sheet 2 of 4



Index Job
Sheet 3 of 4


Mode - List/Range
4A
Read Record Of Z
Value:Value (F) (FVT)
≥
Equal Or Within Range
No
Read FVT Record
EOF
No
Yes
4C
Less
Yes
Build Primary Record File F With 1st R-Value
Write F
Count 1 Occur-rence
4B
4B
Read Record Of Z
EOF
No
Equal Or Within Range
Yes
Put Next R-Value In F Output Buffer
Yes
No
Replace Occurrences in FVT
0 → Occur-rence
4C
Close All Files
4D
4D
4E
Open For Reading F
Read RVIT File Identifier From F
EOF
No
Open For Update (RVIT File)
Read R-Value From F
Seek On R-Value In RVIT
Insert R-Value In RVIT
Yes
No
Close All Files
Exit End Of Job
4F
EOF
Yes
Index Job
Sheet 4 of 4

6.3.2 Remove Index

Job Request: REMOVE-INDEX (field).

6.3.2.1 Functional Description. This job is the inverse of Index. It destroys the directories which cross-reference field values to R-values and marks the field as unindexed in the item list.

6.3.2.2 Inputs. The only input is the term name (qualified) for the field which is to be indexed.

6.3.2.3 Results. The index tables are updated to include the input field.

6.3.2.4 Directories Used

(1) Term Encoding Table.

(2) Item List.

(3) Fields File.

(4) Shadow of Fields File.

(5) Segment Name List.

6.3.2.5 Services Used

(1) Locate IL Entry.

(2) Open for Update.

(3) Seek.

(4) Delete.

(5) Close for Update.

(6) Replace Item.

(7) Retrieve Item.

(8) Term Name to ICC Translation.

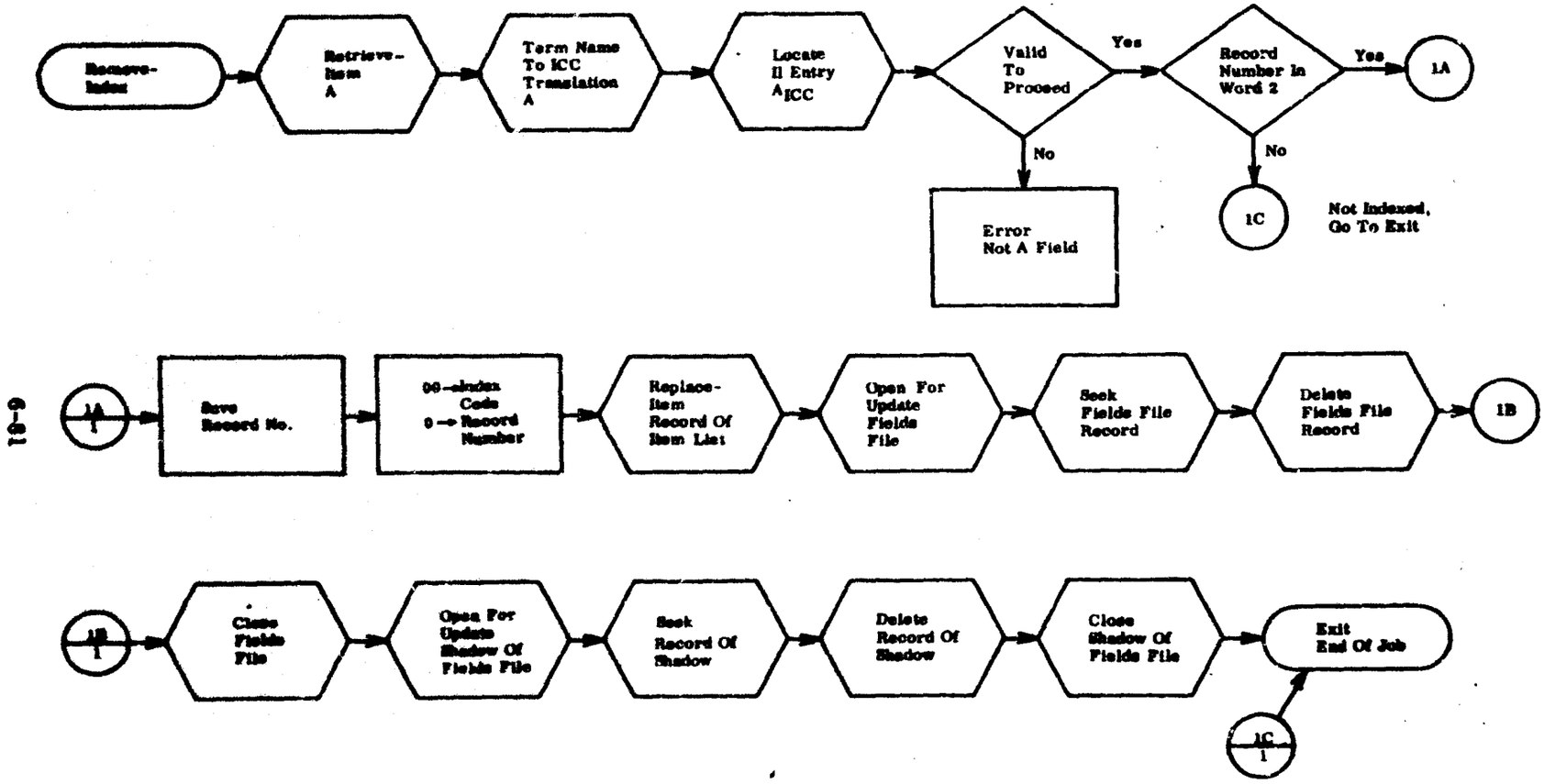
6.3.2.6 Jobs Used. None.

6.3.2.7 **Method of Operation**

(1) The only job input is (field) The program uses the formal name A for the alphanumeric string which contains the term name for the field with whatever qualifiers are necessary for uniqueness. A is retrieved and translated into an ICC. Then the Item List entry is retrieved and the record number link to the Fields file and Shadow of Fields file is saved in storage at connector 1A.

(2) Then, the IL entry is modified to include index code 00, meaning not indexed, and the record number is replaced by zeros.

(3) Using the record number of find the proper place in the fields file, the whole record is deleted. Then the same record number is used to find the proper place in the Shadow of Fields file, and that whole record is deleted. Then the job terminates.



10-01

Remove-Index
Sheet 1 of 1

6.4 LINKAGE

Linkage is a mechanism for expressing a relationship between different data structures. In AIMS, the item definition describes the structure and the data is stored in sequence as structurally defined. If it is desirable to physically separate data and maintain its relationship, or to relate data stored in separate structures, then Linkage is the appropriate mechanism.

The Link job adds source and target link items to the Item List and records the structural relationship in the linkage table of the directory. Remove Link is the inverse of Link.

Before two structures can be linked, each must contain a field of the same size and type, because it is through the field values of the subsumed field that the actual data linkage is accomplished.

6.4.1 Link

Job Request: LINK (source), (target).

6.4.1.1 Functional Description. This job relates or ties an item in one structure to an item in another structure by introducing source and target link items which subsume the link fields. The Link job introduces the source and target link items into the two structures and builds a Linkage Table entry for the directory.

6.4.1.2 Inputs. The phrases (source) and (target) have the form:

link item name (field name)

The source and target link items are defined to the system when the Link job is executed. The source and target field items must have been defined earlier through a Define Item job. If the two field items have the same term name, qualifiers must be used to identify the two structures now being linked.

6.4.1.3 Results. The link items are defined in the system directories.

6.4.1.4 Directories Used

(1) Term Encoding Table.

(2) Item List.

(3) Term List.

(4) Linkage Table.

(5) Segment Name List.

6.4.1.5 Services Used

(1) Locate IL Entry.

(2) Open for Update.

(3) Seek.

(4) Insert.

(5) Close for Update.

(6) Retrieve Item.

(7) Replace Item.

(8) Term Name to ICC Translate.

6.4.1.6 Jobs Used. None.

6.4.1.7 Method of Operation. A is the program's name (formal) for the (source) input, and B is the program's name (formal) for the (target) input.

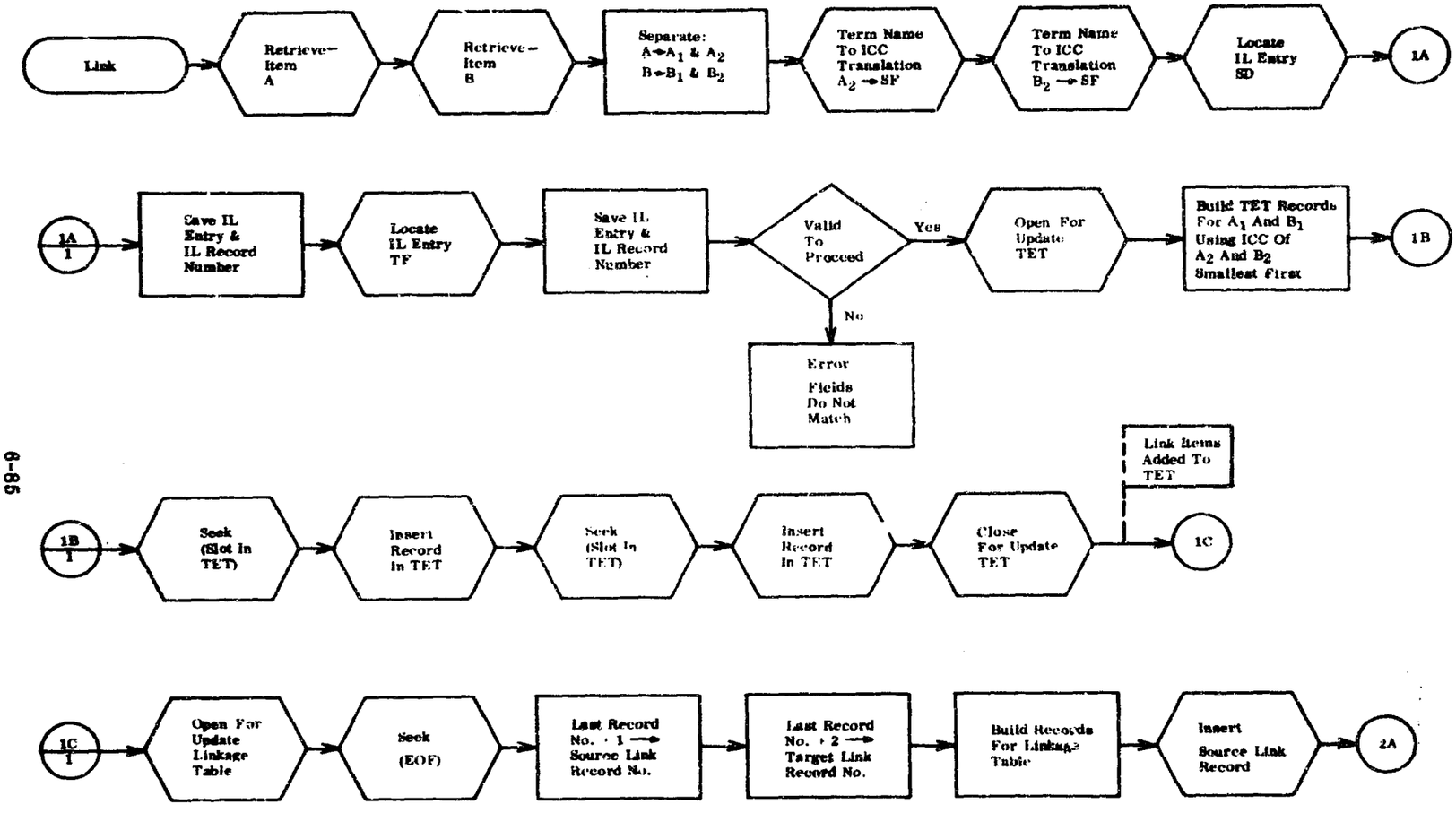
(1) A and B are retrieved and separated into link item name and field name. The two field names are translated into ICC's called SF and TF. Then, the Item List entries for these two fields are retrieved and a check is made to assure that the fields are suitable for becoming the criterial fields for accomplishing linkage.

(2) Next, the term names for the two new link items are added to the Term Encoding Table. The ICC's formerly assigned to the fields are now assigned to the link items. At connector 2A, the fields' ICC's are modified by adding .1. This completes TET updating.

(3) At connector 1C, the Linkage Table is positioned at end of file. Then, two additional records are built for the source and target link items and these are inserted at the end of the Linkage Table file. The record numbers are saved for subsequent use.

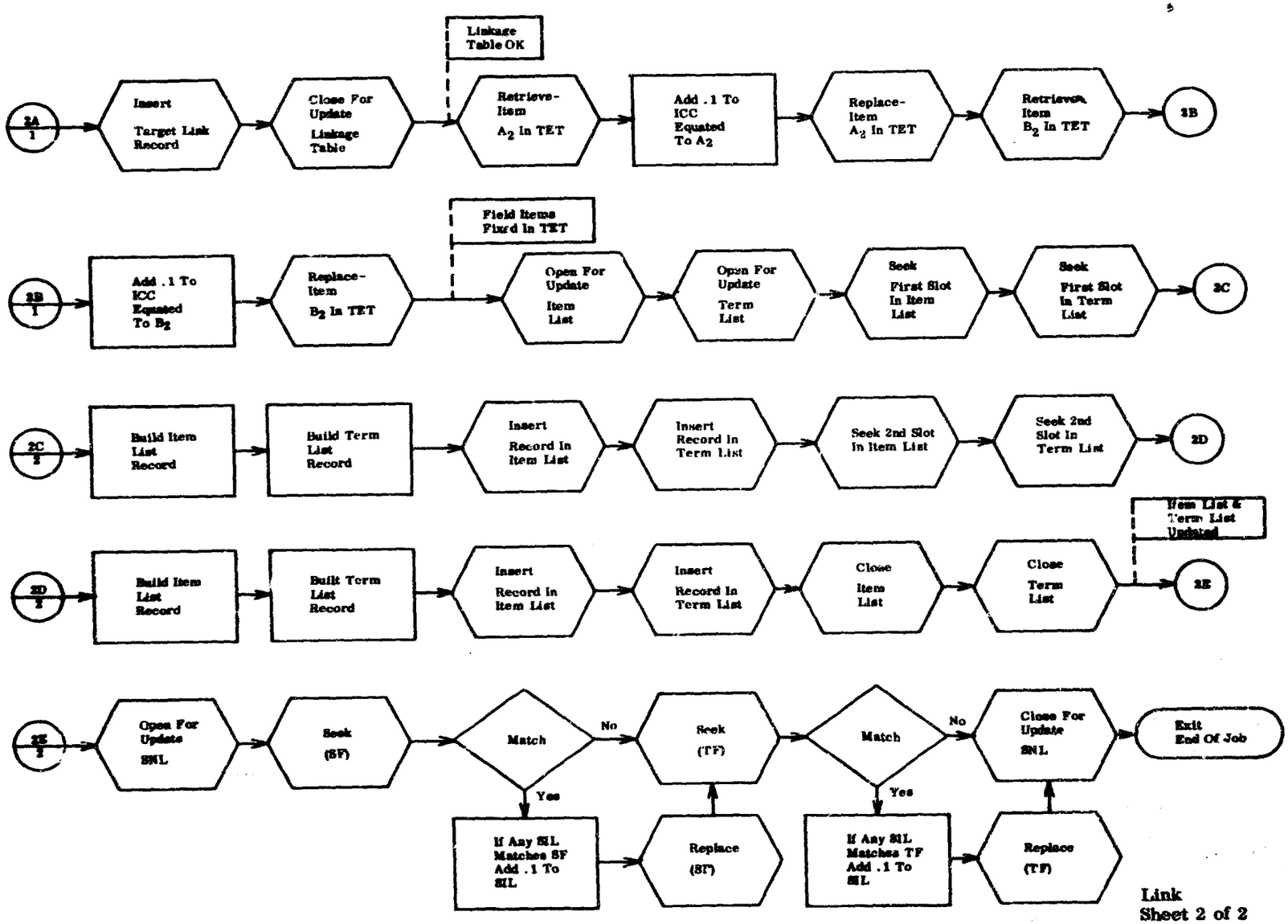
(4) Next, the Item List and Term List are updated in parallel by inserting entries for the source and target link items just in front of SF and TF. These Item List entries include the Linkage Table record numbers assigned in (C).

(5) At connector 2E, the Segment Name List is checked. If any data segments have SF and/or TF as identifiers (i.e., the field is the first data item in the segment), then the identifier has to be modified. The IPC of the field has been changed from x to x.1 by the insertion of the link item into the structure. When the Segment Name List has been updated, if required, the Link job terminates.



6-86

Link
Sheet 1 of 2



Link
Sheet 2 of 2

6.4.2 Delet· Link

Job Reque·: DELETE · NK (source), (target).

6.4.2.1 Functional Description. This job breaks a link by removing the source and target link items from the structures and by deleting the records of the Linkage Table which pertained to this pair of link items.

6.4.2.2 Inputs. The job inputs are:

(1) (source) - the term name for the source link item, qualified if required for uniqueness.

(2) (target) - the term name for the target link item, qualified if required for uniqueness.

6.4.2.3 Results. The link items are removed from the structural definition that is stored in the system directories.

6.4.2.4 Directories Used

(1) Term Encoding Table.

(2) Item List.

(3) Term List.

(4) Linkage Table.

(5) Segment Name List.

6.4.2.5 Services Used

(1) Locate IL Entry.

(2) Open for Update.

(3) Seek.

(4) Delete.

(5) Read.

(6) Replace.

(7) Close for Update.

(8) Retrieve Item.

(9) Delete Data.

(10) Term Name to ICC Translate.

6.4.2.6 Jobs Used. None.

6.4.2.7 Method of Operation. A is the program's name (formal) for the (source) input and B is the program's name (formal) for the (target) input.

(1) A and B are retrieved, translated into ICC's and located in the Item List. A check is made to assure that both items are links and that they are related in a source-target pair.

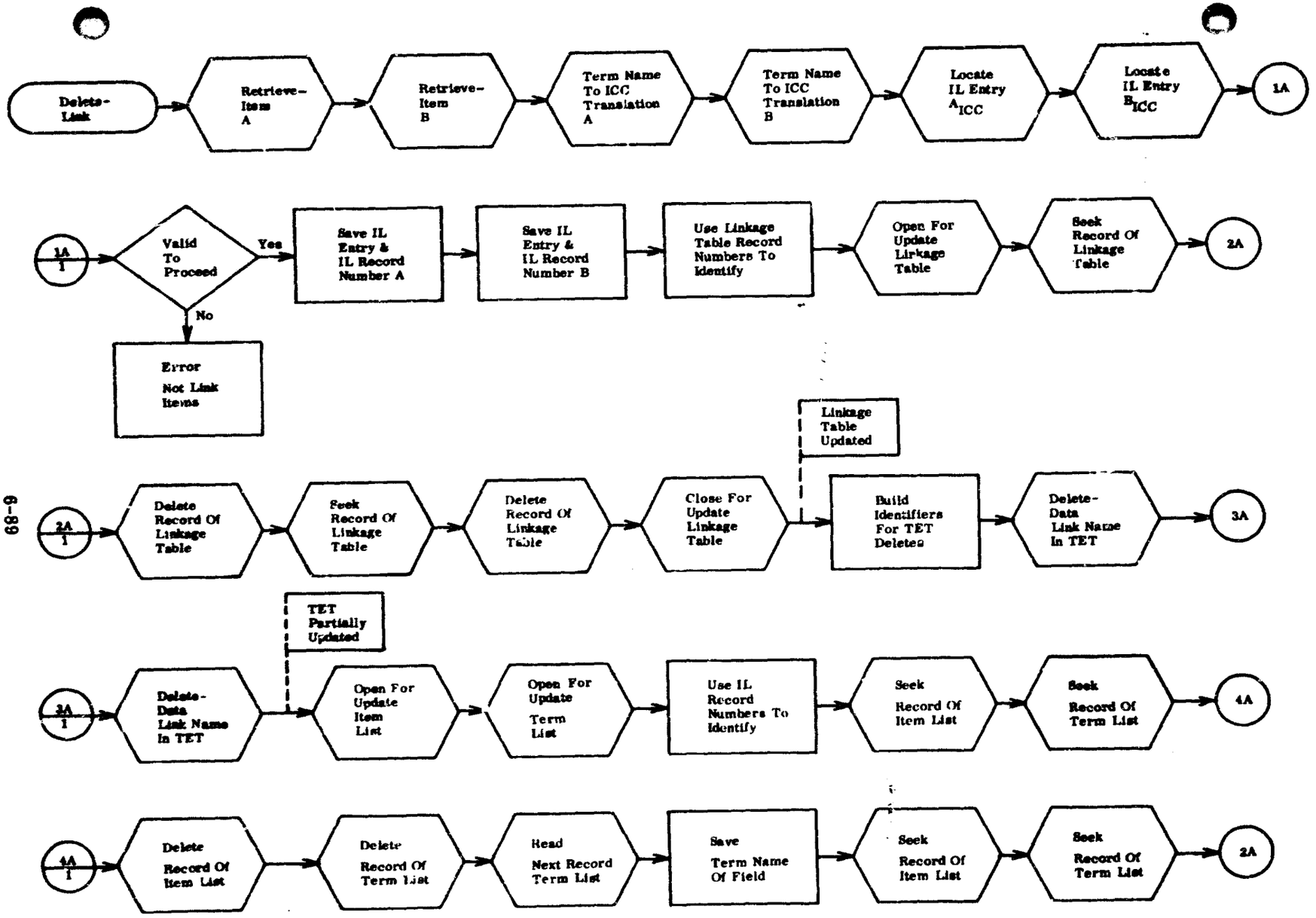
(2) The Linkage Table is updated by deleting the two records pertaining to these two Item List entries.

(3) The term names for the link items are deleted from the Term Encoding Table.

(4) The Item List and Term List are updated in parallel by deleting the two records pertaining to the two link items. The term names of the fields subsumed by the link items are read from the Term List and saved.

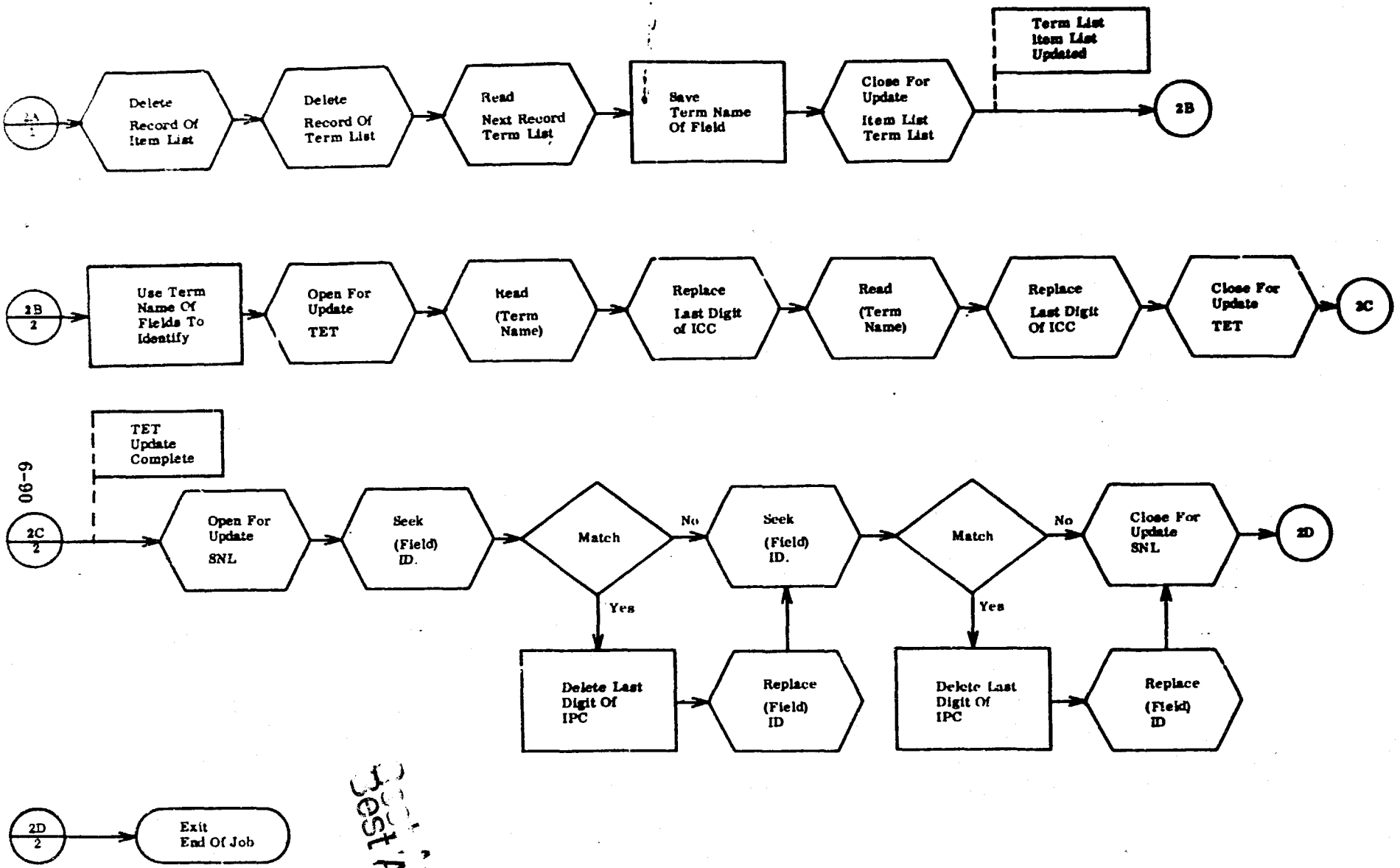
(5) At connector 2B, Linkage Table, Item List, and Term List updating are complete, but in the Term Encoding Table the fields formerly subsumed by the link items have incorrect ICC's. Since the link items have been deleted from the structures, the fields must be "promoted" by deleting the last digit of their ICC's. This is accomplished at connector 2B.

(6) Adjusting the ICC of a field may require adjusting the IPC of the field in the Segment Name List. At connector 2C, the SNL is checked to see if these fields are used as identifiers for any data segments. If so, the last digit of the IPC is deleted. When this is complete, the Delete Link job terminates.



68-89

Delete-Link
Sheet 1 of 2



Best Available Copy

Delete-Link
Sheet 2 of 2

6.5 OTHER MAINTENANCE JOBS

6.5.1 Data Security

Data security is the portion of the data protection system which protects the system user against invasion of privacy by protecting this data against unauthorized read and write operations.

Before read and write operations are executed, security checks are performed. This checking involves comparison of data restrictions against user clearances as they are recorded in the directory. The protection afforded through security checks depends on the accuracy of the restrictions and clearances stored in the directory.

The tools provided to the Data Administrator to protect the data are a set of maintenance jobs designed to put data restrictions and user clearances into the directories so that they can be used for security checks on all data service requests.

6.5.1.1 Data Restrictions. Each item defined to the system (see Define Item, Paragraph 6.1.1) is assigned a security restriction level for access and a security restriction level for modification. These SRL codes range from zero (unrestricted) through six (highest restriction). The only system rule is that in proceeding down a branch of the tree structure, no item may have a higher SRL than its parent.

The Data Administrator assigns the SRL codes when the Define Item job is to be executed. These codes go into the Item List (see Paragraph 2.3) and remain there until modified through a redefinition of the item.

6.5.1.2 User Clearances. Before any user may run a job, he must have been identified to the system through a maintenance job called Add User. The input to the Add User job includes:

(1) (user name)

(2) (user class) - department or group

(3) (priority)

(4) (clearance level-access)

(5) (clearance level-modification)

The Data Administrator assigns the CL codes from one (lowest clearance) through seven (no constraints). When the Add User job is executed, these codes go into the User List (see Paragraph 2.8) and remain there until modified through a Delete User job followed by Add User giving new CL codes.

All data items with SRL codes less than the user's CL code are unconditionally available. If data has an SRL code of zero, all users have free access to it. User CL code 7 has access to all data, and this code should probably be reserved for the Data Administrator.

6.5.1.3 Conditional Access/Modification Rights. When a data item has an SRL code equal to or greater than the user's CL code, the security check will fail unless conditional rights have been given to allow an override of the primary restrictions.

The directory contains Access Rights and Modification Rights Tables (see Paragraph 2.8). When a security check finds data having an SRL code equal to or greater than a user's CL code, the information in the Access or Modification Rights Table will be used to decide if the read or write should be permitted.

The Data Administrator allows conditional rights to certain restricted data by certain users by putting conditional rights into the Access and Modification Rights Tables with the following maintenance jobs:

Add Access Rights

Delete Access Rights

Add Modification Rights

Delete Modification Rights

The input parameters for these jobs are:

(1) (class) - user class, the department or group code.

(2) (list) - one or more term names for the data items (usually files) whose SRL codes are equal to or higher than the user's CL code.

(3) (conditional list) - one or more term names; equality is qualified by a Boolean condition (e.g., TRANSISTOR FILE IF MFR = RCA) which limits the rights to a particular subset of item named.

The permits given, when these jobs are executed, are stored in the Access or Modification Rights Table for the given user class. While the permit remains in the table, the data security checks will pass, even though the data class has a higher SRL than the user's CL.

6.5.2 External to Internal Data Conversion

In the data manipulation jobs, Add Data (Paragraph 6.2.1), Replace Data (Paragraph 6.2.2), and Update Data (Paragraph 6.2.4), the source data input had been translated into internal segmented format before the job was executed. The maintenance jobs which translate external data into internal segmented format are classified as data conversions. Since external data may be received from many different sources, in many different formats, and be recorded on different media, there will be a set of data conversion jobs. The job request has the following form:

(job id), (string), (item),

where:

(1) (job id) - the job name (e.g., EDL2-CONVERSION).

(2) (string) - the job input: a string of external data.

(3) (item) - the job output: ready for use as input to Add Data.

SECTION VII. UTILITY JOBS

The Utility Jobs are divided into two major and three minor jobs. The major jobs include:

(1) Query,

(2) Conditional Reformat.

The minor jobs include:

(1) Conditional Search,

(2) Reformat,

(3) Display.

All Utility Jobs are user-oriented and may be extended indefinitely.

7.1 QUERY

Job Request: QUERY (condition), (qualifier), (format).

7.1.1 Functional Description

A query is a job which consists of several distinct phases or components. In addition to the dialog phase, the basic query (Query job) contains the following components:

(1) Conditional Search,

(2) Reformat,

(3) Display.

It is expected that these basic component jobs will support an additional query capability as well as maintenance jobs with a spectrum of user interfaces, which range from the management-oriented dialog interface to the programmer-oriented data maintenance jobs.

Input to the basic query consists of three user-oriented sentences which may exist within the data pool or which may be entered via remote console. The condition sentence is written in Boolean form and is used to identify the data items of interest in any particular query. The qualifier and format sentence are both written in English-like form and are used to determine the specific fields to be displayed in some desired display format. In addition, the qualifier sentence may be used to restructure the desired items before display.

Results from the basic query consist of some visual display, e.g., console, printer, etc.

7.1.2 Inputs

(1) CONDITION, A, V

(2) QUALIFIER, A, V

(3) FORMAT, A, V

7.1.3 Results

No outputs for the job are specified. The display constitutes the results.

7.1.4 Directories Used

(1) Item List.

(2) Term List.

(3) Fields File.

(4) Shadow of Fields File.

7.1.5 Services Used

(1) Open for Reading.

(2) Close for Reading.

(3) Seek.

(4) Read.

(5) Fix Item.

(6) Open for Writing.

(7) Close for Writing.

(8) Open for Updating.

(9) Close for Updating.

(10) Replace.

(11) Insert.

(12) Delete.

(13) Retrieve Item.

(14) Term Name to ICC Translation.

(15) Retrieve IL Entry.

(16) Retrieve Term List Entry.

7.1.6 Jobs Used

No Job Extensions are used.

7.1.7 Method of Operation

The Query Job is entered into the system via the following Job Description:

Job Name: QUERY

Job Inputs: condition, qualifier, format.

Job Outputs: none specified.

Job Components:

(1) CONDITIONAL-SEARCH: condition, QUERY STATEMENT.

(2) REFORMAT: qualifier, QUERY STATEMENT, QUERY ITEM.

(3) DISPLAY: format, QUERY ITEM.

The internal Job Description of the Query job may be represented graphically as shown in Figure 7-1.

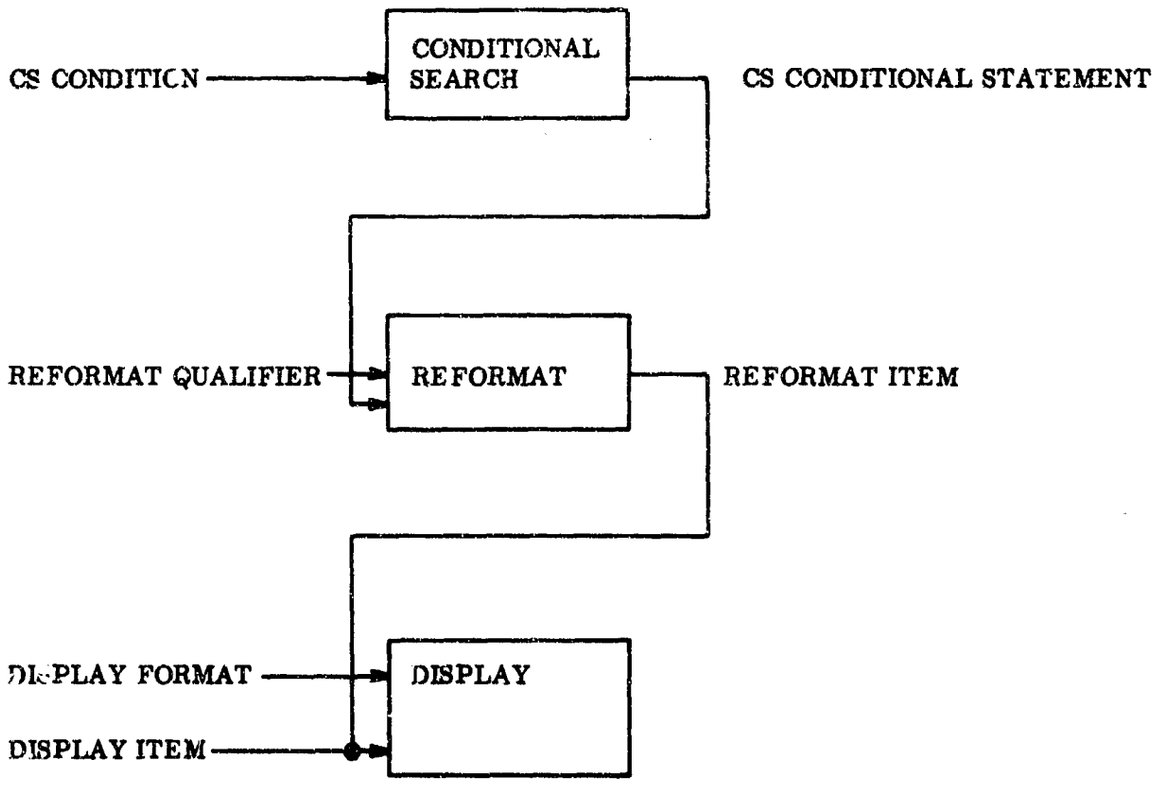


Figure 7-1. Query Job, Internal Job Description

From the condition sentence, the Conditional Search job develops the Conditional Search statement which consists of an ICC encompassing the universe of the discourse and a list of R-values which satisfy the condition. Along with the qualifier sentence, this R-value list constitutes input to the Reformat job. From this, the desired item is fixed in the data pool, and data is written into it. Finally, the desired item is displayed as specified by the Format statement.

7.1.7.1 Conditional Search

Job Request: CS CONDITIONAL-SEARCH: (cs condition);
(cs conditional statement).

7.1.7.1.1 Functional Description. Conditional Search is a job which enables the programmer to find specific records in one or more classes (ICC's) which satisfy some Boolean condition. All of these specific records belong to the universe of the discourse which is represented by the lowest level ICC to subsume all the records.

Four component jobs are needed to form the Conditional Search job. These are:

(1) Condition Translation,

(2) Condition Analysis,

(3) Conjunctive Search,

(4) Disjunctive Search.

A search condition is a specification, in the form of a logical or Boolean statement, of the condition that must be satisfied by an item or items of the data base. The logical statement is made up of terms, parentheses, and logical operators $\neg$ (NOT), $\wedge$ (AND), and $\vee$ (OR). They have the form ϕ, $\neg\phi$, $(\phi \wedge \psi)$, or $(\phi \vee \psi)$, where ϕ and ψ are either terms or statements. A term is a primitive specification of the form $PRv_p\mathcal{E}_p$ where P is a field name (property); R is a relational operator from the set $\{=, \geq, \leq,$ range$\}$; v_p is a value of the field name P; and $\mathcal{E}_p$ (if necessary) is the end of range value. Given an item, a term is true for the item only if it contains the field name P and the value of the field stands in relation R to the specified value $v_p(\mathcal{E}_p)$. In general, $v_p(\mathcal{E}_p)$

may be considered any member of values only one of which is required for the truth of the term. The truth of a logical statement is constructed from the truth of its terms according to the usual rules of Boolean logic.

By considering the Data Base (universe of discourse) as a set whose elements are items, the logical operations can be transformed into set calculus operations on the Data Base. Thus $\neg$ (the logical NOT) is transformed into set complementation which is symbolized as $\neg$; $\wedge$ (the logical AND) is transformed into set intersection which is symbolized as $\cap$; and $\vee$ (the logical OR) is transformed into set union which is symbolized as $\cup$. The situation is complicated by the fact that the Data Base, in addition to simple items (fields), contains compound items (statements, records, and files) which, in turn, contain items. The fact that the Data Base contains compound items permits it to be viewed as a partially ordered set, with the partial ordering relation being "inclusion." Item A is said to include item B (written $A \supseteq B$) if item B is an item within A (e.g., PERSONNEL FILE $\supseteq$ PERSONNEL RECORD, PERSONNEL FILE $\supseteq$ AGE, etc.). The inclusion relation is reflexive ($A \supseteq A$), antisymmetric (if $A \supseteq B$ and $B \supseteq A$, then $A=B$), and transitive (if $A \supseteq B$ and $B \supseteq C$, then $A \supseteq C$), the necessary and sufficient conditions for a partial ordering relation. For two items, A and B, if the IPC of A is a head (stem) of the IPC of B, then A includes B.

Since the Data Base is a particularly ordered set, there is, in general, a hierarchy of items which satisfy each primitive condition (term) in the logical statement. That is, in one interpretation, if B satisfies a condition K, and $A \supseteq B$, then A also satisfies K. But, if B satisfies <u>any</u> condition, then A satisfies it in a trivial way. That is, it is by virtue of the structure of the set that A satisfies every condition that B satisfies, independently of the condition K. Because of this, as a convention, the IPC of the smallest (lowest level) item B that satisfies the condition is taken as the standard representation of an item which "covers" the condition. All items generic to B that also satisfy the condition (by virtue of the structure of the Data Base) can be found by eliminating one or more integers from the tail of the IPC of B. In general, there are cases in which A and B satisfy K, while neither $A \supseteq B$ nor $B \supseteq A$. In this case, A and B are not generically related and they satisfy K independently.

Consider a data base with the structure shown in Figure 7-2. Item H has an ICC of 1R3R1 (ICC(H) = 1R3R1). For a simple condition such as $H = v_h$, the item satisfying the condition has IPC's formed by giving specific values to the two occurrences of R's in the ICC 1R3R1. Likewise, items satisfying simple conditions such as $D = v_d$ have IPC's in the class 1R1. However, for a nonprimitive condition, the situation is slightly more complex.

Consider the condition: $(H = v_h \wedge I = v_i)$. In this case, items satisfying the condition have IPC's in the class 1R3R, particular instances of record G. This is because G is the smallest item which contains both H and I. The general rule, then, is as follows:*

The items satisfying a logical condition are taken as members of the most specific class of items that is generic to all items entering into the logical condition. This class is called the covering item of the condition. For example, items satisfying the condition shown in the "condition" column in Table 7-1 are members of the class identified in the "covering item" column.

TABLE 7-1. LOGICAL CONDITIONS

Condition		Relative Universe of Discourse	Covering Item (Satisfying Condition)	Criterial Record Structure
(1)	I	B	I	C(G)
(2)	E	B	E	C
(3)	D∧E	B	C	C
(4)	D∧H	B	C	C(G)
(5)	D∧L	A	A	C, K
(6)	P∧T	J	K	K(O, S)
(7)	P∧Q	J	O	K(O)

* It is sufficient to consider logical conditions which are a conjunction of terms. A general condition is reduced to a disjunction of conjunctions.

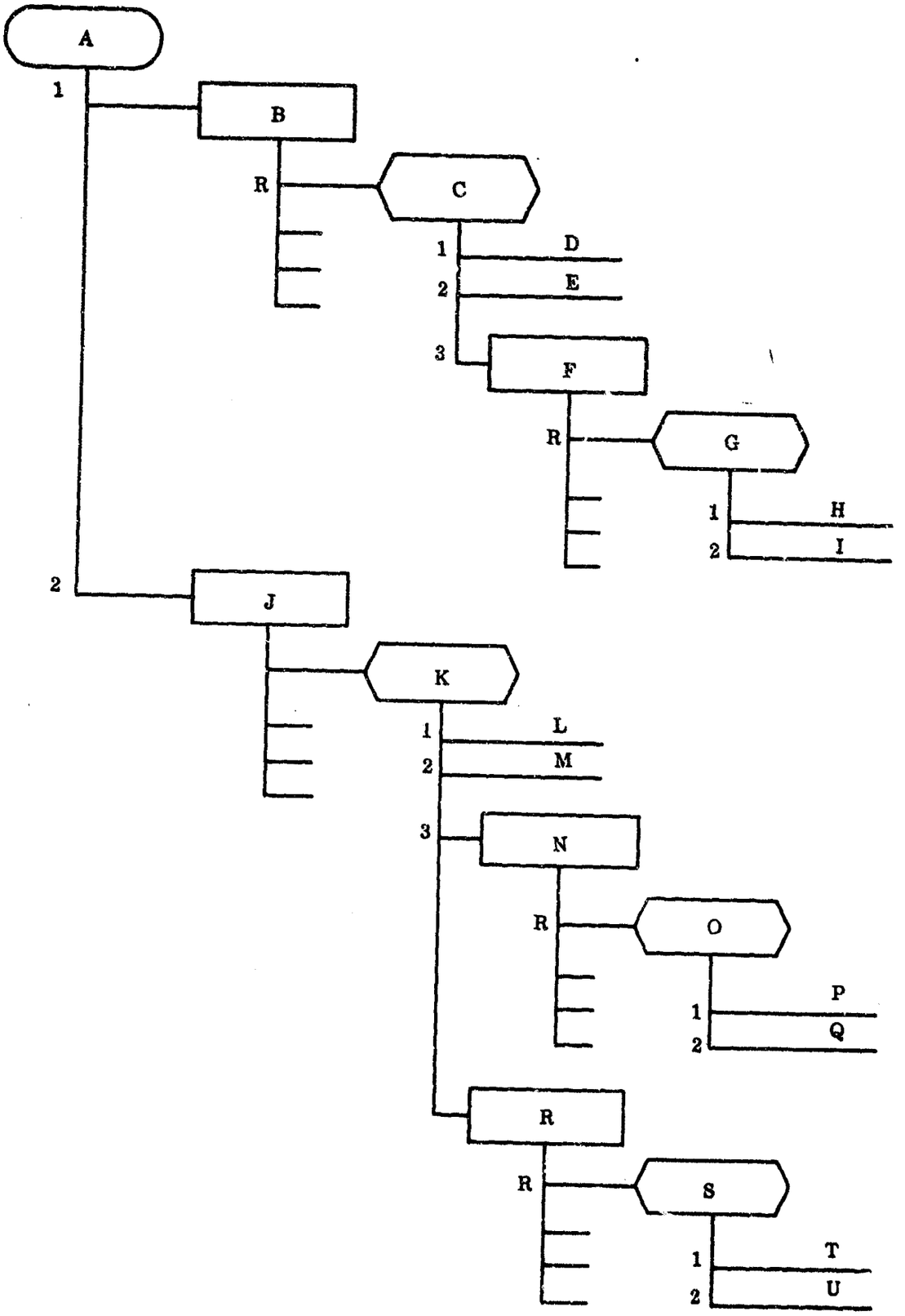


Figure 7-2. Typical Data Structure

The conditional search procedure is developed with use of the symbology, definitions, and theorems of the set calculus. The Data Base is considered as a partially ordered set in which the data items are elements. The operations of set subtraction (or relative complementation, symbolized by A−B), set intersection ($A \cap B$), set union ($A \cup B$), and set complementation ($\neg A$) are defined in terms of the logical concepts of AND($\wedge$), OR($\vee$), NOT($\neg$), and the primitive undefined concept of a set membership (x is a member of A, symbolized $x \in A$). The set calculus definitions are given in Figure 7-3. The notation $\{ x \mid x \in A \}$ is read "the set of all x such that x is a member of A."

With the universe of discourse symbolized by the letter I, the theorem in the set calculus shown in Figure 7-4 can be derived. (These theorems are used in the search procedure which follows.) The definitions and theorems are illustrated by Venn diagrams.

Notice that Theorems T1 through T5 convert expressions involving set subtraction. Subtraction from the universal set I is a costly operation since it involves the largest set I. However, in the search strategy to be employed, it need be performed at most once for any search condition.

In achieving this simplification, the scope of the $\neg$ (NOT) operator in the condition is made a single term. This is accomplished in the condition translation phase by invoking de Morgan's Theorem whenever the scope of $\neg$ is a compound expression, as follows:

$$\neg (A \vee B) = \neg A \wedge \neg B$$

$$\neg (A \wedge B) = \neg A \vee \neg B$$

A second step in the condition translation phase is to transform the logical condition into a disjunction of conjunctions form. This is a two-level conditional expression consisting of $\vee$ (OR's) as the major connectives and $\wedge$ (AND's) as the minor connectives. (Since the scope of each $\neg$ (NOT) has been converted to a single term, they do not influence this discussion.) This form, which will be called the disjunctive normal form, can be obtained from the general conditional expression by applying the theorem

$$A \wedge (C \vee D) = (A \wedge C) \vee (A \wedge D).$$

Definitions		Venn Diagrams
D1	$A \stackrel{=}{\Delta} \{x \mid x \in A\}$	
D2	$A \dashv B \stackrel{=}{\Delta} \{x \mid x \in A \wedge \neg(x \in B)\}$	
D3	$A \cap B \stackrel{=}{\Delta} \{x \mid x \in A \wedge x \in B\}$	
D4	$A \cup B \stackrel{=}{\Delta} \{x \mid x \in A \vee x \in B\}$	
D5	$\neg A \stackrel{=}{\Delta} \{x \mid \neg(x \in A)\}$	

Figure 7-3. Set Calculus Definitions

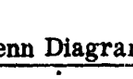
Theorems		Venn Diagrams
T1	$\neg A = I \dashv A$	
T2	$A \cap \neg B = A \dashv (A \cap B)$	
T3	$A \cup \neg B = I \dashv (B \dashv (A \cap B)$	
T4	$\neg A \cap \neg B = I \dashv (A \cup B)$	
T5	$\neg (A \cup \neg B = I \dashv (A \cap B)$	
T6	$\neg \neg A = A$	

Figure 7-4. Set Calculus Theorems

Applying this operation from left to right is the distribution of $\wedge$ over $\vee$. (Applying the theorem from right to left would be equivalent to factoring, which is the converse of distribution.)

7.1.7.1.2 Inputs

CD. S. CONDITION, A, V

7.1.7.1.3 Results

CD. S. CONDITIONAL STATEMENT, S, 2

ICC (UNIVERSE), H, V
R-VALUE, LIST, F

R-VALUE, H, V

7.1.7.1.4 Directories Used

(1) Item List.

(2) Fields File.

(3) Shadow of Fields File.

7.1.7.1.5 Services Used

(1) Open for Reading.

(2) Close for Reading.

(3) Seek.

(4) Read.

(5) Open for Writing.

(6) Close for Writing.

(7) Write.

(8) Open for Updating.

(9) Close for Updating.

(10) Replace.

(11) Insert.

(12) Delete.

(13) Term Name to ICC Translation.

(14) Retrieve IL Entry.

7.1.7.1.6 Jobs Used. No Job Extensions are used.

7.1.7.1.7 Method of Operation. The Conditional Search job is entered into the system via the following Job Description:

Job Name: CONDITIONAL-SEARCH

Job Inputs: cs condition

Job Outputs: cs conditional statement

Job Components:

(1) CONDITION-TRANSLATION: cs condition; cs sum of products.

(2) CONDITION-ANALYSIS: cs sum of products; css, cs SCHK.

(3) CONJUNCTIVE-SEARCH: css, cs scratch list; cs disjunction, cs scratch list.

(4) DISJUNCTIVE-SEARCH: cs disjunction, cs SCHK, cs scratch list; CS CONDITIONAL STATEMENT, cs scratch list.

The internal Job Description may be represented graphically as shown in Figure 7-5.

From the condition sentence, the Condition Translation job develops the sum of products disjunctive normal form of the condition. With this, the Condition Analysis job determines the ICC of each field name, specified within each conjunction of the condition, and orders these in ascending ICC order. Along with this, the Analysis job outputs the CA SCHK which is a list of all incompletely indexed terms for the purpose of a subsequent correlation with the data. The Conjunctive Search job intersects all terms of each conjunction and outputs the resulting list of disjunctions. Through a merge of these, the Disjunctive Search job forms the DS Conditional Statement which includes the list of R-values acceptable to the original condition. In the event of a nonindexed result, the appropriate terms are checked against the data.

In all cases, the scratch list is needed for purposes of internal computations, but may not be required in a given instance.

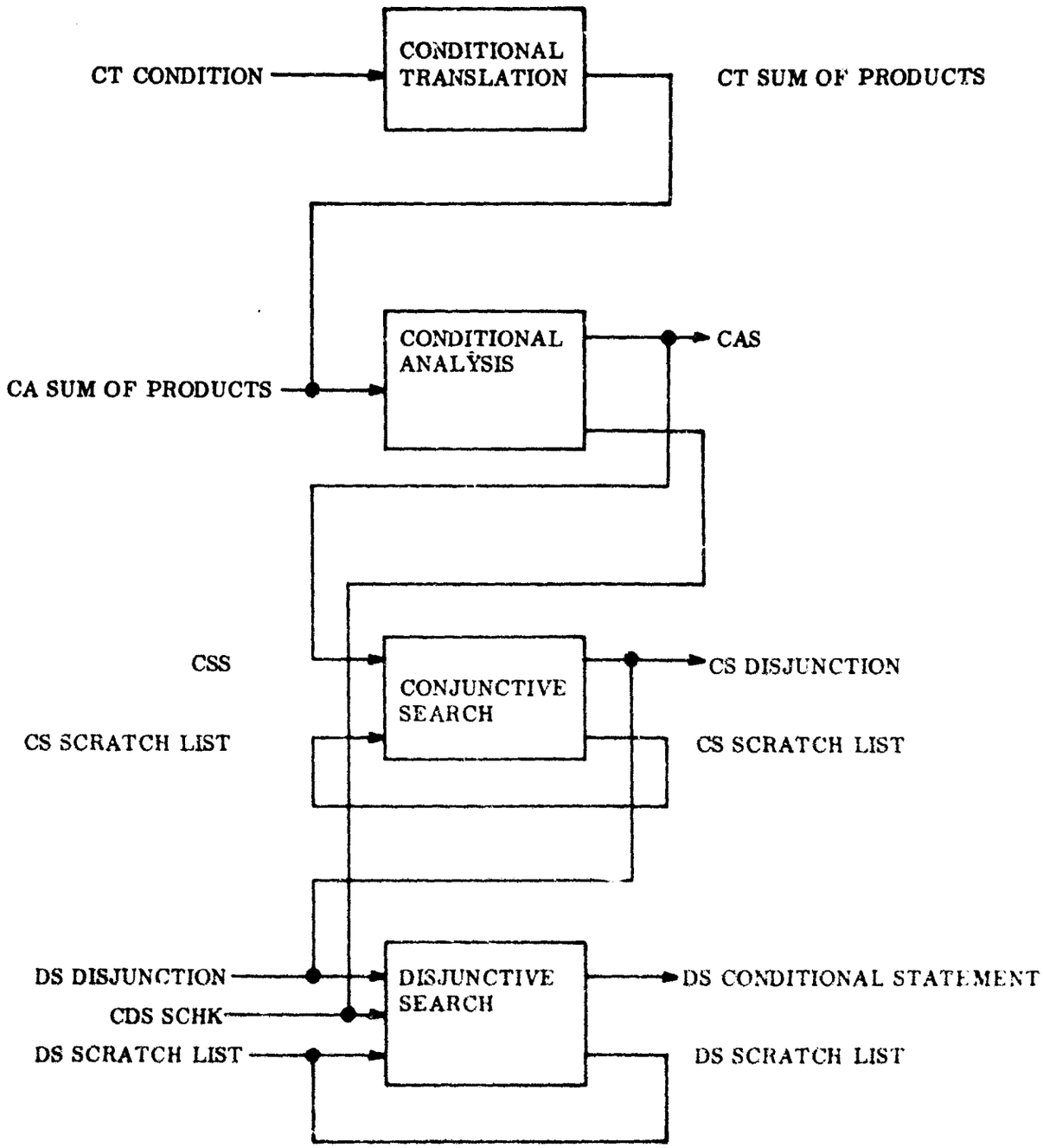


Figure 7-5. Conditional Search, Internal Job Description

7.1.7.1.7.1 Condition Translation

Job Request: CT CONDITION-TRANSLATION (ct condition); (ct sum of products)

7.1.7.1.7.1.1 Functional Description. Condition translation involves two phases:

(1) Translation of the search condition from parenthesized infix operator form to parenthesis – free suffix operator form, and

(2) Translation of the suffix form to the disjunctive normal form.

The suffix translation utilizes INSCAN with the appropriate action-graph, to translate from infix to suffix sentence form. During this process, de Morgan's Theorem is applied to distribute any ¬(NOT) operators so that their scope is a single term. The second phase in the condition translation consists of normalizing the search condition. The following action is taken:

> Distribute $\wedge$ over $\vee$ to form a two-level with $\vee$ (OR) as the major connective. As a result of this process, the terms in each conjunction form a Products File for every disjunction.

7.1.7.1.7.1.2 Inputs

C.T. CONDITION, A, V

7.1.7.1.7.1.3 Results

C.T. SUM OF PRODUCTS, F

PRODUCTS, F

FIELD NAME, A, V
VALUE, B, V
RELATION CODE, B, 2 { 00 => equals; 01 => greater than or equals; 10 => less than or equals; 11 => range
END OF RANGE, B, V
I, B, 1 { 1 =>NOT

7.1.7.1.7.1.4 Directories Used. No directories are used.

7.1.7.1.7.1.5 Services Used

(1) Open for Reading.

(2) Close for Reading.

(3) Seek.

(4) Read.

(5) Open for Writing.

(6) Close for Writing.

(7) Write.

7.1.7.1.7.1.6 Jobs Used. No Job Extensions are used.

7.1.7.1.7.1.7 Method of Operation. The appropriate action-graph is compiled with the Condition Translation program. The graph is of the Compiler-like infix-to-suffix sentence type.

Upon generation of the suffix form, the Sum of Products file is developed and written.

7.1.7.1.7.2 Condition Analysis

Job Request: CONDITION ANALYSIS (ca sum of products); (cas, ca SCHK)

7.1.7.1.7.2.1 Functional Description. A search analysis is performed for each conjunction in the condition. This consists of the following steps:

(1) All Field names are translated to ICC's.

(2) The ICC's are ordered into increasing ICC depth order. Within each depth, the ICC's are ordered into ICC order.

(3) The S file is formed.

(4) If a field is not fully indexed, an SCHK file entry is developed.

7.1.7.1.7.2.2 Inputs

C.A. SUM OF PRODUCTS, F

PRODUCTS, F

FIELD NAME, A, V
VALUE, B, V
RELATION CODE, B, 2
END OF RANGE, B, V
I, B, 1

7.1.7.1.7.2.3 Results

(1) C.A. S, F

T, F

ICC, H, V
DATA CODE, B, 1 { 0 => not indexed; 1 => indexed }
I, B, 1
RVIT R-VALUE

(2) C.A. SCHK, F

TCHK, F

ICC, H, V
VALUE, B, V
RELATION OF CODE, B, 2
END OF RANGE, B, V
I, B, 1

7.1.7.1.7.2.4 Directories Used

(1) Item List.

(2) Fields File.

7.1.7.1.7.2.5 Services Used

(1) Open for Reading.

(2) Close for Reading.

(3) Seek.

(4) Read.

(5) Open for Writing.

(6) Close for Writing.

(7) Write.

(8) Term Name to ICC Translation.

(9) Retrieve IL Entry.

7.1.7.1.7.2.6 Jobs Used. No Job Extensions are used.

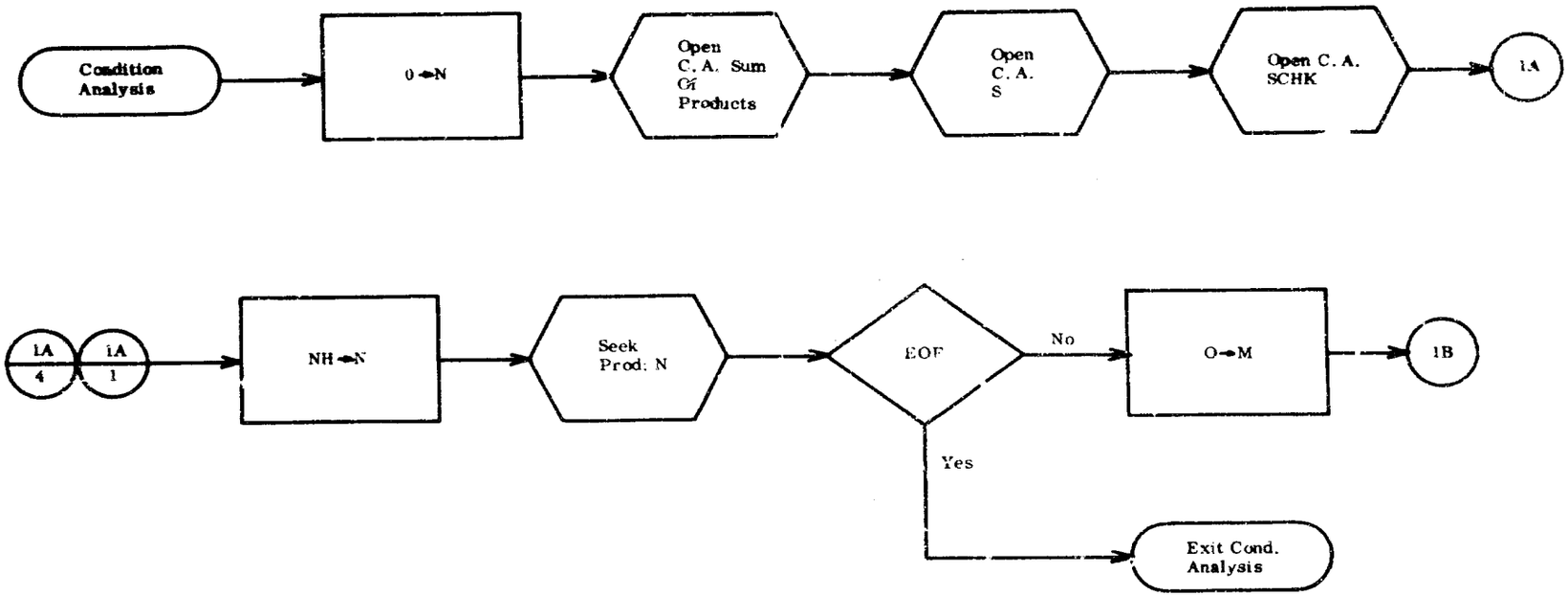
7.1.7.1.7.2.7 Method of Operation. The Field names are translated to ICC's. ICC(MIN) is obtained and the following actions are taken:

(1) If ICC(MIN) is not indexed, write T Record with null RVIT R-VALUE and TCHK Record.

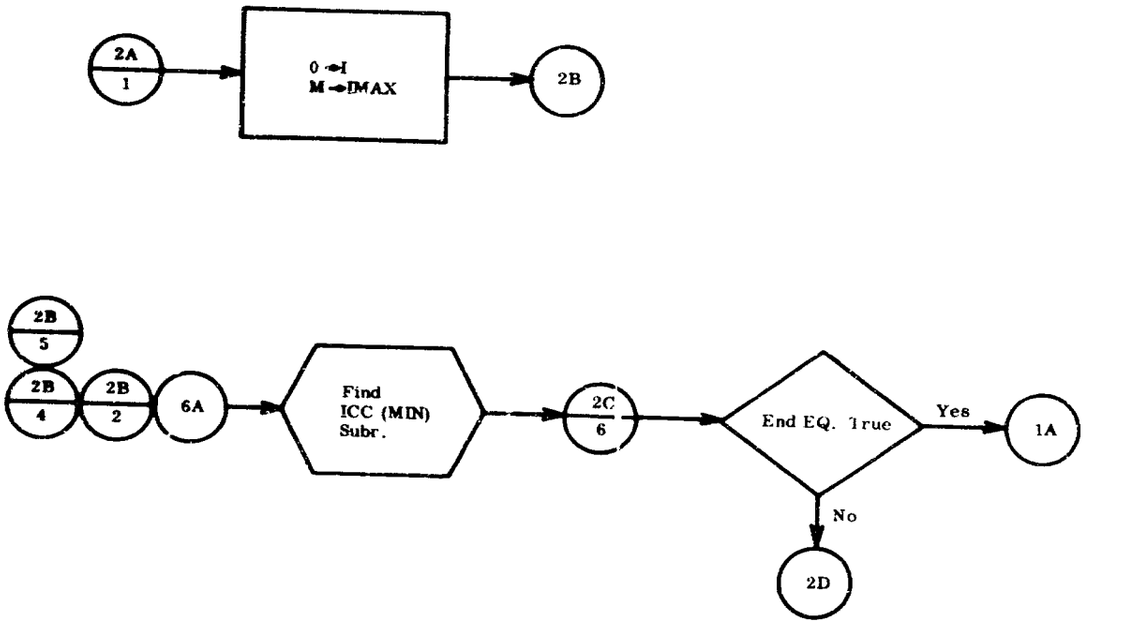
(2) If ALL values of ICC(MIN) are indexed, determine the Relation Code. If "equals", find the particular value in the FVT and write T Record. If a particular value does not exist, terminate the conjunction. The code F or all other relation codes indicate that ICC(MIN) is not indexed.

(3) If LIST values of ICC(MIN) are indexed, determine the Relation Code. If "equals", find the particular value in the FVT and write T Record. If a particular value does not exist or if there is another relation code, assume that ICC(MIN) is not indexed.

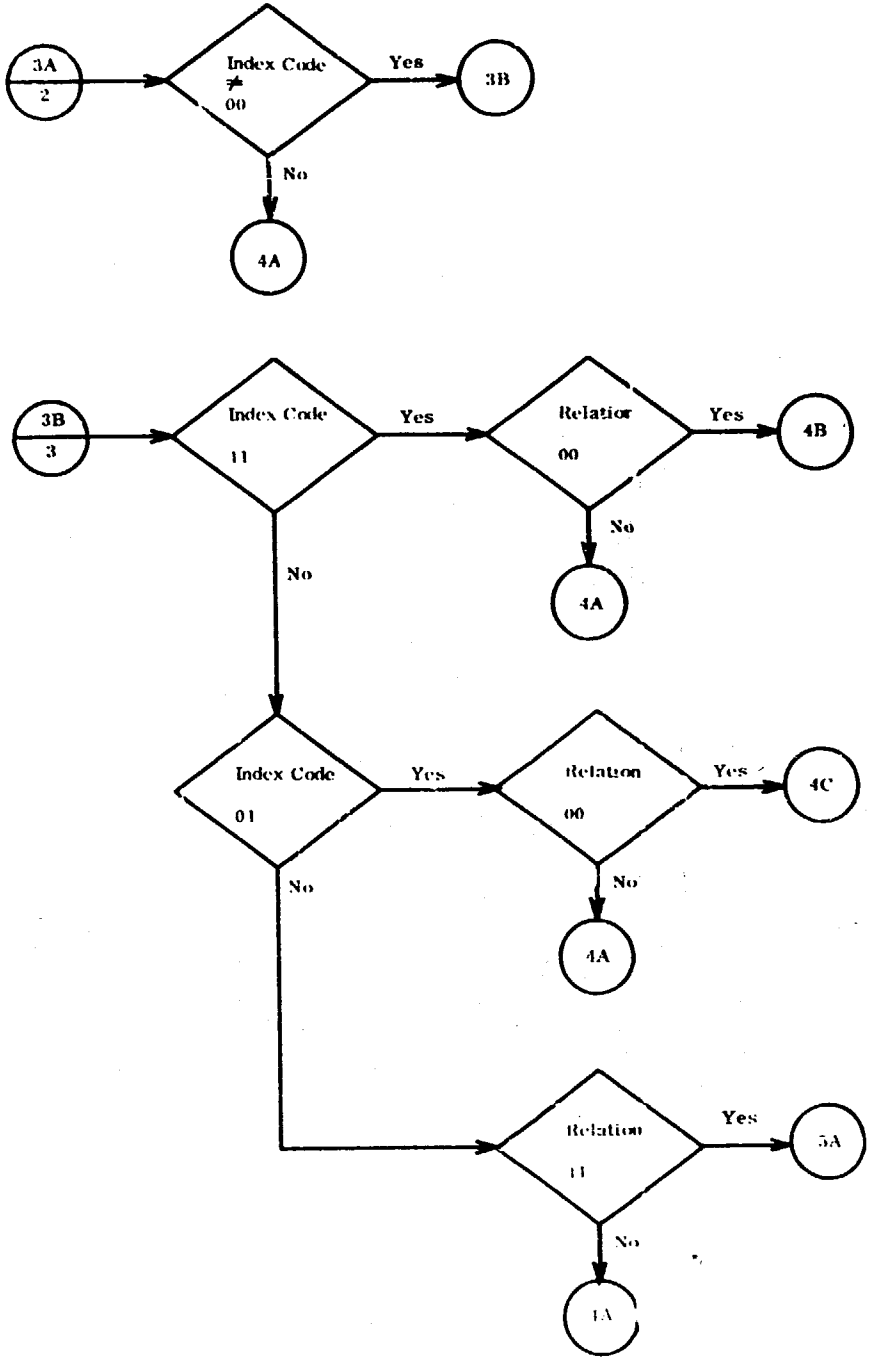
(4) If RANGE values of ICC(MIN) are indexed, determine the Relation Code. If "range", find the particular range in the FVT and write T Record. If a particular range does not exist or if there is another relation code, assume that ICC(MIN) is not indexed.



Condition Analysis Job
Sheet 1 of 6

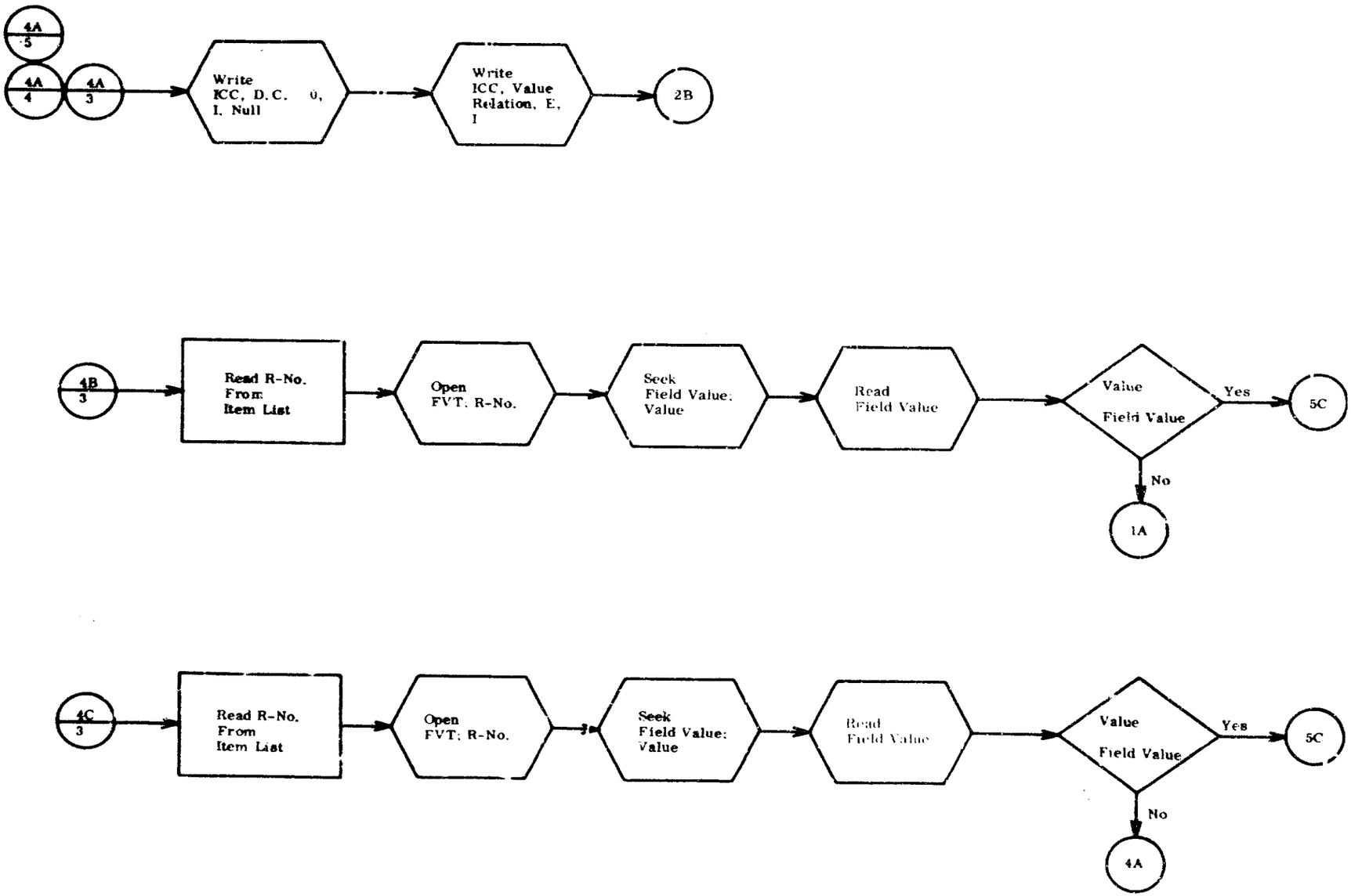


Condition Analysis Job
Sheet 2 of 6

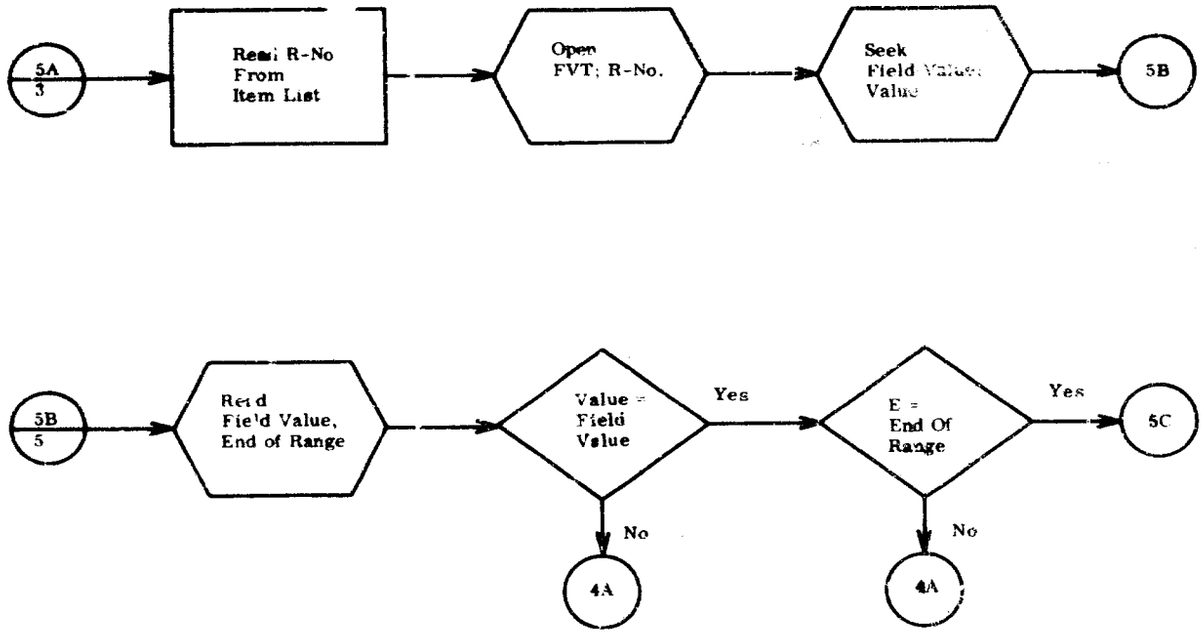


Condition Analysis Job
Sheet 3 of 6

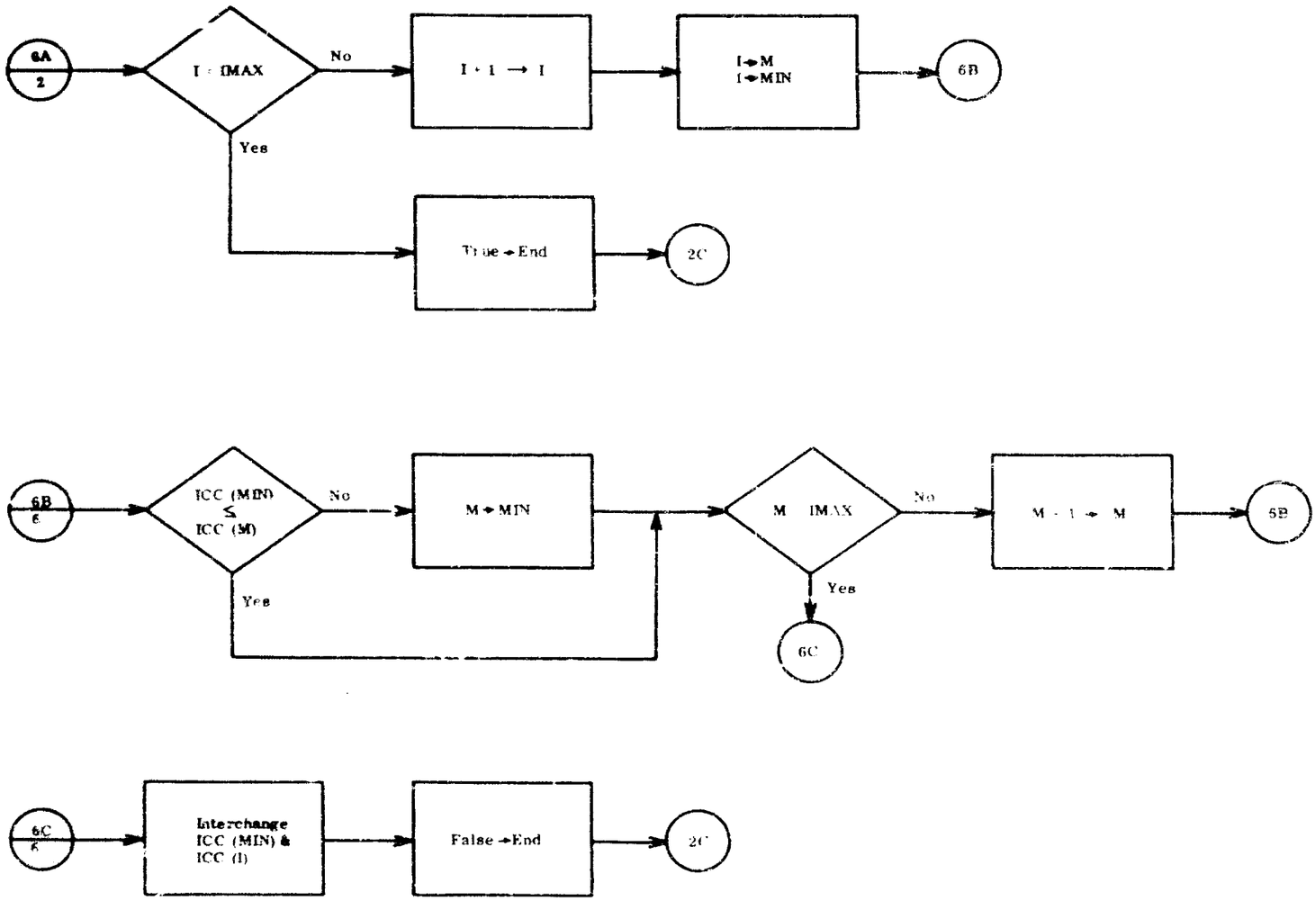
Best Available Copy



Condition Analysis Job
Sheet 4 of 6



Condition Analysis Job
Sheet 5 of 6



Condition Analysis Job
Sheet 6 of 6

7.1.7.1.7.3 Conjunctive Search

Job Request: CONJUNCTIVE-SEARCH (css cs scratch list);
(cs disjunction), (cs scratch list)

7.1.7.1.7.3.1 Functional Description. A conjunctive search is executed for each T file subsumed within the S file. This consists of intersecting the RVIT's of all terms within the conjunction. All R-values are assumed whenever a term is not executed. The result is a file of disjunction.

7.1.7.1.7.3.2 Inputs

(1) C.S.S, F

T, F

ICC, H, V
DATA CODE, B, 1
I, B, 1
RVIT R-VALUE, H V

(2) C.S. SCRATCH LIST, F

R-VALUE, H, V

7.1.7.1.7.3.3 Results

(1) C.S. DISJUNCTION, F

CONJUNCTION, S, 4

ICC (CONJ), H, V
DATA CODE (CONJ), B, 1
I (CONJ), B, 1
R-LIST, F

R-VALUE, H, V

(2) C.S. SCRATCH LIST, F

R-VALUE, H, V

7.1.7.1.7.3.4 Directories Used. Shadow of Fields File.

7.1.7.1.7.3.5 Services Used

(1) Open for Reading.

(2) Close for Reading.

(3) Seek.

(4) Read.

(5) Open for Writing.

(6) Close for Writing.

(7) Write.

(8) Open for Updating.

(9) Close for Updating.

(10) Replace.

(11) Insert.

(12) Delete.

7.1.7.1.7.3.6 Jobs Used. No Job Extensions are used.

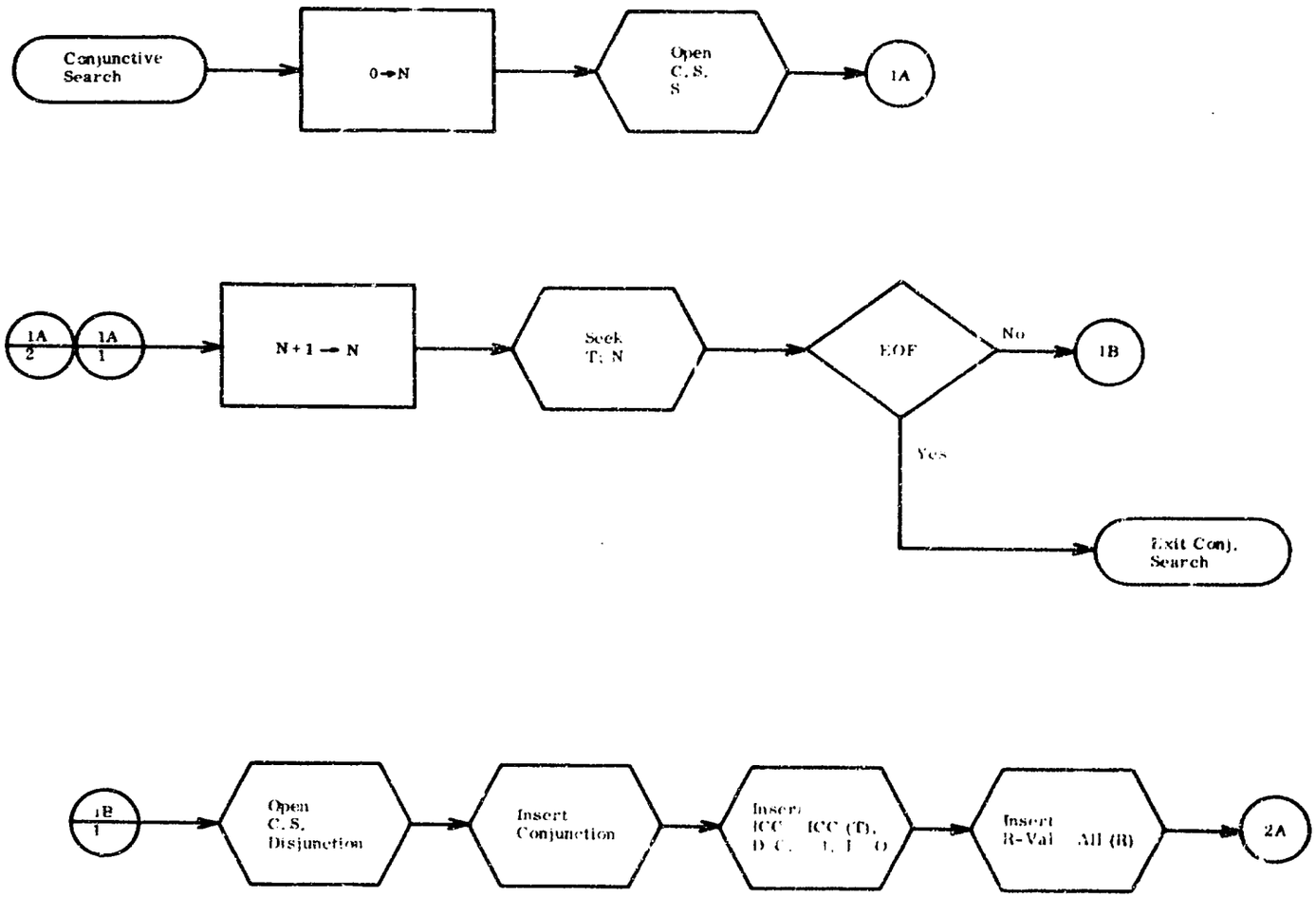
7.1.7.1.7.3.7 Method of Operation. The R-List of each disjunction is initialized to all R-values with I (Conj) = 0. The next T Record is read and the specified RVIT is initialized. The following actions are taken:

(1) If neither I=0, intersect the two R-value lists. [D3 : A∩B]

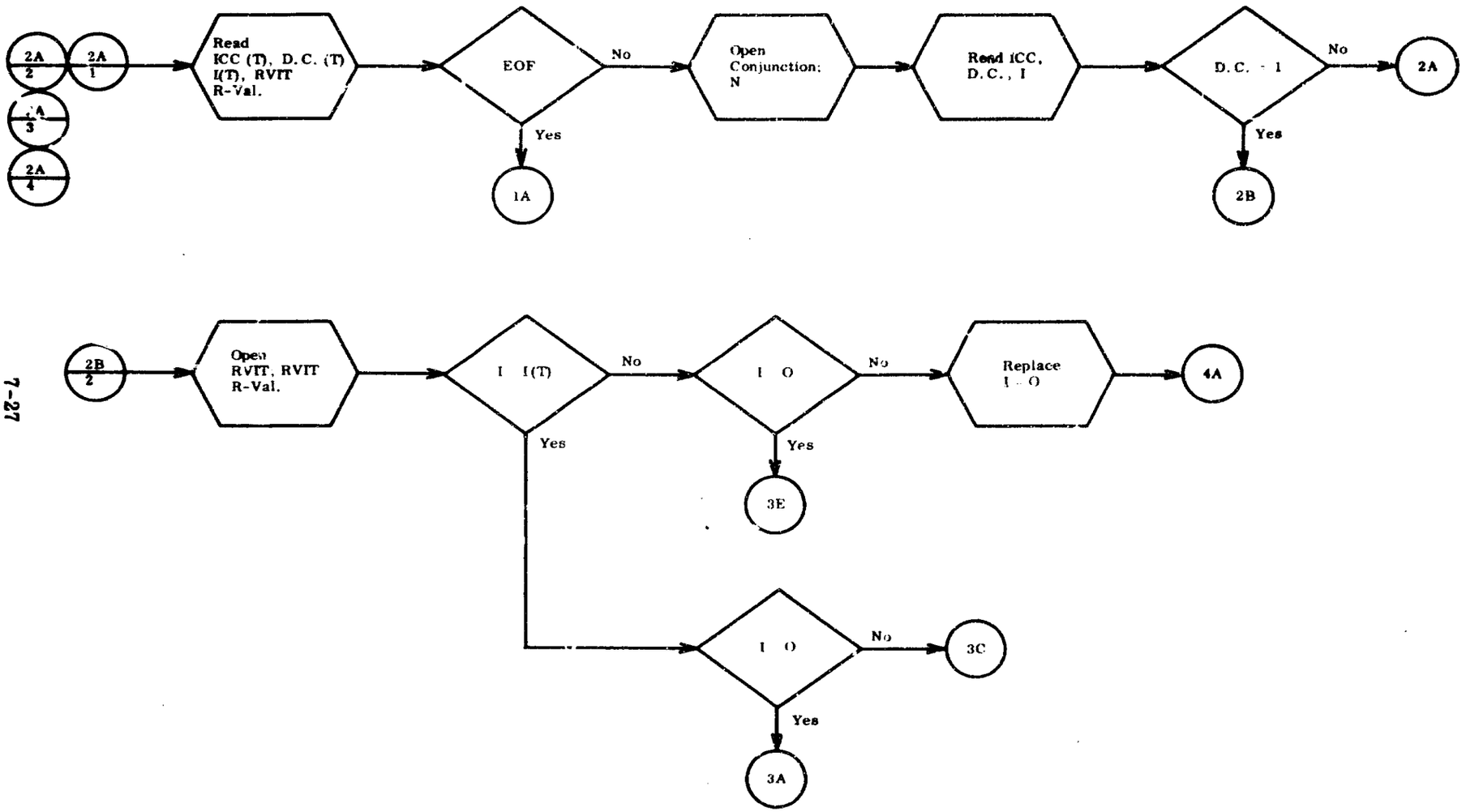
(2) If one I=1, intersect the two R-value lists and subtract the result from the list with I=0. Set I (Conj) = 0. [T2 : A ∩¬B=A ⟶ (A∩B)]

(3) If both I=1, merge the two R-value lists. [T4 : ¬(A∩¬B = I⟶(A ∩ ⁿ)]

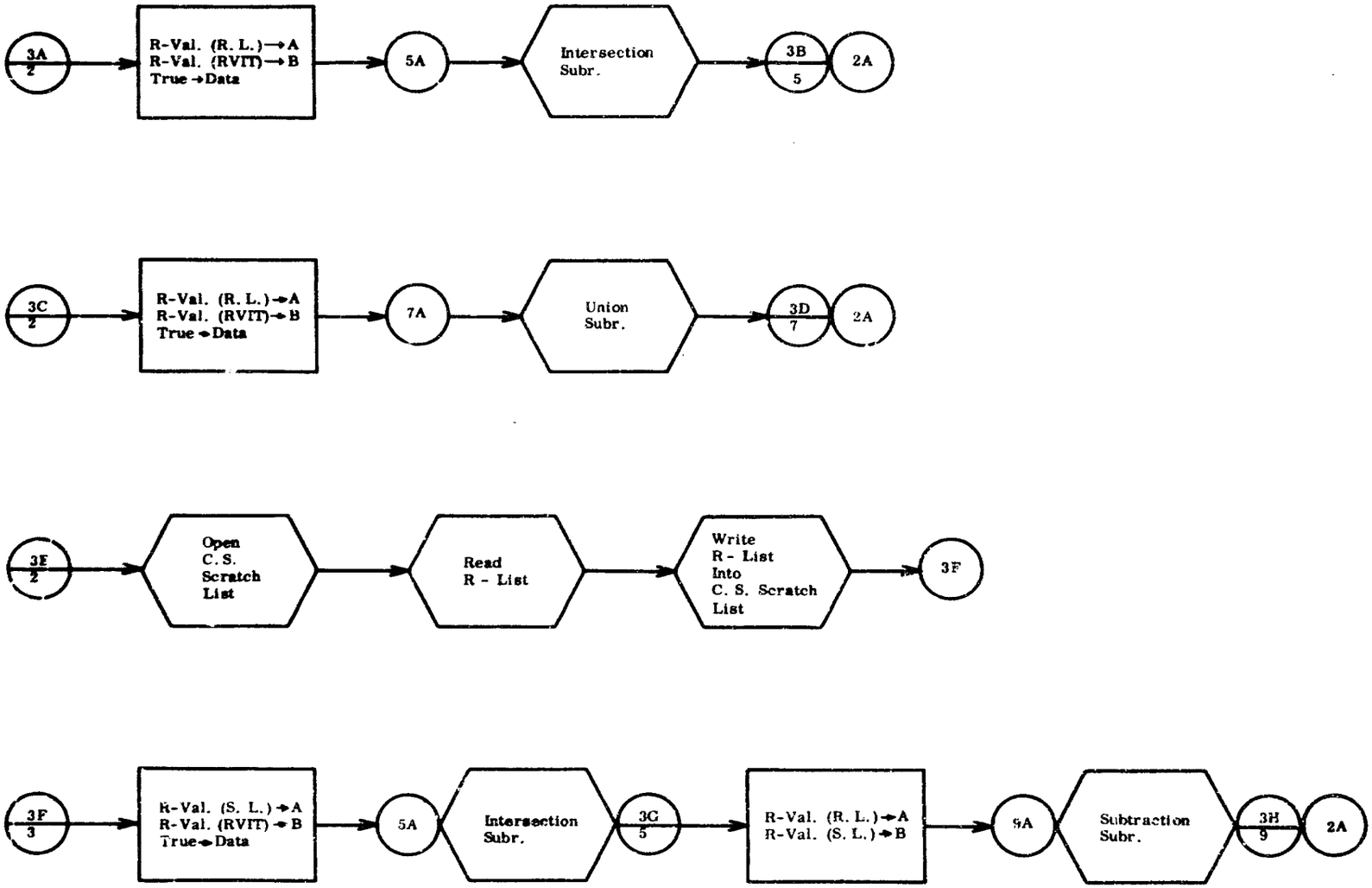
This process is repeated for each conjunction of the disjunction until all are exhausted. The result is a single R-List for each disjunction.



Conjunctive Search Job
Sheet 1 of 9

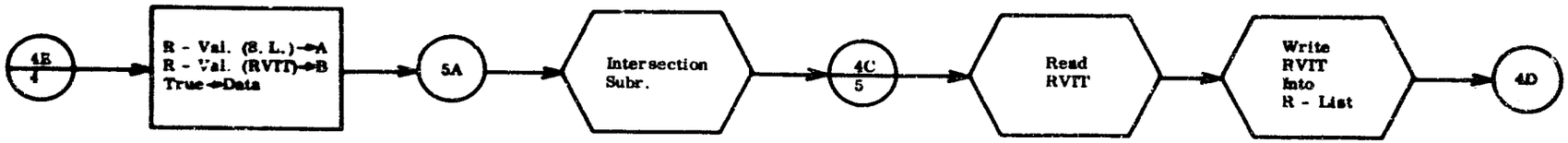


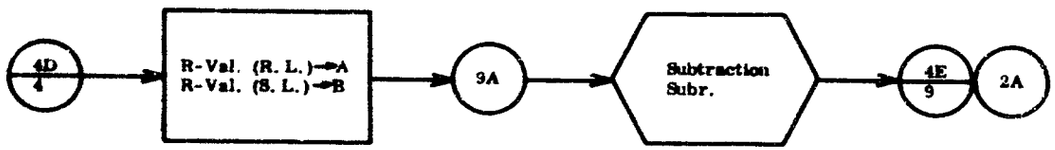
Conjunctive Search Job
Sheet 2 of 9



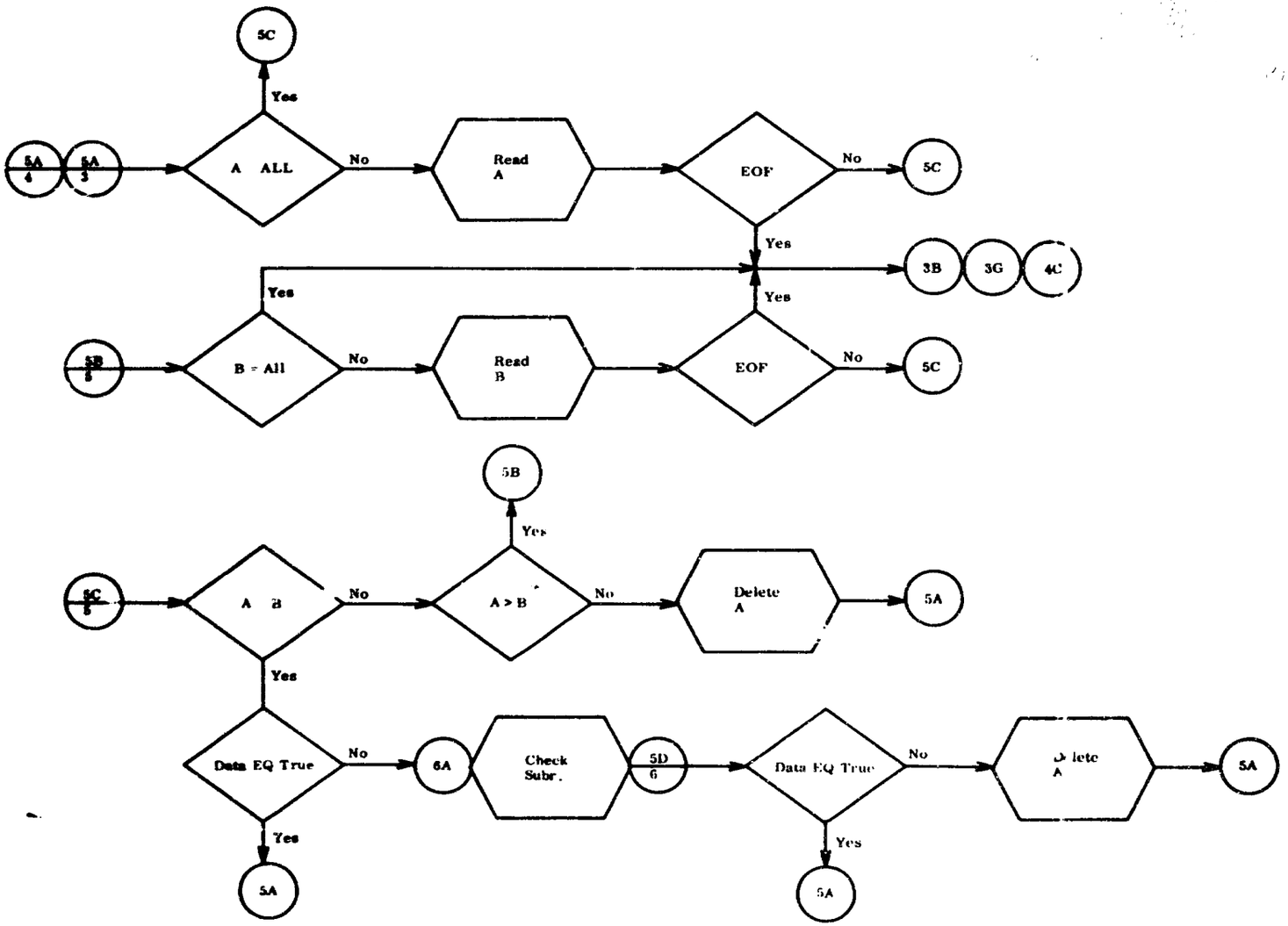
Conjunctive Search Job
Sheet 3 of 9



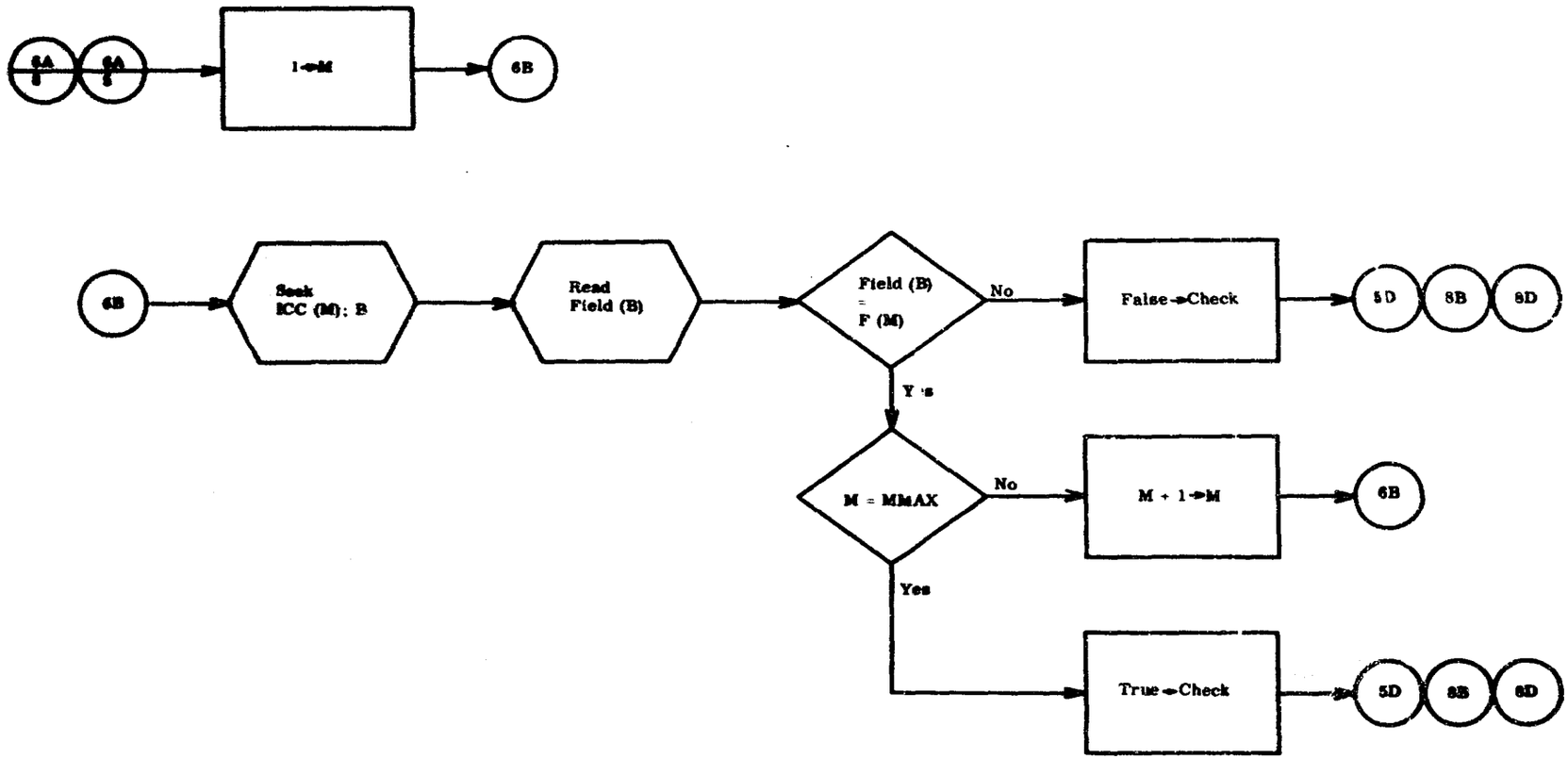




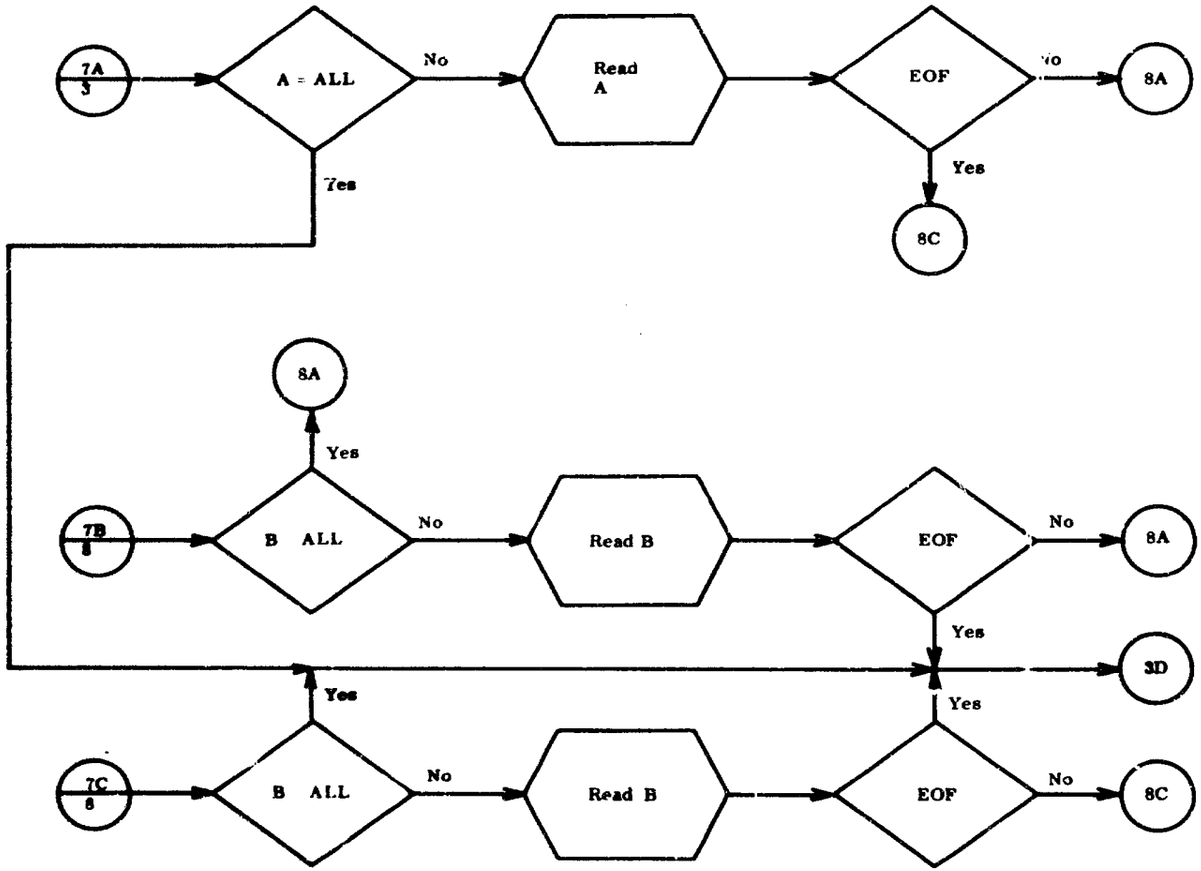
Conjunctive Search Job
Sheet 4 of 9



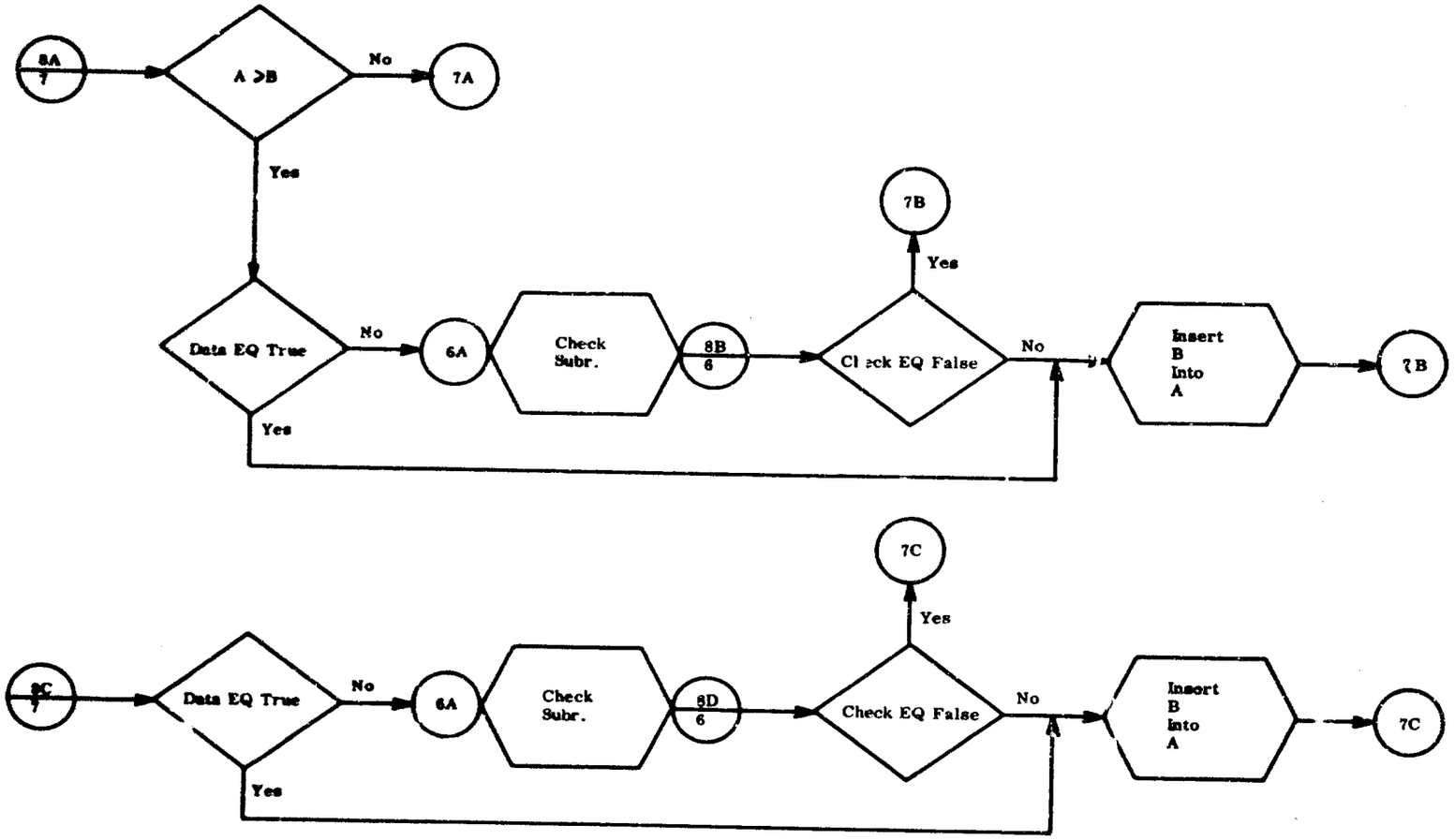
Intersection Subr.
Conjunctive Search Job
Sheet 5 of 9



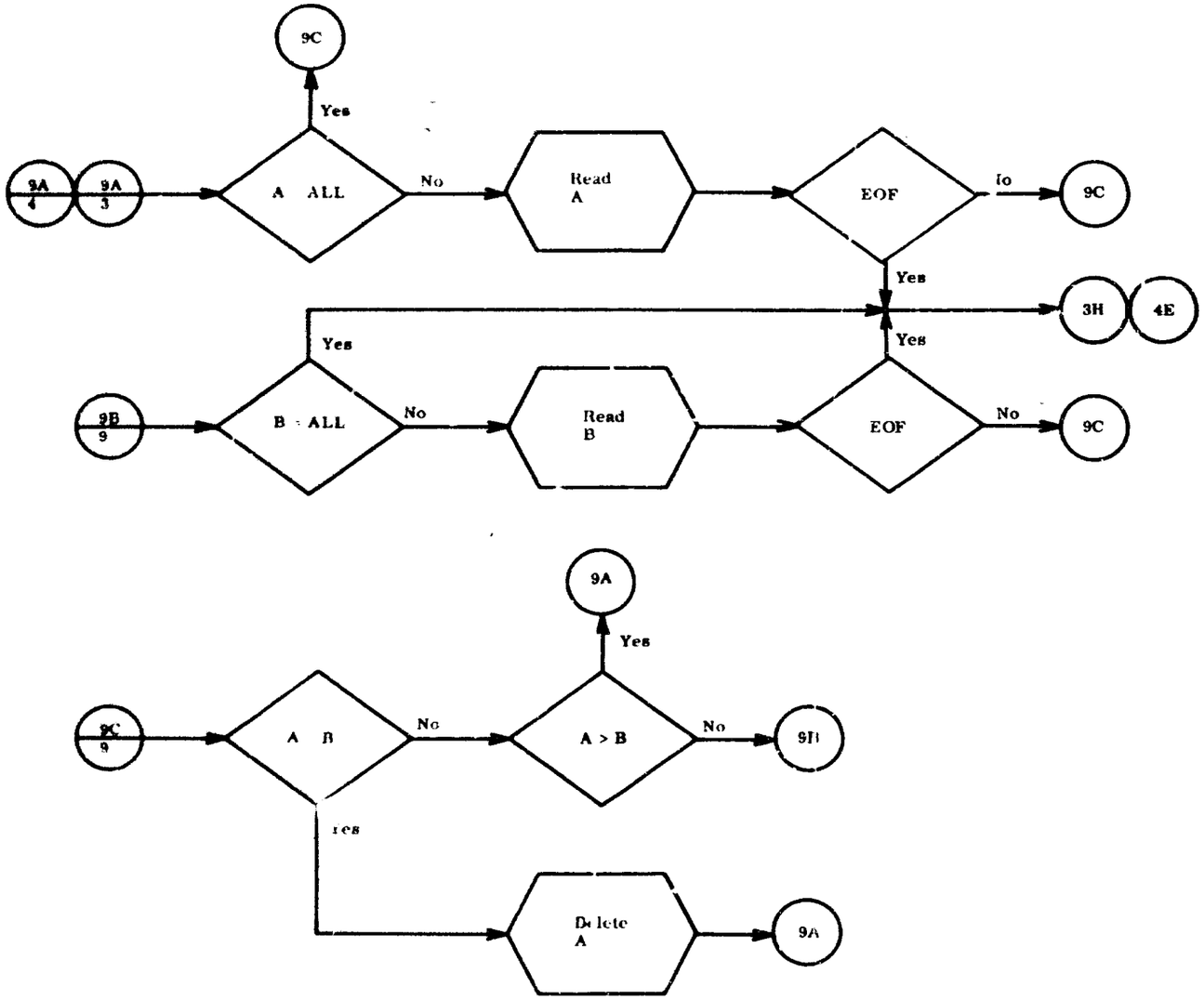
Check Subr.
Conjunctive Search Job
Sheet 6 of 9



Union (1) Subr.
Conjunctive Search Job
Sheet 7 of 9



Union (2) Subr.
Conjunctive Search Job
Sheet 8 of 9



Subtraction Subr.
Conjunctive Search Job
Sheet 9 of 9

7.1.7.1.7.4 Disjunctive Search

Job Request: DISJUNCTIVE-SEARCH (ds disjunction), (ds SCHK),
(ds scratch list);
(ds CONDITIONAL STATEMENT),
(ds scratch list).

7.1.7.1.7.4.1 Functional Description

A disjunctive search is executed on the Disjunction file. This consists of forming ICC(UNIVERSE) and merging all outstanding R-Lists. If necessary, the data field is accessed and checked against the appropriate relation to a particular value (end of range). The result is a simple list of R-values.

7.1.7.1.7.4.2 Inputs

(1) D.S. DISJUNCTION, F

CONJUNCTION, S, 4

ICC(CONJ), H, V
DATA CODE (CONJ), B, 1
I (CONJ), B, 1
R-LIST, F

R-VALUE, H, V

(2) D.S. SCHK, F

TCHK, F

ICC, H, V
VALUE, B, V
RELATION CODE, B, 2
END OF RANGE, B, V
I, B, 1

(3) D.S. SCRATCH LIST, F

R-VALUE, H, V

7.1.7.1.7.4.3 Results

(1) D.S. CONDITIONAL STATEMENT, S, 2

ICC(UNIVERSE), H, V
R-VALUE LIST, F

R-VALUE, H, V

(2) D.S. SCRATCH LIST, F

R-VALUE, H, V

7.1.7.1.7.4.4 Directories Used. No directories are used.

7.1 7.1.7.4.5 Services Used

(1) Open for Reading.

(2) Close for Reading.

(3) Seek.

(4) Read.

(5) Open for Writing.

(6) Close for Writing.

(7) Write.

(8) Open for Updating.

(9) Close for Updating.

(10) Replace.

(11) Insert.

(12) Delete.

7.1.7.1.7.4.6 Jobs Used. No Job Extensions are used.

7.1.7.1.7.4.7 Method of Operation. The Disjunctive Search has the following three phases of operation:

(1) Indexed Phase.

(2) Nonindexed Phase.

(3) Subtraction Phase.

The Indexed and Nonindexed Phases are identical except that the Nonindexed Phase necessitates correlation with the data. This is accomplished by transferring the FALSE value to the DATA indicator.

To start the task, the R-VALUE List is initialized to read EOF with $I(\cup)=0$. The operation proceeds, first, with the Indexed Phase and, then, with the Nonindexed Phase. After the next appropriate R-List is initiated, the following actions are taken:

(1) If neither I=0, merge the two R-value lists.
$[D4 : A \cup B]$

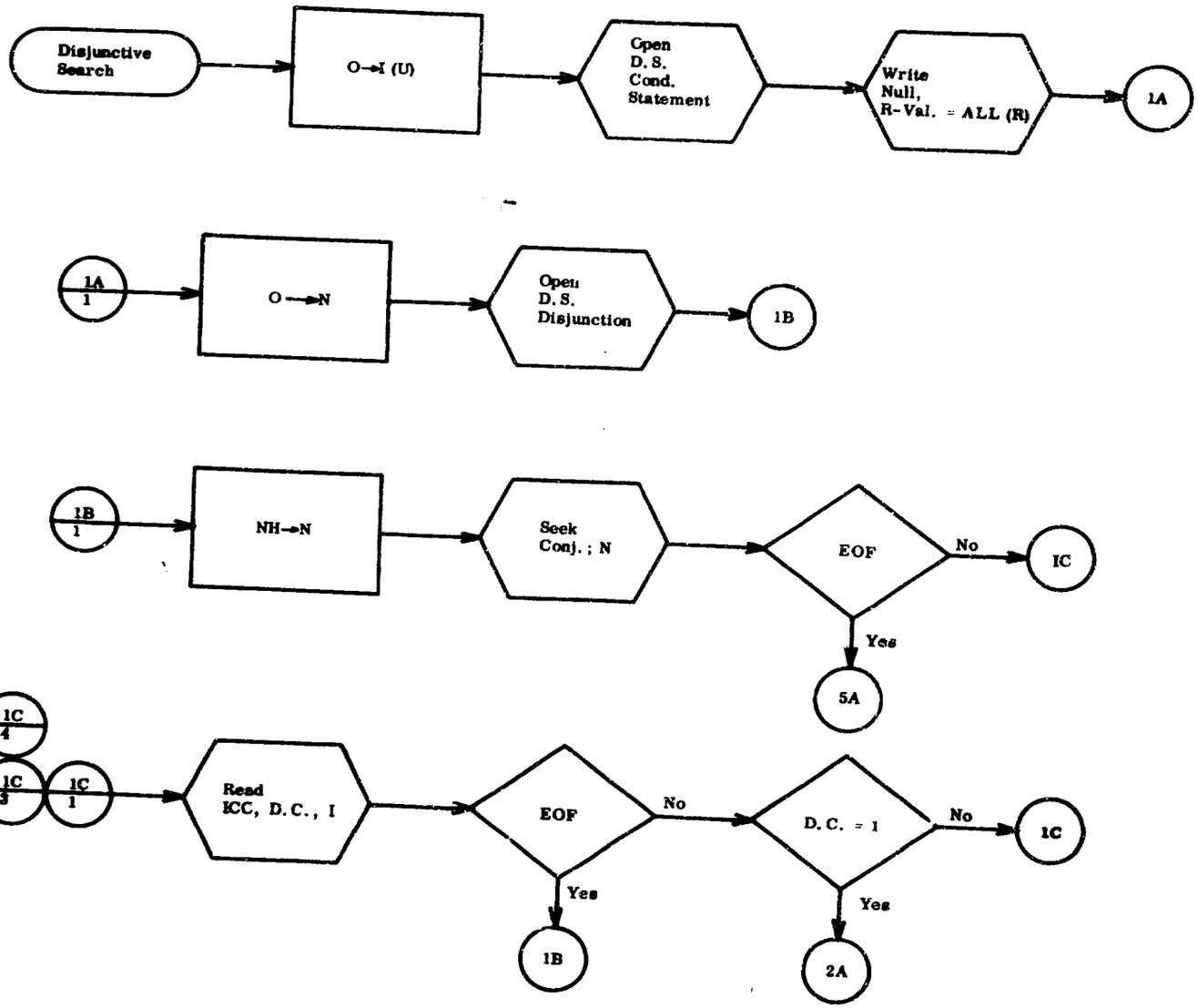
(2) If one I=1, intersect the two R-value lists and subtract the result from the list with I=1.
Set $I(\cup) = 1$. $[T3 : A \cup \neg B = I \dashv (B \dashv (A \cap B))]$

(3) If both I=1, intersect the two R-value lists.
$[T5 : \neg A \cup \neg B = I \dashv (A \cap B)]$

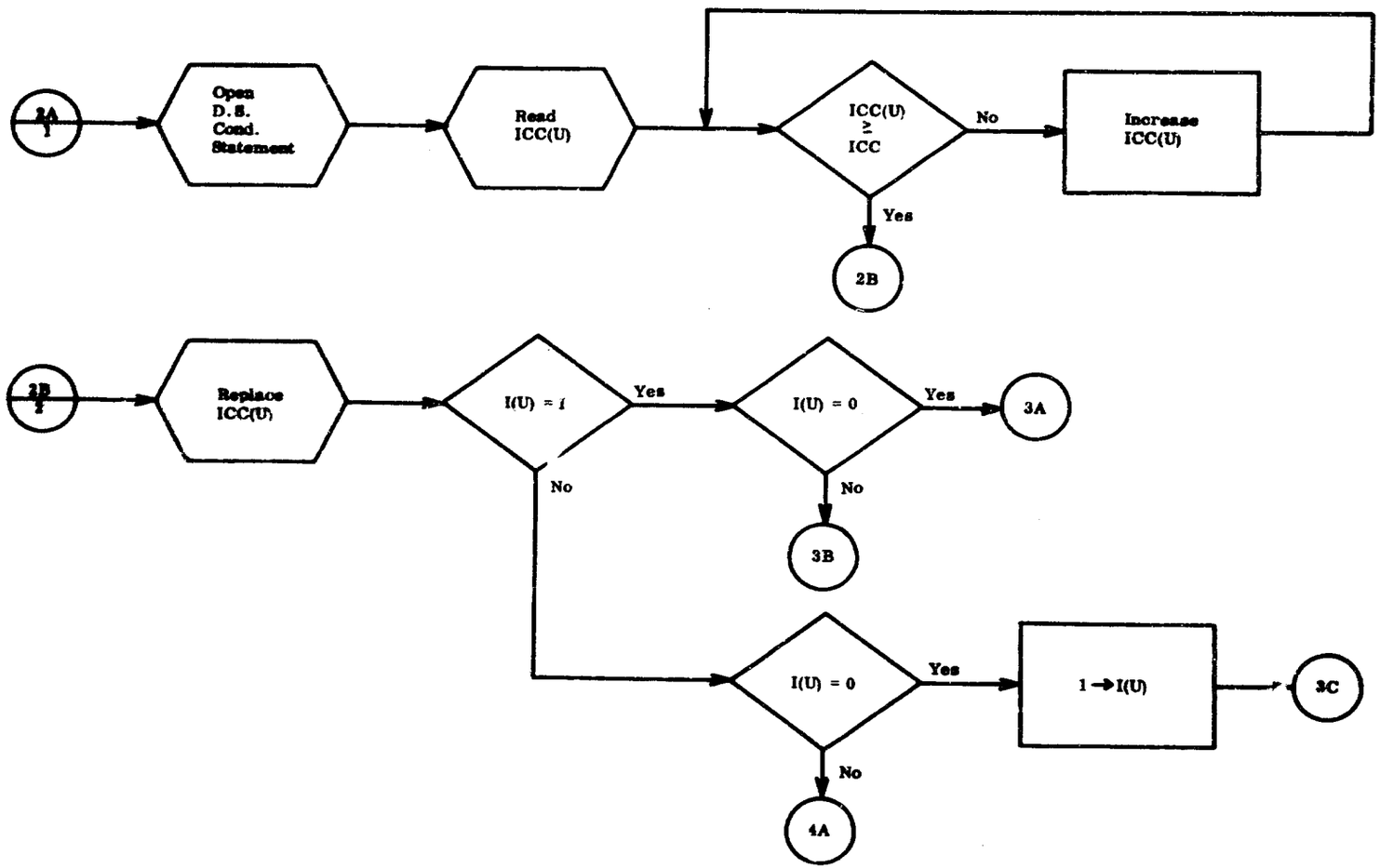
This process is continued until all R-Lists are exhausted. The result is a single R-VALUE List. When this is achieved, the Subtraction Phase is initiated. The following actions are taken:

(1) If $I(\cup)=0$, exit

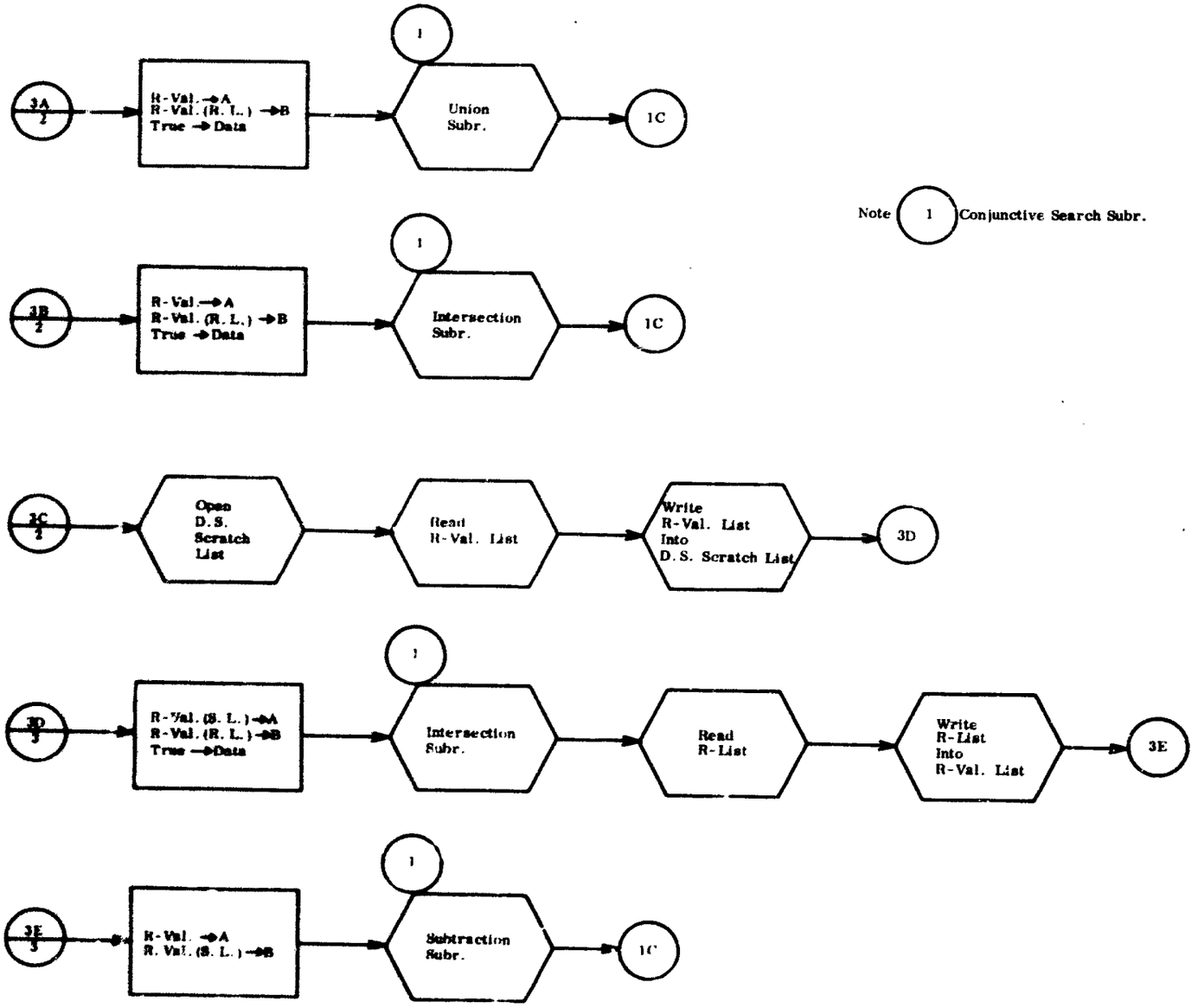
(2) If $I(\cup)=1$, subtract the R-VALUE List from a list of all R-values and exit. $[T1 : \neg A = I \dashv A]$



Disjunctive Search Job
Sheet 1 of 10



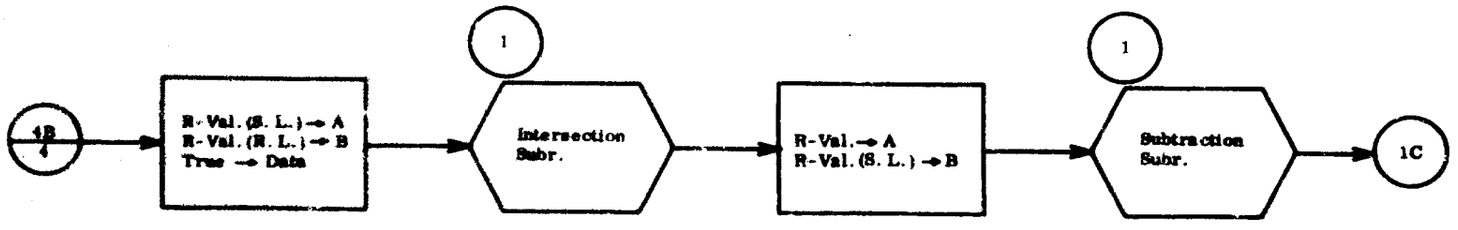
Disjunctive Search Job
Sheet 2 of 10



Note (1) Conjunctive Search Subr.

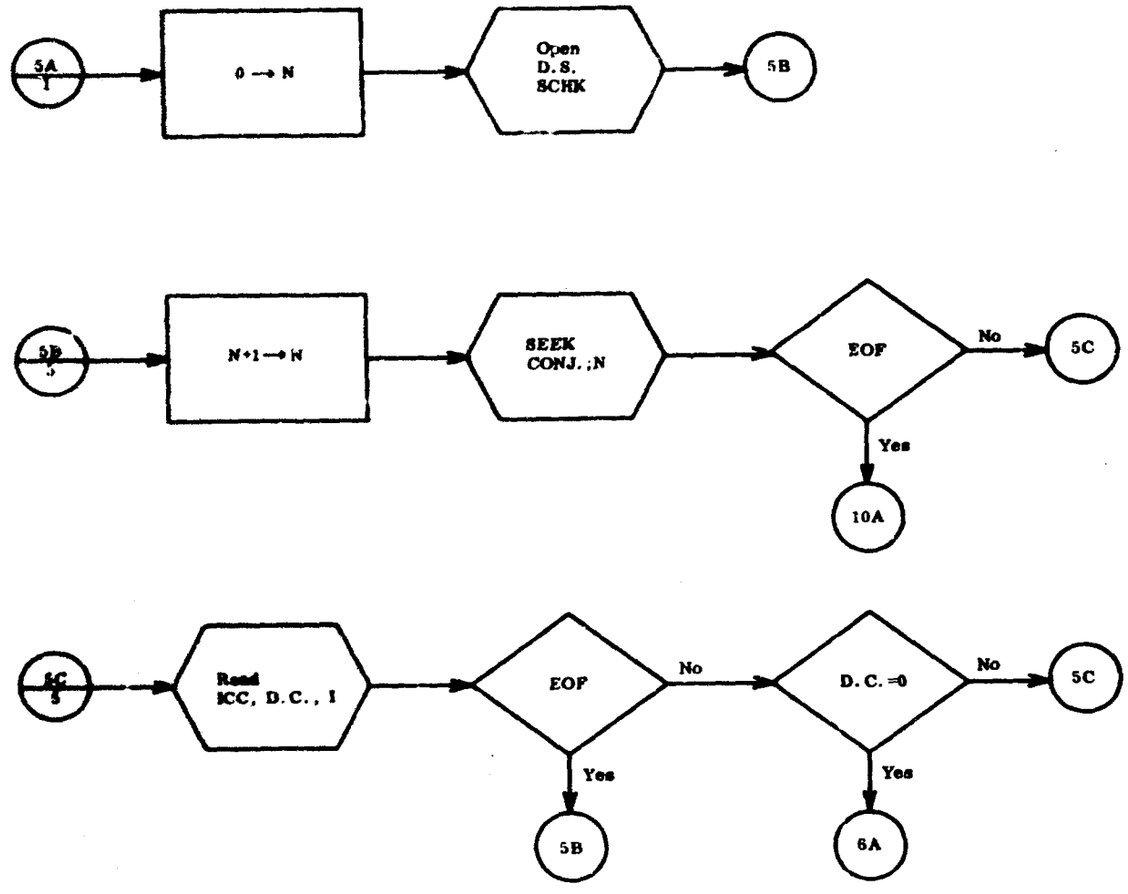
Disjunctive Search Job
Sheet 3 of 10



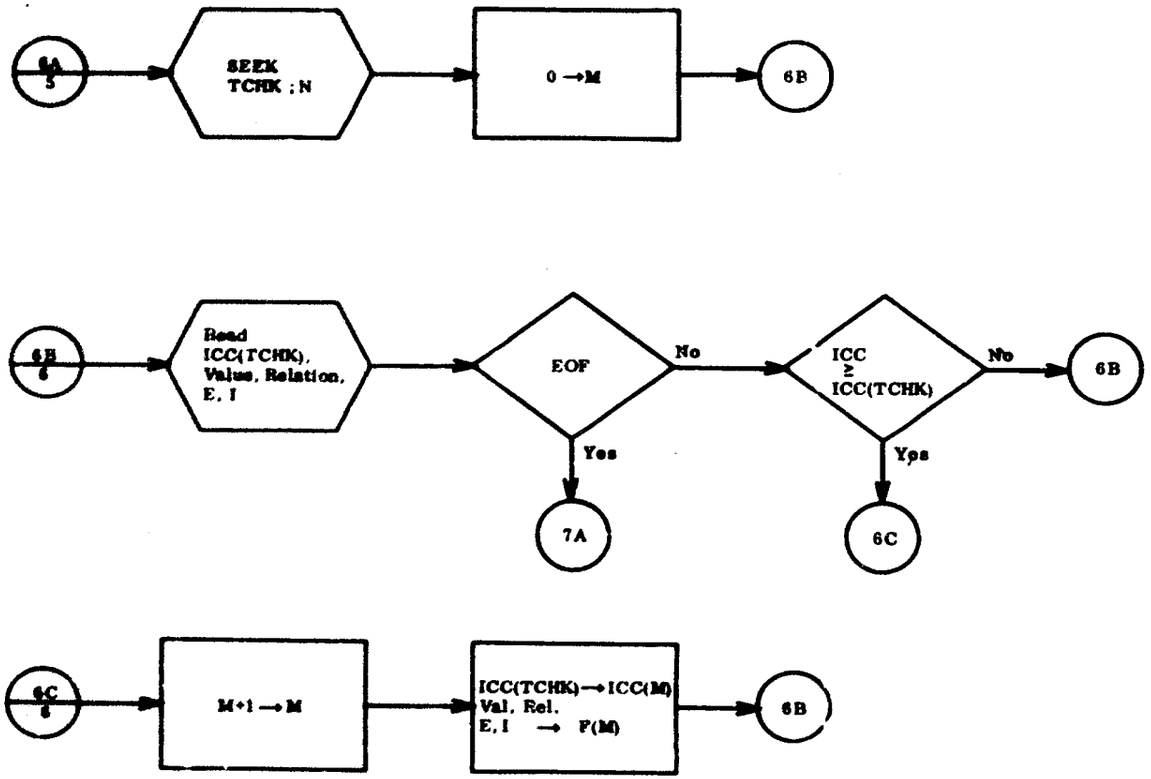


Note (1) Conjunctive Search Subr.

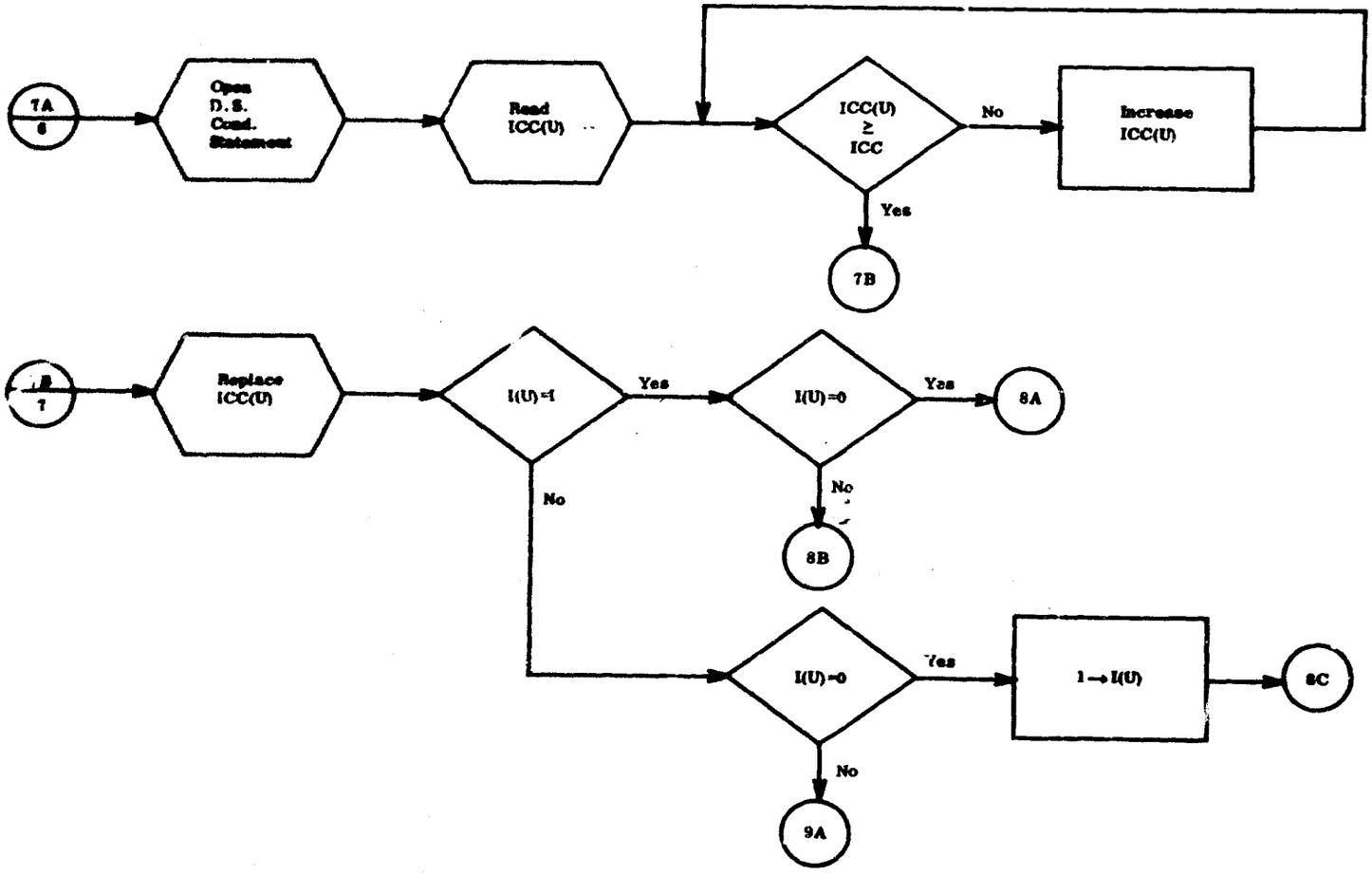
Disjunctive Search Job
Sheet 4 of 10



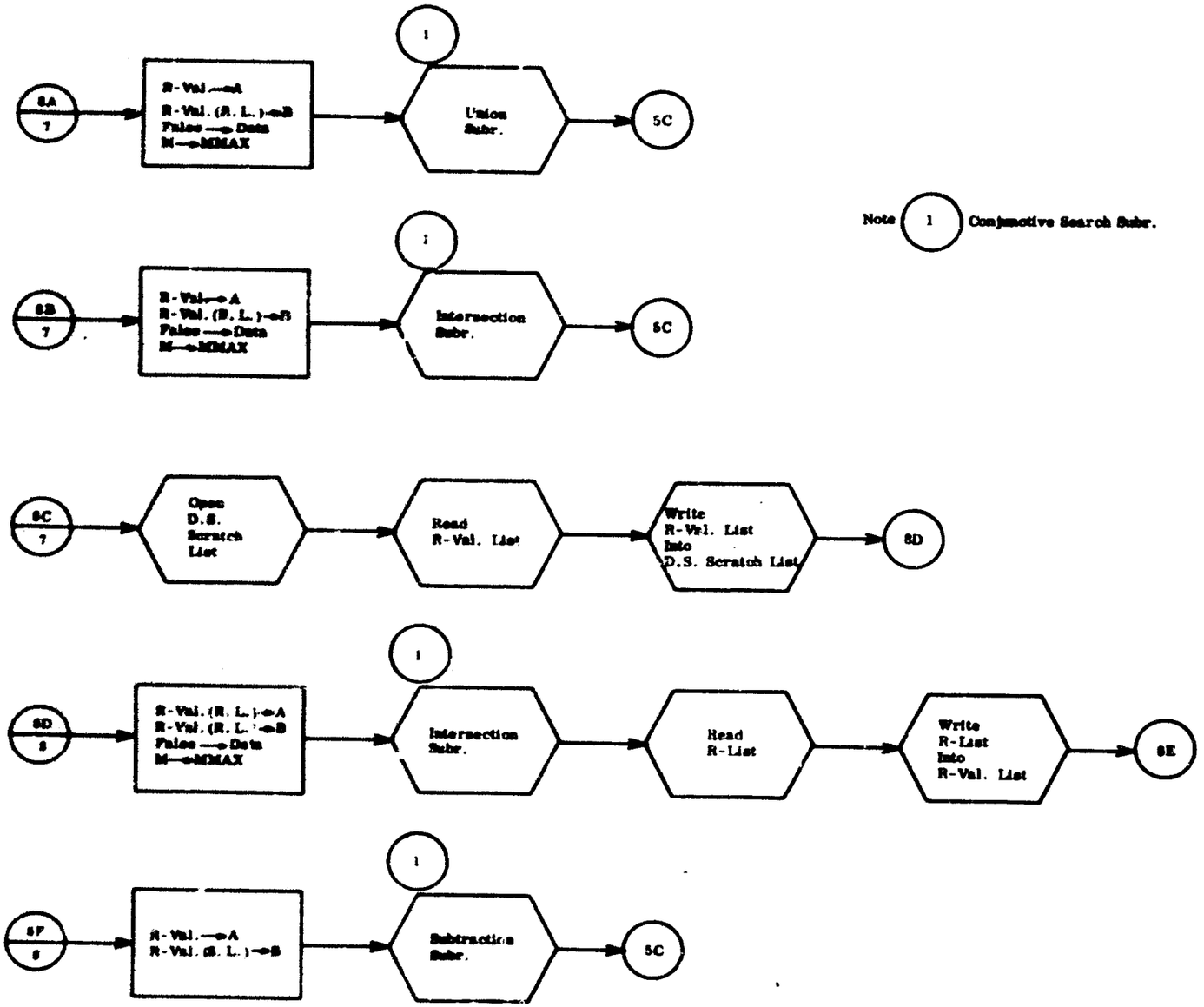
Disjunctive Search Job
Sheet 5 of 10



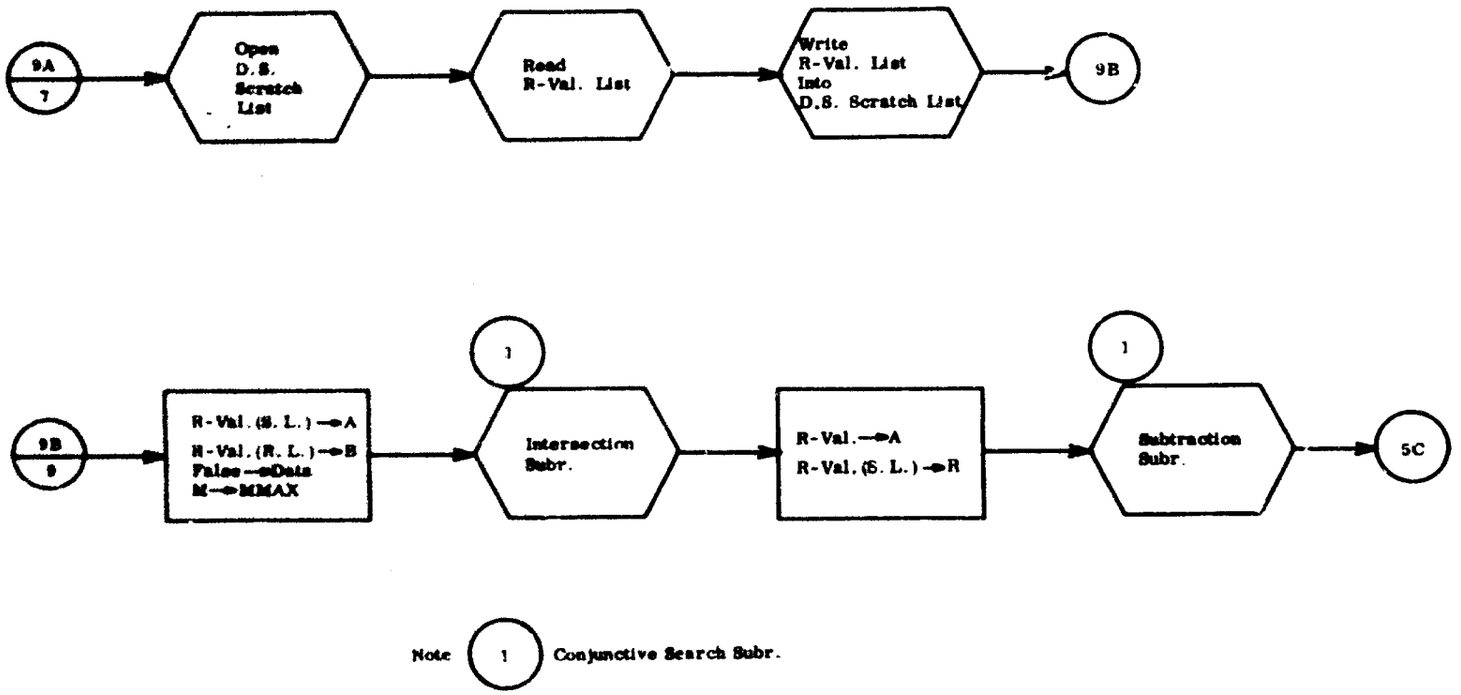
Disjunctive Search Job
Sheet 6 of 10



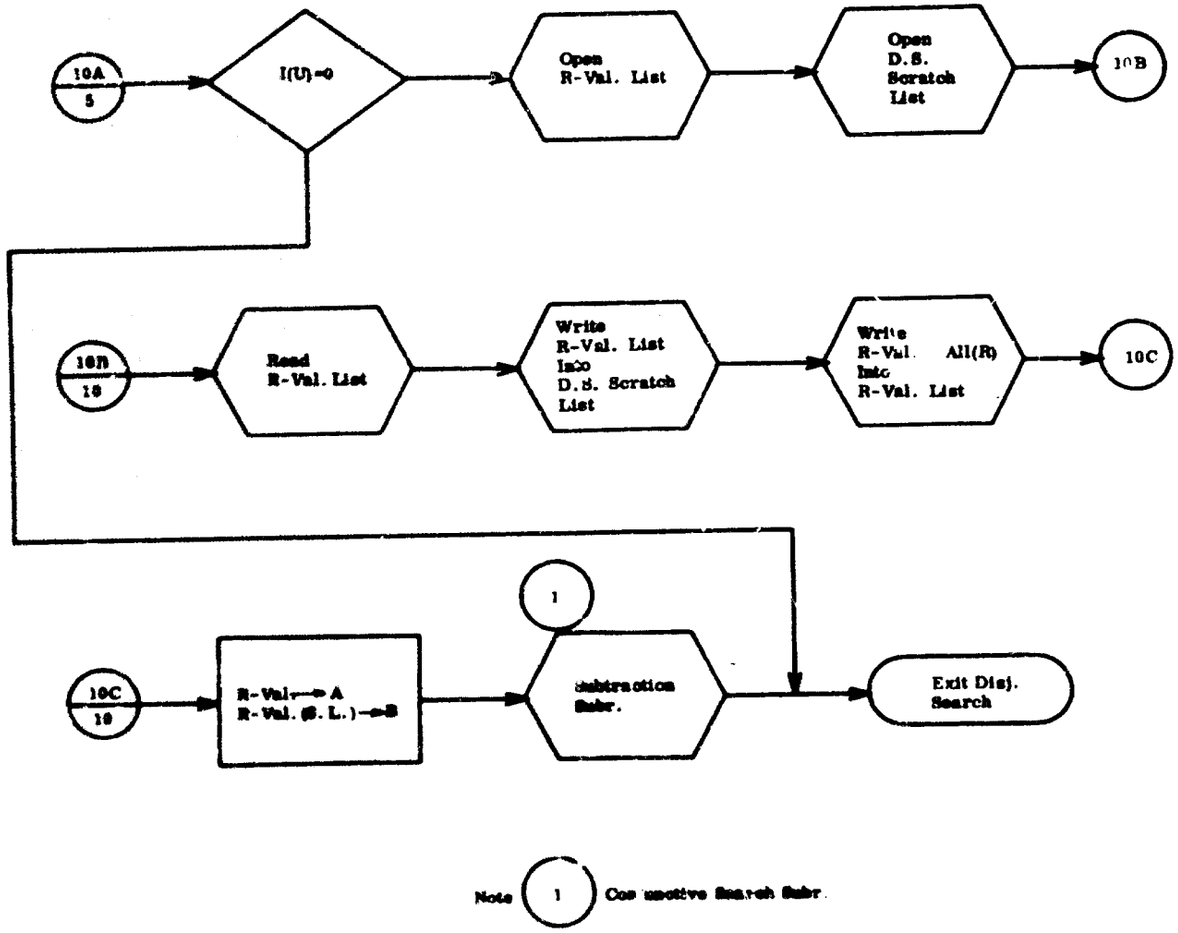
Disjunctive Search Job
Sheet 7 of 10



Disjunctive Search Job
Sheet 8 of 10



Disjunctive Search Job
Sheet 9 of 10


10A
5
I(U)=0
Open
R-Val. List
Open
D.S.
Scratch
List
10B
10B
10
Read
R-Val. List
Write
R-Val. List
into
D.S. Scratch
List
Write
R-Val. All(R)
into
R-Val. List
10C
10C
10
R-Val.→A
R-Val.(S.L.)→B
1
Subtraction
Subr.
Exit Disj.
Search
Note
1
Consecutive Search Subr.

7.1.7.2 Reformat

Job Request: REFORMAT (reformat qualifier), (reformat statement); (reformat item).

7.1.7.2.1 Functional Description

Reformat is a job which enables the programmer to reformat items of the data pool through the use of some English-like qualifier. The specific data items (records) to be mapped into the newly created structure are controlled by a Reformat statement. This statement is identical in item definition to the Conditional statement of the Conditional Search job and may, in fact, be the result of a previous Conditional Search. In those cases where all records of an item are desired, a Reformat statement which is initialized to read all R-values must be provided.

Three component jobs are required to form the Reformat job. These are:

(1) Qualifier Translation.

(2) Extract.

(3) Restructure.

A qualifier is a specification, in the form of English-like statements, of the item structure into which one or more input items are formed. It has two major statement types, both of which are necessary to specify completely any reformat. These are:

(1) Feed Statements.

(2) Field Statements.

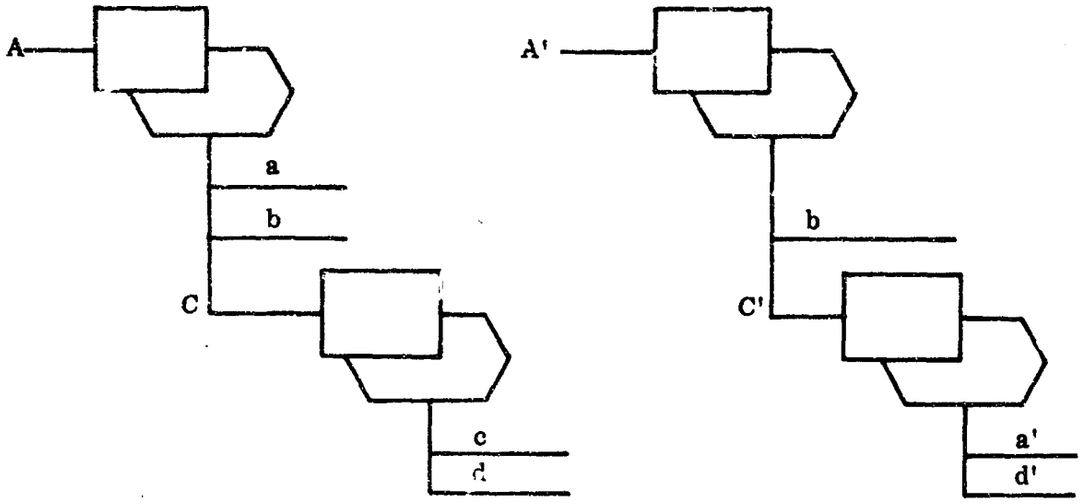
Feed statements define the manner in which the desired item file structure is developed from one or more input items. A Feed statement is required for every file of the new item. The three types are:

(1) One-to-One.

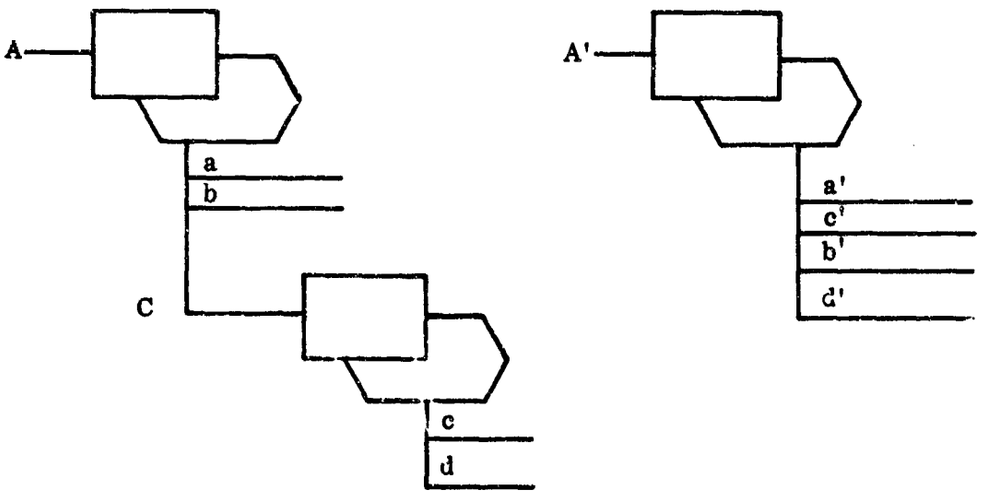
(2) Distribution.

(3) Factorization.

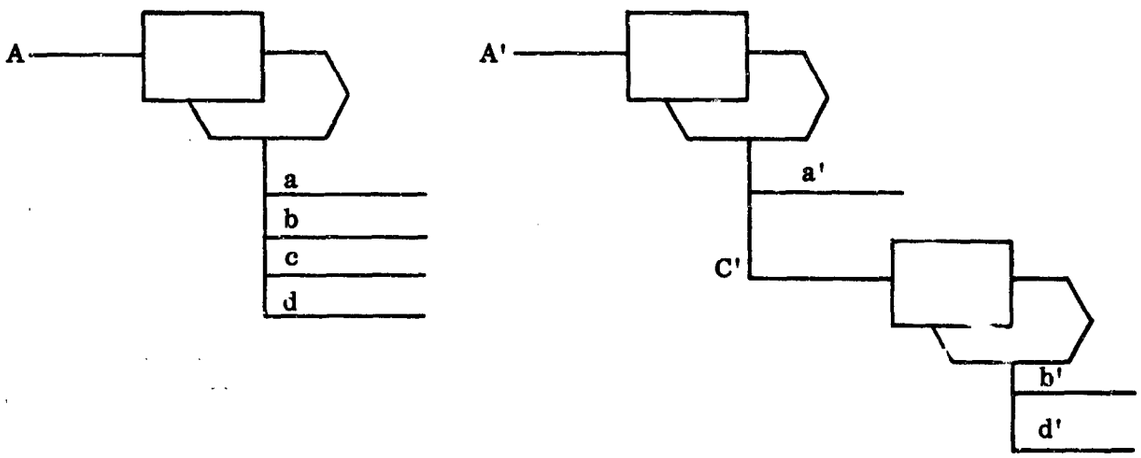
One-to-one Feed statements imply that a particular file of the new item (or any part of this item) is created from a specified file of one or more input items. Thus, in the following example, the primed files are generated by one-to-one Feed statements:



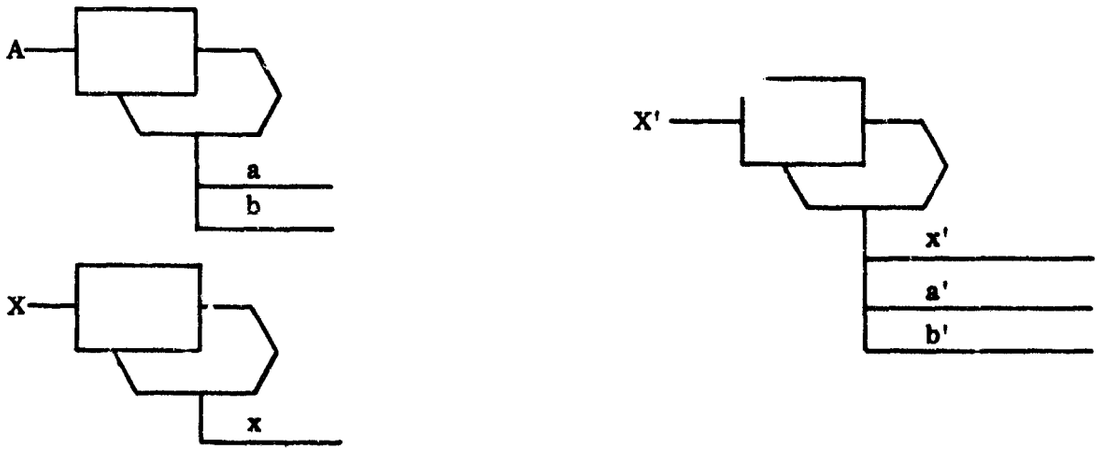
Distribution implies that a particular file of the new item is created from more than one specified file of a single input. Thus, in the following example, the primed file is generated by a Distribution Feed statement:



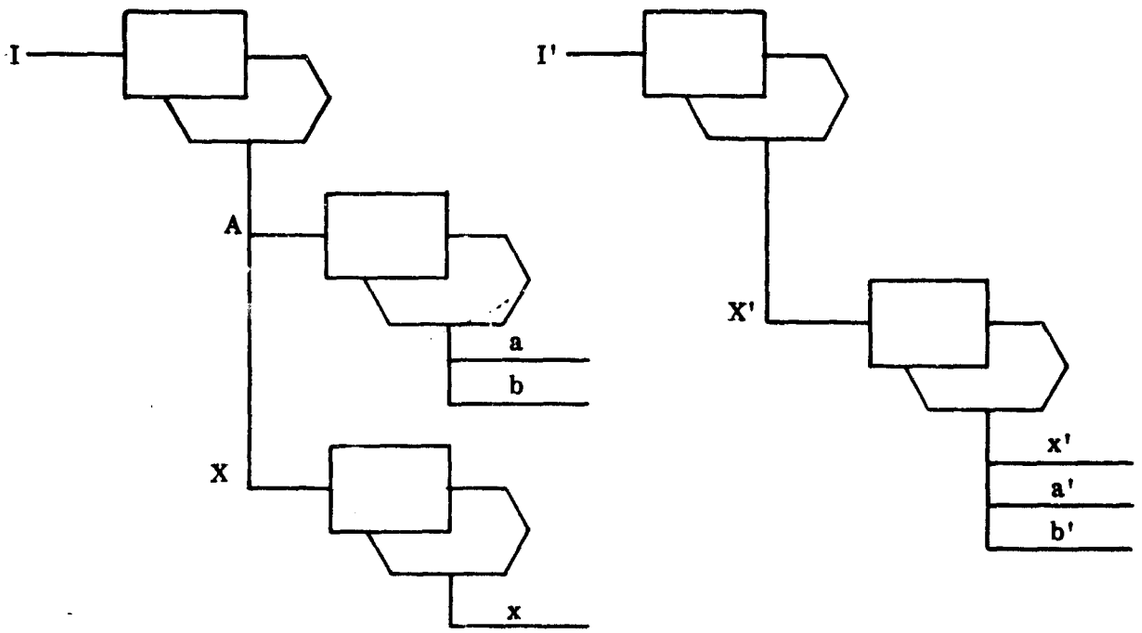
Factorization is the inverse of distribution and implies that a subsumed file of the new item is created without the existence of this file in any input item. To accomplish this, the parent of the subsumed file must be created from a specified file of one input in which a factor control field exists. A value change in this specified field generates the subsumed file. Thus, in the following example the primed subsumed file is generated by a Factorization Feed statement in which the field a is the factor control:



Field statements define the record item structure in terms of ordered field names which originate in the input items. A field statement is required for every field of the new item. The actual values that are transferred to the new item are determined by the R-values of the Reformat statement. Thus, in the following example, fields a and b of the first input item are merged with field x of the second input to form a single record of the new item:

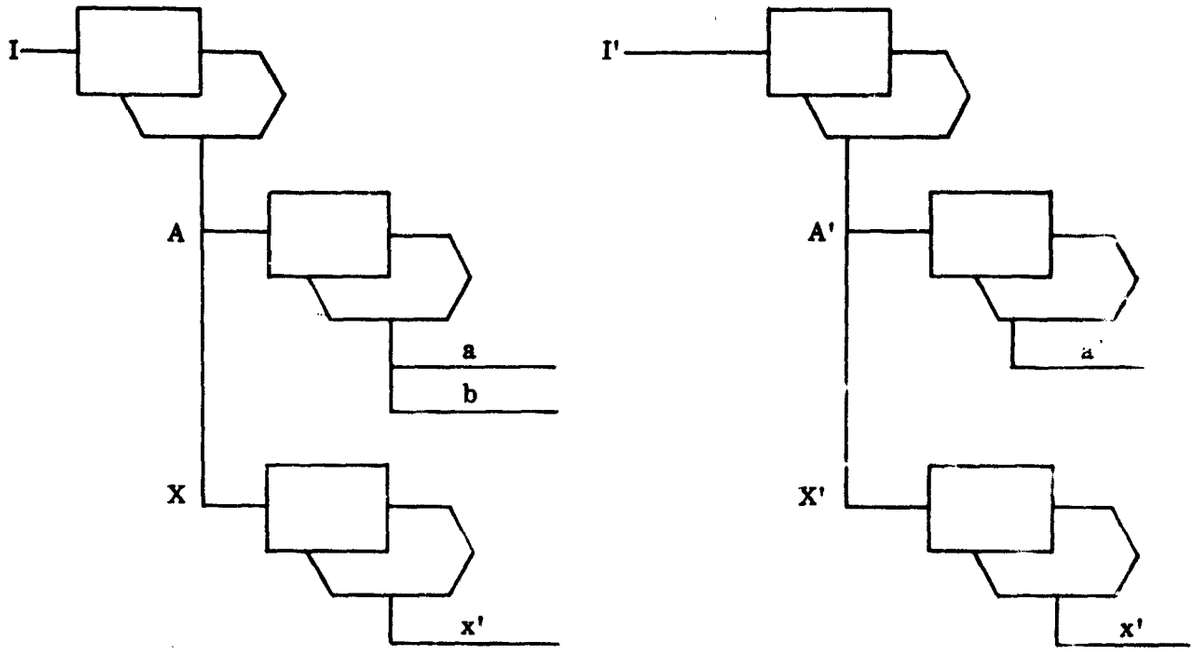


There are no restrictions to the use of Reformat except for that of parallel input files. Parallel files are files of identical item levels within a single item. The restriction is simply that, when parallel files constitute a single input, they may not be merged to form fewer files of a desired item. Thus, in the following example, the Reformat is not permitted when file I constitutes a single input:



Nevertheless, this Reformat may be accomplished if file A and file X constitute two distinct inputs. The objective is demonstrated in the previous example.

It should be noted that there are no restrictions on parallel input files if their respective fields are not merged. Thus, in the following example, no restriction applies:



7.1.7.2.2 Inputs

(1) REFORMAT QUALIFIER, A, V

(2) REFORMAT STATEMENT, S, 2

ICC(UNIVERSE), H. V
R-VALUE LIST, F

R-VALUE, H, V

7.1.7.2.3 Results

(1) REFORMAT ITEM (DYNAMIC)

7.1.7.2.4 Directories Used

(1) Item List.

(2) Term List.

7.1.7.2.5 Services Used

(1) Open for Reading.

(2) Close for Reading.

(3) Seek.

(4) Read.

(5) Fix Item.

(6) Open for Writing.

(7) Close for Writing.

(8) Write.

(9) Term Name to ICC Translation.

(10) Retrieve IL Entry.

(11) Retrieve Term List Entry.

7.1.7.2.6 Jobs Used. No Job Extensions are used.

7.1.7.2.7 Method of Operation. The Reformat job is entered into the system via the following Job Description:

Job Name: REFORMAT

Job Inputs: reformat qualifier, reformat statement.

Job Outputs: reformat item.

Job Components:

(1) QUALIFIER-TRANSLATION: **reformat qualifier;**
reformat k statement, reformat l list.

(2) EXTRACT: **reformat k statement, reformat l list;**
reformat node list.

(3) RESTRUCTURE: **reformat statement, reformat node list;**
reformat item.

The internal Job Description may be represented graphically as shown in Figure 7-6.

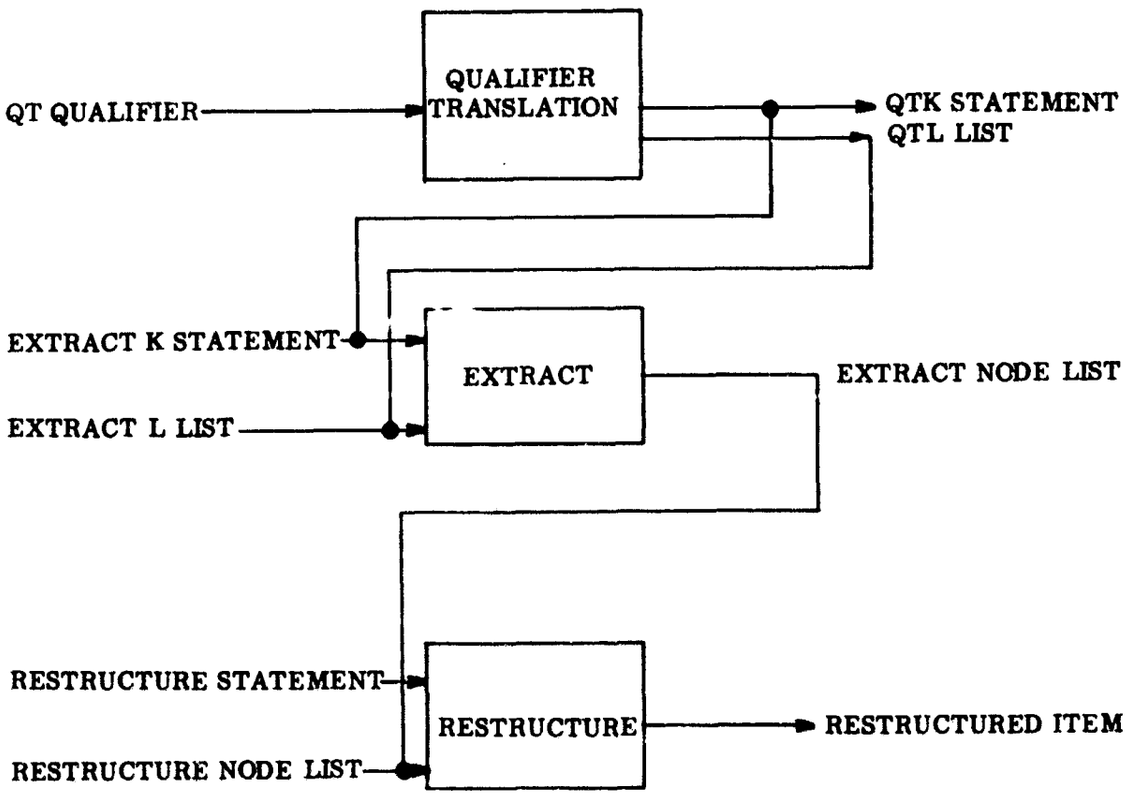


Figure 7-6. Reformat Job, Internal Job Description

From the qualifier sentence the Qualifier Translation job develops the K statement and the L list. The K statement contains two files each of which constitutes an input item. Each file record contains the field name of a desired field, which is to be mapped into the new item. The L list constitutes an internal representation of the desired item structure. With these two results the Extract job forms a detailed node list in the following steps:

(1) All field names are translated to ICC's.

(2) The first input item is broken down into major nodes where a new node is necessitated by each parallel file within the input item.

(3) Each major node is further broken down into K input item levels and L output item levels.

(4) Within every K^{th} level, the ICC's of each input field are ordered in ICC order.

(5) Within every L^{th} level, the output field representations are made to reference a $(K, I)^{th}$ input field representation.

With this detailed guideline the Restructure job creates the desired item in the following steps:

(1) The item is fixed in the data pool.

(2) At the U^{th} major node, the input data items modified by the R-value list are mapped into the output item structure.

(3) The above step is repeated until the major nodes are exhausted.

7.1.7.2.7.1 Qualifier Translation

Job Request: QUALIFIER-TRANSLATION (qt qualifier);
(qtk statement), (qtl list).

7.1.7.2.7.1.1 Functional Description. Qualifier Translation involves two phases:

(1) Translation of the qualifier from English-like statement form to linear algebraic form, and

(2) Translation of the linear algebraic form to the K statement and L List form.

The linear algebraic translation utilizes INSCAN, with the appropriate action-graph, to translate from English-like statement form to linear algebraic form.

In the second phase, the Qualifier Translation consists of separation of the linear algebraic qualifier into input and output representations. The following actions are taken:

(1) Separate all field names into one or two input files.

(2) Separate all field names by Feed statements into L output files.

7.1.7.2.7.1.2 Inputs

Q. T. QUALIFIER, A, V

7.1.7.2.7.1.3 Results

(1) Q. T. K STATEMENT, S, 2

IN1 LIST, F

NAME (IN1), A, V

IN2 LIST, F

NAME (IN2), A, V

(2) Q. T. L LIST, F

FEED, I, 6
FACTOR NAME, A, V
OUT LIST, F

NAME (OUT), A, V

7.1.7.2.7.1.4 Directories Used. No directories are used.

7.1.7.2.7.1.5 Services Used

(1) Open for Reading.

(2) Close for Reading.

(3) Seek.

(4) Read.

(5) Open for Writing.

(6) Close for Writing.

(7) Write.

7.1.7.2.7.1.6 Jobs Used. No Job Extensions are used.

7.1.7.2.7.1.7 Method of Operation. The appropriate action-graph is compiled with the Qualifier Translation program. The graph is of an English-like request language to linear operation type. Upon generation of the linear algebraic form, the K statement and L list are formed and written.

7.1.7.2.7.2 Extract

Job Request: EXTRACT (extract k statement), (extract l list); (extract node list).

7.1.7.2.7.2.1 Functional Description. Input and output linear algebraic representations may be merged into a detailed node specification which symbolically represents the necessary linear steps in creating the output item. This consists of a hierarchy of specific blueprints by which any output item can be viewed. These are:

(1) The U major node specifications by which parallel files are effectively separated into unique items,

(2) The K input and L output item levels of each U^{th} major node, and

(3) The sequential I input and J output fields within each K^{th} input and L^{th} output item level.

7.1.7.2.7.2.2 Inputs

(1) EX, K STATEMENT, S, 2
IN1 LIST, F
NAME (IN1), A, V
IN2 LIST, F
NAME (IN2), A, V

(2) EX, L LIST, F
FEED, I, 6
FACTOR NAME, A, V
OUT LIST, F
NAME (OUT), A, V

7.1.7.2.7.2.3 Results

EX, NODE LIST, F
LEVEL K, F
IN1 LIST, F
IN1, H, V
IN2 LIST, F
IN2, H, V
LEVEL L, F
FEED, I, C
FACTOR, H, V
OUT LIST, H
OUT, H, V

7.1.7.2.7.2.4 Directories Used. No directories are used.

7.1.7.2.7.2.5 Services Used

(1) Open for Reading.

(2) Close for Reading.

(3) Seek.

(4) Read.

(5) Open for Writing.

(6) Close for Writing.

(7) Write.

(8) Term Name to ICC Translation.

7.1.7.2.7.2.6 Jobs Used. No Job Extensions are used.

7.1.7.2.7.2.7 Method of Operation. The following actions are taken:

(1) Translate all input field names to ICC's and order these into the IN1(I) and IN2(I) column matrices through repeated use of the FIND ICC (MIN) subroutine.

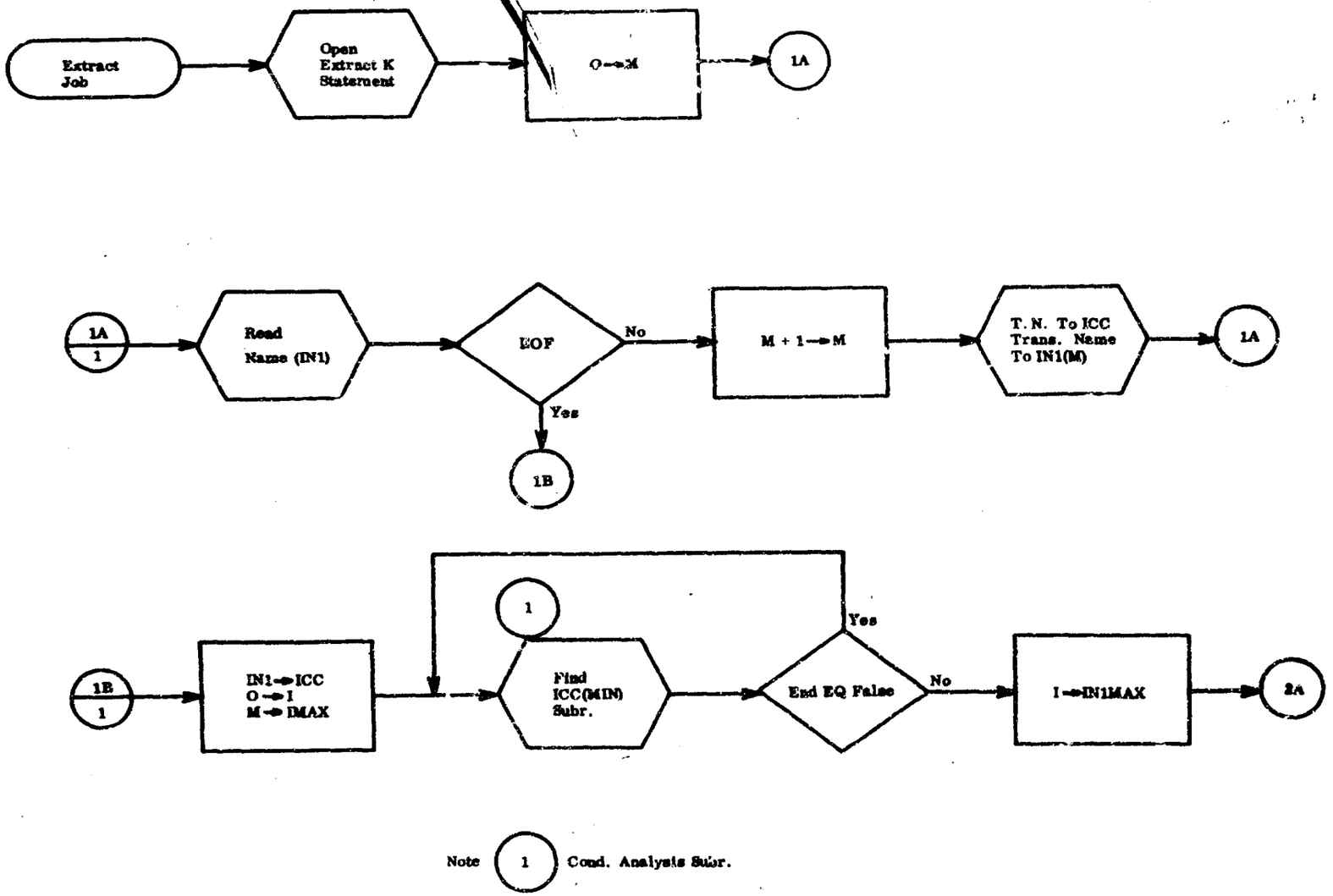
(2) At every L^{th} level, translate the factor name to an ICC, form the FACTOR (L) column matrix, and translate all field names to ICC's of the OUT (L, J) matrix in the order in which they appear.

(3) Group all possible input ICC's to form the next major node of the NODE 1(U) and NODE 2(U) column matrices. Write Node record.

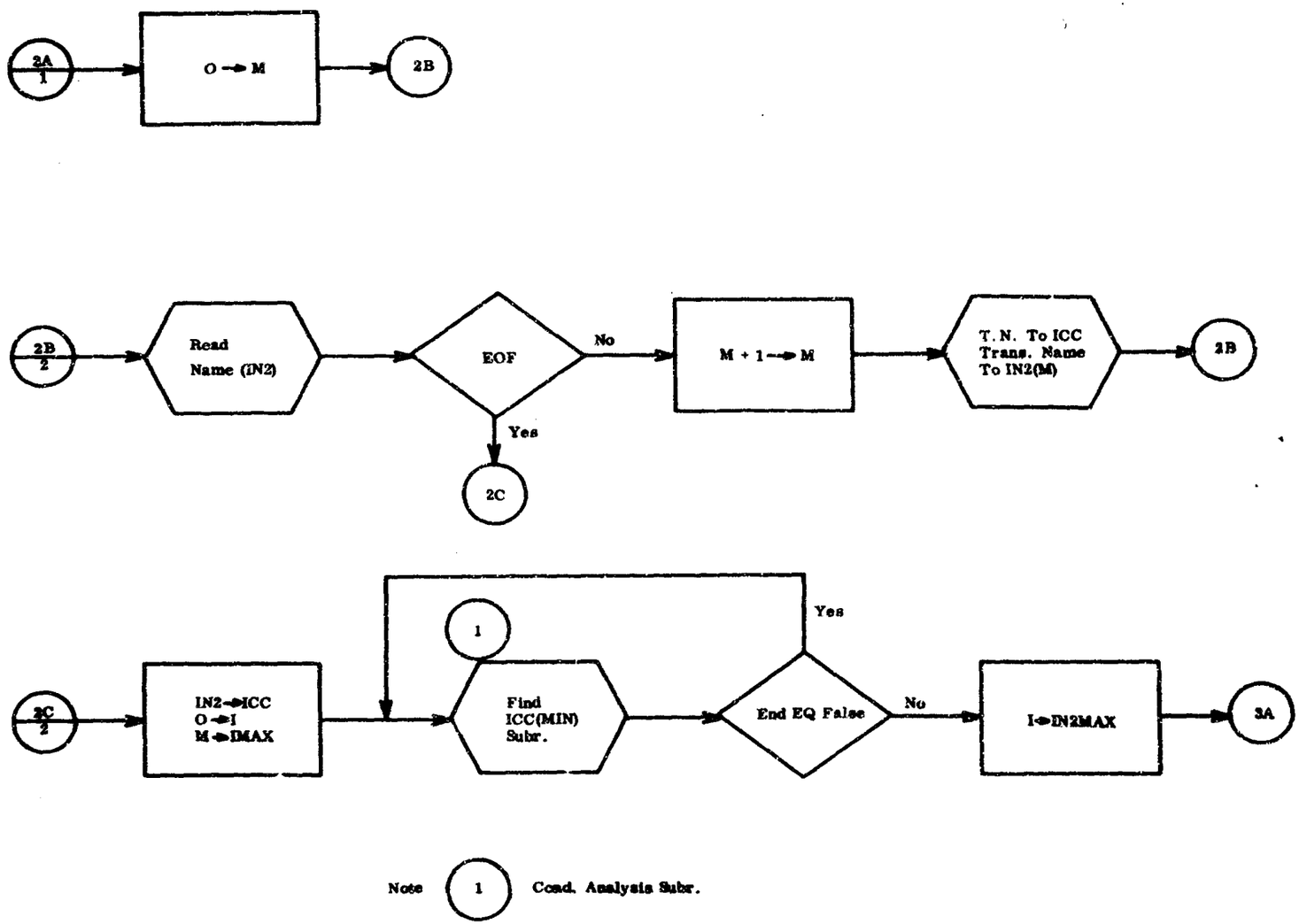
(4) Determine all possible K input levels and form the IN (K, Y) matrix of every possible input ICC. Write Level K record and IN1 and IN2 lists.

(5) Determine the L output levels and reconstruct the FACTOR (L) and OUT (L, J) matrices by replacing their respective elements with (K, Y) values which reference input (source, elements) of the IN (K, Y) values which reference input (source, elements) of the IN (K, Y) matrix. Write Level L record, Feed and Factor statements, and Out list.

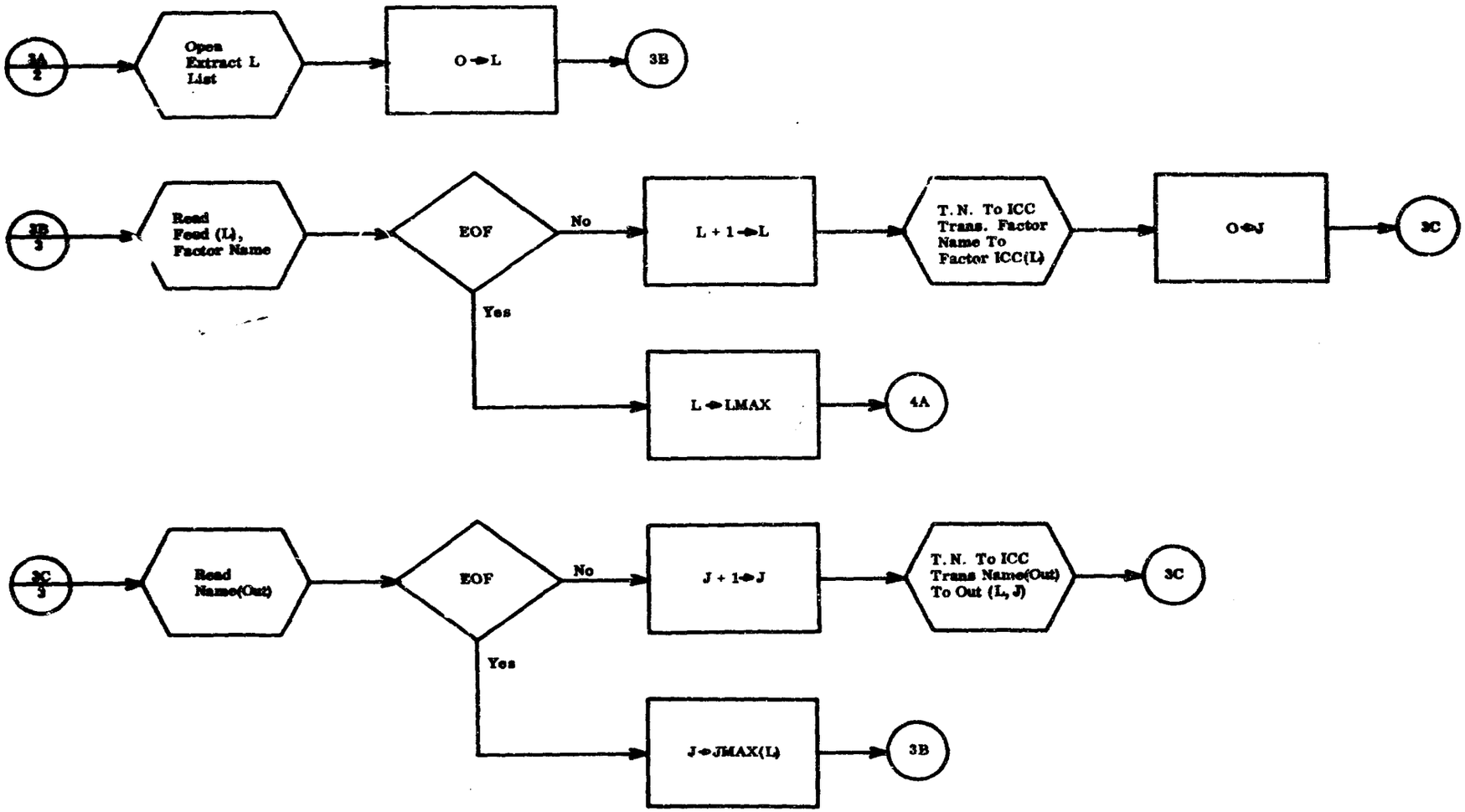
(6) Repeat the last three steps above until all major nodes are exhausted.



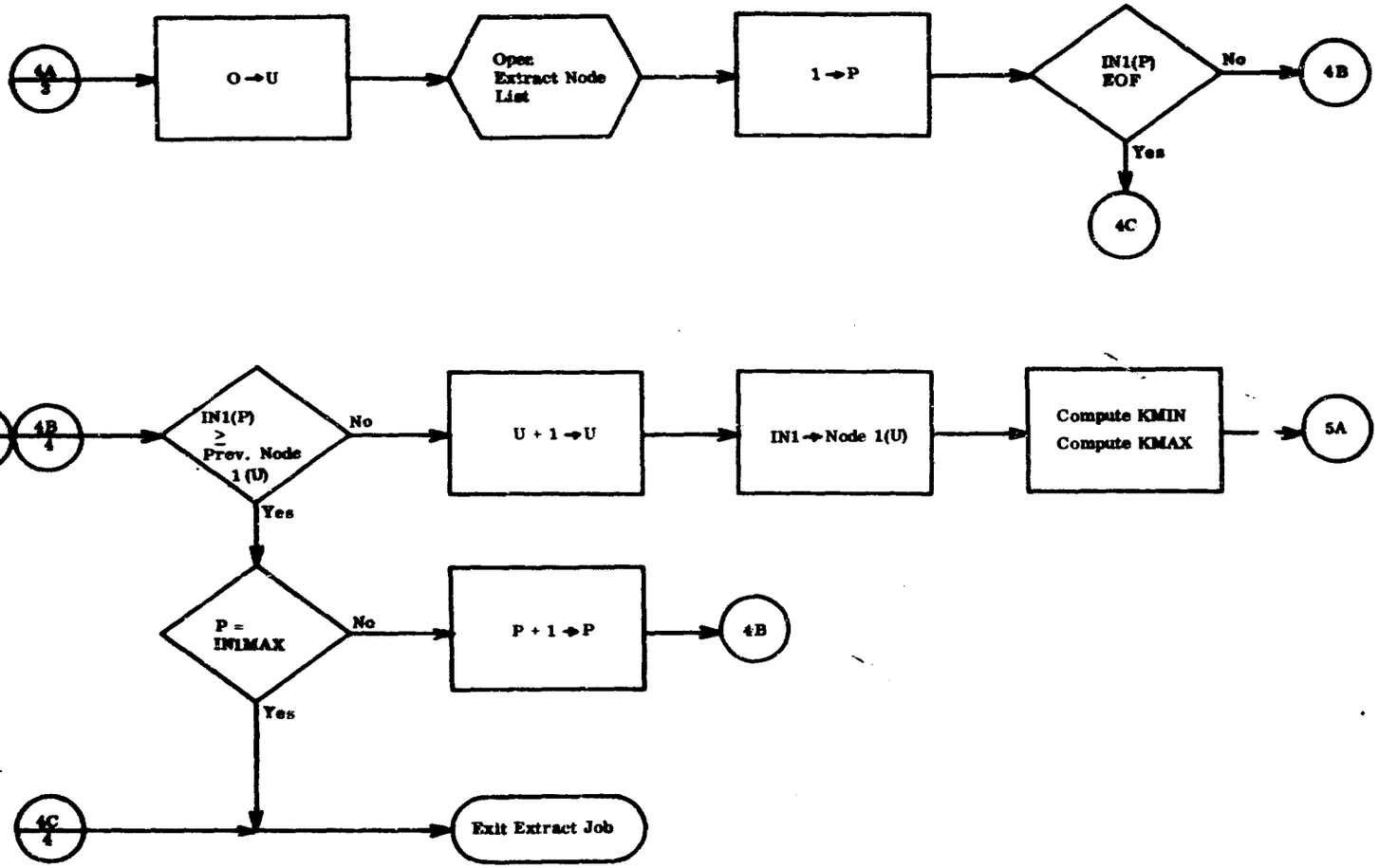
Extract Job
Sheet 1 of 11



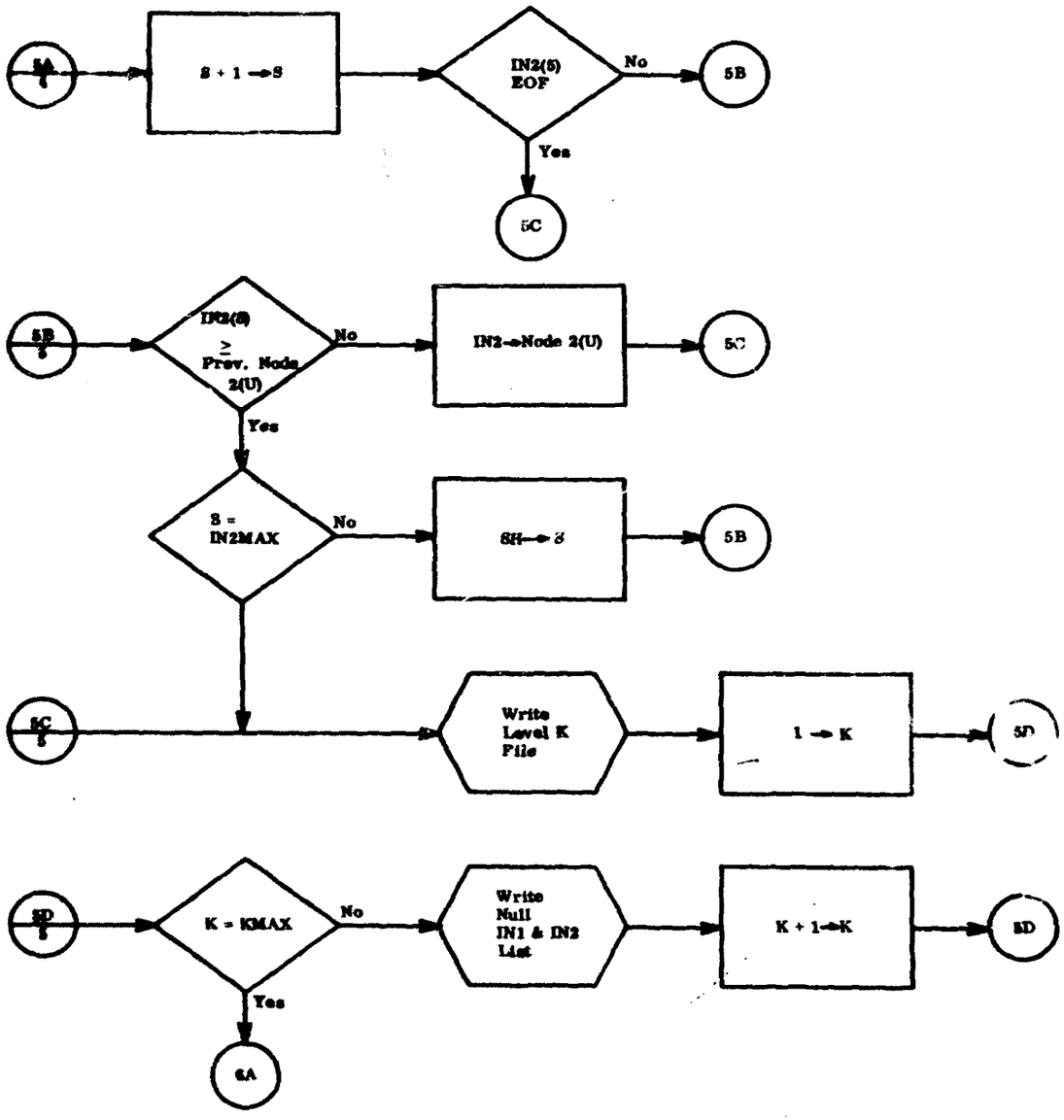
Extract Job
Sheet 2 of 11



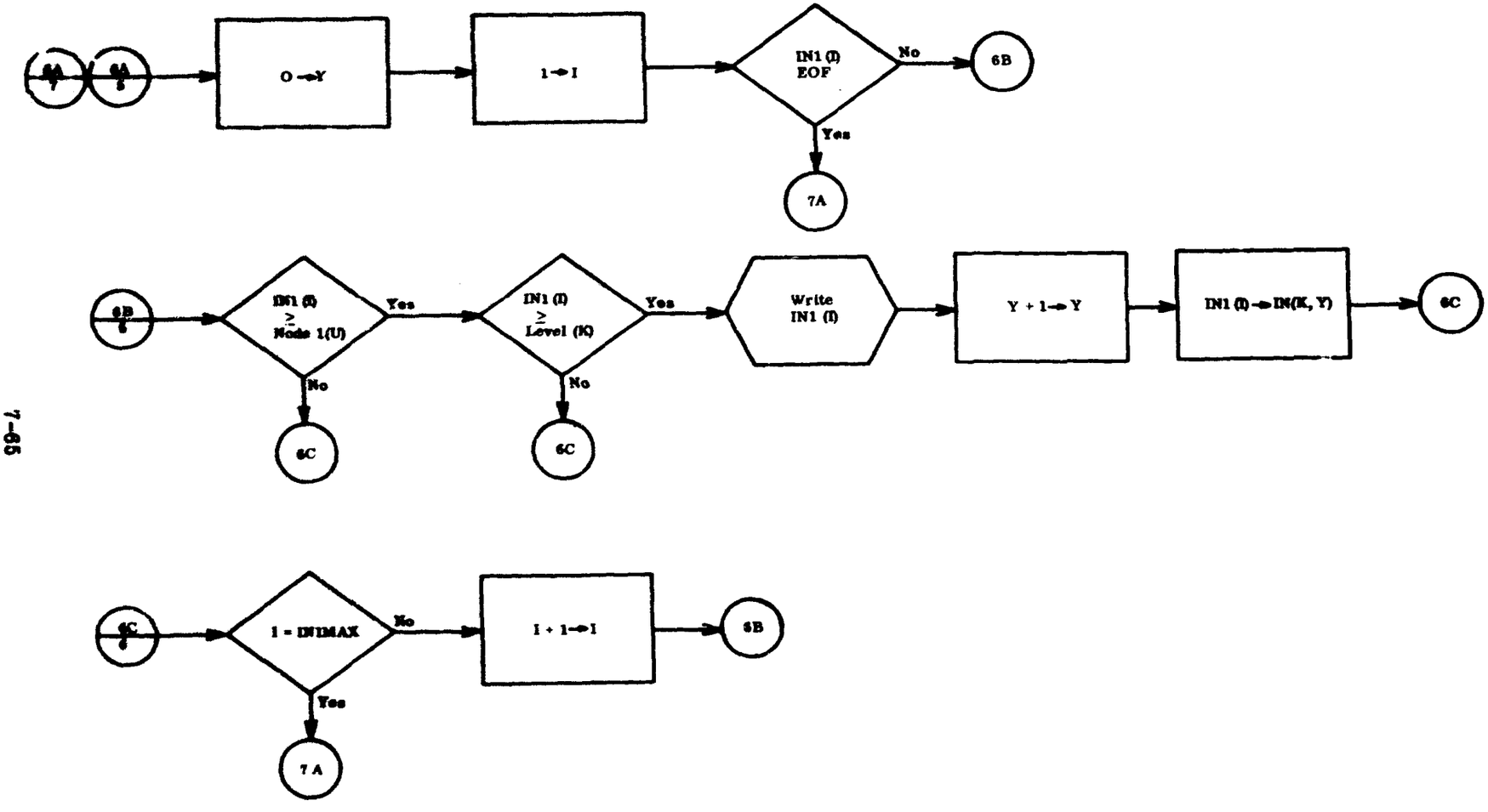
Extract Job
Sheet 3 of 11



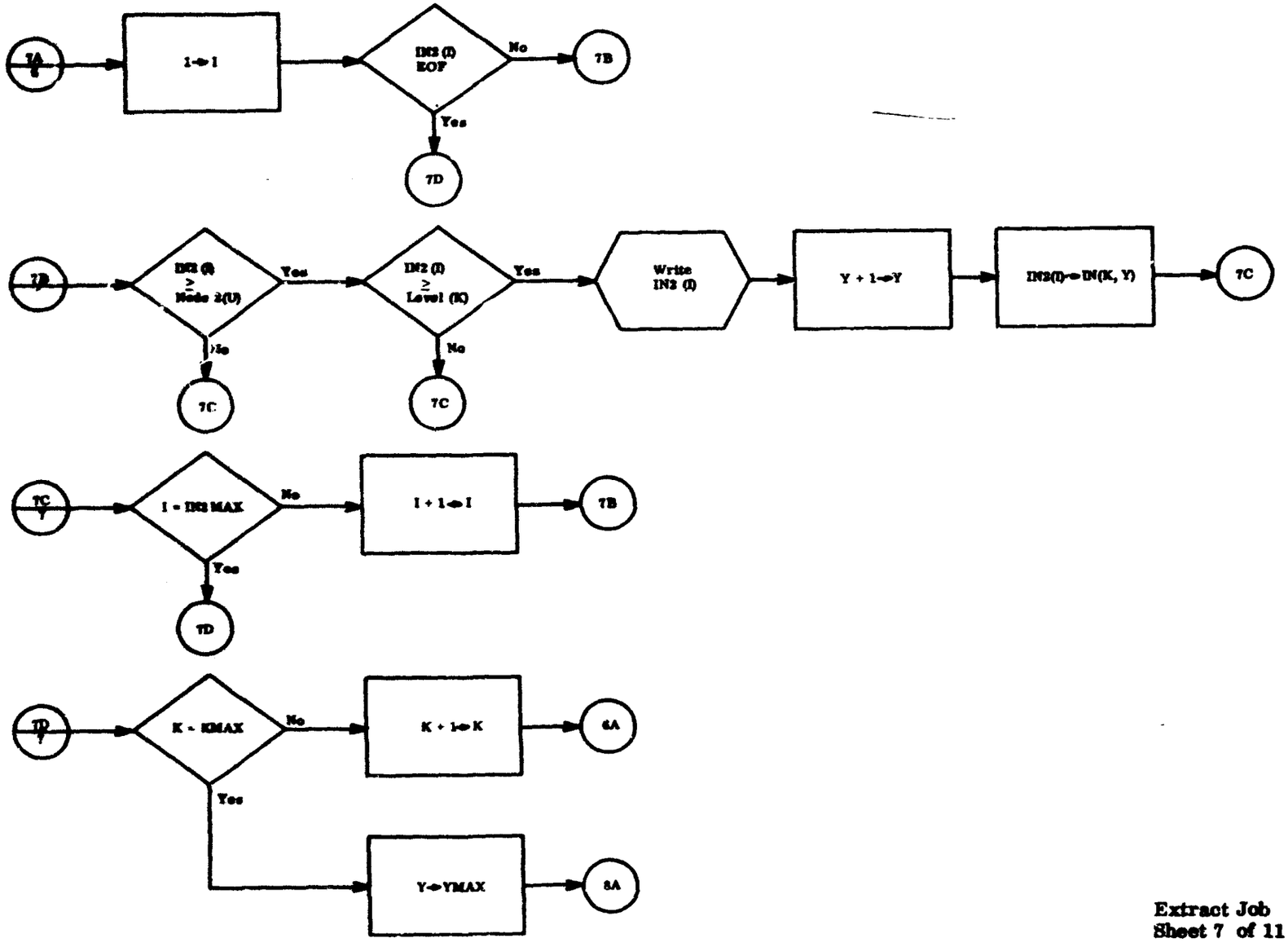
Extract Job
Sheet 4 of 11



Extract Job
Sheet 5 of 11

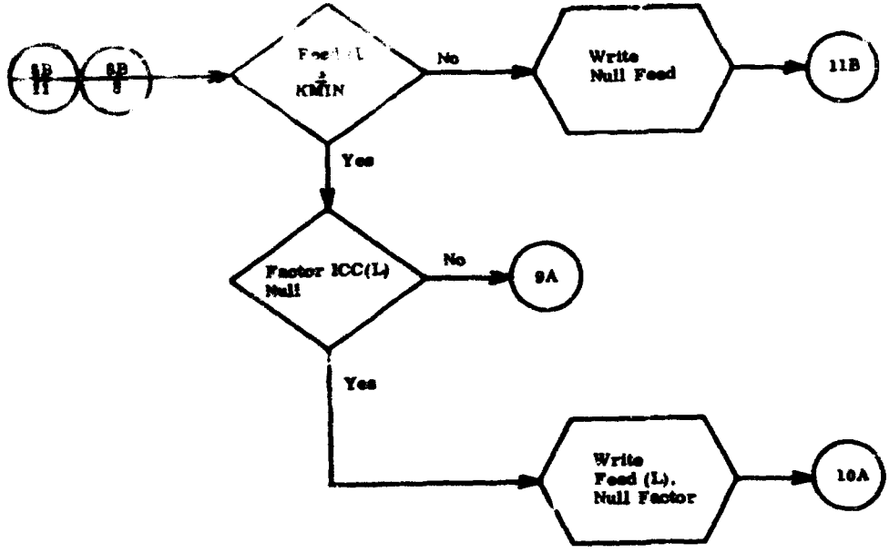


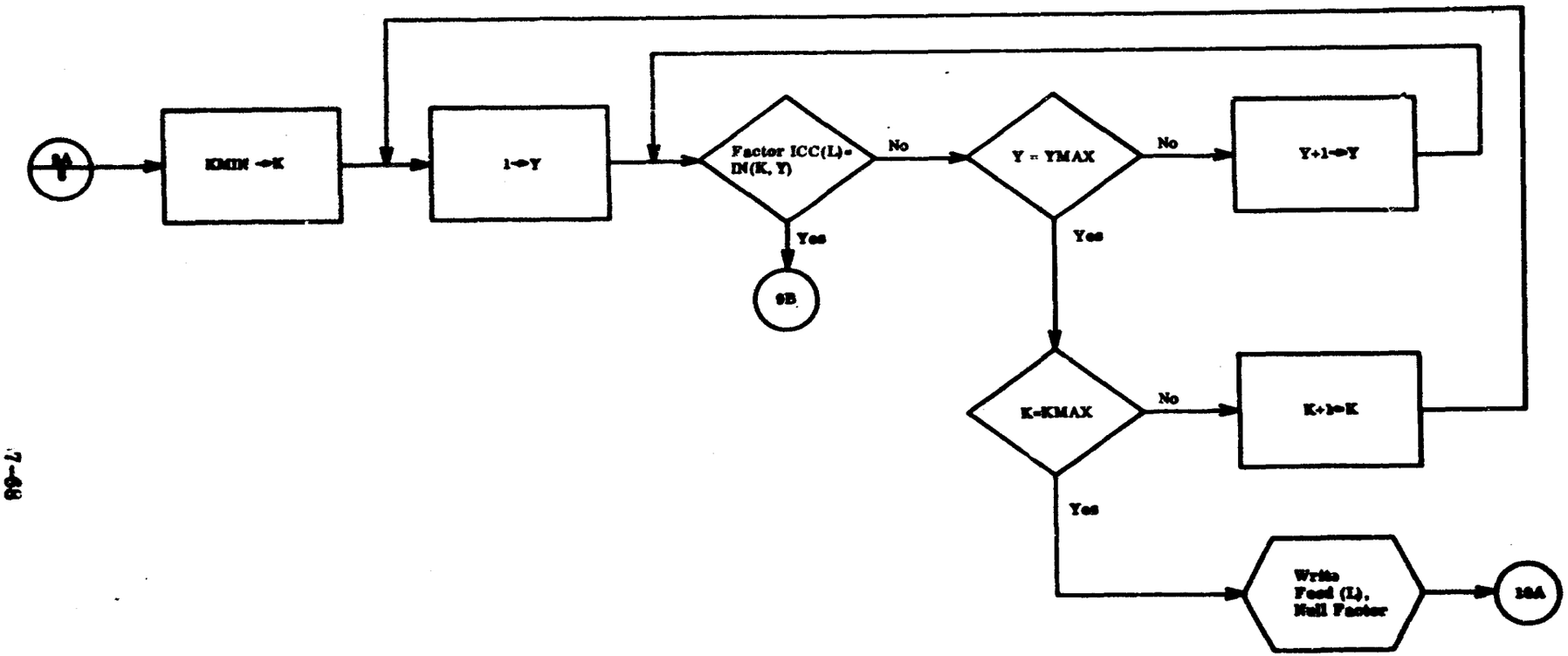
Extract Job
Sheet 6 of 11

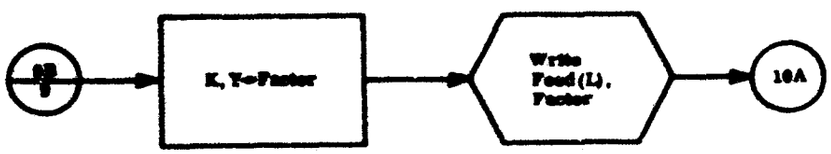


Extract Job
Sheet 7 of 11


8A
1→L
8B

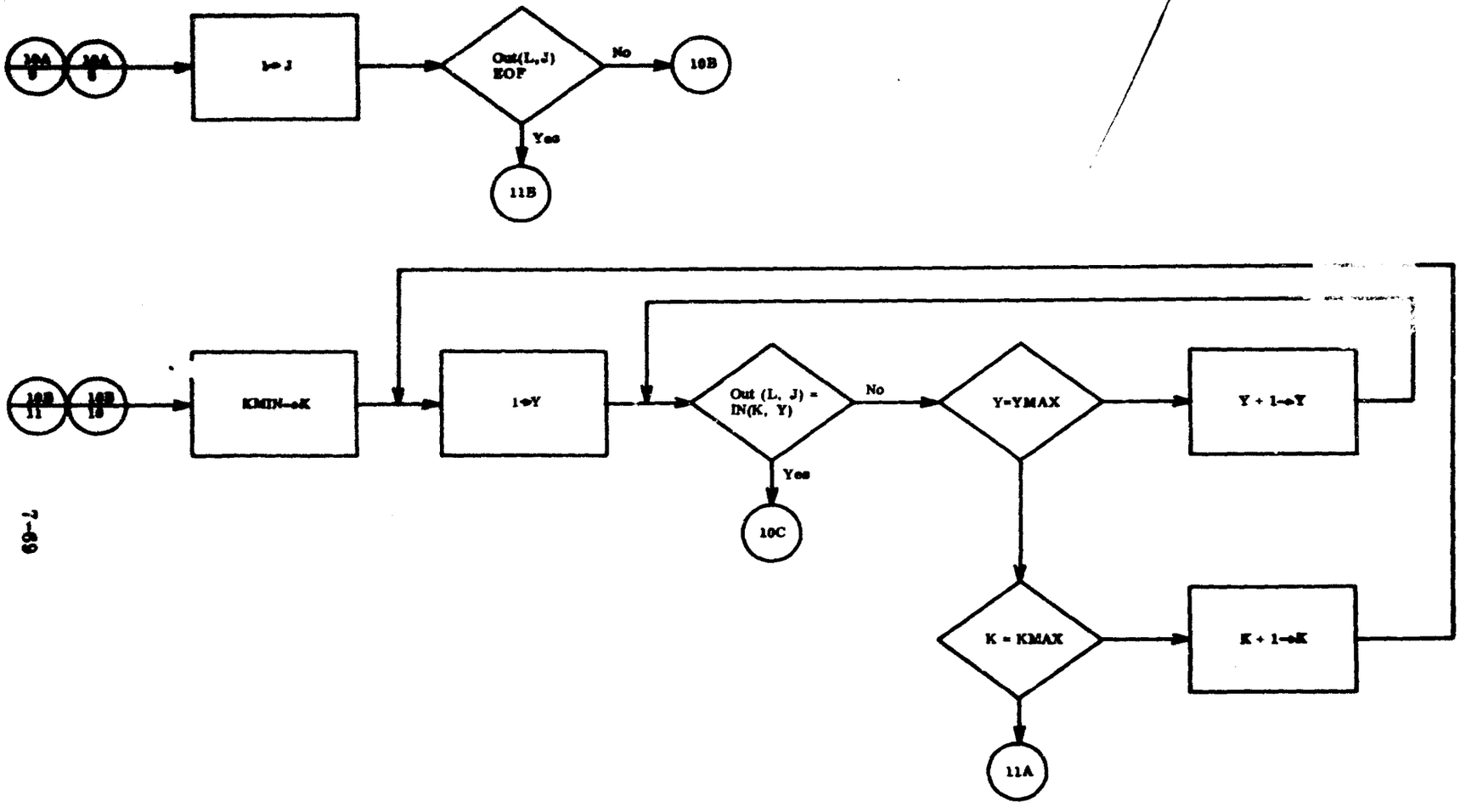

8B
11
8B
8
No
Write
Null Feed
11B
Yes
Factor ICC(L)
Null
No
9A
Yes
Write
Feed (L).
Null Factor
10A

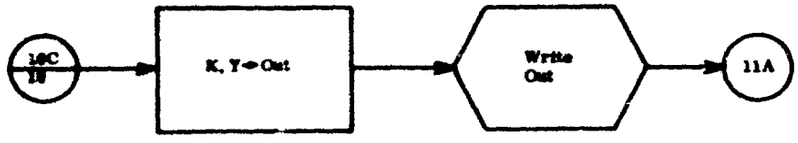




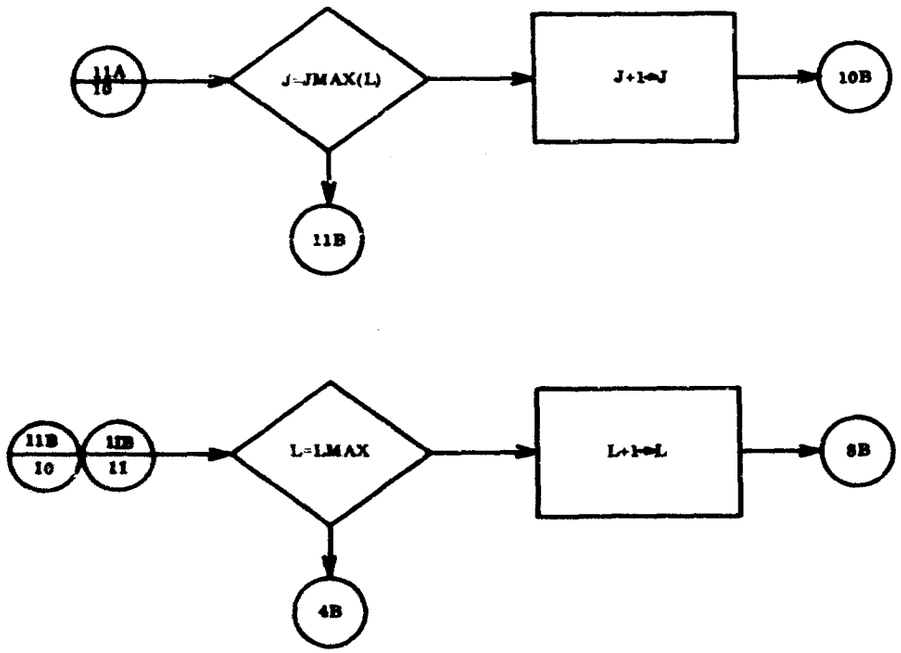
7-68

Extract Job
Sheet 9 of 11





Extract Job
Sheet 10 of 11



Extract Job
Sheet 11 of 11

7.1.7.2.7.3 Restructure

Job Request: RESTRUCTURE (restructure statement), (restructure node list), (restructure item).

7.1.7.2.7.3.1 Functional Description. A node specification represents the necessary linear steps in creating the desired output item. From this blueprint, the Restructure job fixes the item in the data pool. Data is transferred to the item in the manner specified by the node specification under control of the Restructure statement.

At each major node, the R-values of the Restructure statement determine the specific input fields. For every R-value, some L^{th} level data item is written. This L^{th} level is controlled by a specified K^{th} level of the input item.

7.1.7.2.7.3.2 Inputs

(1) RESTR. STATEMENT, S, 2

ICC (UNIVERSE), H, V
R-VALUE, LIST, F

R-VALUE, H, V

(2) RESTR. NODE LIST, F

LEVEL K, F

IN1 LIST, F

IN1, H, V

IN2 LIST, F

IN2, H, V

LEVEL L, F

FEED, I, 6
FACTOR, H, V
OUT LIST, F

OUT, H, V

7.1.7.2.7.3.3 Results

(1) RESTR. ITEM (DYNAMIC)

7.1.7.2.7.3.4 Directories Used

(1) Item List.

(2) Term List.

7.1.7.2.7.3.5 Services Used

(1) Open for Reading.

(2) Close for Reading.

(3) Seek.

(4) Read.

(5) Fix Item.

(6) Open for Writing.

(7) Close for Writing.

(8) Write.

(9) Retrieve IL Entry.

(10) Retrieve Term List Entry.

7.1.7.2.7.3.6 Jobs Used. No Job Extensions are used.

7.1.7.2.7.3.7 Method of Operation. The node specification is read in its entirety and the IN (U, K, I), FEED (U, L), FACTOR (U, L), and OUT (U, L, J) matrices are formed. With this blueprint an Item List and Term List are constructed and fixed in the data pool.

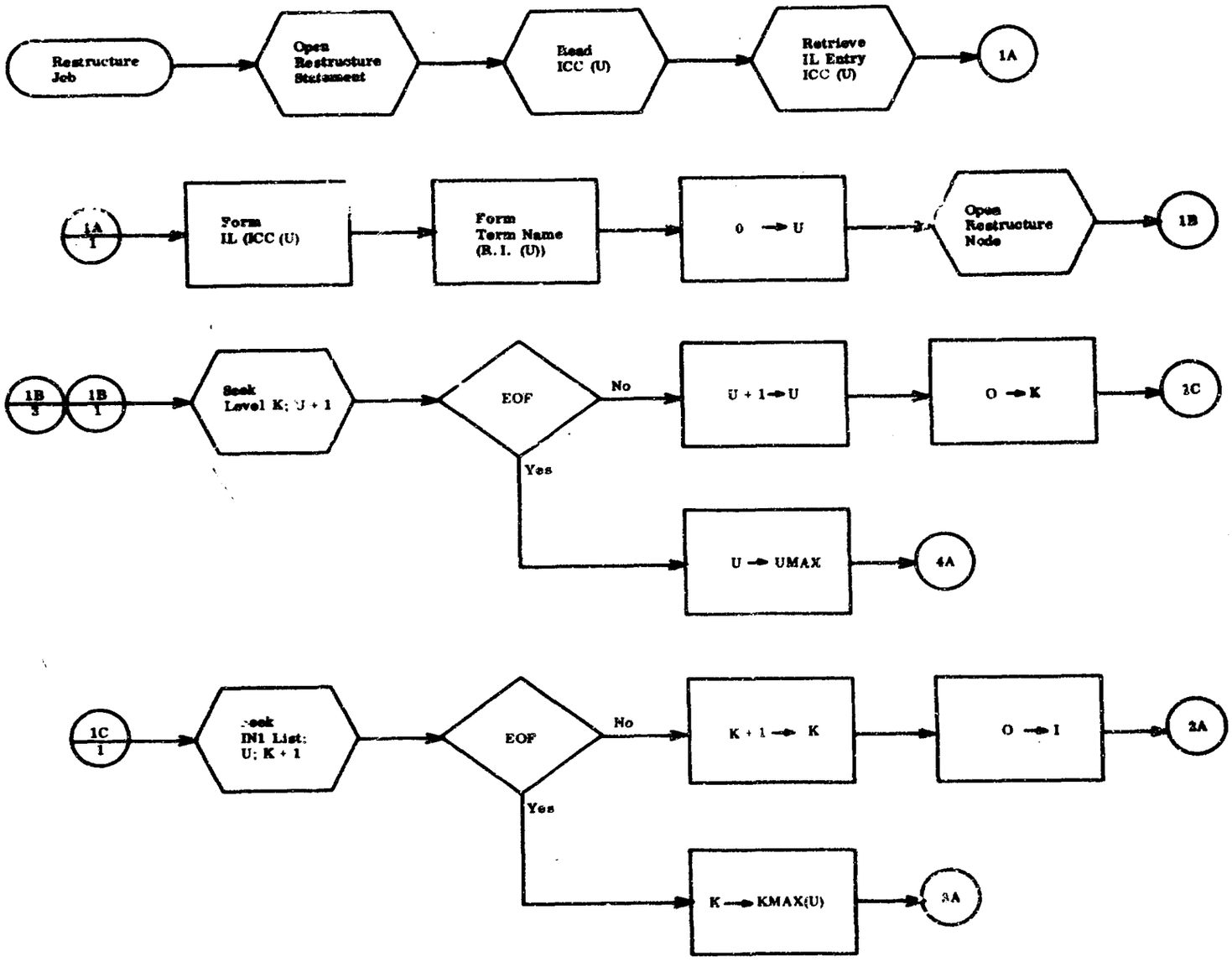
In addition to the above matrices the Restructure program provides storage allocation for the FILE (K) and FIELD (K, I) matrices. FILE (K) indicates EOF at every K^{th} level input in which the input records are exhausted. FIELD (K, I) contains an input field value for every $(K, I)^{th}$ input item, modified by a specific R-value.

At each major node, the R-values are read sequentially until exhausted. For every R-value the following actions are taken:

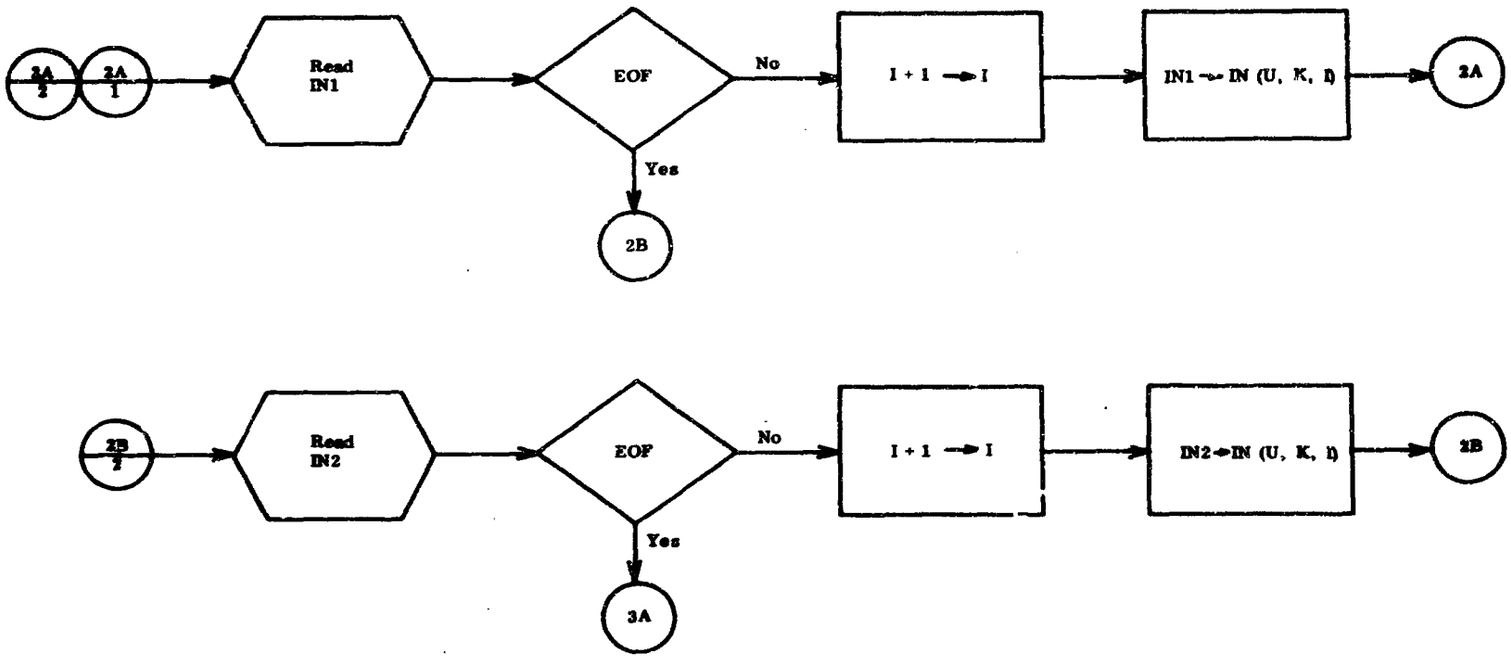
(1) Read all input items IN(U, K, I) into corresponding entries of FIELD (K, I) until all K levels are exhausted.

(2) If FILE (U, K) indicates EOF and FEED (U, L) specifies the K^{th} level, write file (U, L) EOF, decrement L, and, after the last such L^{th} level, decrement K. If FACTOR (U, L) indicates an (X, Y) value where FIELD (X, Y) has exhibited a change in value, decrement L but not K.

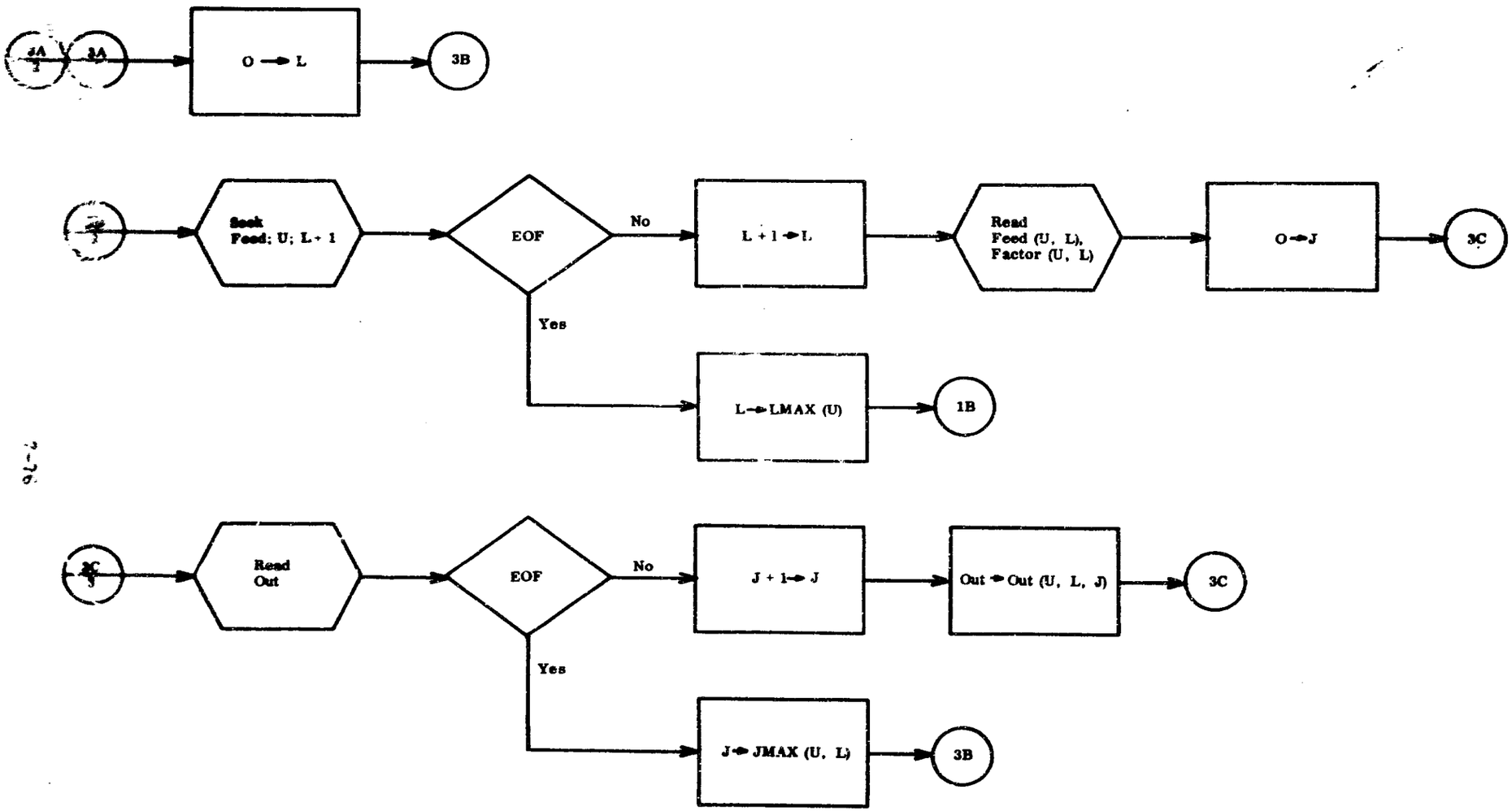
(3) Determine all (X, Y) values of OUT (U, L, J) and write their corresponding output data items FIELD (X, Y) until all L levels are exhausted.



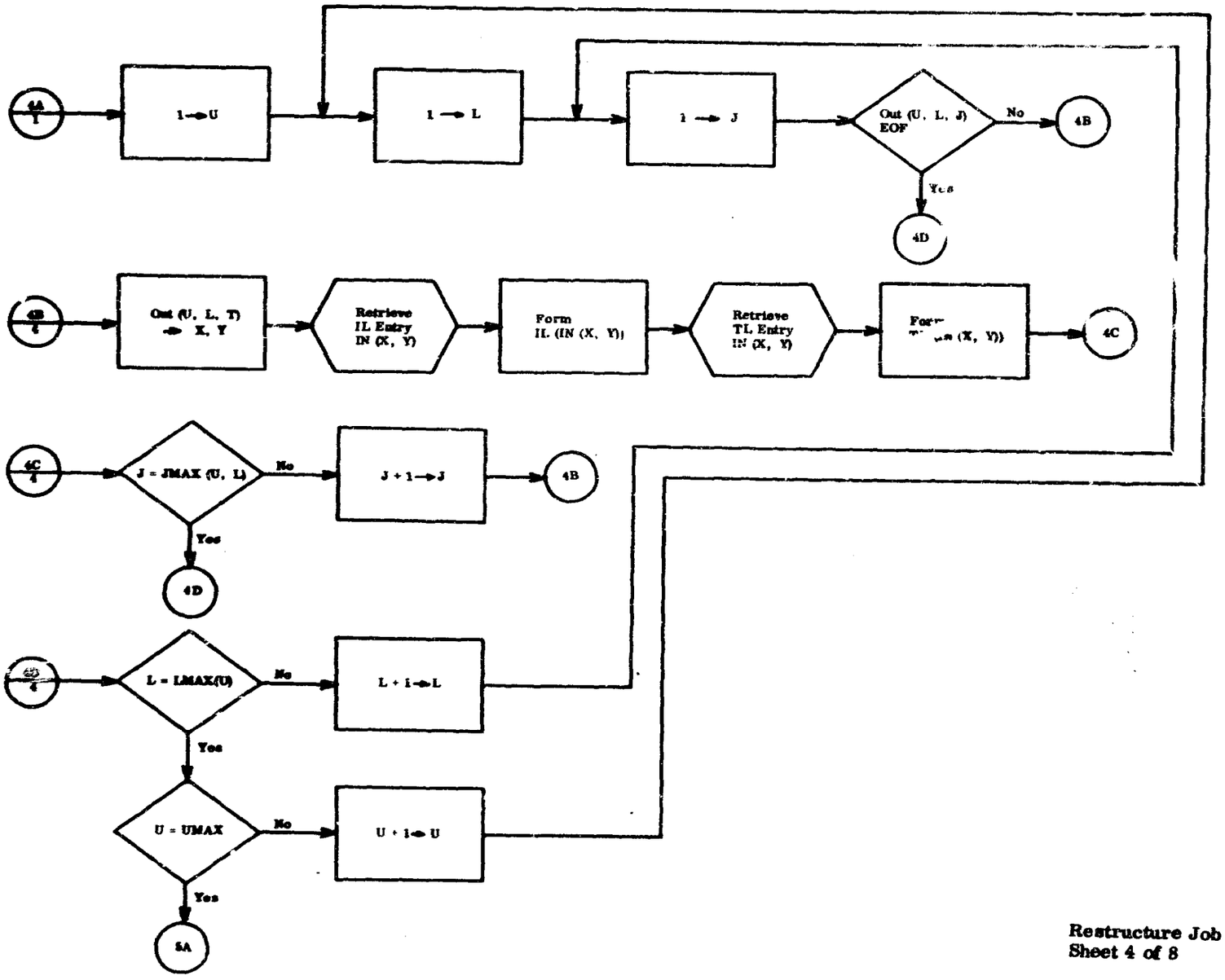
Restructure Job
Sheet 1 of 8



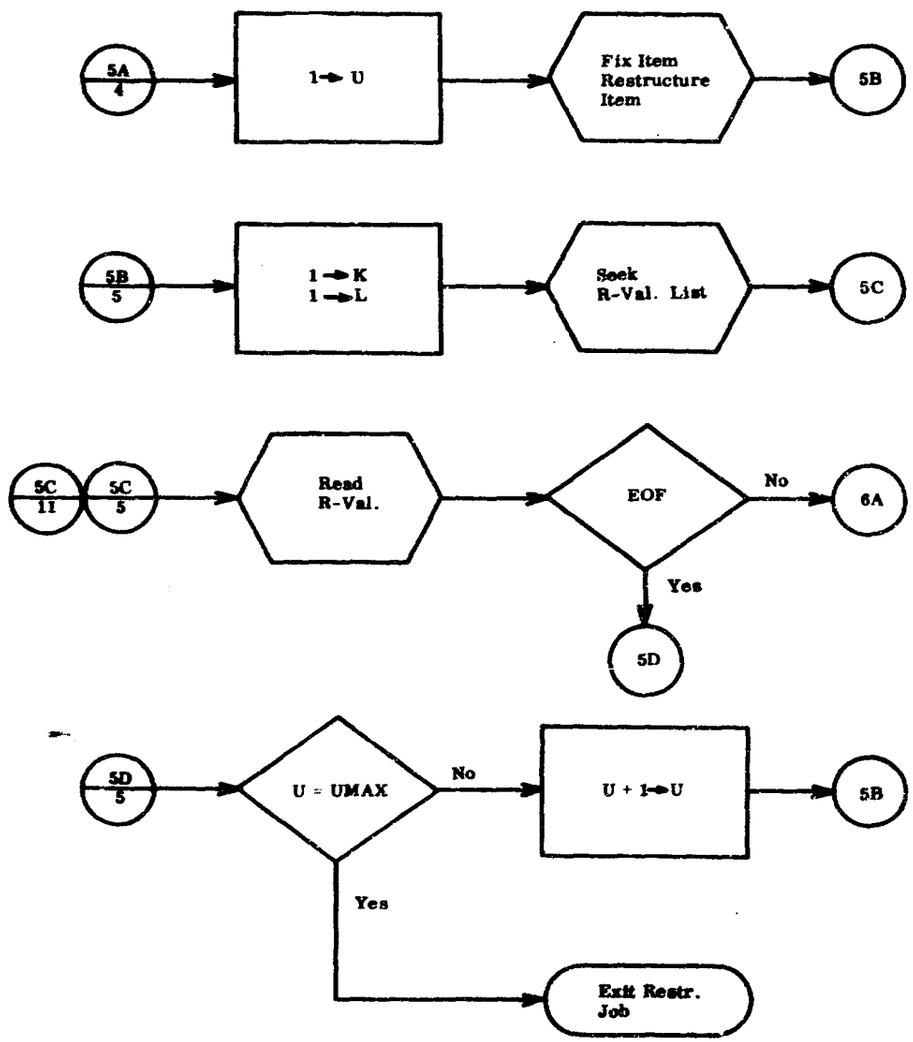
Restructure Job
Sheet 2 of 8



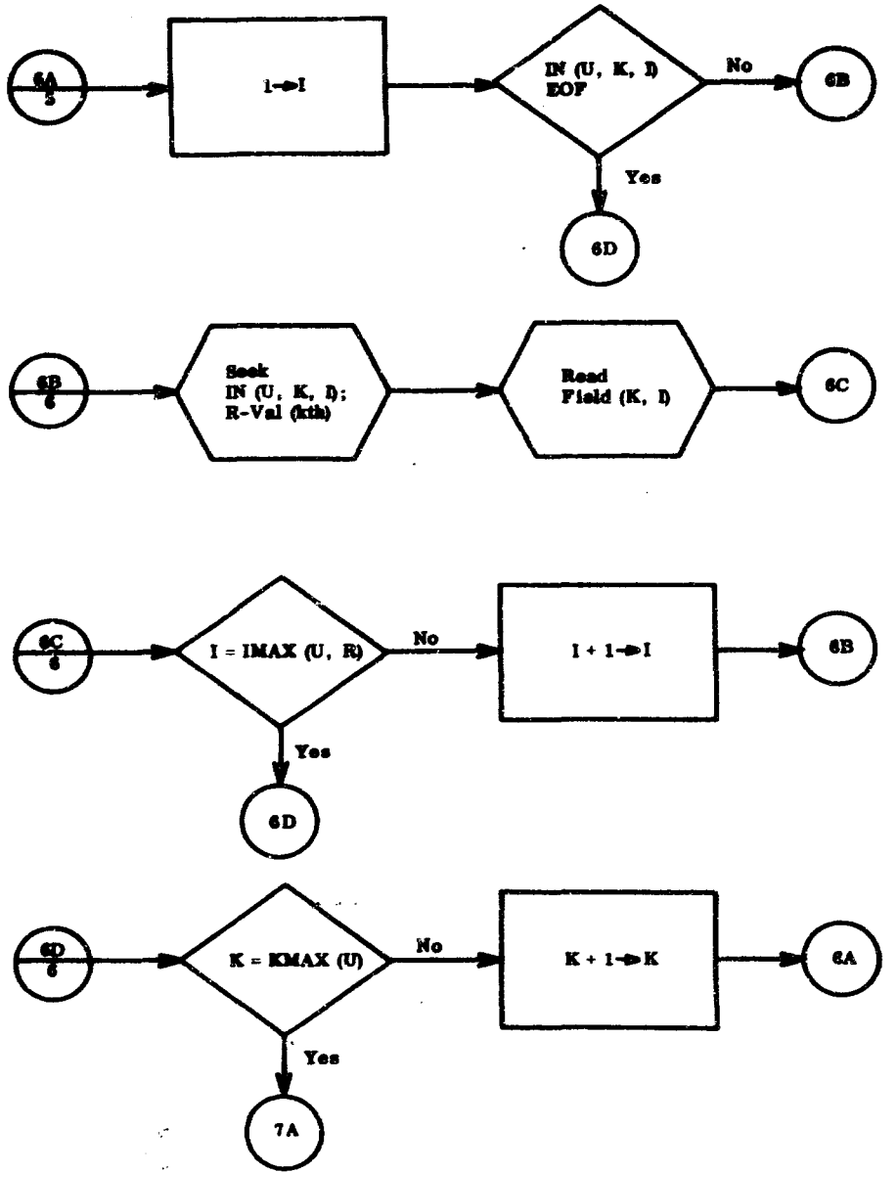
Restructure Job
Sheet 3 of 8

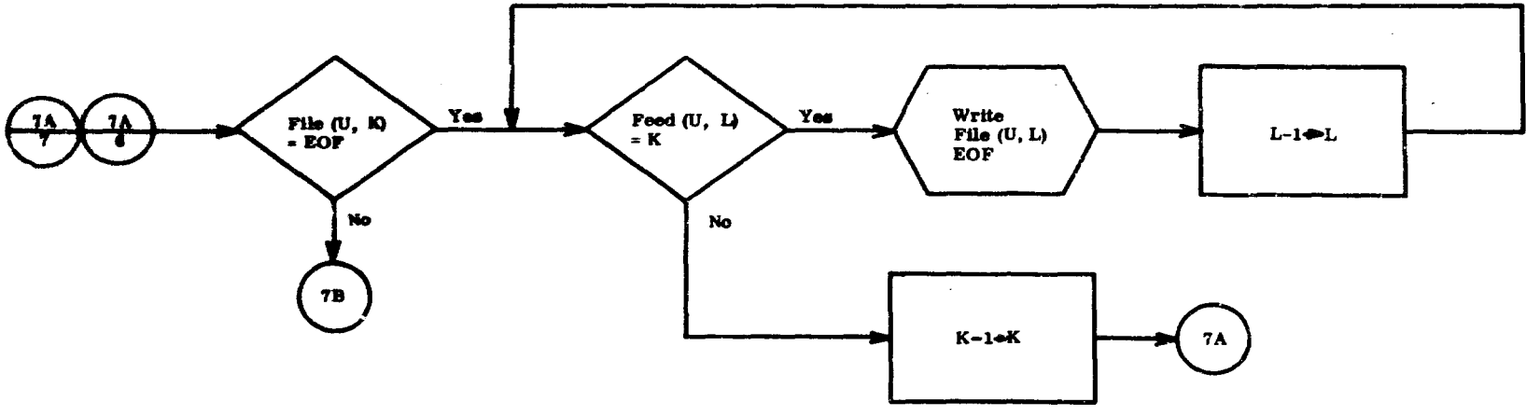


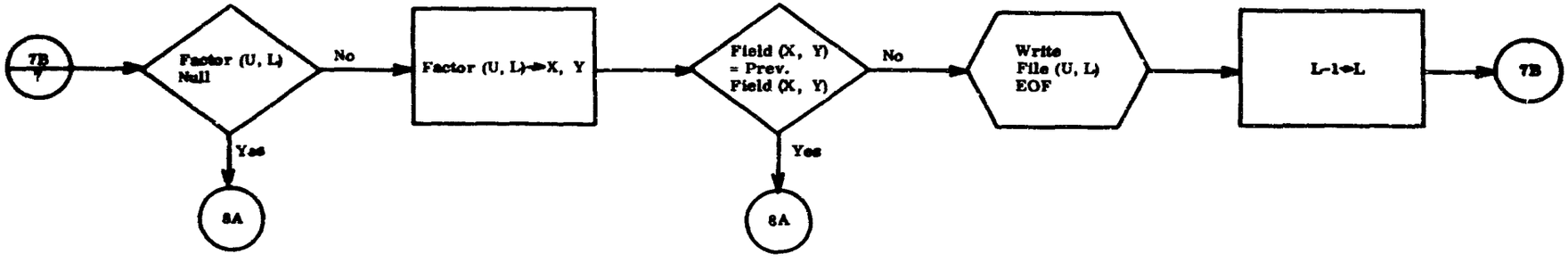
Restructure Job
Sheet 4 of 8



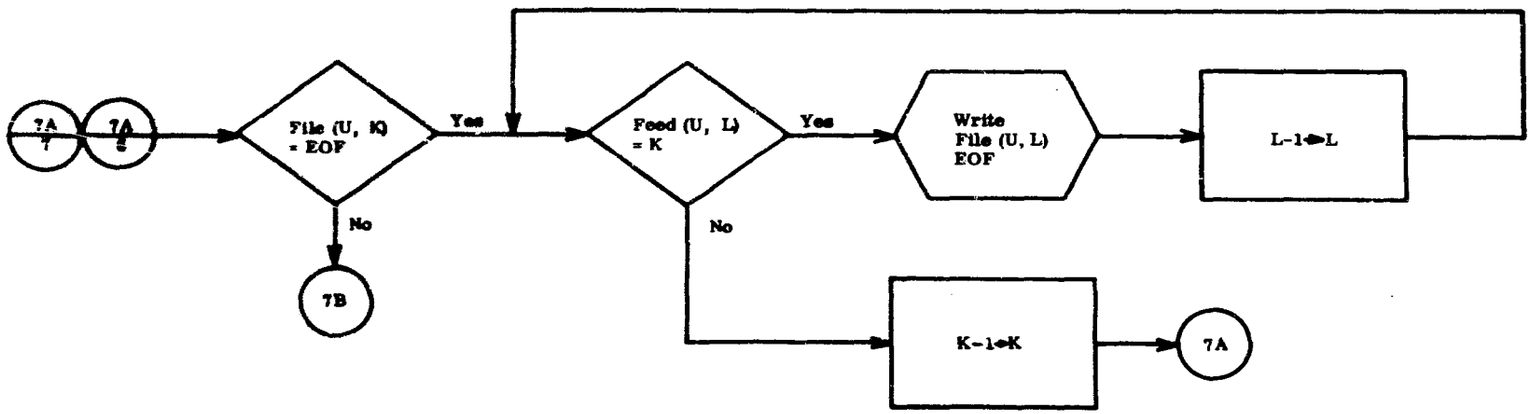
Restructure Job
Sheet 5 of 8

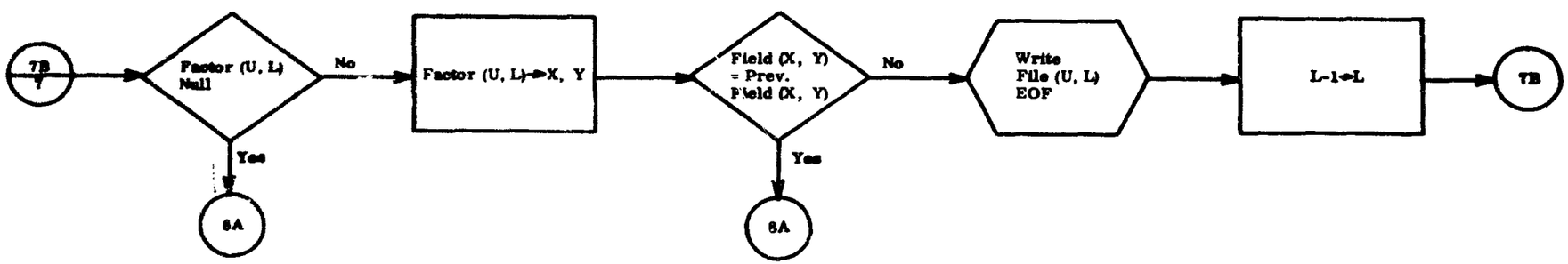

6A
1→I
IN (U, K, I)
EOF
No
6B
Yes
6D
6B
Seek
IN (U, K, I);
R-Val (kth)
Read
Field (K, I)
6C
6C
I = IMAX (U, K)
No
I + 1→I
6B
Yes
6D
6D
K = KMAX (U)
No
K + 1→K
6A
Yes
7A



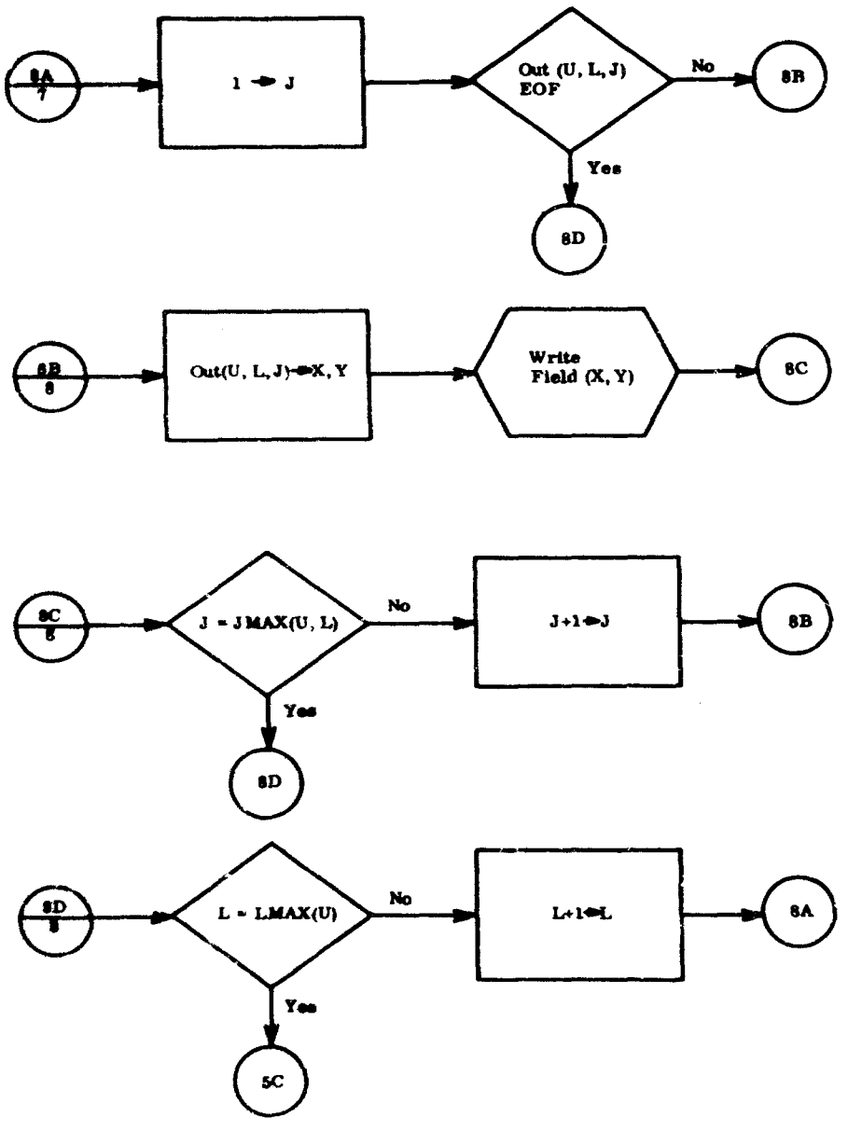


Restructure Jcb
Sheet 7 of 8





Restructure Job
Sheet 7 of 8



Restructure Job
Sheet 8 of 8

7.1.7.3 Display

Job Request: DISPLAY (display format), (display item).

7.1.7.3.1 Functional Description. Display is a job which enables the user to display items which are bound to this job. An optional English-like Format statement allows wide flexibility in the display characteristics.

There are three phases to the Display job. These are:

(1) Format Translation,

(2) Display Analysis and Heading,

(3) Display Data.

The format translation distinguishes three major statement types. These include:

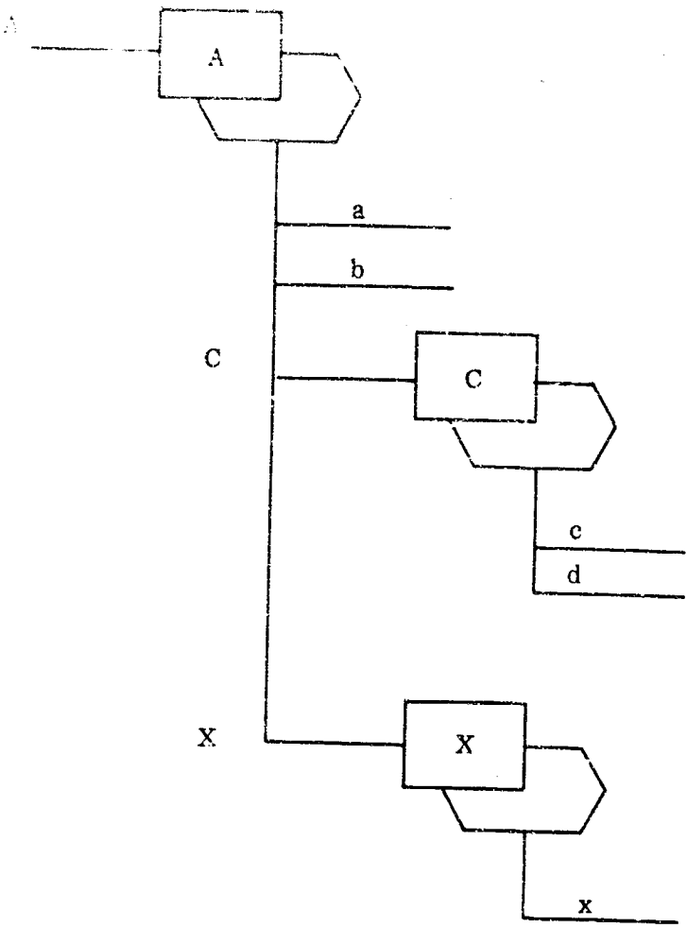
(1) Header,

(2) Data,

(3) Eject.

A Header statement provides the user with the capability of fixing a constant heading on each display. The Data statement consists of the field names of those fields which the user desires to display. An Eject statement indicates the subsumed levels of the data at which the user desires to initiate a new display.

From the field names of the Data statement, the Display program analyzes the corresponding item structure and generates a fixed subheader. This consists of a horizontal listing of the field names beneath corresponding file names. In this manner the subheader displays the structure of the subsequent data. Thus, in the following example, Item A is displayed in its entirety:



A					subheader
a	b	C		X	
		c	d	x	
V_a	V_b	V_c	V_d	V_x	data
		V_c	V_d	V_x	
		V_c	V_d	V_x	
		V_c	V_d	V_x	

In the following display only selected fields of Item A are displayed:

A		subheader
C	X	
d	x	
V_d	V_x	data
V_d	V_x	
V_d	V_x	
V_d	V_x	

Finally, in the third phase, data is displayed. If an Eject statement exists, this is reflected whenever the data requires.

7.1.7.3.2 Inputs

(1) DISPLAY FORMAT, A, V

(2) DISPLAY ITEM (DYNAMIC)

7.1.7.3.3 Results. No outputs for the job are specified. The display itself constitutes the results.

7.1.7.3.4 Directories Used. Term List.

7.1.7.3.5 Services Used

(1) Open for Reading.

(2) Close for Reading.

(3) Seek.

(4) Read.

(5) Retrieve Item.

(6) Term Name to ICC Translation.

(7) Retrieve Term List Entry.

7.1.7.3.6 Jobs Used. No Job Extensions are used.

7.1.7.3.7 Method of Operation. The Format statement is read and translated into Header, Data, and Eject statements through the use of Inscan (refer to Section VIII, The Input Scanner). The Header is fixed in the DISPLAY (H, W) buffer and the field names of the Data statement are translated to ICC's of the ICC (I) matrix. ICC (MIN) is obtained, and the following actions are taken:

(1) If ICC(I) is not subsumed within the present file, form a new node and fix the file name in the DISPLAY (H, W) buffer.

(2) Repeat the above step if ICC (I) represents a more embedded level.

(3) Transfer ICC (I) to IN (L, J) and W to OUT (L, J), where J is the J^{th} field at the L^{th} level and W is the horizontal position of that field on the display.

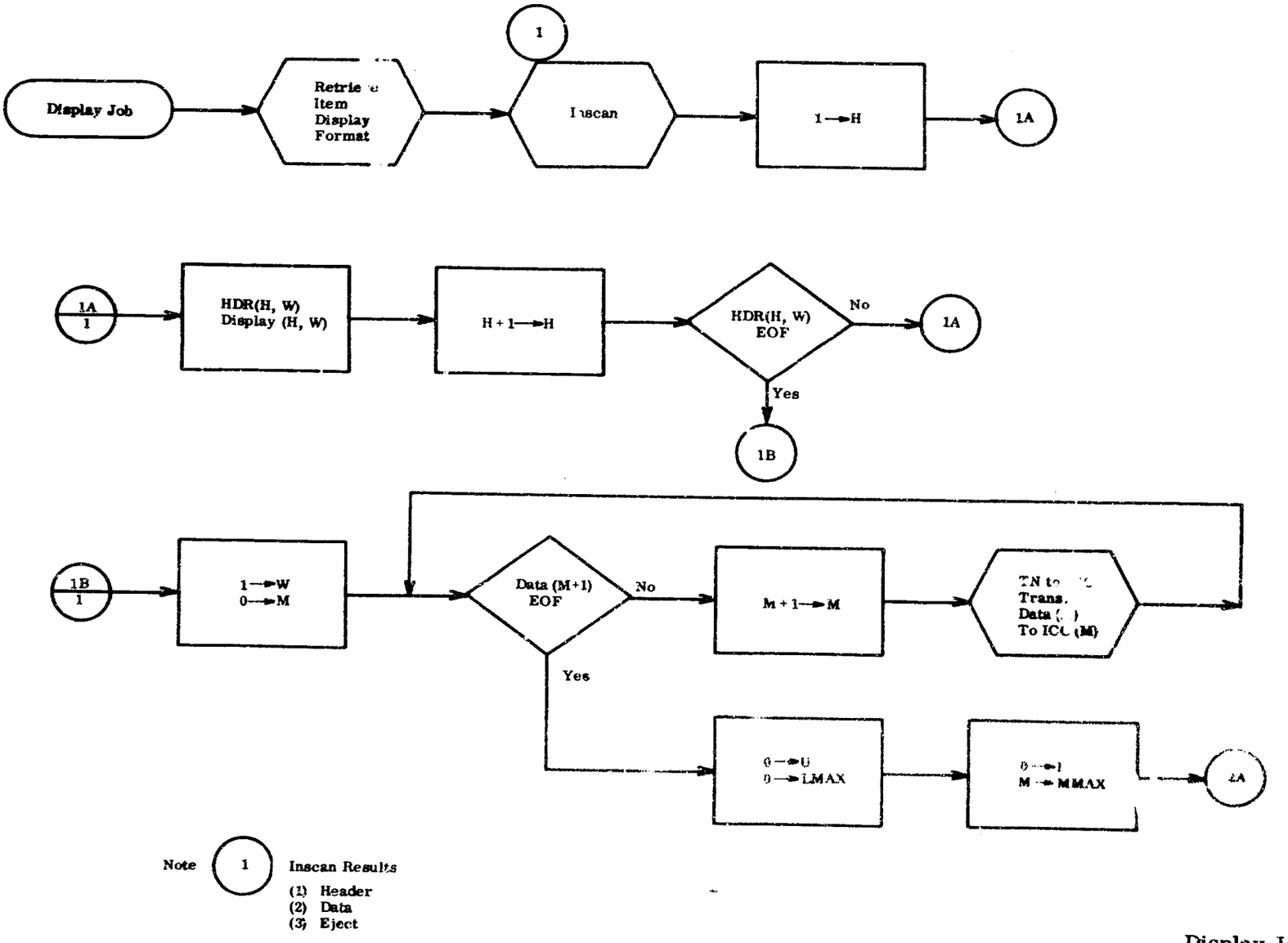
(4) Fix the corresponding field name in the DISPLAY (H, W) buffer.

Upon fixing the subheader in the DISPLAY (H, W) buffer, the data is accessed and the following actions are taken:

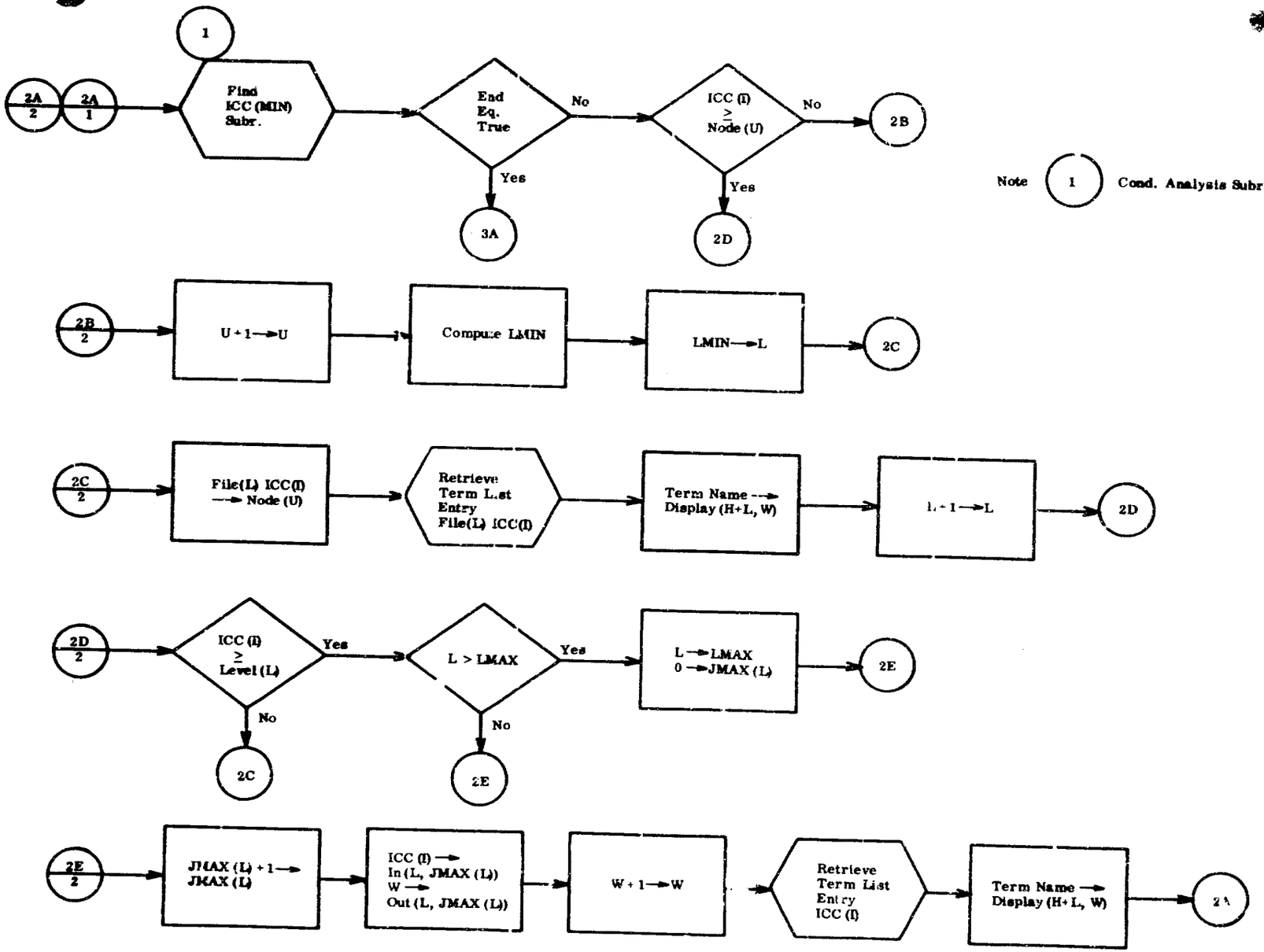
(1) Read the J^{th} field at the L^{th} level into the DISPLAY (H, W) buffer, where W is determined by OUT (L, J) and H is the line number.

(2) Repeat the above step until all levels are exhausted and the line is complete.

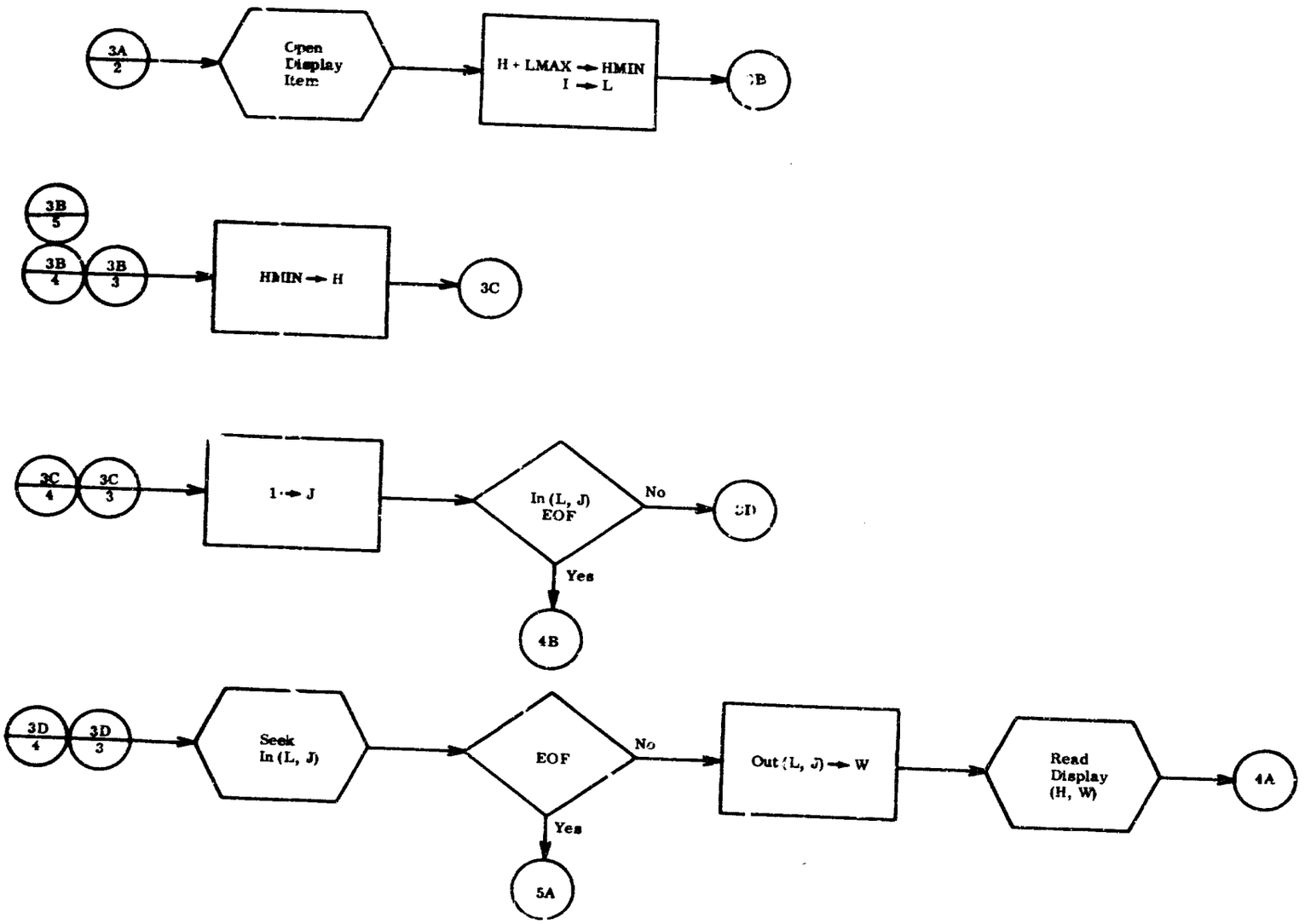
(3) If either the display buffer is exhausted or EOF is reached with EJECT (L) true at that level, display the buffer contents.



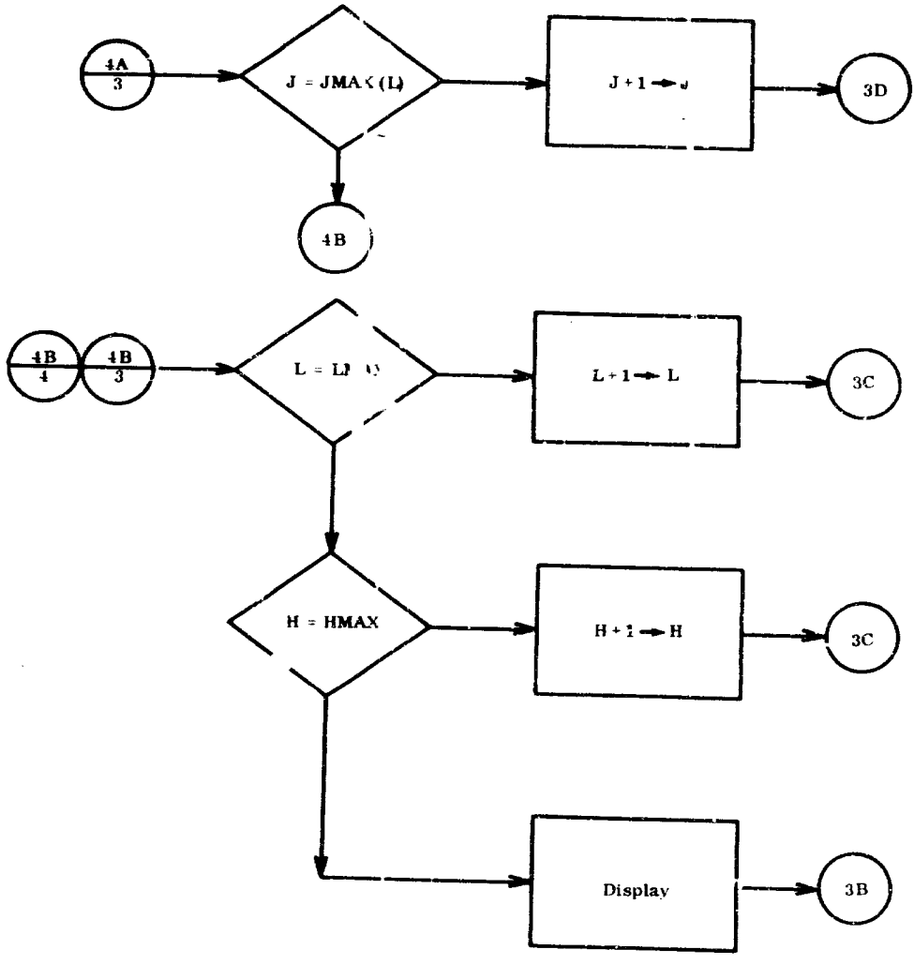
Display Job
Sheet 1 of 5



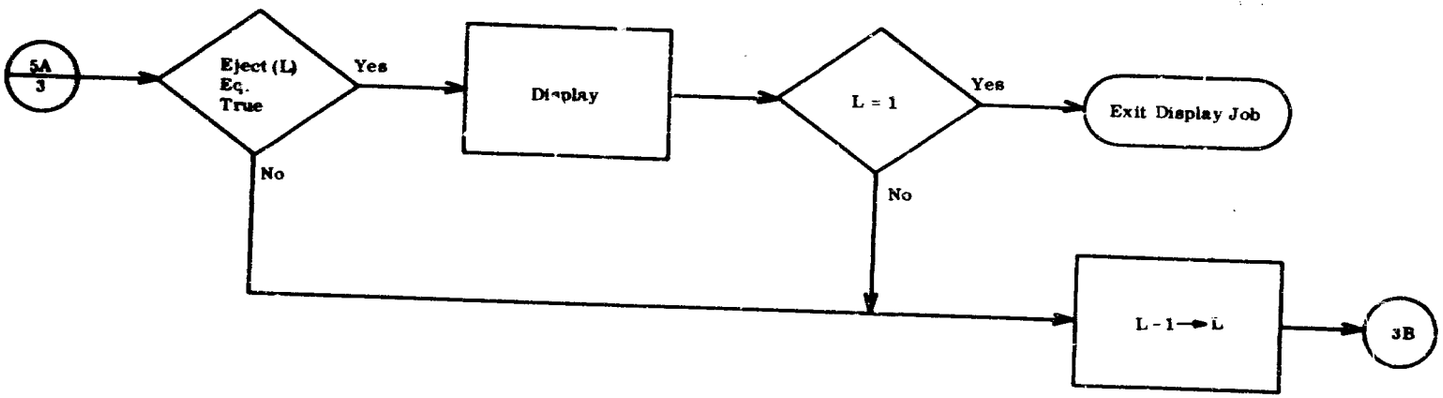
Display Job
Sheet 2 of 5



Display Job
Sheet 3 of 5



Display Job
Sheet 4 of 5



Display Job
Sheet 5 of 5

7.2 CONDITIONAL REFORMAT

Job Request: CONDITIONAL-REFORMAT (condition), (qualifier); (item).

7.2.1 Functional Description

Conditional Reformat is identical to the Query job except for the Display Component. The dynamic item generated by the Reformat Component constitutes the job output. In this manner, items within the data pool may be conditionally reformatted to satisfy any item description.

7.2.2 Inputs

(1) CONDITION, A, V

(2) QUALIFIER, A, V

7.2.3 Results

ITEM (DYNAMIC)

7.2.4 Directories Used

(1) Item List.

(2) Term List.

(3) Fields File.

(4) Shadow of Fields File.

7.2.5 Services Used

(1) Open for Reading.

(2) Close for Reading.

(3) Seek.

(4) Read.

(5) Fix Item.

(6) Open for Writing.

(7) Close for Writing.

(8) Write.

(9) Open for Updating.

(10) Close for Updating.

(11) Replace.

(12) Insert.

(13) Delete.

(14) Term Name to ICC Translation.

(15) Retrieve IL Entry

(16) Retrieve Term List Entry.

7.2.6 Jobs Used

No Job Extensions are used.

7.2.7 Method of Operation

The Conditional Reformat job is entered into the system via the following Job Description:

Job Name: CONDITIONAL-REFORMAT

Job Inputs: condition, qualifier

Job Outputs: item

Job Components:

(1) CONDITIONAL-SEARCH: condition; conditional reformat statement.

(2) REFORMAT: qualifier, conditional reformat statement; item.

The internal Job Description may be represented graphically as shown in Figure 7-7.

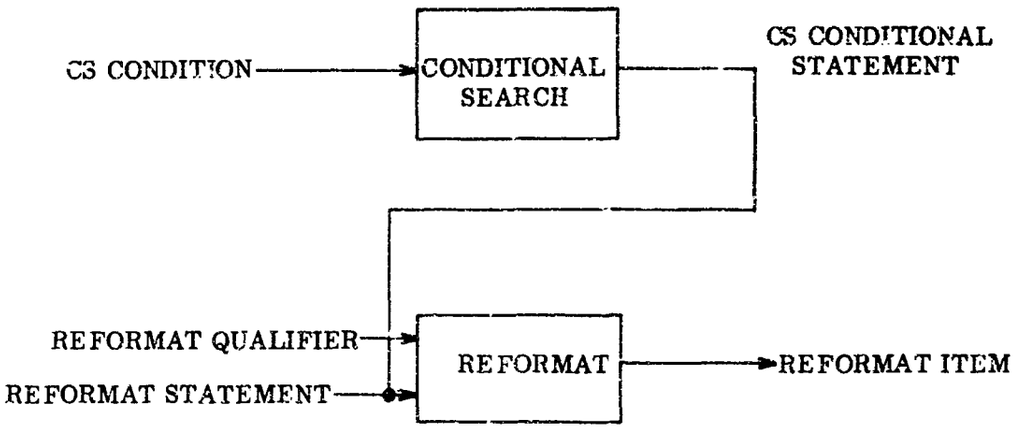


Figure 7-7. Conditional Reformat Job, Internal Job Description

SECTION VIII. THE INPUT SCANNER

The Input Scanner is a powerful, high-level, general-purpose routine. The inputs to Inscan are the input string to be interpreted and the action-graph defining the syntax and actions to be taken. The results from Inscan consist of either a data exit plus an output list containing pointers to the parameters generated at action-points or an error exit.

The use of a generalized Input Scanner can be seen to be a powerful tool in allowing the system to broaden its language handling capability economically. It allows the system to introduce (with no changes in the Input Scanner) a query language or other user-oriented and specialized languages.

The advantages to the programmer in generating an action-graph are that the graph can be written directly from the syntax. This philosophy has been rigorously maintained in the more detailed design which follows.

8.1 ACTION-GRAPHS

Action-graphs are first assumed to be loaded in their named locations. If necessary, provisions for effectively accomplishing this may easily be added at a later

date. Second, the word length of every machine address is assumed to be 18 bits, of which the last 12 bits can specify a machine address. This specific assumption may be altered as a function of the machine word.

There is a one-to-one correspondence between an action-graph symbol as drawn in graph form, this symbol as entered into the machine, and a machine word. The machine entries for any one graph must be sequential in the manner in which they are drawn, except where indicated. Each of these entries is comprised of a 6-bit shape code S(b) and 12-bit content C(b), always a machine address, loaded to location b (Figure 8-1).

LOCATION b	SHAPE CODE S(b) 6	ADDRESS C(b) 12
graph name		
graph name + 1		
graph name + 2		
1		
1		
1		

Figure 8-1. An Action-Graph

The shape codes and their corresponding action-graph definitions are listed in Figure 8-2.

If the action-graphs are compiled in symbolic machine language, each entry can be written symbolically as:

Symbolic Location ‖ Code, Symbolic Address

This statement is entered into the machine as an address constant.

The advantages to this technique are simply that the programmer can describe an action-graph in a symbolic language for which a compiler must already exist.

Implementation problems should be minimized by the fixed, single, machine-word entry.

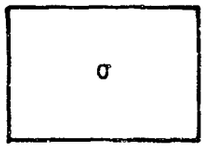
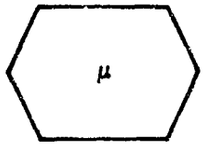
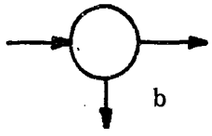
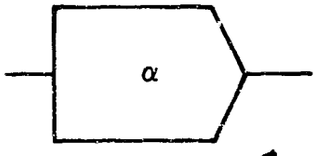
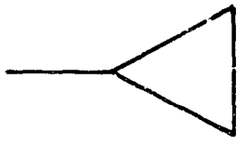
Graph Symbol	Code	Definition
	S1	Symbol Code: σ is the location of some number of basic input symbols.
	S2	Variable Code: μ is the name and location of an action-graph.
	S3	Choice Code: b is the location of a branch entry, should the next sequential entry fail.
	S4	Action Code: α is the name and location of a closed subroutine.
	S5	Terminate Code
	S6	Branch Code: b is the branch entry location.

Figure 8-2. Shape Codes

8.2 ADDRESS LISTS

Design of the Input Scanner requires two address lists of indefinite length. If a level depth ℓ is assigned to a specific action-graph, the level depth $\ell + 1$ is assigned to any graph (including itself) entered from the first. Thus, the same graph may have many levels of execution. The Input Scanner maintains a history of each level depth in the address lists (Figure 8-3).

LOCATION	NEXT ADDRESS	LINK ADDRESS
ℓ	P (ℓ)	L (ℓ)
0		
1		
2		
1		
1		
1		

Figure 8-3. The Address Lists

If the initial level depth is zero, the level depth ℓ defines the relative location of each entry in the address lists as well.

The P(ℓ) entry contains the address of the next sequential location in some action-graph at level depth ℓ whenever the present level depth is greater than ℓ. The L (ℓ) entry contains the address of a branch (link) location at level depth ℓ.

8.3 INSCAN

8.3.1 Functional Description

Inscan (refer to Figure 8-4) scans an input string, such as a job request, job data, or external format data, and, in concert with an action-graph, it checks for the syntactic acceptability of the input string and controls the subroutines which take the actions specified by the action graphs. For example, in the case of a job request, these actions are normally the preparation of parameter-list structures for the job specified.

8.3.2 Inputs

The inputs to Inscan are the input string to be interpreted and the name of the action-graph.

8.3.3 Results

The results from Inscan consist of either a data exit plus an output list containing pointers to the parameters generated from the action codes or an error exit.

8.3.4 Indicator Cells Required

(1) b ■ the location of the next action-graph entry.

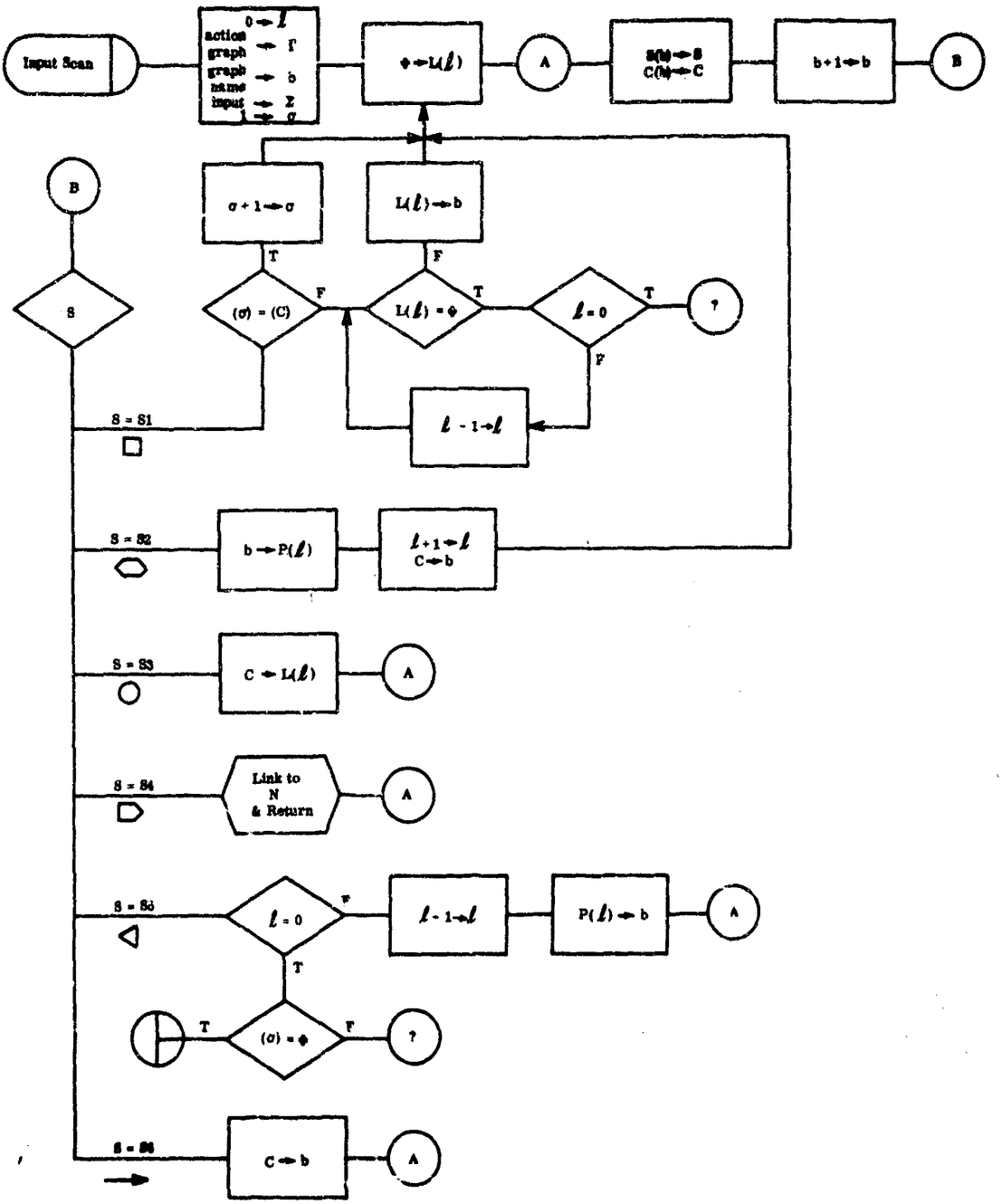


Figure 8-4. Inscan Flow Chart

(2) $\ell \equiv$ the relative location (level depth) of the address list entries.

(3) $\sigma \equiv$ the relative location of the current input symbol.

8.3.5 Tables Read

Action-Graphs.

8.3.6 Tables Modified

None.

8.3.7 Method of Operation

(1) The input string Σ is scanned with an action-graph Γ. The action-graph name is loaded into cell b, indicating the location of the next graph entry; the level depth indicator ℓ is made zero; the input symbol pointer σ is made 1, pointing effectively to the first input symbol.

(2) The ℓ^{th} level depth branch (link) address $L(\ell)$ is made empty, indicating that no choice is available should an entry at this level fail.

(3) The shape code S(b) is brought into S and the address C(b) is brought into C. Cell b is incremented to the location of the next sequential entry in the action-graph and the following action is taken:

(a) If S = S1 (Symbol Code), test if the input symbol at address σ equals the action-graph symbol at address C. If true, increment σ to the next input symbol and return to (2); if false, load the branch address $L(\ell)$ into cell b, indicating the next graph entry. If no choice is available at this level ($L(\ell)$ is empty), back up to the previous level.

(b) If S = S2 (Variable Code), remember the next graph entry of the current level depth in $P(\ell)$, increment the level depth ℓ, make address C the next graph entry, and return to (2).

(c) If S = S3 (Choice Code), remember the branch address C in $L(\ell)$ and return to (3).

(d) If S = S4 (Action Code), execute the closed subroutine at address C and return to (3).

(e) If S = S5 (Terminate Code), test if level depth is zero. If true, exit; if false, decrement the level depth ℓ, recall the location of the next entry in this graph from P(ℓ), and return to (3).

(f) If S = S6 (Branch Code), branch to address C and return to (3).

8.4 INSCAN EXAMPLES

Figure 8-5 is the action-graph for a language called JM, whose acceptable input strings consist of a string of JOHN's followed by the same number of MARSHA's. The graph is first shown in graph form. The graph is then shown in symbolic code as it would appear to the language compiler. A sample input string is now considered to be:

JOHN JOHN MARSHA MARSHA

The final diagram gives a photographic view of the working cells as Inscan scans the input string.

The S-entry describes the machine state as Inscan switches on the shape code: The S-action describes the corresponding machine action.

Each machine action is arbitrarily numbered (#) and corresponding comments are given below:

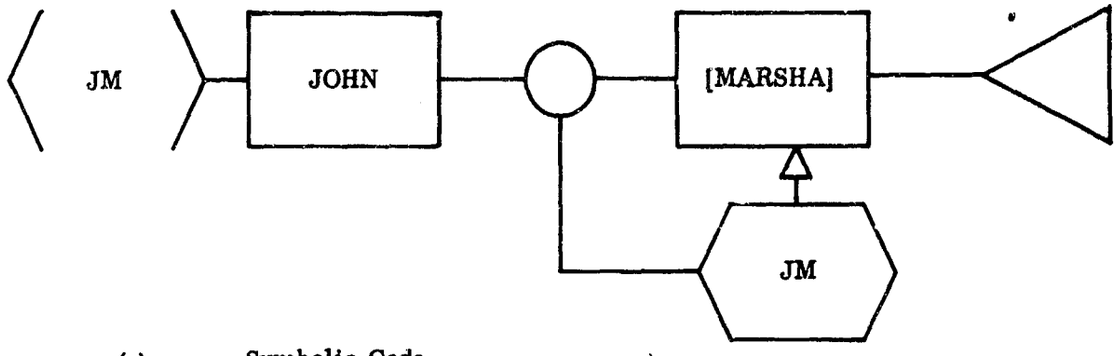
(1) The machine is initialized.

(2) The shape code is the symbol code. The symbol JOHN is equal to the graph symbol. Point to the next input symbol. Remove any branch address entry at the current level (No Back-Up Condition). Advance.

(3) The shape code is the Choice Code. Remember the branch address JM + 4 in L(0), the zero level depth entry of L(ℓ). Advance.

(4) The shape code is the Symbol Code. The symbol JOHN is not equal to the graph symbol MARSHA. Recall the branch address from L(0), and remove this entry. Advance.

(5) The shape code is the Variable Code. Remember the next sequential ent. of this graph in P(0). Increase the level depth, and branch to JM. Advance.



(a) Symbolic Code

JM	S1, [John]
	S3, JM2
JM1	S1, [Marsha]
	S5
JM2	S2, JM
	S6, JM1

(b) Input String

(1) JOHN (2) JOHN (3) MARSHA (4) MARSHA (5) Φ

(c) Inscan Operation

S - Entry			S - Action					
b	S	C	#	ℓ	P(ℓ)	L(ℓ)	b	σ
			1	0		∅	JM	(1)
JM + 1	S1	[John]	2	0		∅		(2)
JM + 2	S3	JM + 4	3			JM + 4		
JM + 3	S1	[Marsha]	4			∅	JM + 4	
JM + 5	S2	JM	5	1	JM + 5		JM	
JM + 1	S1	[John]	6			∅		(3)
JM + 2	S3	JM + 4	7			JM + 4		
JM + 3	S1	[Marsha]	8			∅		(4)
JM + 4	S5		9	0			JM + 5	
JM + 6	S6	JM + 2	10				JM + 2	
JM + 3	S1	[Marsha]	11			∅		(5)

Figure 8-5. Inscan Example

(6) The shape code is the Symbol Code. The symbol JOHN is equal to the graph symbol. Point to the next input symbol. Remove any branch address entry at the current level. Advance.

(7) The shape code is the Choice Code. Remember the branch address JM + 4 in L(1). Advance.

(8) The shape code is the Symbol Code. The symbol MARSHA is equal to the graph symbol. Point to the next input symbol. Remove the branch address entry at the current level depth 1. Advance.

(9) The shape code is the Terminate Code. The level depth is not zero. Decrease the level depth. Recall the next sequential entry at this level from P(0). Advance.

(10) The shape code is the Branch Code. Branch to JM + 2. Advance.

(11) The shape code is the Symbol Code. The symbol MARSHA is equal to the graph symbol. Point to the next input symbol. Remove any branch address entry at the current level. Advance.

(12) The shape code is the Terminate Code. The level depth is zero. EXIT.

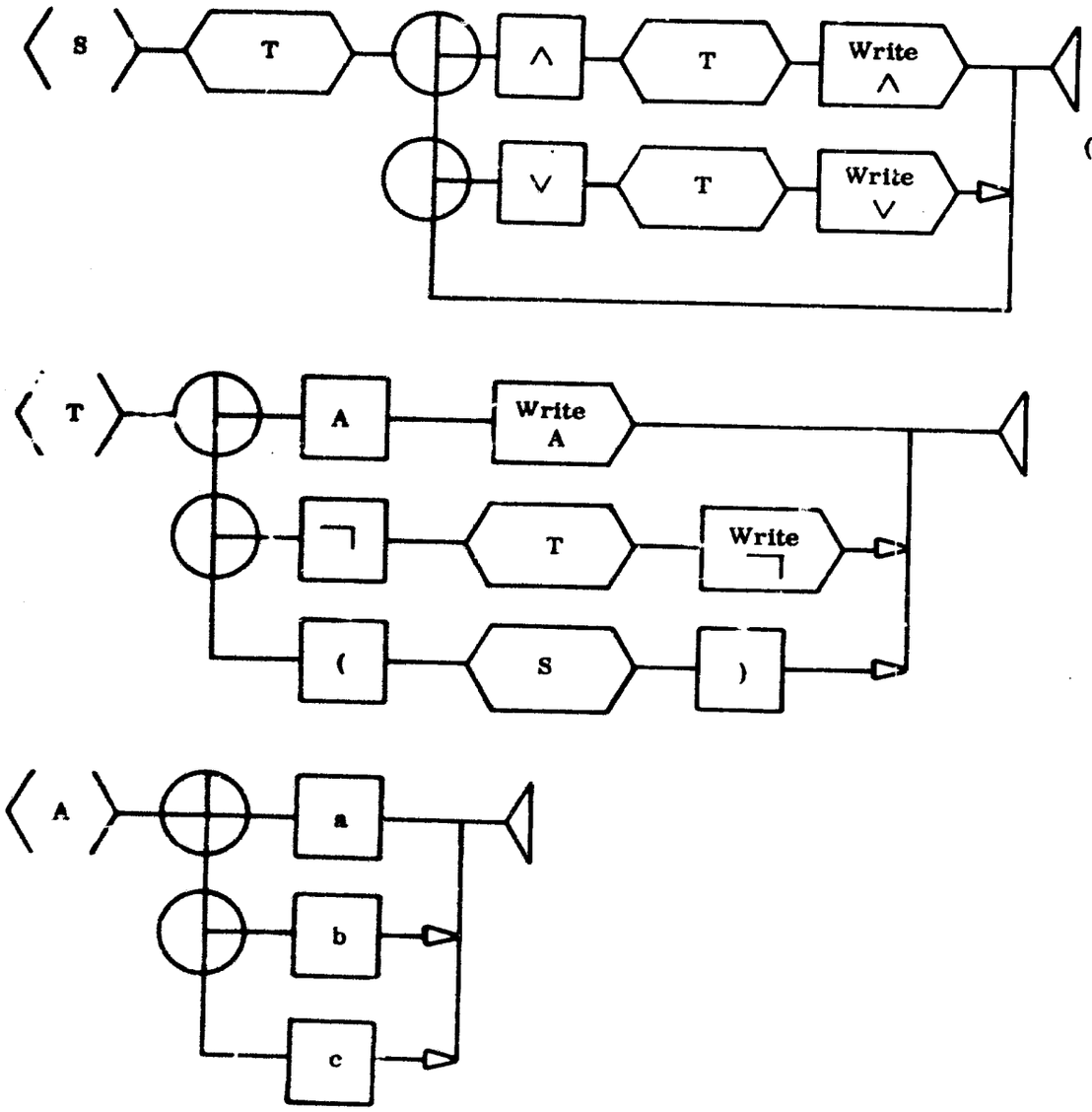
Figure 8-6 is the action-graph of a language called S (Sentence). It is the graph of the Infix to Suffix Operator Translator intended for use in the system. The necessary variables T (Term) and A (Alpha) are also shown.

The entire graph is then shown in symbolic code. A sample input string is now considered to be:

$$\neg a \vee \rightarrow (b \wedge c).$$

The resultant parameter list when this input string was desk-checked appeared as:

$$a \neg b\ c \wedge \rightarrow \vee.$$



(a) Symbolic Code

```
S           S2, T
            S3, SENT 1
            S1 [∧]
            S2, T
            S4, [WRITE ∧]
SENT X      S5
SENT 1      S3, SENT X
            S1, [∨]
            S2, T
            S4, [WRITE ∨]
            S6, SENT X

T           S3, TERM 1
            S2, A
            S4, [WRITE A]
TERM X      S5,
TERM 1      S3, TERM 2
            S1, [¬]
            S2, T
            S4, [WRITE ¬]
            S6, TERM X
TERM 2      S1, [(]
            S2, S
            S1, [)]
            S6, TERM X

A           S3, ALPHA 1
            S1, [a]
ALPHA X     S5
ALPHA 1     S3, ALPHA 2
            S1, [b]
            S6, ALPHA X
ALPHA 2     S1, [c]
            S6, ALPHA X
```

Figure 8-6. Infix to Suffix Operator Translator

SECTION IX. THE DIALOGUE PROCEDURE

9.1 INTRODUCTION

This appendix presents the detailed design of the Dialogue Query, a system job in DM-1, describes the dialogue procedure, and specifies the programs of the Dialogue Query job by means of flowcharts and descriptions. The dialogue procedure provides a multistage interchange of information between the DM-1 data pool and an inquirer at a Query Response Communications Console (QRCC). Through console displays, the dialogue programs guide the user to select the attributes which describe the object, event, or process in which he is interested, and to specify the properties which identify the individual data values relevant to his information need. Once defined, the need may be satisfied by the DM-1 query processors, which retrieve, restructure, and display the pertinent data.

9.1.1 Need for a Dialogue

The DM-1 data pool is a large, complex structure that contains data on many facets of an overall application. A user with a need for information from the data pool cannot be expected to be familiar with the entire structure of the data pool and the detailed interrelationships among the various items of data. A dialogue with the DM-1 system informs him of the information which is available and shows him the relationship of the data on his subject to the other data maintained by the system.

The inquirer who approaches the data pool is like a researcher performing a search in a library. The researcher is rarely able to specify in advance the precise information he seeks. Very often, he is unaware of the precise nature of the available information. He has a vaguely formed idea of his information requirement. Perhaps he begins by searching a subject index to determine the names of books which deal with his subject. Each book he reviews leads him to other sources and narrows his search. He applies his judgment in each situation to define his next step. His information requirement is never really specified until the search is completed.

Through a dialogue with the DM-1 system, an inquirer is provided with information he can use on the way to defining his need. He is presented with a generic set of names for the categories of information in the data pool. He selects a category for deeper probing, and he is presented with the next level of detail. He is constantly in a position to apply his judgment in relating his need to the available information. His information requirement is not defined until he finishes the dialogue. By then, the system has developed, internally, a formal statement identifying the subject of his query and the logical condition which must be met for retrieval to take place.

The dialogue procedure relieves the user of concern with the rules of conditional logic. It may be used by an inquirer who knows nothing about the contents of the data pool. Thus, it meets the need for making the data pool available to a wide audience with little or no training in the languages or mechanisms of the system.

9.1.2 The Dialogue Principle

The dialogue procedure involves the interaction of the inquirer and the system in focusing-in on the data items pertinent to the inquirer's need by progressive enlargements of the scale of detail. A small-scale representation of a large area of the

Best Available Copy

data pool is displayed by the system. The inquirer selects a smaller area, and that area is presented in a larger scale for further probing. When the inquirer encounters an item whose value in particular cases would satisfy his information need, the system adds the item to a developing list of desired attributes. Whenever the inquirer is finished with the more detailed display in any area, the larger area which contains it is presented again in a smaller scale. This allows the user to investigate another area in more detail.

When the inquirer finishes defining the relevant attributes of the object of his interest, he is offered the opportunity to specify limiting properties which define the particular cases for which he wants the selected information. For example, he might have selected the name and population as the things he wants to know about cities; he might now specify that he is only interested in cities in New York, New Jersey, or Pennsylvania which have a population over 100,000. The homing-in process occurs again and the inquirer selects items which define the particular cases. When an item is selected, the system displays ranges of values, thus permitting the user to select the values which define the desired property. If values for the selected item are not available (the field is not indexed), the system displays an example and asks the user to key-in the appropriate values.

After a property is specified, the user indicates whether he has more limiting properties to specify. If he does, the process is repeated. The user is also given the opportunity to specify alternative properties. When he is finished, a full statement of the information specification is presented. He may modify it or use it to obtain the results.

Throughout the dialogue, the user's participation is limited to making a choice among the alternatives presented to him wherever possible. The alternatives are presented in digestible groups (usually eight) for his review. This keeps the process simple, yet provides for focusing rapidly (logarithmically) on the area of interest.

9.2 THE DIALOGUE IN DM-1

The Dialogue Query is a job in the DM-1 system. It is initiated by a DM-1 job-run request; it uses the system service routines and the information in the system directories and interfaces with other system components. The dialogue procedure uses the structural organization of the DM-1 data pool to thread through its series of displays. It relates to the DM-1 Query job, because the object of the dialogue is to develop a formal query which can be processed by the Query job. The dialogue procedure is also useful in relation to other parts of the DM-1 system. These relationships to the DM-1 system are discussed in the following paragraphs.

9.2.1 Use of the Data-Pool Structure

The key element in the dialogue procedure is the organization and structure of the DM-1 data pool. The data pool is basically a tree structure. It is a single item with a series of subitems emanating from it. Any of these subitems may, in turn, contain more subitems, and so on, until terminal items (fields) are reached.

This aspect of the DM-1 data pool is used to partition the data into applicable and nonapplicable subsets with each user decision. The DM-1 directory is the guide to the hierarchical structure of the data pool; it shows an incremental increase in detail at each level in the hierarchy. The dialogue procedure uses the directory in developing the displays and determining which items to display, based on the user's selection. No reference to the data is required.

9.2.2 Object of the Dialogue Procedure

A specification of information in the data pool contains two elements: the names of the items which carry the information and the conditions which define the value occurrences which are significant. Each of these elements relates to an entity which is described by the information. The object of the dialogue is to determine the names of the relevant items and the conditions which define the pertinent entities.

Suppose that an inquirer needs information about transistors. This makes transistors the subject of the inquiry (entity). There are many kinds of information about transistors. The inquirer may be interested in their type designator and power rating. These are the attributes which carry the information he needs (names). However, he doesn't need the information for all transistors in existence. There might be a minimum power rating and a maximum cost which will meet his needs; he might also need a short mean-time-to-failure (conditions). The dialogue would permit him to probe the area of the data pool dealing with transistors and select the type and power rating as the desired items. Then he would be guided to specify the minimum acceptable power rating, the mean-time-to-failure, and the maximum cost. After the dialogue, the system has a formal statement of a query, which might look like:

```
RETRIEVE      TRANSISTOR TYPE,          POWER RATING
   IF      POWER RATING ≥ 100, AND
           COST ≤ 20, AND
           MTTF ≥ 50
```

9.2.3 Uses of the Dialogue Procedure

The dialogue procedure may be used to fulfill a number of information needs with respect to DM-1. Since the product of the dialogue is a formulated query, the

dialogue may be used wherever a query could be used. However, other elements of the DM-1 also function with a condition. The identification of the individual data items to be affected by a maintenance operation is accomplished with a condition. The selection of data as input to a job in a job request may also be conditional. The dialogue can be used to probe the data pool to define the precise elements for these conditional operations by developing a condition with system guidance.

At the completion of the dialogue, the user is asked what is to be done with the information developed. He may display the formulated query, proceed to the Query job, store the query, or store only the condition. If the query is stored, the user may call the Query job at any later time, by specifying the stored query as the indirect input. Similarly, if the condition is stored, the user may bind it later to any job which accepts a condition as input.

An inquirer who approaches the dialogue with the need for an answer from the data pool elects to go on to the Query job after the dialogue. He would specify that the results of the query are to be displayed to him or printed on hard copy.

A user who wants to perform further analysis on the information he has selected also selects the Query job as the last stage of the dialogue. He provides a name for the information and requests that the results be stored in the work area for further processing. He may specify a structure for the results, or he may accept the structure derived by the Query job from the relationships among the desired items in the data pool. When the query is finished, the newly created item is available in the work area. It may be bound to any job for further processing by specifying the user-assigned name.

Another user might have some new data to be added to the data pool, or he might have some other maintenance operation to perform. If he is uncertain of the precise nodes in the structure which should be affected by the operation, he may perform a dialogue. While determining the structure in the pertinent area of the data pool, the user may develop a condition which defines the precise data he wishes to change. He may request that the condition be stored with an assigned name. After the dialogue, the user may call the appropriate maintenance job and refer to the condition by name.

These examples show the range of uses for the dialogue procedure. Its primary value is to help the lay user of the system formulate a specification of his information needs. But it may be used for other purposes, ranging from a review of the structure to the development of a private file for analysis.

9.3 THE DIALOGUE PROCEDURE

9.3.1 Structure of the Dialogue Query

Notification that a dialogue is to be performed is made by a standard DM-1 job-run request, which is inserted at the console. The Dialogue Query job is called into the processing system, and it controls the communication with the console, the reading of the keyboard inputs, and the preparation of displays. The Dialogue Query job is performed as five distinct steps. These are:

Phase 1

(1) Selection of Relevant Items

(2) Display of Selected Items

Phase 2

(1) Specification of Limiting Properties

(2) Display of the Condition

(3) Selection of the Output Process

The following description explains the overall structure of the dialogue process and emphasizes what the inquirer is trying to accomplish in each step. The dialogue facilities available to the user are described where applicable. The discussion is presented under the headings of the five distinct steps of the dialogue query.

9.3.1.1 Phase 1: Item Selection Phase. The first phase of the dialogue procedure guides the inquirer in selecting those items whose value or content he wishes to examine. There are two major steps involved in this phase.

9.3.1.1.1 Step 1: Selection of Relevant Items. The selection process is a step-by-step dialogue with the DM-1 system in which the user selects items of interest from screen displays. The initial display made to the user at the start of the dialogue will contain up to the first eight items subsumed by the highest node. If there are more than eight items, an "ADDITIONAL" selection will be offered on the screen display. The user may select an item by indicating its number; he may choose "ADDITIONAL"; he may depress the OPTION button; or he may choose to press the NONE button.

(1) If he selects a nonterminal (not a field) item, a new display of up to the first eight items subsumed by the selected item will be made. If there are more than eight items, an "ADDITIONAL" selection will be offered.

(2) If the user selects a terminal item, it will be stored, and the same screen display will be reissued, with the exception that the previously selected item will be indicated.

(3) If "ADDITIONAL" is chosen, a display of up to the next eight items of the highest node are made. The "PRIOR" selection is added to the screen, since there are prior items on this node, and, if there are also additional items, the "ADDITIONAL" selection is added to the display.

(4) If OPTION is selected, a display of the available options is made. The user may select an option or NONE. If NONE is selected, the last screen display of the items of the present node is redisplayed. The only option at this stage of processing is the homograph option. The term homograph is used to indicate that there are items in different areas of the data

structure with the same name. If the user selects "HOMOGRAPH," the last screen display of the items of the present node will be shown. The user selects an item, and a screen display is made of all parent-node names for items with the same name as the item selected. The user may now choose one of these names. This will cause a new display of up to the first eight items subsumed by the selected item. If there are more than eight items, an "ADDITIONAL" item will be offered.

(5) If the user initially selects NONE, he indicates that he does not have any interest in any of the major nodes of the data base. Therefore, the first step of Phase 1 ends, and the dialogue proceeds to the second step.

Each time the user steps to a different level of the structure, a display offering him a choice of items is developed. The user may select an item; he may choose "ADDITIONAL" or "PRIOR" if they are available; he may select OPTION; or, he may choose NONE. All steps basically work the same as with the first display. However, if "PRIOR" is selected, the previous eight items of this node are displayed. If these are not the first subitems, the "PRIOR" selection is offered again; since there are "ADDITIONAL" items, this selection will be added. If the NONE selection is made, the user indicates that no items on this level are of interest. Therefore, if there is a higher level, up to the first eight items subsumed by the higher node are shown, with, if necessary, an "ADDITIONAL" selection. If, however, the current display is the highest level, the user has stepped through the data base to the point where he feels that he has made all of his Phase 1 selections. This ends the first step of Phase 1, and the dialogue proceeds to the second step.

9.3.1.1.2 Step 2: Display of Selected Items. The display of all Phase 1 selected items allows the user to modify the selections and determine what he wishes to do with his choices. The display will exhibit all Phase 1 selections in a structural format. That is, all selected items will be shown in an indented format so that the subsumed relationships are quite apparent to the user. If no Phase 1 selection were made, this fact will be shown to the user. After the inquirer depresses CONTINUE, a display is made requesting selection of the next step.

(1) The user may select from two steps if there were no Phase 1 selections, namely, restart of Phase 1, or transfer to Phase 2. If, however, there were Phase 1 selections, the user may select from four steps, namely, the above two

steps and the additional steps of depressing OPTION or printing a copy of all Phase 1 selections. A description of each of these termination steps follows:

(a) If restart Phase 1 is selected, all selections are erased, and the entire selection process is restarted at the first step of Phase 1.

(b) If a transfer to Phase 2 is desired, the user wishes to give a set of conditions. These conditions might be used to narrow the search for Phase 1 selections, or, if no Phase 1 selections were made, Phase 2 will develop a condition that can be stored for use in another job. In any case, the dialogue proceeds to the first step of Phase 2.

(c) If OPTION is chosen, a display of available options is made for the user's choice. The user may select from the Delete-Item option, the Add-Item option, and the Redisplay of all Phase 1 Selections option. The Delete-Item option will display all Phase 1 selections, and it will allow the user to delete any items he wishes. Upon conclusion, the user is again requested to select the next step at (1). The Add-Item option returns control to the first step of Phase 1 so that further selections can be made. The Redisplay Phase 1 Selections option will cause a display of all Phase 1 selections after a return to the beginning of the second step of Phase 1.

(d) If a request for the printing of a copy of all Phase 1 selections is made, this service is provided. At this point, the Dialogue Query job terminates.

9.3.1.2 Phase 2: Condition Specification Phase. The second phase of the dialogue is concerned with the specification of a set of logical conditions which will restrict the search. There are three major steps involved in this phase.

9.3.1.2.1 Step 1: Specification of Limiting Properties. The specification of limiting properties begins with a selection process quite similar to that of Phase 1; however, the items selected in this phase are items for which the user will select values so as to create a condition. The initial display and the displays that follow are similar to those in Phase 1.

The differences occur when OPTION is selected, when no selections are made from the highest display, and when a terminal item is chosen.

(1) If OPTION is selected, the available option, namely, redisplay of all Phase 1 selections, is offered. When selected, the display will take place, and when the user depresses CONTINUE, the display which was shown when the OPTION request was made is re-displayed. At that time, another selection can be made.

(2) If NONE is selected, and if the highest level of the structure is currently displayed, the user indicates that he has selected all desired Phase 2 items. The dialogue proceeds to the second step of Phase 2 to display the total condition to the user.

(3) If a terminal item (field) is chosen, the user is requested to supply values and a relationship ($=, \leq, \geq, \neq$) between the field and each value. If the field is indexed, that is, if the system has a list of values of the selected field, they are displayed in a digestible form, and the user may select as many as he wishes, supplying a relation selection for each immediately after a choice. The user may have to (if the field is not indexed), or wish to (if the field is only partially indexed), key-in one or more values for the field. If so, a sample format is given, and the user keys-in the values. The relations are indicated in much the same manner as for indexed fields. When the user specifies that he wishes to impose no further restrictions for this field, he is asked if he wishes to narrow or broaden the search.

   (a) If the search is to be narrowed, that is, if the user wishes to select other fields for specification of values, the last display of the item structure is redisplayed, and the user may make further selections.

   (b) If the search is to be broadened, the user wishes to impose an entirely new condition. The dialogue proceeds to the first step of Phase 2.

   (c) If neither narrowing nor broadening is desired, the user has supplied all Phase 2 selections, and the dialogue proceeds to the second step of Phase 2 so that the user may see all of his Phase 2 choices.

9.3.1.2.2 Step 2: Display of the Condition. The display of the condition step will allow the user to check if the evolved condition meets his criteria. After the display of the total condition, the user can specify that the condition is satisfactory, the condition is to be narrowed, or that the condition is to be broadened.

(1) If the condition is satisfactory, that is, if the user feels that he has chosen all conditions, the dialogue proceeds to the third step of Phase 2.

(2) If the condition is to be narrowed, the user determines which term of the condition is to be affected, and the dialogue proceeds to the start of Phase 2.

(3) If the condition is to be broadened, that is, if an alternative condition is to be evolved, a transfer to the start of Phase 2 is made, with the appropriate information concerning the fact that this is a broadening request.

9.3.1.2.3 Step 3: Selection of the Output Process. This step allows the user to specify what he wishes to do with the Phase 1 (if any) and Phase 2 (if any) selections he has made.

(1) There are four conditions that determine which output processes are available:

(a) If there were no Phase 1 or Phase 2 selections, the Terminate process is available;

(b) If there were no Phase 1 selections, Terminate, Type Copy of the Search Condition, Restart Phase 1, and Restart Phase 2 are available;

(c) If there were no Phase 2 selections, Terminate, Store Query, Query, Conditional Reformat, Restart Phase 1, and Restart Phase 2 are available.

(d) Otherwise, all output processes are available.

(2) A description of the output processes follows:

(a) Terminate - The job is terminated.

(b) Type Copy - A copy of the Phase 2 Search Condition is produced, and the job is terminated.

(c) Store Condition - The condition is stored, and the job is terminated.

(d) Store Query - The entire query is stored, and the job is terminated.

(e) Develop Display - The Query job is called so as to develop a display of all items that meet the condition. The job is then terminated.

(f) Store Developed Item - The Conditional Reformat job is called to retrieve the items that meet the condition into a work-area item. The job is then terminated.

(g) Restart Phase 2 - All Phase 2 selections are erased, and the dialogue proceeds to the beginning of Phase 2.

(h) Restart Phase 1 - All selections are erased, and the dialogue proceeds to the beginning of Phase 1.

9.3.2 Console Dialogue Example

The following example shows (1) how data may be maintained in DM-1, (2) what the succession of displays shown on the console might look like, and (3) the general procedure of the statement composition.

The inquirer who prepares the statement at the console need not be familiar with the features and design aspects of any mechanisms; the Dialogue Query job will make all features available to him without his having to be concerned with them.

A relatively small data base is considered in the example. However, the implications of a larger data base on the query procedure and display are obvious.

Example:

Query Statement: The inquirer wishes to find the Purchase Order (P. O.) Number(s) of those orders which are for GE or RCA, and for which the individual order is greater than or equal to $10,000.

Using this statement, which may either be prepared in writing or kept in mind, the inquirer would identify that the item being sought is Purchase Order Number (Phase 1); the rest of the items are conditions of the search (Phase 2).

Best Available Copy

Let it be assumed that purchasing data is one of the major segments of the data base maintained in the system. The purchasing information data base is structured as shown in Figure 9-1. It would be infeasible to display all elements of the actual data structure. Therefore, a level-by-level approach is employed.

(1) Thus, when the job-run request "Dialogue" is inserted at the console, Phase 1 of the dialogue begins (Item Selection Phase), and the major segments appear as in Display 1. A request for the selection of a desired output item is made.

ITEM SELECTION

Please Select Output Item Desired.

1 Purchasing Information

:

5 Reliability Information

:

8 Production Information

DISPLAY 1

The inquirer is first concerned with indicating the information he is seeking (Purchase Order Number) and he considers it most probable that it is in the area of PURCHASING INFORMATION which he selects.

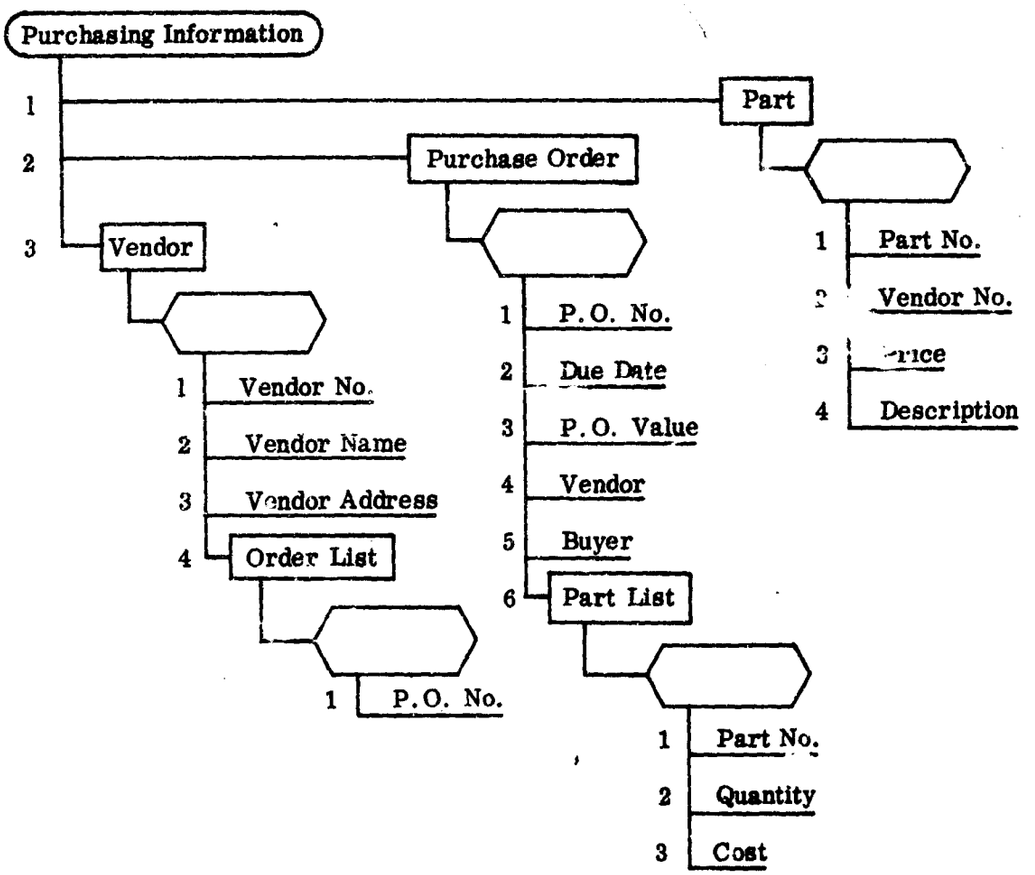
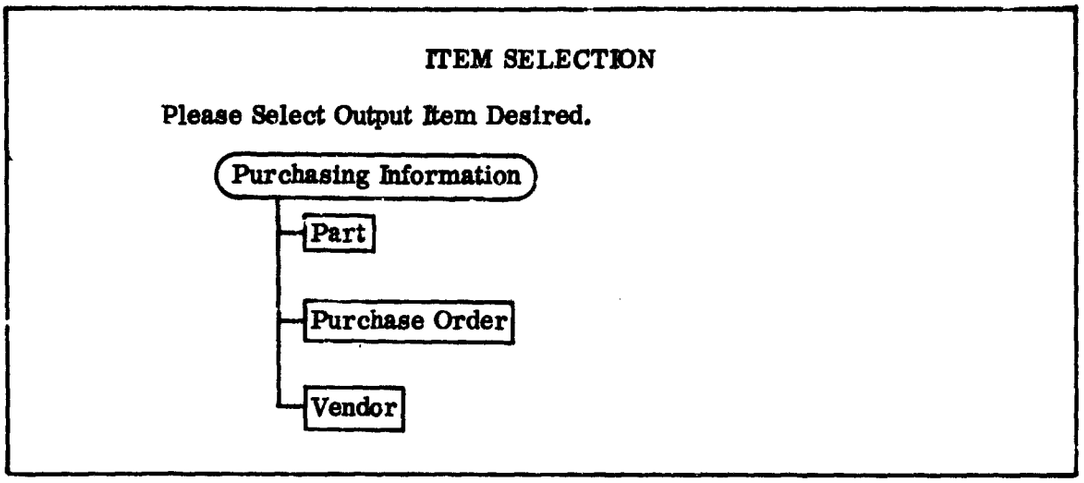


Figure 9-1. Structure of Purchasing Data Base

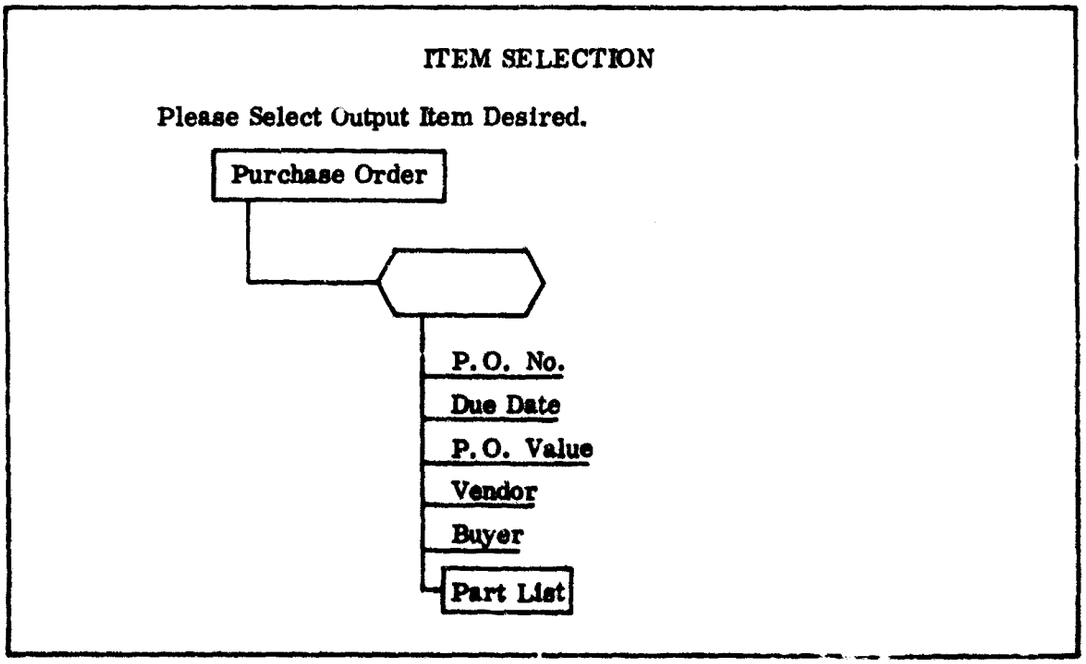
(2) The next level is displayed (see Display 2) after the inquirer selects PURCHASING INFORMATION.



DISPLAY 2

PURCHASE ORDER is the type of item being sought, and it is indicated by the inquirer, whereupon Display 3 appears.

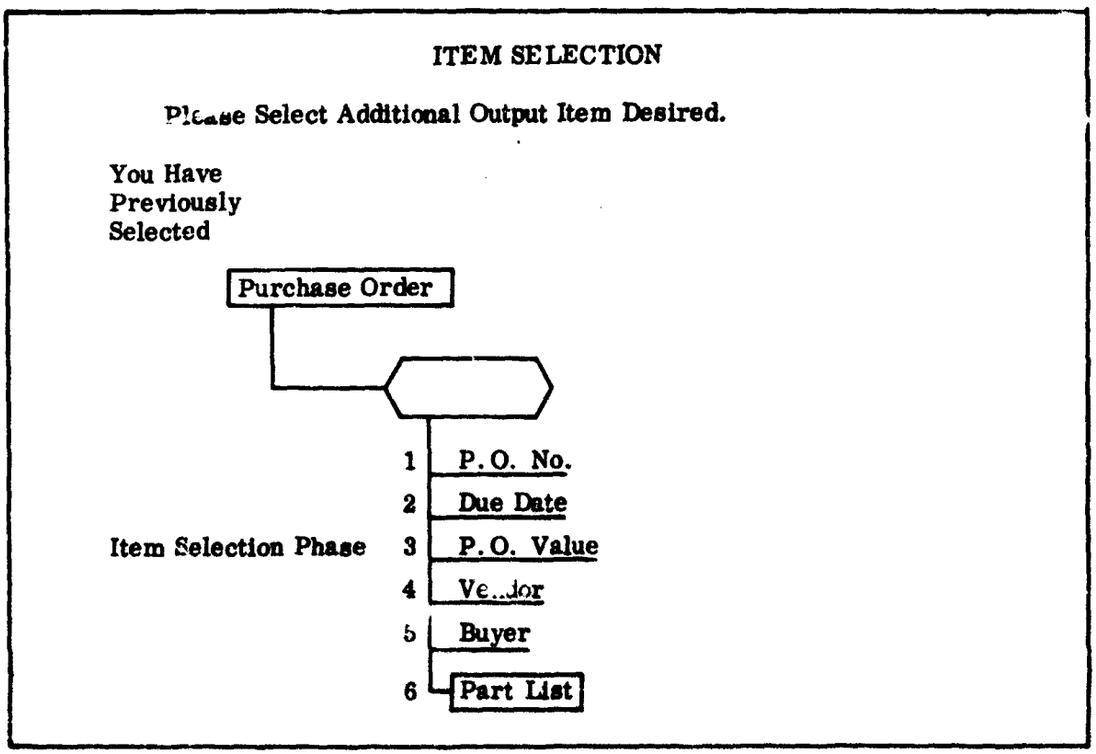
(3) The user recognizes that part of the data base that contains the item he is seeking, Purchase Order Number (P.O. No.).



DISPLAY 3

He thereupon selects the P.O. No. to indicate that it is the item sought.

(4) A display almost identical to the last is made. This time, however, the request is for an additional output item selection, and P.O. No. is indicated as having been previously selected in the Item Selection Phase.



DISPLAY 4

The user chose NONE, since he has selected all Phase 1 items of interest on this level.

(5) Therefore, we advance to the next higher level in the tree structure and request the selection of an additional item. The file PURCHASE ORDER is indicated as having been previously selected in the Item Selection Phase.

ITEM SELECTION

Please Select Additional Output Item Desired.

You Have
Previously
Selected

Purchasing Information

Item Selection Phase

1 Part
2 Purchase Order
3 Vendor

DISPLAY 5

Again the response is NONE, since no further probing is required to define the desired items.

(6) Therefore, the next higher level in the structure (in this case, the highest level) is redisplayed. A request for an additional selection is made.

ITEM SELECTION

Please Select Additional Output Item Desired.

You Have
Previously
Selected

Item Selection Phase

1 Purchasing Information
5 Reliability Information
8 Production Information

DISPLAY 6

Again, NONE is the user response.

(7) As no further selections have been made from the highest level, a display of all selected items is made, showing structural relationship among the chosen items.

ITEM SELECTION

You Have Selected The Following Item(s):

1 Purchasing Information

2 ---Purchase Order

3 ---P.O. No.

DISPLAY 7

Since the user is satisfied with this display, and does not wish to add or delete items, he chooses CONTINUE.

(8) A display requesting the selection of the next step is made.

ITEM SELECTION

Select Next Step:

1 Condition Specification Phase

2 Type Hard Copy And Terminate

3 Erase And Restart Item Selection Phase

4 Option

DISPLAY 8

Step 1 is selected, thereby indicating that the user wishes to go to Phase 2 to impose conditions.

(9) Phase 2 (the Condition Specification Phase) starts with a display of the highest node and a request for a choice for condition specification. An indication of previously selected items is made.

CONDITION SPECIFICATION

Choose An Item Whose Value Will Be Specified As A Condition Of Retrieving The Output Items Desired. Key In ITEM NUMBER or NONE.

You Have Previously Selected

Item Selection Phase

1 Purchasing Information

5 Reliability Information

8 Production Information

DISPLAY 9

The user's choice is PURCHASING INFORMATION, since he feels that the items he wishes to use as conditions are subsumed by this node.

(10) A display of the items subsumed by the item selected in the preceding display is made, with a request for choice. PURCHASE ORDER is indicated as having been a prior selection.

CONDITION SPECIFICATION

Choose An Item Whose Value Will Be Specified As A Condition Of Retrieving The Output Items Desired. Key In ITEM NUMBER or NONE.

You Have Previously Selected

Purchasing Information

Item Selection Phase

1 Part

2 Purchase Order

3 Vendor

DISPLAY 10

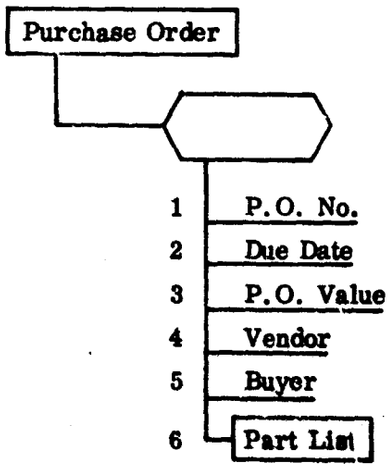
The choice is PURCHASE ORDER, since the user wishes to get to the subsumed items of this node.

(11) A display of the items of the subsumed node, with a request for choice, is made. A previously selected item is so indicated.

CONDITION SPECIFICATION

Choose An Item Whose Value Will Be Specified As A Condition Of Retrieving The Output Items Desired. Key In ITEM NUMBER or NONE.

You Have Previously Selected



Item Selection Phase

1 P.O. No.
2 Due Date
3 P.O. Value
4 Vendor
5 Buyer
6 Part List

DISPLAY 11

The user selects VENDOR, since he desires to specify certain vendors' names to identify the pertinent purchase order numbers.

(12) Since the item chosen in the last display is a field, and since VENDOR is a fully indexed attribute, all values are displayed, with a request for choice.

CONDITION SPECIFICATION

Please Choose One Of The Following Values Of:

VENDOR

1	GE
2	IRC
3	RCA

DISPLAY 12

GE is chosen as a value for the condition.

(13) A display of the four relationships, and the selected value of VENDOR is created. An appropriate message is also displayed.

CONDITION SPECIFICATION

Please Choose One Of The Following Relations Between VENDOR And The Value Specified.

		RELATION	VENDOR
1	=	=	GE
2	$\neq$		
3	$\geq$		
4	$\leq$		

DISPLAY 13

After the selection, the appropriate relationship is placed in the column marked "Relation"; that is, to the left of the respective value of VENDOR. CONTINUE is then selected by the user.

(14) Another request for a choice of a value for VENDOR is made.

CONDITION SPECIFICATION

Please Choose One Of The Following Values Of:

VENDOR

1	GE
2	IRC
3	RCA

DISPLAY 14

RCA is now chosen as an appropriate restrictive value.

(15) A request for a relation description gives the following display.

CONDITION SPECIFICATION

Please Choose One Of The Following Relations Between VENDOR And The Value Specified.

		RELATION	VENDOR
1	=	=	RCA
2	$\neq$		
3	$\geq$		
4	$\leq$		

DISPLAY 15

The "=" conditions is chosen, as indicated on the display, and the user signals CONTINUE.

Best Available Copy

(16) Another request for a value of VENDOR is made.

CONDITION SPECIFICATION

Please Choose One Of The Following Values Of:

VENDOR

1 GE
2 IRC
3 RCA

DISPLAY 16

NONE is the response, since the user has selected all values of interest.

(17) The total condition (underlined) is displayed, and a question concerning additional restrictions is posed. A "YES" or "NO" answer is available.

CONDITION SPECIFICATION

The Total Condition Is Now:

VENDOR = GE OR RCA

Can You Specify Additional Terms Which Will Narrow The Underlined Part Of The Condition?

1 YES
2 NO

DISPLAY 17

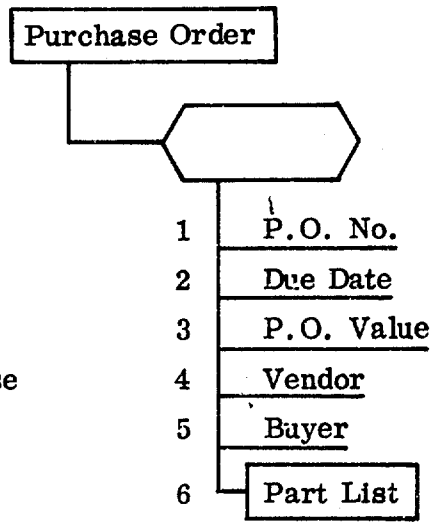
The response is "YES", since another attribute is to be examined by the user.

(18) Since "YES" is selected as a response to the previous display, a redisplay of the last data structure exhibited is made. However, VENDOR is now noted as having been selected in the Condition Specification Phase.

CONDITION SPECIFICATION

Choose An Item Whose Value Will Be Specified As A Condition Of Retrieving The Output Items Desired. Key In ITEM NUMBER Or NONE.

You Have Previously Selected



Item Selection Phase

Condition Specification Phase

DISPLAY 18

The choice for the above is P.O. VALUE, because the user is interested in purchase orders whose values are greater than or equal to $10,000.

(19) Since the item chosen in the last display is a field, and since P.O.VALUE is not an indexed attribute, a sample format is given, and a request for a key-in is made.

CONDITION SPECIFICATION

Please Key-In One Value For Property:

P.O. VALUE

The Format Of This Field Is:

$00,000

DISPLAY 19

$10,000 is the keyed-in value for the P.O. VALUE condition.

(20) A display of the four relationships and the selected P.O. VALUE is created. An appropriate message is also displayed.

CONDITION SPECIFICATION

Please Choose One Of The Following Relations Between P.O. VALUE And The Value Specified.

		RELATION	P.O. VALUE
1	=	≥	$10,000
2	≠		
3	≥		
4	≤		

DISPLAY 20

After the selection, the appropriate relationship is placed in the column marked "Relation"; that is, it is placed to the left of the value of P.O. VALUE, and it is stored in the CVT.

(21) A request for a key-in of a value for P.O. VALUE is issued again.

CONDITION SPECIFICATION

Please Key-In One Value For Property:

P.O. VALUE

The Format Of This Field Is:

$00,000

DISPLAY 21

The response now is NONE, since the user has specified all conditions.

(22) The total condition (underlined) is displayed, and a question concerning additional terms to be supplied so as to narrow the underlined part of the condition is posed. A "YES" or "NO" answer is available.

CONDITION SPECIFICATION

The Total Condition Is Now:

VENDOR = GE OR RCA

AND

P.O. VALUE ≥ $ 10,000

Can You Specify Additional Terms Which Will Narrow The UNDERLINED PART Of The Condition?

1 YES

2 NO

DISPLAY 22

"NO" is the response, since all conditions have been selected.

(23) The total condition is displayed, and a question concerning broadening the condition is posed. A "YES" or "NO" answer is available.

CONDITION SPECIFICATION

Total Condition Is Now:

VENDOR = GE OR RCA

AND

P.O. VALUE ≥ $10,000

Do You Wish To Specify A Condition Which Will Broaden The Search?

1 YES

2 NO

DISPLAY 23

The user chooses "NO," since he has selected all limiting properties for purchase orders of interest.

(24) The total condition is displayed, and a question concerning further modifications is posed.

CONDITION SPECIFICATION

Total Condition Is Now:

VENDOR = GE OR RCA

AND

P.O. VALUE ≥ $10,000

Select One Of The Following:

1. The condition is SATISFACTORY
2. The search is to be NARROWED
3. The search is to be BROADENED

DISPLAY 24

Selection 1 is the choice, since the user is satisfied with the condition.

(25) The Condition Specification Phase is terminated. A display is made with a request for the selection of the next step.

CONDITION SPECIFICATION

Select Next Step:

1. Query
2. Conditional Reformat
3. Type Hard Copy And Terminate
4. Erase And Restart Condition Specification Phase
5. Erase And Restart Item Selection Phase
6. Store Query
7. Store Condition

DISPLAY 25

The user selects the Query job to retrieve the pertinent P.O. No.'s and to display them.

9.3.3 Display Development in the Example

The example in Paragraph 9.3.2 steps through a succession of displays and user decisions in a dialogue query to formulate a statement of an information retrieval request. This section describes the use of the system directories behind the scenes in developing the displays, in responding to the inquirer's selections, and in developing the information retrieval statement.

The dialogue begins when the job displays the highest level of the structure. This is accomplished readily because the highest level items have the Item Class Codes 1.1, 1.2, 1.3, ... 1.N. These codes are converted to the display through the system directories. Table 9-1 is a schematic of the directories for the data base (refer to Figure 9-1) used in the example. The Item List is the center of the directory system and other parts of the directory are parallel to it or are linked to it.

TABLE 9-1. DIRECTORY SCHEMATIC

Term List	ICC	Item List		
Purchasing Information	1.1	(Statement)	S, 3	
Part	1.1.1	(File)	F, V	
Part Record	1.1.1.R	(Record)	R, 4	
Part No.	1.1.1.R.1	(Field)	I, V	
Vendor No.	1.1.1.R.2	(Field)	I, 4	
Price	1.1.1.R.3	(Field)	E, 6	
Description	1.1.1.R.4	(Field)	A, V	
Purchase Order	1.1.2	(File)	F, V	
P.O. Record	1.1.2.R	(Record)	R, 6	FVT
P.O. No.	1.1.2.R.1	(Field)	I, 6	
Due Date	1.1.2.R.2	(Field)	D, 6	
P.O. Value	1.1.2.R.3	(Field)	A, V	
Vendor	1.1.2.R.4	(Field)	A, V	
Buyer	1.1.2.R.5	(Field)	E, V	GE
Part List	1.1.2.R.6	(File)	F, V	IRC
Part List Record	1.1.2.R.6.R	(Record)	R, 3	RCA
Part No.	1.1.2.R.6.R.1	(Field)	I, V	
Quantity	1.1.2.R.6.R.2	(Field)	I, 5	
Cost	1.1.2.R.6.R.3	(Field)	E, 7	
Vendor	1.1.3	(File)	F, V	
Vendor Record	1.1.3.R	(Record)	R, 4	
Vendor No.	1.1.3.R.1	(Field)	I, 4	
Vendor Name	1.1.3.R.2	(Field)	A, V	
Vendor Address	1.1.3.R.3	(Field)	A, V	
Order List	1.1.3.R.4	(File)	F, V	
Order Record	1.1.3.R.4.R	(Record)	R, 2	
P.O. No.	1.1.3.R.4.R.1	(Field)	I, 6	
.	.	.		
.	.	.		
.	.	.		
Reliability Information	1.5	(Statement)	S, 4	
.	.	.		
.	.	.		
.	.	.		
Production Information	1.8	(Statement)	S, 5	

The Item Class Code (ICC) is the key to the Item List. Since the Dialogue job knows the ICC's for the highest level items, it can find the Item List entries corresponding to those items. For ICC 1.1, the job finds from the Item List that the item is a statement and from the parallel Term List that its name is PURCHASING INFORMATION. The job looks up the type and name for each of the other top-level items and finds that ICC 1.5 corresponds to a statement named RELIABILITY INFORMATION and that ICC 1.8 is a statement named PRODUCTION INFORMATION. After retrieving the type and name for each top-level item, a display is developed and written on console. The next step in the dialogue is up to the inquirer.

In the example, the user selects PURCHASING INFORMATION as an item of interest. The Dialogue job must now respond to him by displaying the items directly subsumed by the selected item. Since the selected item has the ICC 1.1 and is a statement, the subsumed items have ICC's 1.1.1, 1.1.2, 1.1.3, ... 1.1.N. The item types can be retrieved from the Item List since the ICC's are known. In Table 9-1, it is seen that 1.1.1 is a file whose name is PART, 1.1.2 is a file named PURCHASE ORDER, and 1.1.3 is a file named VENDOR. This information is formatted into Display 2 and is written on the console. Again the job waits for a user selection.

Now, the inquirer selects the PURCHASE ORDER file. To display its subitems, the Dialogue job develops the ICC's of the subitems. Since a file was selected, the direct subitems are the records of the file, but this is no information for the user, so the subitems of the record are displayed next. These have the ICC's 1.1.2.R.1, 1.1.2.R.2, ... 1.1.2.R.N, since the selected file has the ICC 1.1.2. The information needed to develop Display 3 is retrieved from the Item List and the Term List as before. The fields P.O. NO., DUE DATE, P.O. VALUE, VENDOR, and BUYER, and the file PART LIST are written on the console. The inquirer indicates that P.O. NO. is the desired item. The choice is saved with its record number, and the job awaits the user's next move.

Since the user seeks no other Phase 1 items, he indicates that he wishes no further selections on this level. Therefore, since the current display contains items with the ICC's 1.1.2.R.1, 1.1.2.R.2, ... 1.1.2.R.N, we retreat to the parent node and obtain the ICC's of the next higher level, namely, 1.1.1, 1.1.2, and 1.1.3. By means of this information, Display 5 is created. This display will note that the PURCHASE ORDER file has already been evaluated by the user.

If he wished to investigate another branch of the structure in Phase 1, the inquirer could select another item from Display 5 and the system would respond by displaying the subitems on that branch. In the example, the selection of desired output items is complete; thus, the user again indicates no selections. This develops Display 6 in a manner similar to the development of Display 5, in case another branch is to be followed from the highest level. Again, the user indicates no selections, and this signals the end of Phase 1 selections.

For the Phase 2 process, the Dialogue job directs the user so that he may specify restrictions to get the Phase 1 items with the desired characteristics. The highest level of the structure is displayed; for example, Display 9.

Since the user wants to restrict the search to certain vendors, the user reselects the PURCHASE ORDER line. The program responds as in Phase 1 by displaying the subitems. This results in Display 11 containing P.O. NO., DUE DATE, P.O. VALUE, VENDOR, and BUYER. The user chooses VENDOR because he wants only those purchase orders relating to GE or RCA as vendors.

The chosen item is an indexed field. This means that there is a Field Value Table (FVT) linked to its item list entry. (Refer to Table 9-1 where the FVT for VENDOR is shown schematically.) The Dialogue job responds to the user's choice by retrieving the Field Value Table for VENDOR and displaying the values in digestible groups. The remaining displays in Phase 2 are generated by using the system directories in a similar way.

9.4 EXPLANATION OF LOGICAL DESIGN

The formulation of an Information Retrieval Statement is performed in two distinct phases, each of which involves a level-by-level discourse through part of the directory. In the first phase, the inquirer selects those items whose value (content) he wishes to examine. In the second phase, the inquirer specifies a set of conditions which will restrict the search for the values he wishes to examine (items selected in first phase). Figures 9-2 through 9-7 show the logical design for the steps explained in the following paragraphs.

9.4.1 Phase 1: Display and Select

The inquirer selects those items whose value (contants) he wishes to examine.

(1) The highest node of the data structure is used as the display node to begin the dialogue. The names of its subitems are displayed on the console. If there are more than eight, only the first eight are displayed.

(2) The inquirer may select an item of interest:

(a) If a nonterminal item, that is, an item not a field, is chosen, the display will be modified to show, at maximum, the first eight available

items on the next lower node which are subsumed by the selected item. Any previously selected item will be so indicated. Another selection can now be made in Step 9.4.1.(2).

(b) If "ADDITIONAL" or "PRIOR" is chosen, the appropriate display of up to eight additional or prior items of the current node will be created. Any previously selected item will be so indicated. Another selection can now be made in Step 9.4.1.(2).

(c) If OPTION is selected, the appropriate options will be displayed for choice. Different options will be available at certain times within the dialogue query procedure. A particular option or NONE may be selected. The termination of the chosen option will create a display of some area of the data structure on the console screen, and another selection can then be made in Step 9.4.1.(2).

(d) If NONE is the choice, a display of up to the first eight items of the next higher node will occur. Another selection can then be made in Step 9.4.1.(2). However, if the highest node of the data structure is currently displayed, transfer is made to Step 9.4.2.(1), so as to terminate Phase 1.

(e) At the time of a display, a terminal item, that is, an item that does not have any subsumed items, may be chosen. This means that the inquirer wants values of this item as a result of the query. Using the same display from which this item was selected, another selection can then be made at Step 9.4.1.(2).

9.4.2 Phase 1: Termination

The inquirer decides whether or not he has chosen the Phase 1 items he desired, and what step should not be followed.

(1) A display of all items chosen in Phase 1 is made. Various options, for example, delete an item, add an item, and redisplay all Phase 1 selections, are now available. These can be selected by choosing OPTION and then selecting a definite item from the available list of options.

(2) A request for the selection of the next step is made. The usual procedure is to go to Phase 2, Step 9.4.3.(1). However, several other choices are available.

9.4.3 Phase 2: Display and Select

The inquirer selects items for which he will supply values, and which will then act as restrictions on the search of Phase 1 selections.

(1) The highest node of the data structure is used as the display node to begin this phase of the dialogue. The names of its subitems are displayed on the console. If there are more than eight, only the first eight are displayed. It should be noted that on any display, previously selected items will be so indicated; i.e., Phase 1, Phase 2, or Both Phases.

(2) The inquirer may select an item of interest:

(a) If a nonterminal item is chosen, the display will be modified to show, at maximum, the first eight available items subsumed by the selected item on the next lower level. Another selection can then be made in Step 9.4.3.(2).

(b) If "ADDITIONAL" or "PRIOR" is chosen, the appropriate display of up to eight additional or prior items on the current level will be created. Another selection can then be made in Step 9.4.3.(2).

(c) If OPTION is selected, the appropriate options will be displayed for choice. A particular option or NONE may be selected. The termination of the chosen option will create a display of some area of the data structure on the console screen, and another selection can then be made in Step 9.4.3.(2).

(d) If NONE is the choice, a display of up to the first eight items of the next higher level will occur. Another selection can then be made in Step 9.4.3.(2). However, if the highest level of the data structure is currently displayed, transfer is made to 9.4.3.(2).

(d) If NONE is the choise, a display of up to the first eight items of the next higher level will occur. Another selection can then be made in Step 4.3 (2).

(e) At the time of a display, a terminal item may be chosen. This means that the inquirer wants to specify value(s) of that item. A transfer to 9.4.4.(1) is made.

9.4.4 Phase 2: Value Selection

The inquirer selects values for the Phase 2 attribute just selected in Step 9.4.3.

(1) If the item selected is indexed and if there are more values than can be displayed at once, a display is made of ranges of values. By breaking selected ranges into successively smaller groups, a display of individual values will eventually

evolve. However, if all values could have been initially displayed at once, this would have been done. In any case, when a particular value is selected, it is stored, and a request for a relation specification is made. For example, a particular value can be related to an attribute by =; a range by $\geq$ or $\leq$; and an exception by $\neq$. The selected relation is stored. At this time, a return to Step 9.4.4.(1) is made so as to redisplay all values. If, at any time, the user selects NONE, a check is made to see if all values of the particular item were indexed. If so, go to Step 9.4.5.(1), since all values have been selected by the user. Otherwise, a transfer to Step 9.4.4.(2) is made to determine whether the user wants to key-in any values.

(2) If the field was not indexed, or not fully indexed, the user may key-in a value. If he does, the keyed-in value is stored and a request for a relation selection is made. When this is performed, the condition is stored, and a return to Step 9.4.4.(2) is performed. If NONE is the response, the user has chosen all values, and a transfer to Step 9.4.5.(1) is made.

9.4.5 Phase 2: Condition Development

The inquirer is presented with the evolving condition, and he is asked if he wishes to develop the condition further.

(1) At this time, a display of the condition with a request for further narrowing (AND) of the search occurs. If narrowing is desired, the screen display shown at the selection of this terminal item is redisplayed, and another selection can then be made in Step 9.4.3.(2). If no further narrowing restrictions are to be given, the inquirer is asked if a broadening (OR) of the search is to occur. If no further input is to be made, transfer to Step 9.4.6.(1). Otherwise, the condition is broadened by going to the start of Phase 2, Step 9.4.3.(1).

(2) The total condition is displayed.

(a) If found to be satisfactory, go to Step 9.4.6.(1) to terminate Phase 2.

(b) If the condition is to be narrowed, a determination of which term of the condition is to be narrowed is made. When the term is located, a transfer to Step 9.4.3.(1) is made. If no term is selected, a transfer to Step 9.4.5.(2) is made.

(c) If the condition is to be broadened, transfer to Step 9.4.3.(1) to repeat Phase 2 and develop an alternative condition.

(d) If none of these selections is made, the display at Step 9.4.5.(2) is repeated.

9.4.6 Phase 2: Termination

The inquirer is presented with the list of available next steps.

(1) The user will select the next step he wishes to perform with the Phase 1 and Phase 2 selections, just chosen. If the user chooses to repeat either Phase 1 or Phase 2, the appropriate transfer takes place. Otherwise, the dialogue transmits an output according to the selected output step.

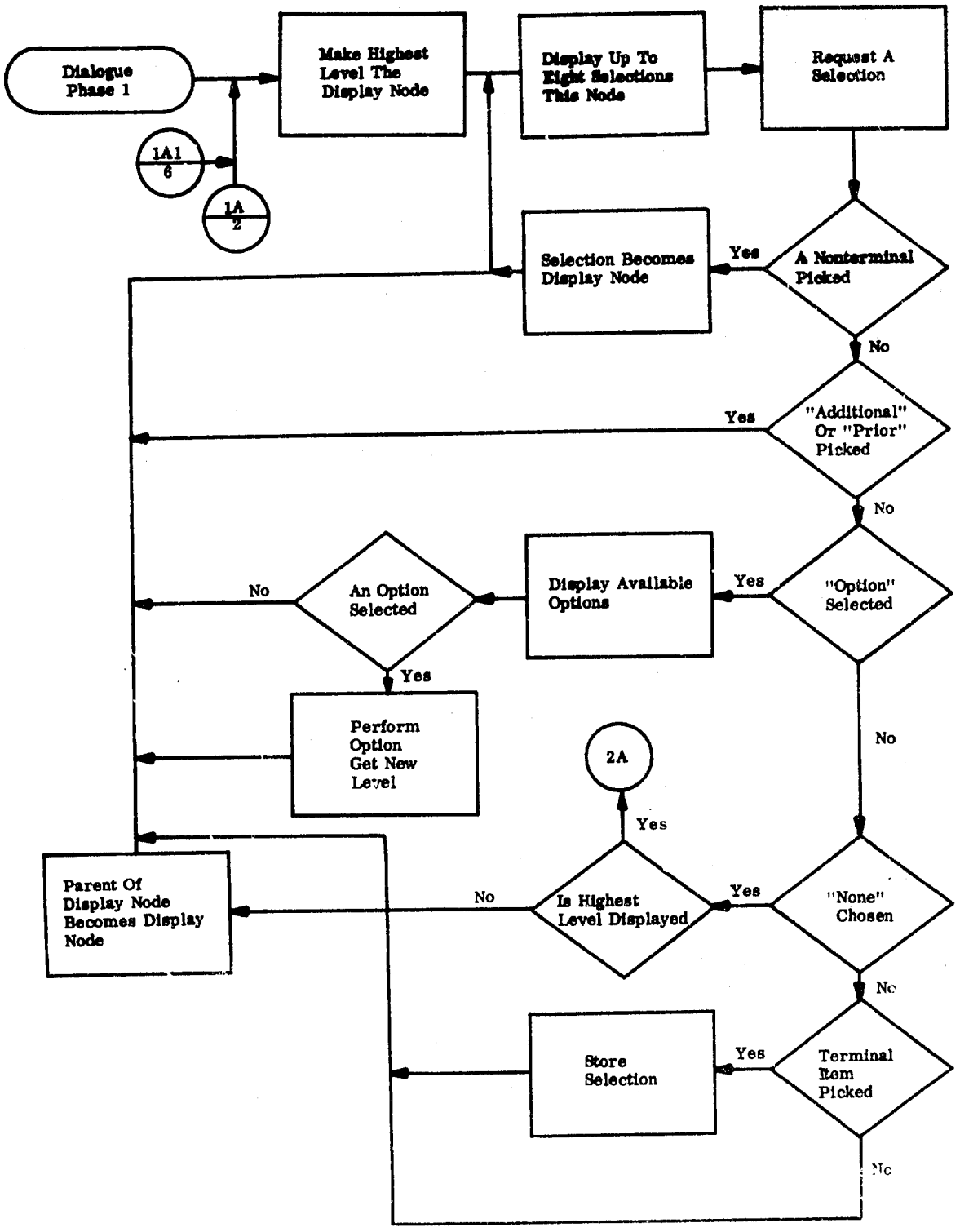
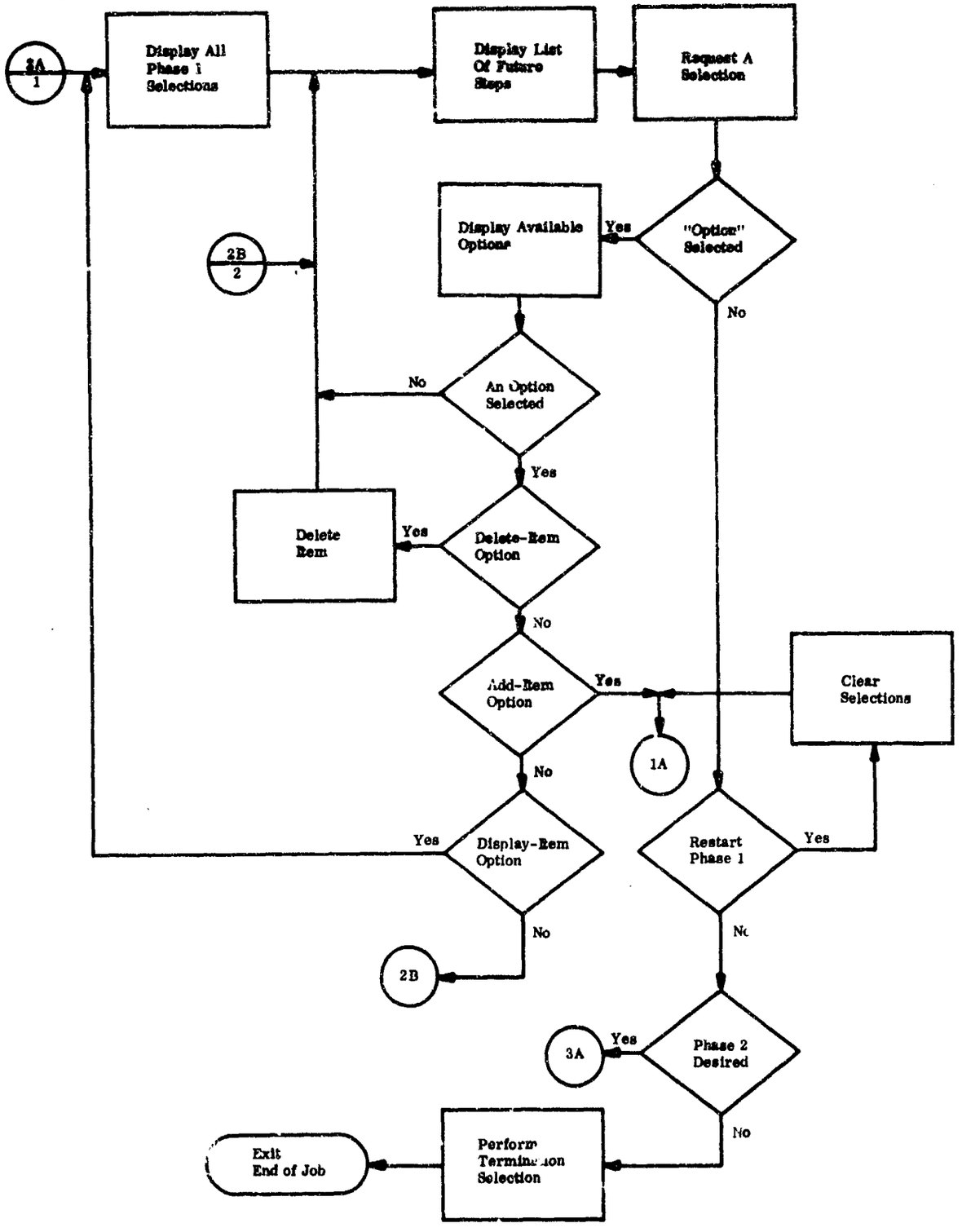


Figure 9-2. Di[illegible]nd Select



. . Figure 9-3. Phase I: Termination

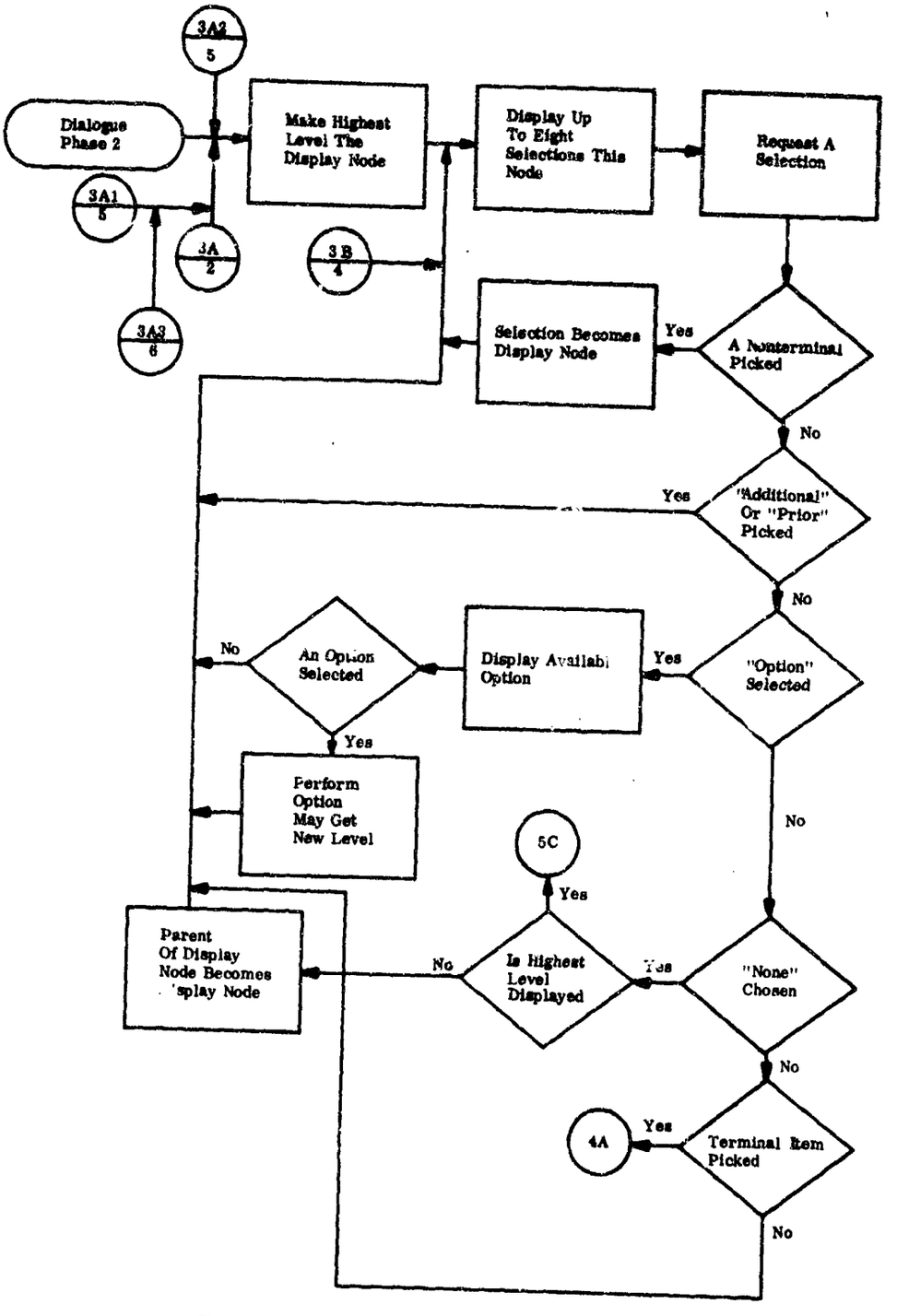


Figure 9-4. Phase 2: Display and Select

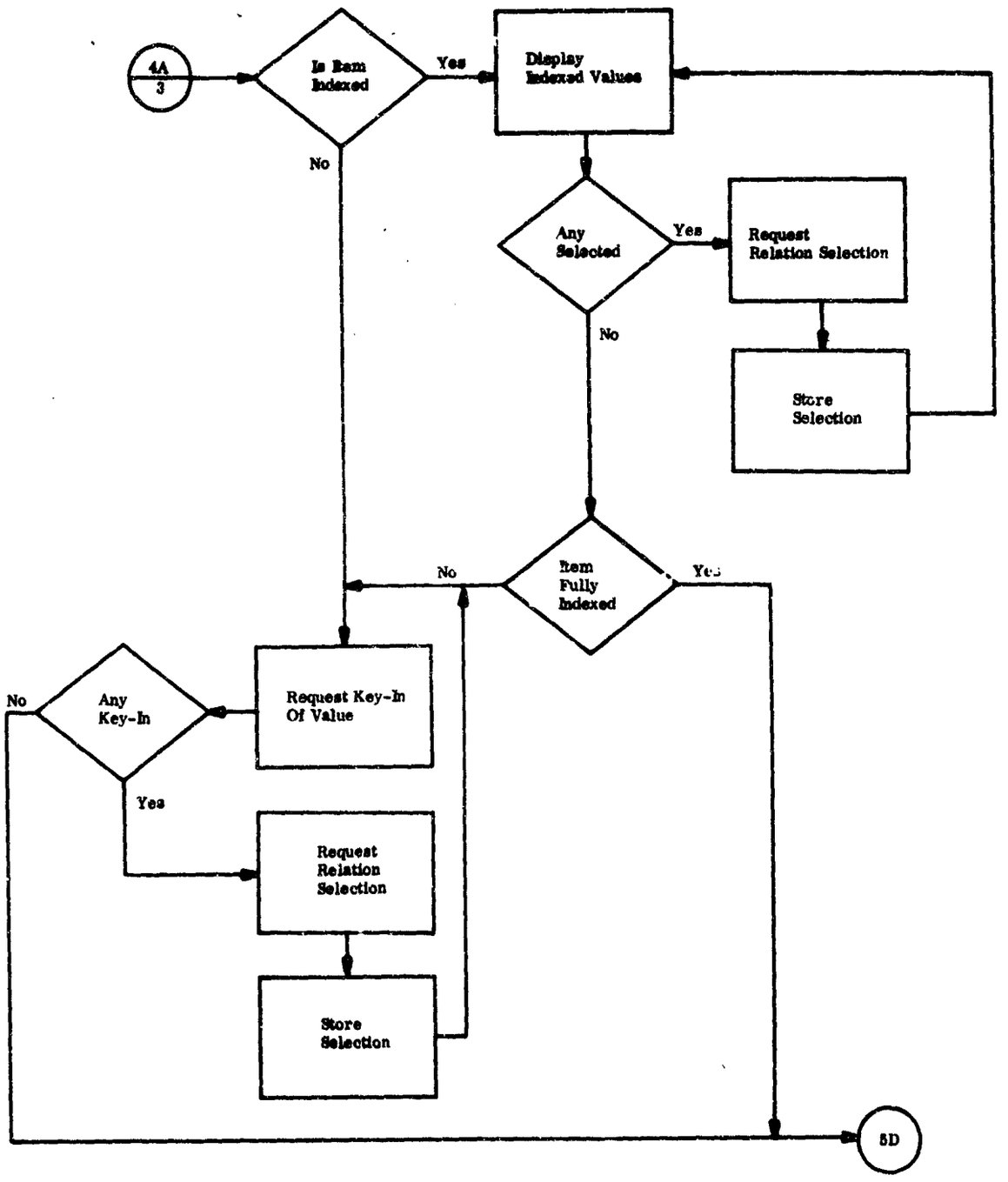


Figure 9-5. Phase 2: Value Selection

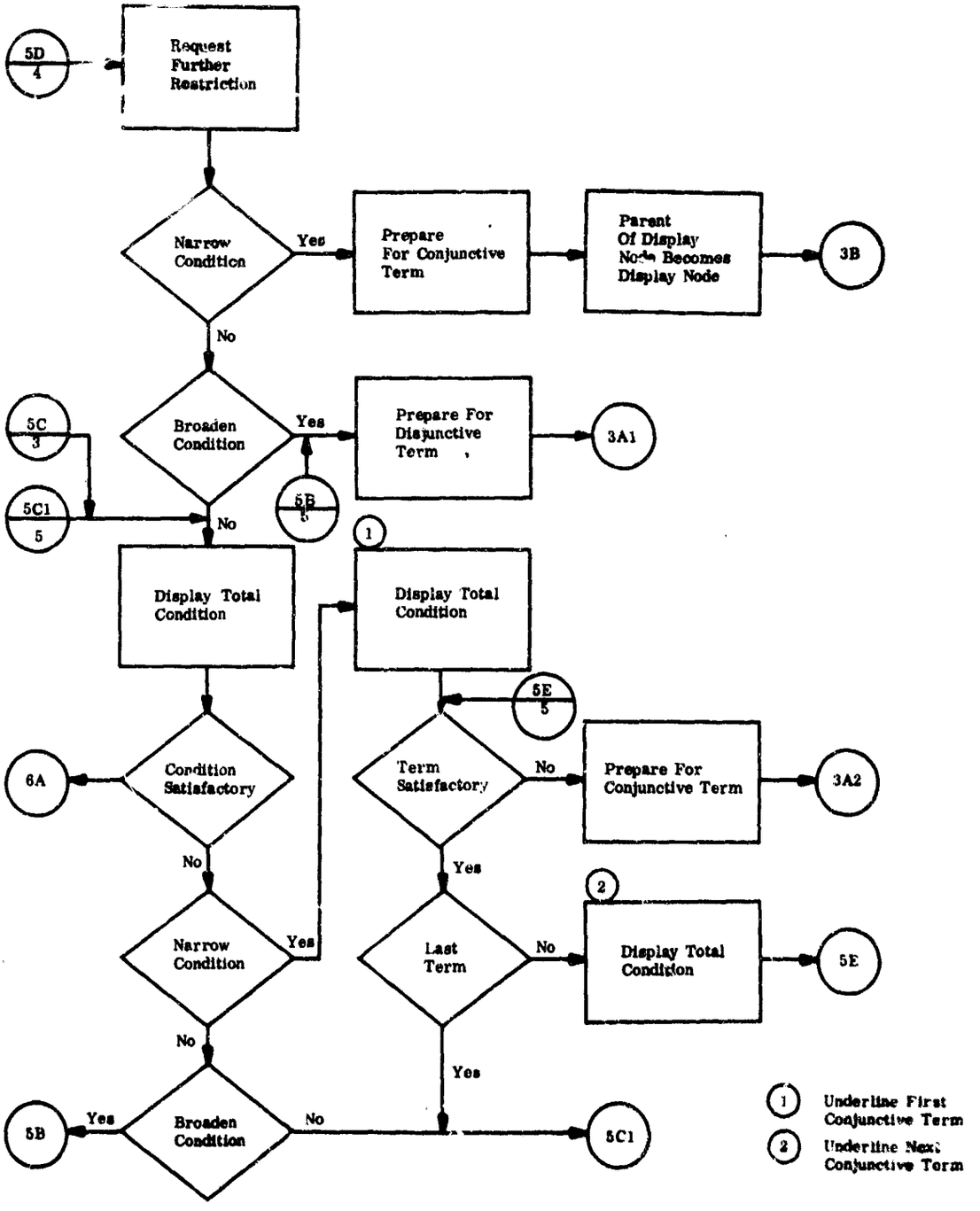


Figure 9-6. Phase 2: Condition Development

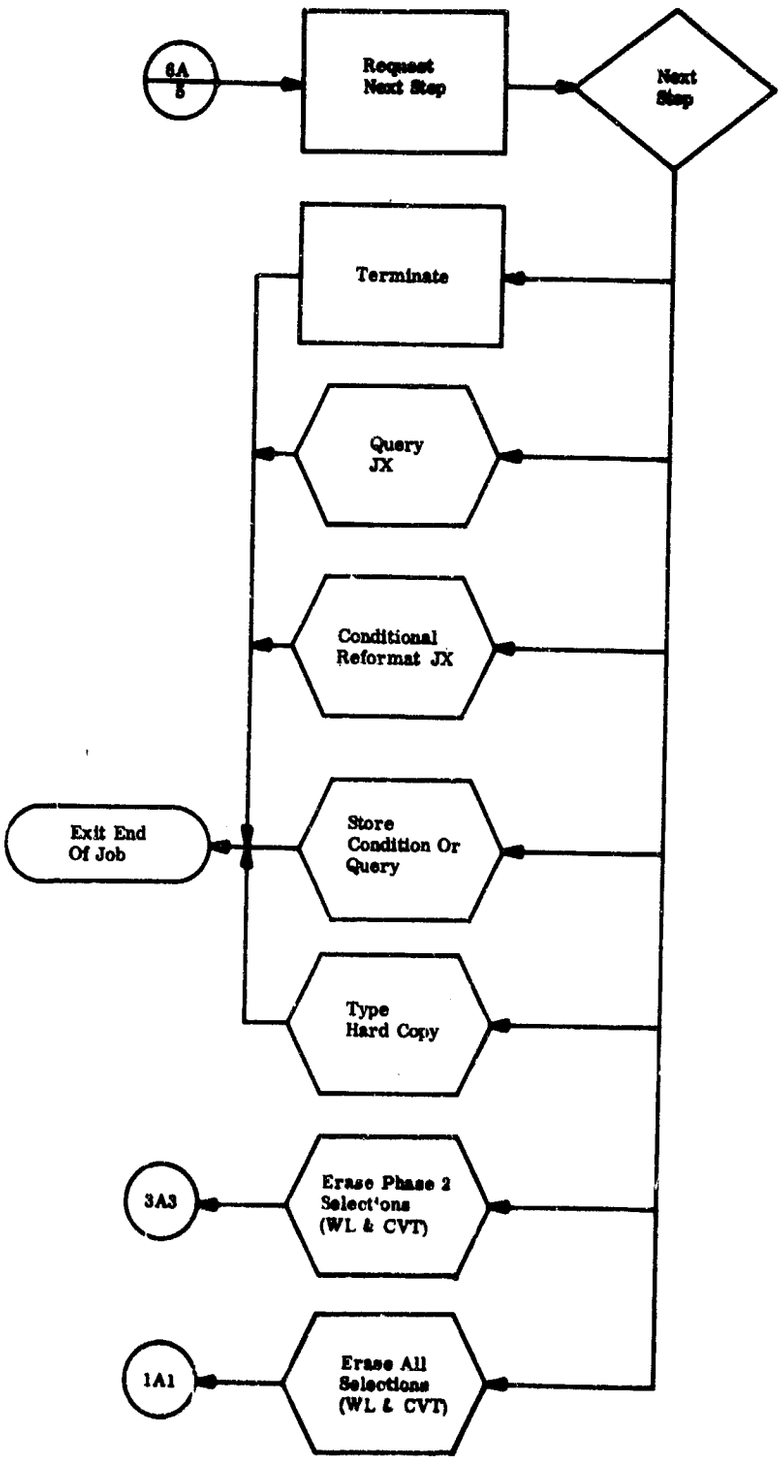


Figure 9-7. Phase 2: Termination

9.5 DETAILED DESIGN

The detailed design of the Dialogue Query is contained in the flowcharts on Figures 9-8 through 9-34. The steps in the program are explained in detail in the following paragraphs. Where appropriate, the chart connectors are referenced in the text with the explanation of the step performed at the point of the connector. The terms and parameters used in the description are explained in Paragraph 9.6.

9.5.1 Phase 1: Initialization and DWT Setup

The program begins by initializing the Want List (WL) and the Display Work Table (DWT) for the Phase 1 operation. Then the DWT is set up for the first display.

(1) (1H) The word counter in the WL is cleared to indicate an empty Want List.

(2) (1E) The node item ICC in the DWT is set to 1. This is the logical identifier for the highest item in the data pool. Its subitems are the first items to be displayed. The node level in the DWT is set to one, the level of the node item in the hierarchical structure.

(3) (1K) The DWT node pointer is set to 1. This will direct the DWT and display routines to start the display with the first subsumed item of the node item.

(4) (1B) A subroutine is called (Figure 9-25) to put the Item List record number and description for the node item into the DWT.

(5) (1D) A subroutine is called (Figure 9-26) to retrieve the Item List record numbers and descriptions for the items subsumed by the node item, starting with the subitem identified in the DWT node pointer. A maximum of eight entries are made in the DWT for the subitems.

(6) (1F) If there are more subitems of the node item, beyond the entries set into the DWT, the node size in the control word (CW) is not yet zero. An indicator is set in the CW to signal the display routine to inform the inquirer that there are additional subitems. Similarly, if the DWT node pointer is not at 1, the next set of subitems to be displayed is not the first. An indicator is set in the CW to signal the display routine that there are prior subitems. The user will be permitted to step forward or backward on the same level if there are additional or prior subitems. After the analysis, control passes to Figure 9-10, Step 1, (3D) to issue the display and detect the inquirer's response.

(7) (1C) This is an entry point to begin displaying a new level after the inquirer has made a selection or has finished searching within a given level. The CW is cleared and a transfer is made to Step 3, (1K) to begin the new display.

9.5.2 Phase 1: Selection Diagnosis

With each display, the inquirer makes a selection. His selection, if any, is recorded in the CW and, after the appropriate setting of the selection bit, control is passed to the selection diagnosis in Figure 9-9. He may have selected a specific subitem for deeper probing or as a desired attribute (if it was a field); he may have chosen to see additional or prior subitems on the same level, if these exist; or he may have chosen NONE, thereby indicating that he is finished with the subitems on this level and wants to step back to a higher level. The selection diagnosis determines his selection, acts on it, and directs the production of the appropriate next display.

(1) (2A) A check is made to see if anything was selected from the screen display (performed by examining the selection bit in the CW). If a selection was made, control is transferred to Step 3, (2Y) to act on the selection.

Otherwise, the DWT node item ICC is examined to see if it is at level one. If so, the highest node was displayed. Since the inquirer made no selection, he is probably finished with Phase 1. Control passes to Figure 9-16, Step 1, (9C) to terminate Phase 1. If the node displayed was not the highest node, the process continues with the next step.

(2) (2B) The termination symbol in the node item ICC is moved back one level, and the node level is decremented by one to produce the ICC of the parent of the node item. If the highest level of the new ICC designates a record, the process is repeated to get the ICC of its parent file. Control goes to Figure 9-8, Step 7, (1C) to set up the DWT to display the subitems of the higher level node item identified by the new node item ICC.

(3) (2Y) When a selection is detected in the CW in Step 1, (2A), an analysis of the selection is made:

(a) If "ADDITIONAL" items was the selection, the DWT node pointer is incremented by eight so that the next group of subsumed items is indicated. Transfer is made to Figure 9-8, Step 5, (1D) to set up the DWT for the additional set of subitems.

(b) If "PRIOR" items was the selection, eight is subtracted from the DWT node pointer. The CW node size is incremented by eight plus the number of items in the present display. These steps set the parameters to the proper point so as to display the group of eight subitems preceding the group just displayed. Transfer is made to Figure 9-8, Step 5, (1D) to set up the DWT for the prior set of subitems.

(c) If neither of the selections was made, a specific subitem was selected, and its Item List record number, level number, and subnode number are inserted in the Want List (WL), if an entry for the item is not already there. This identifies it as an item selected in Phase 1. The WL is kept in record-number order. A check of the item type is made. If it is a field, the terminal bit is set in the WL, and a transfer is made to Figure 9-10, Step 1, (3D) to reproduce the same display as the previous one so that the inquirer may make a further selection. For other items, the Phase 1 bit is set in the WL. If the item selected was a statement, the process continues with Step 4, (2W). Otherwise, the selected item is a file; thus, a record symbol is inserted in the next level of the node item ICC, and the node level is incremented by one.

(4) (2W) The subitem number of the selected item is inserted after the last level of the node item ICC. The subitem number is the current node pointer plus the selection number of the selected item. The node level is incremented by one. This sets the node item ICC to the ICC of the selected item so that the next display will show its subitems. Control goes to Figure 9-8, Step 7, (1C) to set up the DWT for the subitems of the selected item.

9.5.3 Phase 1: Console Display

This process is entered from Figure 9-8 after the DWT has been set up for a new display. It is also referenced from Figure 9-9 to repeat a display after a field was selected; from Figure 9-11 to repeat a display when the inquirer makes no selection in the homograph option; and from Figure 9-14 to repeat a display when the inquirer makes no selection after requesting an option. This chart describes the process of creating a display from the DWT, displaying it, and recording the inquirer's response in the CW.

(1) (3D) A subroutine is called (Figure 9-27) to create the output buffer by converting the record numbers in the DWT to names, inserting appropriate symbols in the output lines, and including appropriate messages if any subsumed items to be displayed have already been chosen (all chosen items are noted in the WL). The subroutine will also place an appropriate message in the output buffer.

(2) (3E) A display on the console is performed using the output buffer just set up. An examination of the response is made:

(a) If an item 0 through 9 was selected and if such an item existed in the output buffer, we go to Step 3, (3A) on this figure so as to set up the CW.

(b) If the user selected NONE, we proceed to Step 4, (3C) on this figure so as to indicate to the system that no selection was made.

(c) If the choice was OPTION, the inquirer wishes to make use of one of the special services of the system. At this time, the available option is the homograph option. The appropriate bit allowing the routine on Figure 9-14 to make the option available is set in the option control word (OCW), and a transfer is made to Figure 9-14, Step 1, (7A).

(d) none of the preceding choices were made, a transfer to Step 2, (3E) is made.

(3) (3A) The item-selected bit in the CW is set so that the selection-diagnosis routine will know that an item currently displayed has been chosen. The item number (e.g., 0 through 9) is inserted in the appropriate position of the CW, and a transfer to Figure 9-9, Step 1, (2A) occurs.

(4) (3C) As no item was selected from the current display, the selected bit in the CW is cleared. This information is then available to the selection-diagnosis routine. A transfer to Figure 9-9, Step 1, (2A) takes place, so that further displays can be created.

9.5.4 Phase 1: Homograph Dialogue - I

If the OPTION selection is made, and if the homograph option is selected, this routine is entered (from Figure 9-15). This part of the homograph dialogue requests selection of an item for which the user wishes homographs, gets a name of the item selected, and prepares it for a Term Encoding Table (TET) search so that all ICC's with the same name will be located.

(1) (4A) The homograph option in the CW is set to one, so that certain subroutines will recognize that the homograph dialogue is currently in control. A call on a subroutine (Figure 9-27) is performed so as to create an output buffer by converting record numbers in the DWT to names, inserting appropriate symbols, and by including the proper message.

(2) (4C) A display on the console is made of the same items that were shown when the user selected the OPTION response. There will be no "ADDITIONAL" or "PRIOR" choices avilable. If a selection of an item is made, transfer to Step 3, (4F). Otherwise, a determination is made if NONE was chosen. If so, the homograph option bit in the CW is cleared, since no item was selected as being of interest for a homograph search, and a transfer to Figure 9-10, Step 1, (3D) is made so that a request for another item selection can be made.

(3) (4F) As a selection was made, the ICC, record number, and other information about the item selected for the homograph routine are placed in the node of the DWT. The node level is updated to reflect the level of the selected ICC. The node pointer is set to 1 so as to indicate which TET record within the selected TET file is to be examined. The record number of the selection is prepared for a Term List (TL) search so that its name can then be read. The

name of the selected item is prepared for a TET search so that the appropriate record within the TET file can be retrieved. Transfer to Figure 9-13, Step 2, (6A) is performed.

9.5.5 Phase 1: Homograph Dialogue - II

This part of the homograph dialogue routine is entered from Figure 9-13 after a display of the homograph items is made. This section will analyze the selection made in response to the display; if a selection was made, the ICC of the selection will be retrieved.

(1) (5A) A determination is made if a selection of an item was made. If not, a transfer to Step 2, (5X) is made. If a selection was made, an evaluation of the proper DWT node pointer entry for the selection is made. This will indicate the record number of the chosen homograph item within the TET file. This record number is prepared for a TET seek and read, and, thereby, the ICC of the selected homograph item is retrieved. It is placed in the DWT node, and the level of the node ICC is placed in the DWT. The homograph bit in the CW is cleared, to indicate that an exit from the homograph routine is being made, and the TET is closed for reading. A transfer to Figure 9-9, Step 2, (2B) is made so that the chosen ICC can be replaced by its parent. This will cause an ensuing display to contain as a subitem the term whose ICC was just selected in the homograph routine.

(2) (5X) As the user did not select a specific item, an analysis is made to see if additional items are available and if they are indicated as being desired. If so, a transfer to Figure 9-13, Step 1, (6A) is made. If not, a check is performed to see if prior items are available and if they are desired by the user. If so, a transfer to Figure 9-13, Step 2, (6B) is made. If neither, control is passed to Figure 9-13, Step 6, (6M) so that another display with a request for selection can be made.

9.5.6 Phase 1: Homograph Dialogue - III

This area of the homograph dialogue routine is concerned with setting up the output buffer of up to eight parent names for the term for which homographs are desired. It is necessary to provide the parent names, since the user must have some means to be able to differentiate among the various items with similar names. Therefore, a homograph display will consist of an item and a list of direct parents.

(1) (6B) This entrance provides for the selection of the "PRIOR" selection on Figure 9-12, Step 2, (5X). In this case, the DWT node pointer is reduced by eight so that the eight prior TET entries will be redisplayed.

(2) (6A) This step is entered from Figure 9-11 for an initial display of homograph items, from Figure 9-12 for further displays, and from Figure 9-13, Step 1, (6B). The step sets an output line counter to one. The present value of the DWT node pointer is used as a record number for a TET seek.

(3) (6J) A TET read is now performed.

(4) (9Y) A check is made to see if an end-of-file (EOF) was encountered. If so, there are no further items; the appropriate switch is cleared in the CW; and a transfer to Step 5, (6K) is made. However, assuming no EOF was encountered, the parent of the ICC just retrieved is prepared for a retrieve of its Item List (IL) entry. This will provide a record number, type, and other information that is placed in the DWT. The node pointer is incremented by one as another TET record has been processed. A test is made to see if the output line counter is equal to eight. If not, the counter is incremented by one, and a transfer to Step 3, (6J) is made. If the counter equals eight, the DWT is full. Therefore, another read of the TET is performed. This is to check if there are any more entries. If an EOF is not read, the additional bit is set in the CW. Otherwise, transfer to Step 5, (6K).

(5) (6K) If there are prior items, that is, if the DWT node pointer is greater than eight, the prior item switch is set in the CW. Now, a transfer to the subroutine in Figure 9-27 is made so that an output buffer of names, symbols, and a message can be established.

(6) (6M) A display is made of the output buffer, and a transfer to Figure 9-12, Step 1, (5A) is executed.

9.5.7 Phase 1: Option

This area is entered from Figure 9-10 if OPTION was selected, and from Figure 9-16 upon the same occurrence. The section will set up a screen display of available options, and an associated transfer table for the start of the options.

(1) (7A) The homograph bit in the OCW is examined. If set, the homograph option is added to the screen setup, and the transfer address of the homograph routine is placed

in a table. The delete bit in the OCW is examined. If set, the delete option is added to the screen setup, and the transfer address of the delete-item routine is placed in a table. The add-item bit in the OCW is examined. If set, the add-item option is added to the screen setup and the transfer address of the add-item routine is placed in a table. The Display Phase 1 Selections bit in the OCW is examined. If set, the display-item option is added to the screen setup and the transfer address of the display item is placed in a table.

(2) (7X) A display with a request message is issued, and an analysis is made to see if an available option was selected. If so, transfer to Figure 9-15, Step 1, (8C) is made. Otherwise, a determination of whether or not NONE was selected is made. If so, control passes to Figure 9-10, Step 1, (3D), where a redisplay of the last display of DWT entries will be set up for selection. If neither an item nor NONE was chosen, transfer goes to Step 2, (7X).

9.5.8 Phase 1: Transfer or Delete

This step transfers control to the proper option routine. It is called by the option evaluation routine, Figure 9-14.

(1) (8C) Appropriate transfers are made for option selections:

(a) Homograph - Transfer to Figure 9-11, Step 1, (4A)

(b) Delete Item - Transfer to Step 2, (8A)

(c) Add Item - Transfer to Figure 9-8, Step 2, (1E)

(d) Display Phase 1 Selections - Transfer to Figure 9-16, Step 1, (9C)

(2) (8A) A call on a subroutine, Figure 9-29, is made. All items will be displayed, and selected items will be deleted in an hierarchical manner.

(3) (8B) A transfer to Figure 9-16, Step 1, (9C) is made so that further termination of Phase 1 may take place.

9.5.9 Phase 1: Termination

The user is provided with a display of all Phase 1 selections, if any, and is then given a choice of further processing steps. The routine is entered from Figure 9-9,

when the highest level is displayed and no further selections are made, and from Figure 9-15, when display of all Phase 1 selections was requested or after the deletion of selected Phase 1 items took place.

(1) (9C) A call on a subroutine, Figure 9-29, is made so that all Phase 1 selections will be displayed in a readily understandable, hierarchical manner.

(2) (9D) A request for the next step is made. If there were Phase 1 selections, the user may select OPTION, the printing of a hard copy of all Phase 1 selections, transfer to Phase 2, or restart of Phase 1. If OPTION is chosen, the OCW bits for add item, delete item, and display all Phase 1 items are set, and a transfer to Figure 9-14, Step 1, (7A) is made. If a typed hard copy is desired, it is produced. The Term List (TL) is closed for reading, and an exit from the Dialogue job is made. If Phase 2 is desired, conditions are to be introduced by the user, and a transfer to Figure 9-17, Step 1, (10H) is made. If restart of Phase 1 is desired, a transfer to Figure 9-8, Step 1, (1H) is made. If there were no Phase 1 selections, only the steps Transfer to Phase 2 and Restart of Phase 1 will be available to the user.

9.5.10 Phase 2: Initialization and DWT Setup

The start of Phase 2 is almost identical to that of Phase 1, Figure 9-6. Therefore, only an outline is provided.

(1) (10H) The OCW is cleared.

(2) (10E) The node item of the DWT is set to one (the relative universe of discourse is the node with the ICC of 1), and the node level is also set to one.

(3) (10K) At this stage, the DWT node pointer is set to one. This will direct attention to the proper subsumed item(s).

(4) (10B) A transfer to Figure 9-25 is executed so as to retrieve the node IL entry.

(5) (10D) The next set of ICC's is created, and their IL entries are obtained by transferring to the subroutine in Figure 9-26.

(6) (10G) An analysis of whether or not there are additional or prior subsumed ICC's of the parent now listed as the node ICC of the DWT (we are just displaying eight ICC's at a time) is made. Appropriate bits are set in the CW, and control goes to Figure 9-19, Step 1, (12D).

(7) (10C) The CW is cleared, and control is passed to Step 3, (10K).

9.5.11 Phase 2: Selection Diagnosis

The selection process of Phase 2 is quite similar to that of Phase 1, Figure 9-9. The major difference occurs in Step 3, (11Y)c. This is the case of the selection of a field, whose chosen values will help form a condition.

(1) (11D) A check is made to see if anything was selected from the screen display (performed by examining the selection bit in the CW). If so, control is transferred to Step 3, (11Y). Otherwise, an examination is made to see if the highest node is currently displayed. If so, control is passed to Figure 9-22, Step 7, (15F).

(2) (11A) The term symbol in the node ICC is moved back one level, and the node level is decremented by one. A check is made to see if the new level is a record. If so, control goes to Step 2, (11A). Otherwise, transfer to Figure 9-17, Step 7, (10C).

(3) (11Y) An analysis of the selection is made:

(a) If "ADDITIONAL" items was the selection, the DWT node pointer is incremented by eight, so that the next subsumed items are correctly indicated. Transfer is made to Figure 9-17, Step 5, (10D).

(b) If "PRIOR" items was the selection, eight is subtracted from the DWT node pointer, and the number of currently displayed items plus eight is added to the CW node size. This will permit the eight prior items (there will always be eight prior items) to be displayed when a transfer to Figure 9-17, Step 5, (10D) is executed.

(c) If a field was selected, the record number is moved from the DWT to the Control Value Table (CVT). The incomplete bit and word-count bits are set. If the first bit in the OCW is set, the Boolean "OR" bit in the CVT entry is set so as to indicate a broadening condition, the OCW is cleared, and a transfer to Figure 9-21, Step 1, (14C) is made so as to get the user's parameter values for the chosen attribute (selection).

(d) If neither of the preceding selections were made, a statement or file was chosen. Therefore, if the record number of the selection is not already in the Want List (WL), it is inserted in its proper numerical

position. The Phase 2 bit, indicating that this record number was chosen in Phase 2, is set, regardless of whether the record number was in the WL or was just placed in the WL. In order to build a subsumed ICC, an analysis is made whether or not a statement was selected. If not, a file was chosen. Therefore, the next level of the parent ICC has an "R" (record) entry. The node level is increased by one. At this time, the same path is followed as if a statement had been chosen. Namely, the correct value of the DWT node pointer for the selection is inserted in the next level of the ICC of the parent node of the DWT. The node level is incremented by one. A new ICC has now been formed, and, in order to display subsumed items, a transfer to Figure 9-17, Step 7, (10C) is made.

9.5.12 Phase 2: Console Display

The routine is quite similar to the one described in Phase 2, Figure 9-10. The basic steps are as follows:

(1) (12D) Control is passed to the subroutine on Figure 9-27 so that the output buffer can be created.

(2) (12E) A display is made. An analysis of the response will be made:

(a) If a legitimate item 0 - 9 is selected, transfer to Step 3, (12A).

(b) If the NONE selection is made, control goes to Step 4, (12B).

(c) If OPTION is chosen, the OCW is setup for the redisplay option, and a transfer to Figure 9-20, Step 1, (13A) is made.

(d) If none of these, go to Step 2, (12E) for a redisplay.

(3) (12A) The item-selected bit in the CW is set, and the item number of the selection is placed in the CW. A transfer to Figure 9-18, Step 1, (11D) is made.

(4) (12B) The selected bit in the CW is cleared, and a transfer to Figure 9-18, Step 1, (11D) is made.

9.5.13 Phase 2: Option

This routine is very similar to Phase 1, Figure 9-14. The major difference is that the display of all Phase 1 selections is the only available option.

(1) (13A) The OCW is examined for possible options. At this time, the only available option is the redisplay of all Phase 1 selections. The option is added to the screen display, and its transfer address is put in a table.

(2) (13X) A message is created, and a display is made. If a legitimate item is chosen, go to Step 3, (13B). If NONE is chosen, go to Figure 9-19, Step 1, (12D). If neither, go to Step 2, (13X).

(3) (13B) Using the item number selected and the transfer table previously set up, a jump is executed.

(4) (13Y) If DISPLAY is chosen, go to the subroutine on Figure 9-29 for the display of all Phase 1 selections.

(5) (13D) Upon return from the subroutine, transfer to Figure 9-19, Step 1, (12D).

9.5.14 Phase 2: Terminal Selection - I

This routine will deal with the situation of a selected field being either fully or partially indexed.

(1) (14C) A determination is made if the selected attribute (a field) is indexed. This is determined by checking the index indicator in the DWT. Certain counters are cleared; a transfer to a subroutine on Figure 9-31 is made so as to set counter NIV (Number Indexed Values) equal to its value.

(2) (14E) The start of the first range in the Value Range Table (VRT) is set to one, since the entire range is currently to be considered. Counter DNIV1 (Divided Number Indexed Values) is set equal to the contents of NIV.

(3) (14F) The contents of DNIV1 are divided by eight (so as to secure eight ranges of values). DNIV2 is set equal to DNIV1 plus any remainder received from the division.

(4) (14Y) A transfer to a subroutine on Figure 9-32 is made. Here, the values of DNIV1, DNIV2, and the first entry in VRT will be used to form a display of eight ranges.

(5) (14A) An evaluation is made if a legitimate selection was made. If not, go to Step 6, (14X). Otherwise·

(a) If the last range was selected, DNIV1 is set equal to the contents of DNIV2. This will compensate for DNIV2 possibly being greater than DNIV1.

(b) If DNIV1 is now equal to 1, a value and not a range has been selected. Therefore, a request for a relation selection is made, and the value and relation are stored in the CVT. Control is passed to Step 4, (14Y), where the same display of values just issued is redisplayed.

(c) If DNIV1 is not equal to 1, the start of the selected range is placed in the first entry of VRT, and a transfer to Step 3, (14F) to break the chosen range into smaller chunks is performed.

(6) (14X) If NONE was not selected, a return to Step 2, (14E) is made so as to start the process again. If NONE was chosen, a check is made to see if the highest display is being made. If not, go to Step 2, (14E) to issue that display. If the highest display is being made and NONE was chosen, the Field Value Table (FVT) is closed for reading, and, if the attribute is fully indexed, a transfer to Figure 9-22, Step 6, (15C) is made as no key-in of values is necessary. If, however, the attribute is not fully indexed, a transfer to Figure 9-22, Step 2, (15B) is made for the key-in of additional values.

9.5.15 PHASE 2: TERMINAL SELECTION - II

This part of the terminal selection routine primarily deals with the case of the unindexed attribute (field), where a key-in of value(s) is required. It also performs a display of the total condition and evaluates the next step.

(1) (15A) A request is setup in the output buffer for the key-in of a value. A transfer to Step 3, (15X) is made.

(2) (15B) A determination is made if any value was chosen during the operation on Figure 9-21. If not, go to Step 1, (15A). Otherwise, a request is setup in the output buffer for the key-in of an additional value.

(3) (15X) All "R's" (record indicators) in the selected ICC are set to 1. This will obtain an IPC, which is prepared for the retrieve-item subroutine.

(4) (15E) After this operation, the obtained value is translated to external form and is placed in the output buffer, labeled as a sample.

(5) (15Y) A display is made and a determination of whether or not a key-in occurred is made. If not, go to Step 6, (15C). Otherwise, a request for a relation selection is made, and the keyed-in value with the relation selection is stored in the CVT. A return to Step 5, (15Y) is made so as to request further key-ins.

(6) (15C) A request for a further restrictive condition is made. If narrowing is not desired, go to Step 7, (15F). Otherwise, a preparation for a conjunctive (Boolean "AND" or narrow condition) term is made, and a transfer to Figure 9-18, Step 2, (11A) is made.

(7) (15F) If broadening is not required as the start of a further restrictive condition, a transfer to Figure 9-23, Step 1, (16A) is made.

(8) (15D) Otherwise, a preparation for a disjunctive (Boolean "OR" or broaden condition) term is made. The first bit (bit 0) in the OCW is set so as to indicate this "OR" condition, and the WL is restored to its status at the start of Phase 2 by clearing all Phase 2 selection bits and by removing all items not selected in Phase 1. A return to Figure 9-17, Step 2, (10E) is made to restart Phase 2. The only thing saved from the prior Phase 2 pass is the CVT.

9.5.16 Phase 2: Termination - I

This part of the termination routine deals with the determination by the user if he is satisfied with the evolved condition. The routine is entered from Figure 9-22, when no further narrowing or broadening is indicated by the user.

(1) (16A) A display of the total condition is made. If there were no Phase 2 selections, a message to that effect is indicated. The user is requested to respond as to whether the condition is satisfactory, the condition is to be narrowed, or whether the condition is to be broadened.

(a) If the condition is satisfactory, a transfer to Figure 9-22, Step 1, (17A) is made so as to allow the user to select the final termination selection.

(b) If the condition is to be narrowed, a check is made to see if any Phase 2 selections were made. If not, a return to Figure 9-17, Step 2, (10E) is made. Actually, this will restart Phase 2. If Phase 2 selections were made, a transfer to Step 2, (16B) is made so that a determination of which term is to be narrowed is made.

(c) If the condition is to be broadened, a transfer to Figure 9-22, Step 8, (15D) is made. This will set up the proper steps for a disjunctive term.

(2) (16B) The total condition is displayed, with the first conjunctive term underlined.

(3) (16X) If the user does not find this term satisfactory, a preparation for a conjunctive term is made, and a return to Figure 9-17, Step 2, (10E) is made.

(4) (16Y) Otherwise, a check is made to see if the last conjunctive term is now underlined. If not, the total condition is again displayed, with the next conjunctive term now being underlined, and a transfer to Step 3, (16X) is made. If the last term was presently underlined, a return to Step 1, (16A) for further display evaluation is made.

9.5.17 Phase 2: Termination - II

This part of the termination routine determines what path the user wants to follow with the selections he made with the help of the Dialogue query.

(17A) The next step is requested. Available steps are dependent on what selections the user made. As shown in Table 9-2, if the user has only made Phase 1 selections, he may select those items as indicated by "X" in column 2. If the user has only selected Phase 2 items, he may select those items as indicated by "X" in column 3. If he chose Phase 1 and Phase 2 selections, he can choose any step as shown in column 4.

9.5.18 Retrieve Node IL Entry (Subroutine)

This subroutine retrieves the item list (IL) entry of the parent node ICC of the DWT, stores the record number and other information in the DWT, and puts the node size in the control word (CW).

TABLE 9-2. PHASE 2: TERMINATION

Step	Phase 1 Selections, No Phase 2 Selections	Phase 2 Selections, No Phase 1 Selections	Phase 1 and Phase 2 Selections	Action	Close For Reading Term List	Transfer To:
Terminate	X	X	X	Terminate No Action	Yes	End Job
Type Copy		X	X	Type Hard Copy	Yes	End Job
Store Condition		X	X	Store Condition	Yes	End Job
Store Query	X		X	Store Query	Yes	End Job
Develop Display	X		X	Call Query Routine	Yes	End Job
Store Developed Item	X		X	Call Conditional Reformat Routine	Yes	End Job
Restart Phase 2	X	X	X	Erase Phase 2 Selections From WL and CVT	No	Figure 9-17, Step 1, (10H)
Restart Phase 1	X	X	X	Erase All Selections From WL and CVT	No	Figure 9-8, Step 1, (1H)

ENTRANCES (from:)		EXITS (to:)
1.	Figure 9-8 (18A)	Figure 9-8 (1D)
2.	Figure 9-17 (18A1)	Figure 9-17 (10D)

(18A) The node ICC is prepared for the retrieval of its IL entry. A call on the retrieve IL entry subroutine yields the following desired information: the record number, type, and index indicator, which are placed in the DWT; the size of the node, which is placed in the CW. A return is now made.

9.5.19 Create ICC's and Obtain IL Entries (Subroutine)

This subroutine uses the contents of the DWT to create the next set of ICC's, and it then retrieves the IL entries for these ICC's.

ENTRANCES (from:)		EXITS (to:)
1.	Figure 9-8 (19A)	Figure 9-8 (1G)
2.	Figure 9-17 (19A1)	Figure 9-17 (10G)

(1) (19A) A switch is either cleared (Phase 1 entry) or set (Phase 2 entry), so as to later be able to determine which phase is in control.

(2) (19Y) Using the ICC of the node level in the DWT, and the value of the DWT node pointer, a new ICC is created by taking the value of the node pointer plus the location (1-8) within the DWT table which is now being processed to the ICC. A check is made to see if the segment of the item list (IL) containing the record number of this ICC is in memory. If so, all that is required is a locate of the IL entry. Otherwise, a somewhat lengthier operation, namely, retrieve IL entry, takes place. After either operation, the desired information about the ICC is in memory.

(3) (19B) The record number, type, and index indicator are placed in the appropriate slot of the DWT. The node size in the CW is decremented by one. If this was a Phase 2 entry (switch initially set), the Field Value Table (FVT) record number and index indicator are placed in the DWT1 in the same relative position as the last entry placed in the DWT. The node size in the CW is examined. If equal to zero, all items have been examined. An exit is then performed. Otherwise, a check is made to see if

all eight entries of the DWT have been processed. If not, a return to Step 2, (19Y) is made so as to process the next DWT entry. Otherwise, an exit is performed.

9.5.20 Setup Output Buffer - I (Subroutine)

This portion of the subroutine obtains the term names for all entries of the DWT, inserts symbols, and sets up an output buffer. It operates in conjunction with Figure 9-28.

ENTRANCES (Figure 9-27; from:)		EXITS (Figure 9-28; to:)
1.	Figure 9-10 (20A)	Figure 9-10 (3E)
2.	Figure 9-11 (20A2)	Figure 9-11 (4C)
3.	Figure 9-13 (20A3)	Figure 9-13 (6M)
4.	Figure 9-19 (20A1)	Figure 9-19 (12E)

(1) (20A) If this subroutine is entered from the Homograph Dialogue III (Figure 9-13), switch D is set. Otherwise, switches C and D are cleared. If the term list was never opened for reading during this job, it is now opened. The record number of the parent node of the DWT is prepared for a term list seek and read. This will allow the term name to be moved to the output buffer. Symbols to indicate the type of the term, for example file, statement, or field, are inserted around the term name. This line will act as the parent of the subsumed items to be displayed.

(2) (20E) The next subsumed record number in the DWT is prepared for a TL seek and read. The term name and appropriate symbols are placed in the output buffer. An analysis is made to determine if the record number for which the term name was just read is in the WL or CVT. If this item was not chosen previously, go to Step 3, (20Y). If an item was previously selected, an indication is placed in the output buffer so that the user will know that he has previously chosen this subsumed item in Phase 1, Phase 2, or Both Phases. Switch C is set so that further processing will know that some item on the screen display has previously been selected.

(3) (20Y) If there are more record numbers in the DWT to be analyzed, a transfer to Step 2, (20E) is made for further processing. Otherwise, the contents of the DWT has been setup for a screen display, and a transfer to Figure 9-28, Step 1, (21A) is made.

9.5.21 Setup Output Buffer - II (Subroutine)

This portion of the subroutine will add the "ADDITIONAL" and "PRIOR" selections to the screen setup, if required, and will add an appropriate message to the output buffer.

(1) (21A) The prior and additional bits in the CW are examined. If either is set, the appropriate options of additional or prior are added to the screen display. At this point, an analysis is made to determine if the homograph bit in the CW is set. If so, transfer to Step 2, (21X). Otherwise, a check is made to see if switch C is set. If not, a message requesting a selection is placed in the output buffer. If switch C is set, a message requesting the selection of an additional item is made. After either case, an exit is made.

(2) (21X) An analysis is made to determine whether or not switch D is set. If not, a message requesting selection of an item for the homograph option is made. If switch D is set, a message requesting a homograph item selection is made. After either case, an exit is made.

9.5.22 Display (Delete) Selected Items - I (Subroutine)

This subroutine creates a screen display of all Phase 1 items, and, if the user selected the delete option, the subroutine will permit item(s) to be deleted.

DISPLAY		
ENTRANCE (from:)		EXIT (to:)
1. 2.	Figure 9-16 (22A) Figure 9-20 (22A1)	Figure 9-16 (9D) Figure 9-20 (13D)

DELETE		
ENTRANCE (from:)		EXIT (to:)
1.	Figure 9-15 (22D)	Figure 9-15 (8B)

(1) (22A) This entrance to the subroutine is for the display of all Phase 1 selection entries. A determination is made whether or not there were any Phase 1 choices. If not, a display is made stating "NO PHASE 1 SELECTIONS," and an exit is made. Otherwise, switch D is cleared so as to indicate a display entrance. Transfer is made to Step 3, (22F).

(2) (22D) In order to indicate a delete entrance, switch D is set. It should be noted that this portion of the routine would not be entered if there were no Phase 1 selections.

(3) (22F) A counter used to step through the WL is set to two (so as to bypass the first two words of the WL).

(4) (22G) Using the counter and the address of the WL, the next Phase 1 record number is obtained. The counter is incremented by two for each entry. A seek-and-read of the term list (TL) is performed on the record number of the Phase 1 selection. The name is shifted to its proper hierarchical output buffer slot, and the counter is incremented by two. A check is made to see if eight output items have been processed. If so, a transfer to Step 6, (22C) is made so as to process the output buffer for a display.

(5) (22E) A check is made to see if there are more Phase 1 selections. If so, transfer to Step 4, (22G) to process the next Phase 1 record number.

(6) (22C) If additional or prior Phase 1 items exist, the appropriate option is added to the output buffer. If switch D is set (delete entrance) transfer to Figure 9-30, Step 3, (23B). Otherwise (display entrance) transfer to Figure 9-30, Step 1, (23A).

9.5.23 Display (Delete) Selected Items - II (Subroutine)

This part of the subroutine will set up a display, and will allow the user either to run through a display of the items or a delete of selected items, depending upon the initial entrance.

(1) (23A) This portion of the program provides for the display of all Phase 1 selections. An appropriate message is placed in the output buffer.

(2) (23X) A display is made. Three responses are available.

(a) If CONTINUE is selected, an exit is made;

(b) If "PRIOR" is chosen, go to Step 6, (23E);

(c) If "ADDITIONAL" is picked, go to Step 5, (23F);

(d) If none of the above, return to Step 2, (23X).

(3) (23B) This portion of the program provides for the deletion of appropriate Phase 1 selections. A message is placed in the output buffer.

(4) (23Y) A display is made. Four responses are available.

(a) If NONE is selected, an exit is made;

(b) If an item is selected, go to Step 7, (23H);

(c) If PRIOR is chosen, go to Step 6, (23E);

(d) If ADDITIONAL is picked, go to Step 5, (23F);

(e) If neither, go to Step 4, (23Y).

(5) (23F) The output area is cleared, and a transfer to Figure 9-29, Step 4, (22G) is performed so as to setup another display.

(6) (23E) Sixteen plus two times the number of currently displayed record numbers is subtracted from the counter. This will allow for the redisplay of the prior eight items. A transfer to Figure 9-29, Step 4, (22G) is performed. so that the prior items will be redisplayed.

(7) (23H) A determination of which item was selected for deletion is made. All Phase 1 elements in the WL, starting with the current selection, are deleted until an entry with the same or higher level is encountered. The WL is repacked as the deletion is taking place. A transfer to Figure 9-29, Step 3, (22F) is made so that the user may doublecheck if he wishes to delete any other items.

9.5.24 Set NIV = Number Indexed Values (Subroutine)

This subroutine determines the number of indexed values of an attribute by examining the count of the number of entries in a particular Field Value Table (FVT) record number.

ENTRANCE (from:)		EXIT (to:)
1.	Figure 9-21 (24A)	Figure 9-21 (14E)

(1) (24A) Using the correct FVT record number from the DWT1, the appropriate portion of the FVT is opened for reading. Using a seek EOF, the number of indexed values is retrieved. This number is placed in cell NIV. The VRT table is cleared, and the first entry is set to one, so as to indicate the start of range. An exit is made.

9.5.25 Display Eight Ranges - I (Subroutine)

This subroutine will create a display of values for an indexed attribute, after certain values, i.e., NIV, DNIV1, and DNIV2 are calculated (Figure 9-21) and after the start of the first range is placed in the VRT. This section takes care of the case where there are eight or more values.

ENTRANCE (from:)		EXIT (to:)
1	Figure 9-21 (25A)	Figure 9-21 (21A)

(1) (25A) A check is made to see if DNIV1 is equal to zero. This would indicate there is only one value or range. If so, a transfer to Figure 9-34, Step 1, (27A) is made. Otherwise, counter F is set to one so as to indicate the first output line.

(2) (25X) The VRT is filled by adding successive powers of DNIV1 to previous entries in the VRT. This is performed seven times. Then DNIV2 is added to the eighth entry to obtain the ninth entry.

(3) (25B) A determination is made if DNIV1 is equal to one. If so, part (or all) of the display will be values as opposed to ranges. If true, go to Figure 9-33, Step 1, (26A). Otherwise, counter F is set to one, and the first record number in the VRT is prepared for a FVT seek.

(4) (25C) The FVT record number is read and the value obtained is stored.

(5) (25D) The end-of-range record number is prepared for a FVT seek. A seek and a read are performed. The output line (range) number as per counter F is prepared. A check is made to determine if F is equal to eight. If so, go to Step 6, (25E). Otherwise, F is incremented by one, and a transfer is made to Step 4, (25C).

(6) (25E) A message is inserted in the output area, and eight ranges are displayed. A return is then made.

9.5.26 Display Eight Ranges - II (Subroutine)

This section deals with the case where there are from eight to fifteen values indexed in the FVT.

(1) (26A) Counter F is set to one.

(2) (26X) The record number of the next value in the VRT is prepared for a seek. A seek-and-read of the FVT record are performed. The output line is prepared as per the value of output line indicator F.

(3) (26B) A test is made for F being greater or equal to seven. If not, transfer to Step 2, (26Y). Otherwise a test is made to see if DNIV2 is equal to one. If so, there is no final range, and we go to Step 5, (26W). Otherwise, the record number in the eighth word of VRT is prepared for a FVT seek-and-read. The value of the FVT is obtained, and preparation of the ninth record number for a seek-and-read is made. The output range in line eight of the buffer is setup, and a transfer to Figure 9-32, Step 6, (25E) is made.

(4) (26Y) F is incremented by one, and a transfer to Step 2, (26X) is made.

(5) (26W) A test is made to determine whether or not F is equal to eight. If it is, go to Figure 9-32, Step 6, (25E). Otherwise, transfer to Step 4, (26Y).

9.5.27 Display Eight Ranges - III (Subroutine)

This section deals with the case where there are less than eight indexed values. Therefore, there will be just one display of values.

(1) (27A) F is set to 1. The first record number in the VRT is prepared for a FVT seek.

(2) (27X) A FVT read is executed.

(3) (27B) The output line specified by F is set up in the output buffer. A check is made to see if F is equal to DNIV2. If so, transfer to Figure 9-32, Step 6, (25E). Otherwise, the next entry in the VRT is set equal to the first entry plus F. F is incremented by one, and a transfer to Step 2, (27X) is performed.

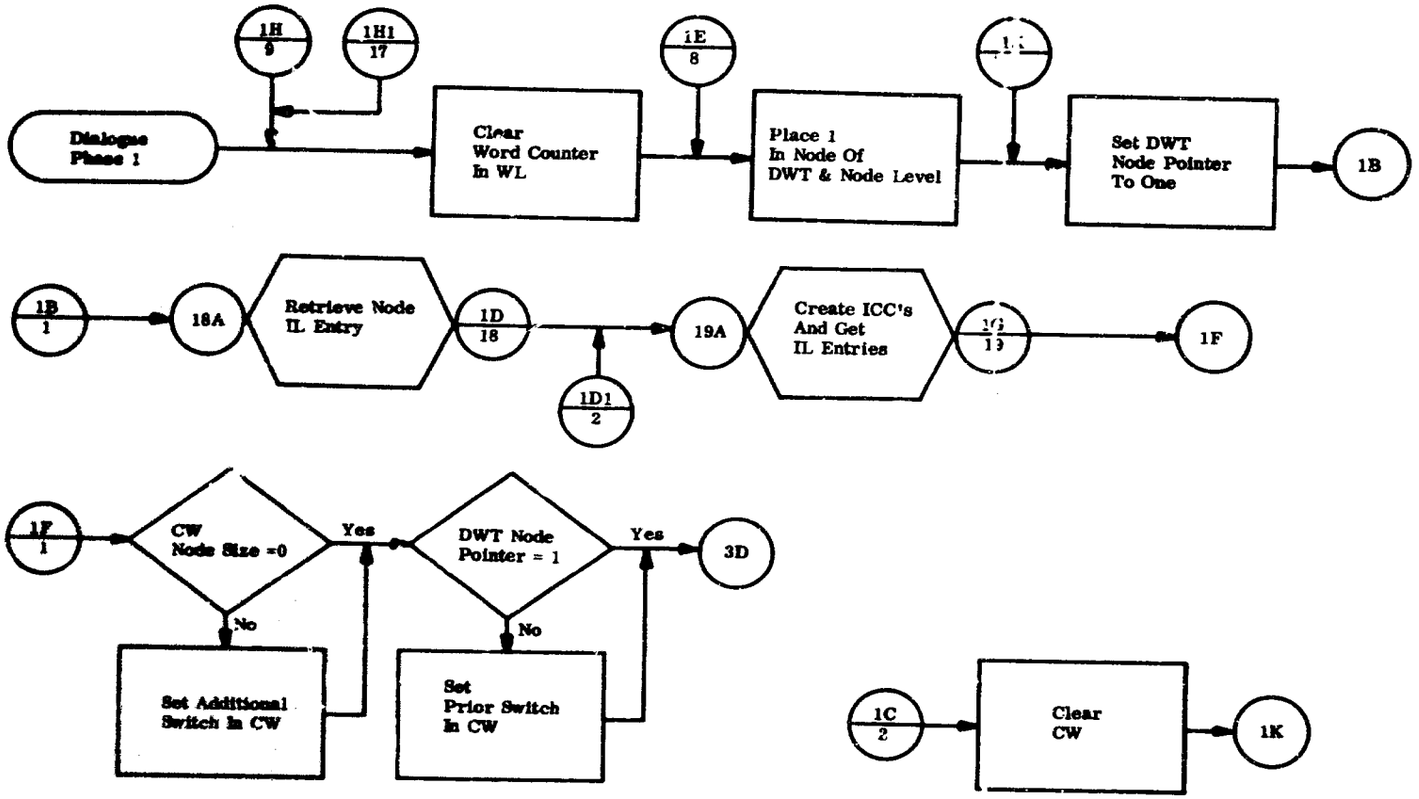


Figure 9-8. Phase 1: Initialization and DWT Setup

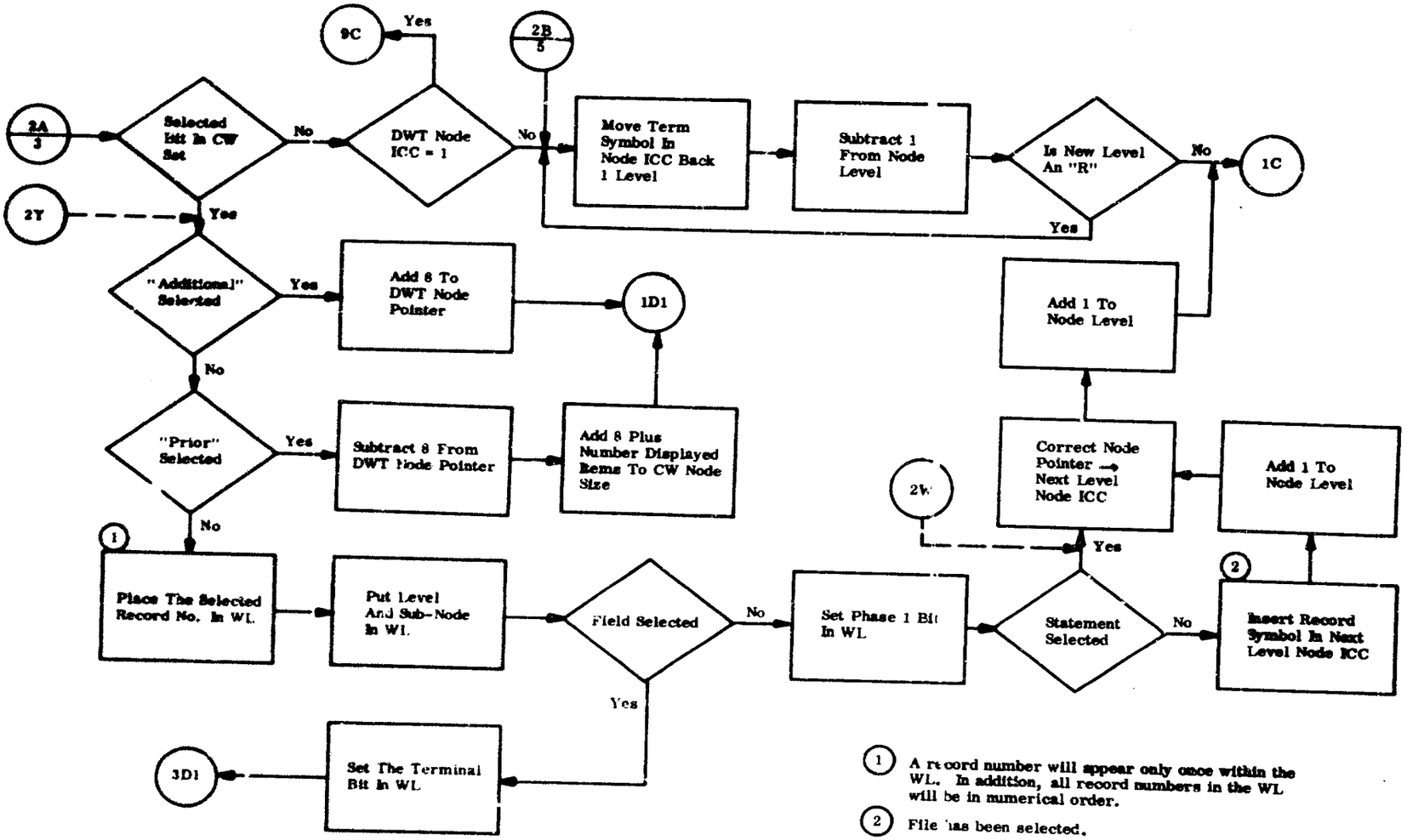


Figure 9-9. Phase 1: Selection Diagnosis

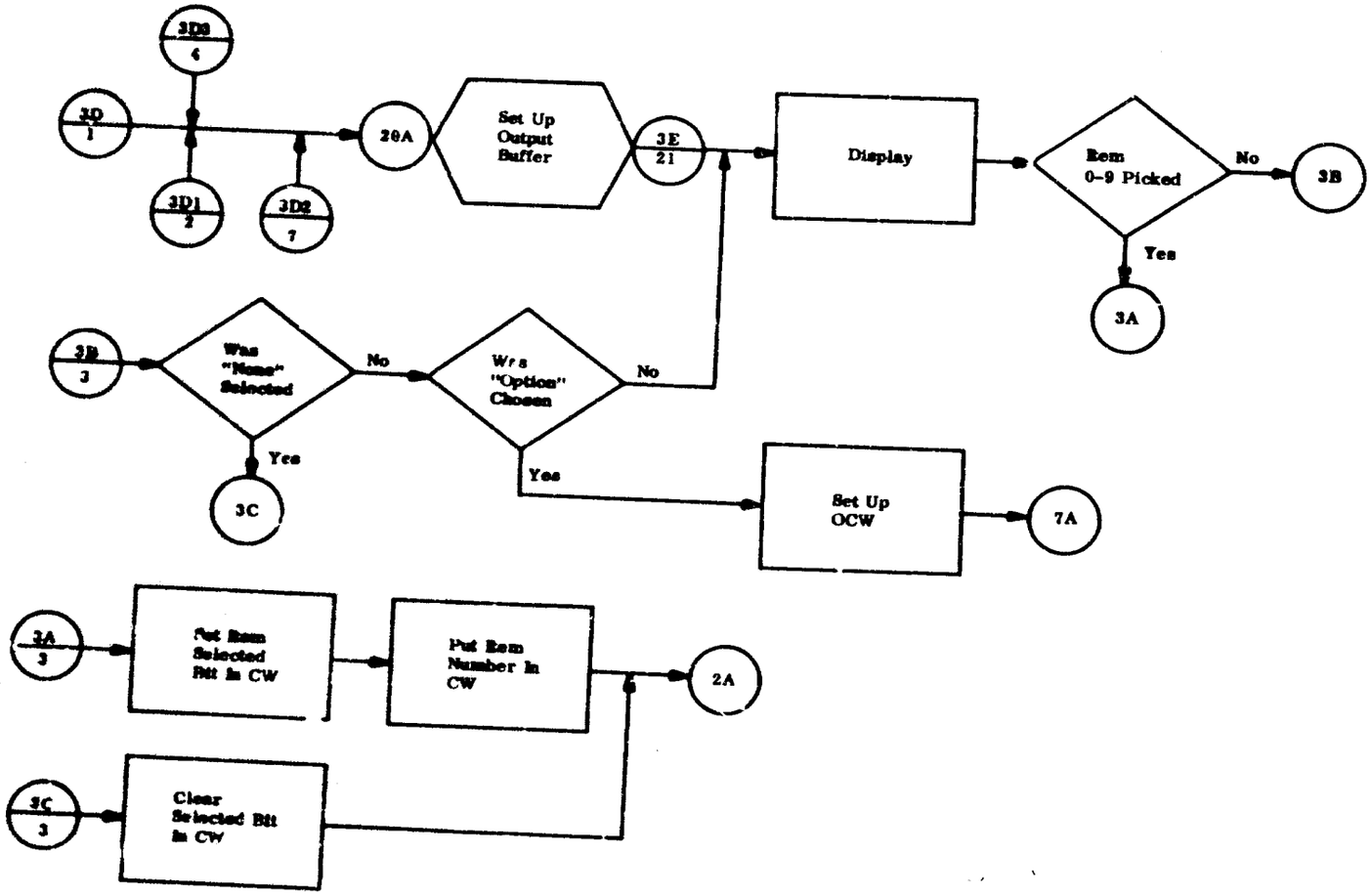
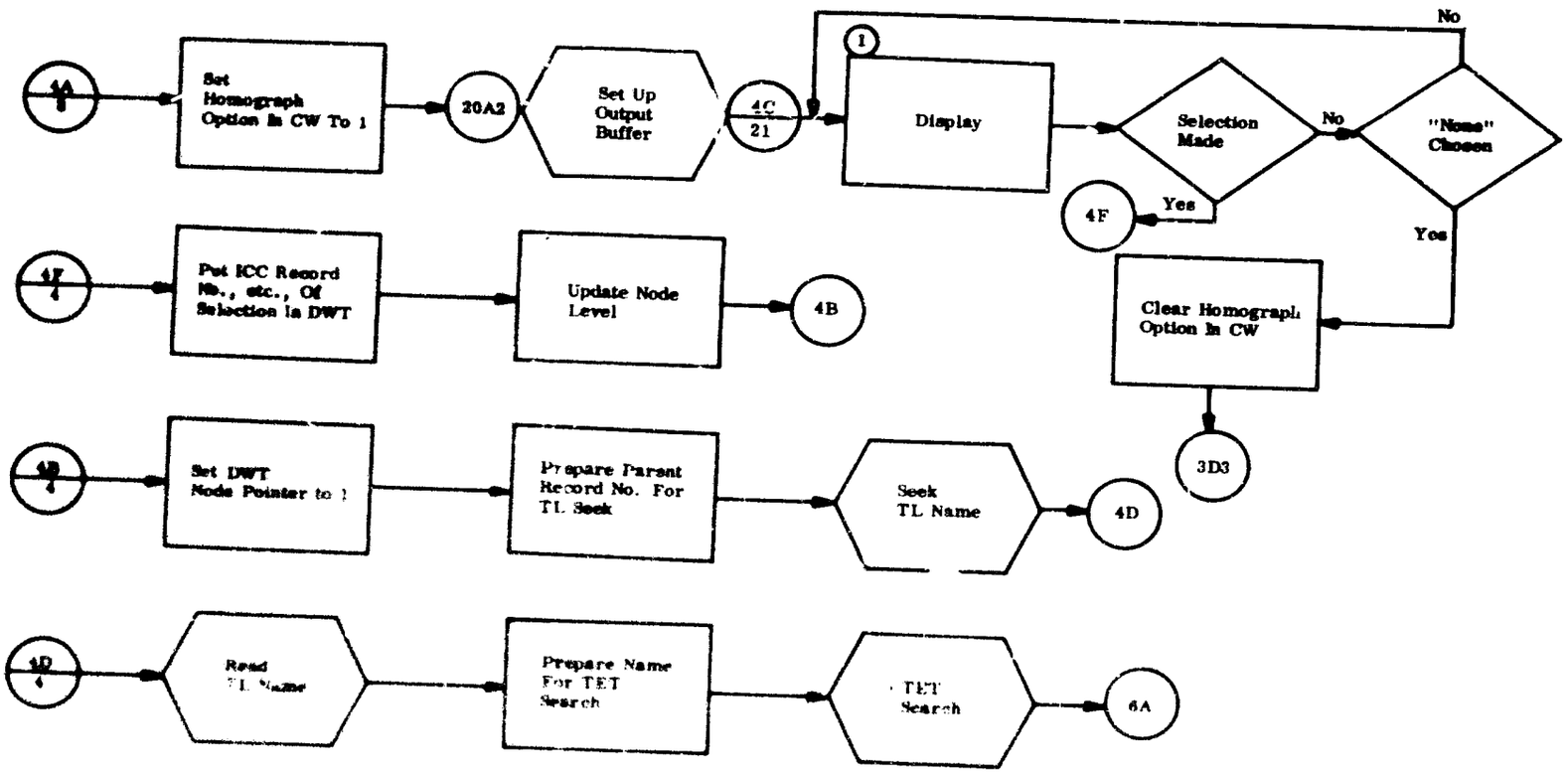


Figure 9-10. Phase 1: Console Display


4A
Set Homograph Option In CW To 1
20A2
Set Up Output Buffer
4C
21
I
Display
Selection Made
No
"None" Chosen
No
Yes
4F
Yes
Clear Homograph Option In CW
3D3
4F
Put ICC Record No., etc., Of Selection In DWT
Update Node Level
4B
4B
Set DWT Node Pointer to 1
Prepare Parent Record No. For TL Seek
Seek TL Name
4D
4D
Read TL Name
Prepare Name For TET Search
TET Search
6A

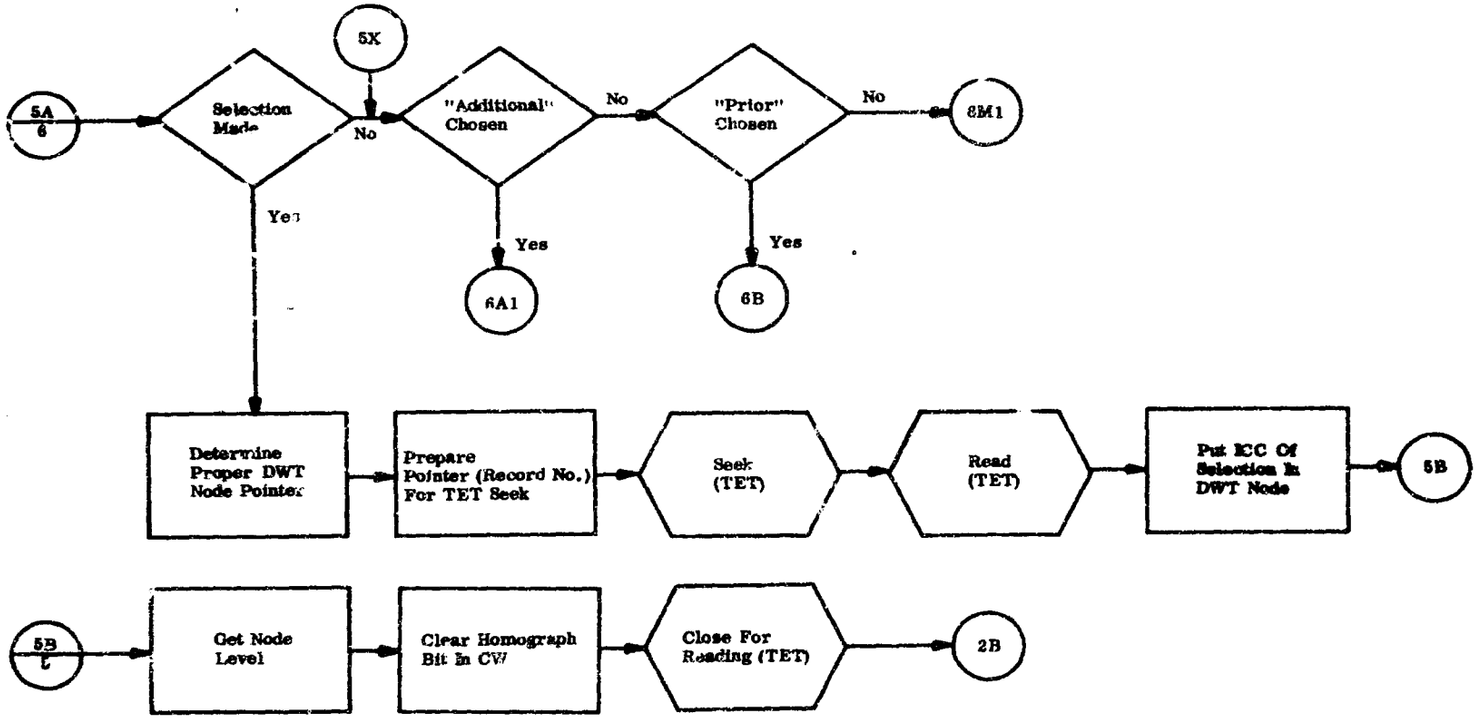


Figure 9-12. Phase 1: Homograph Dialogue - II

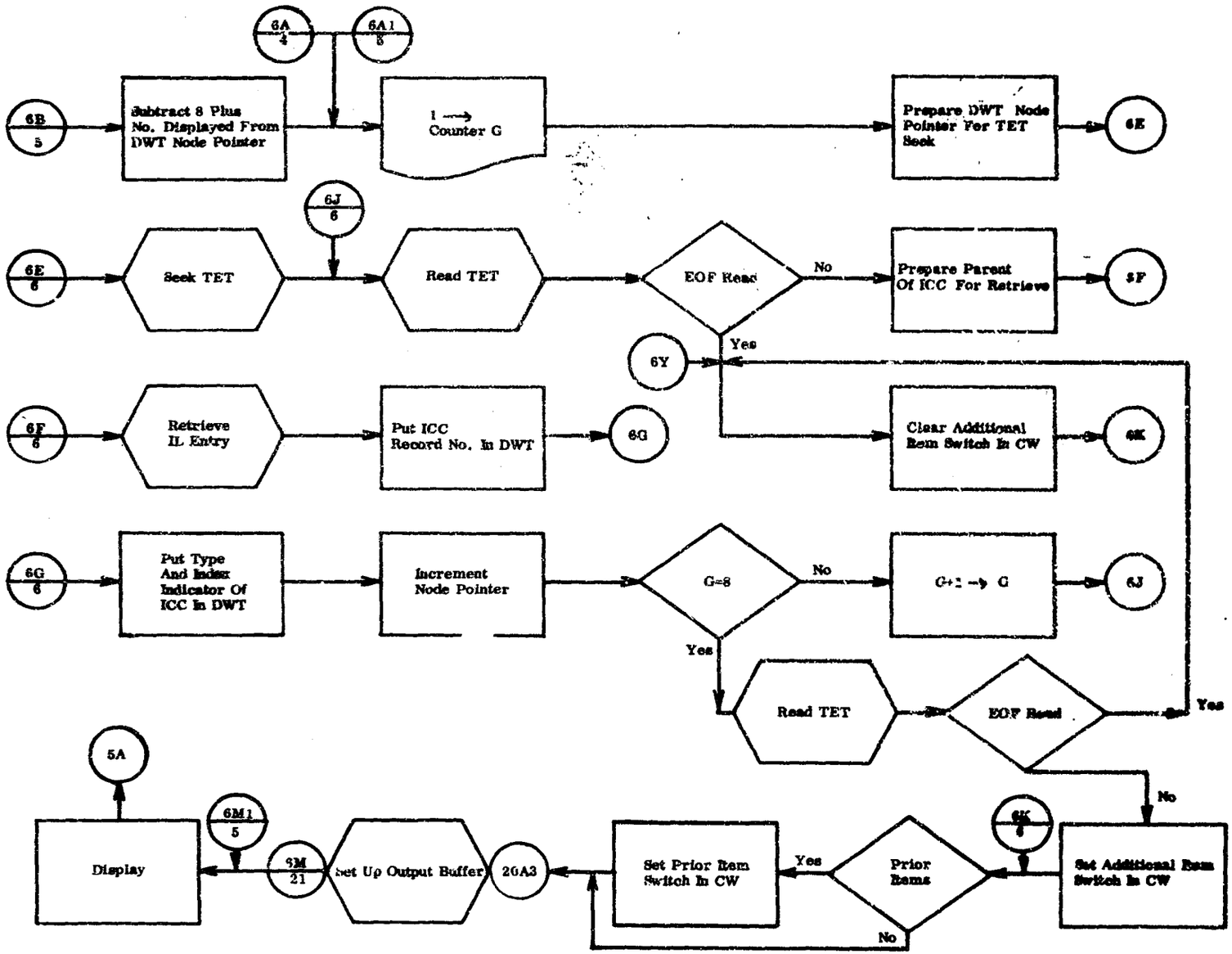


Figure 9-13. Phase 1: Homograph Dialogue - III

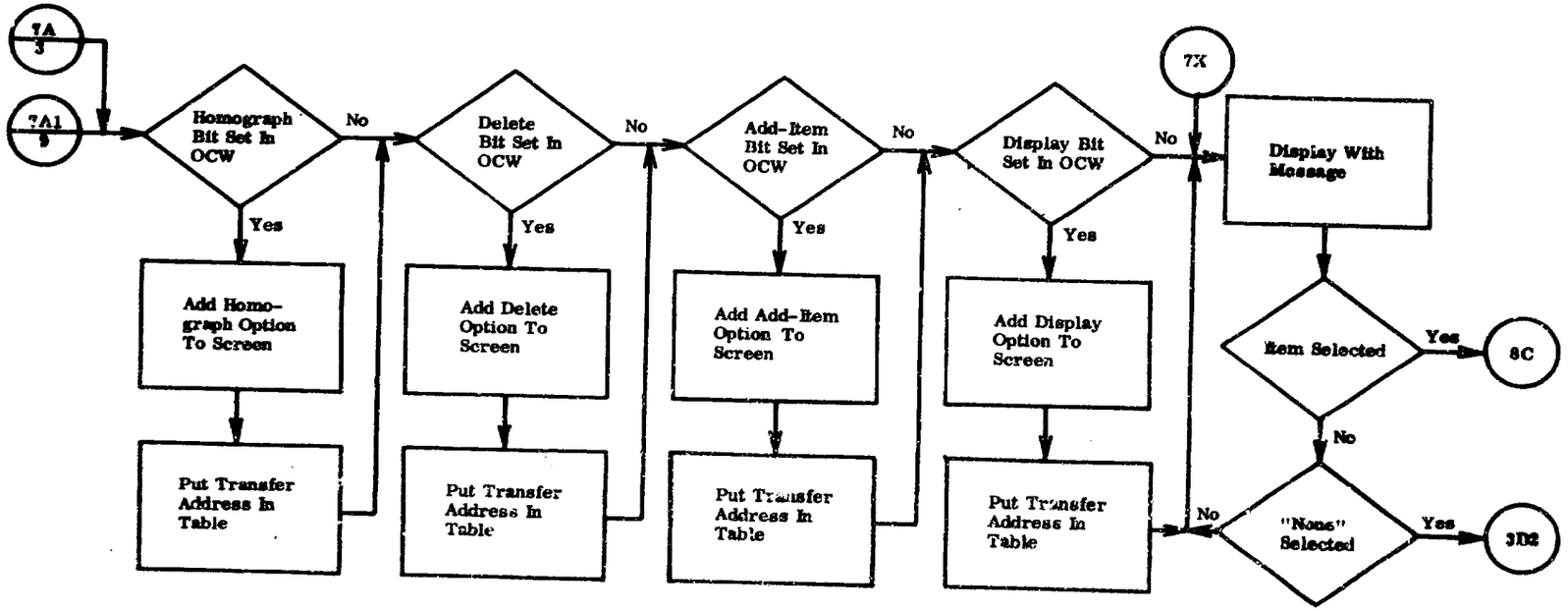


Figure 9-14. Phase 1: Option

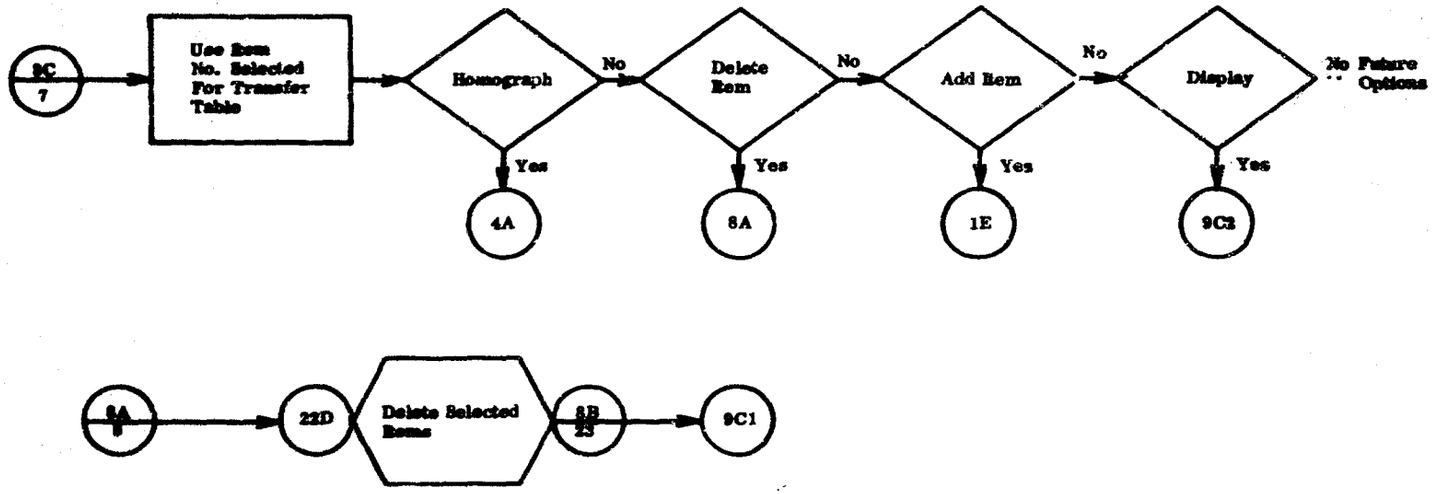


Figure 9-15. Phase 1: Transfer or Delete

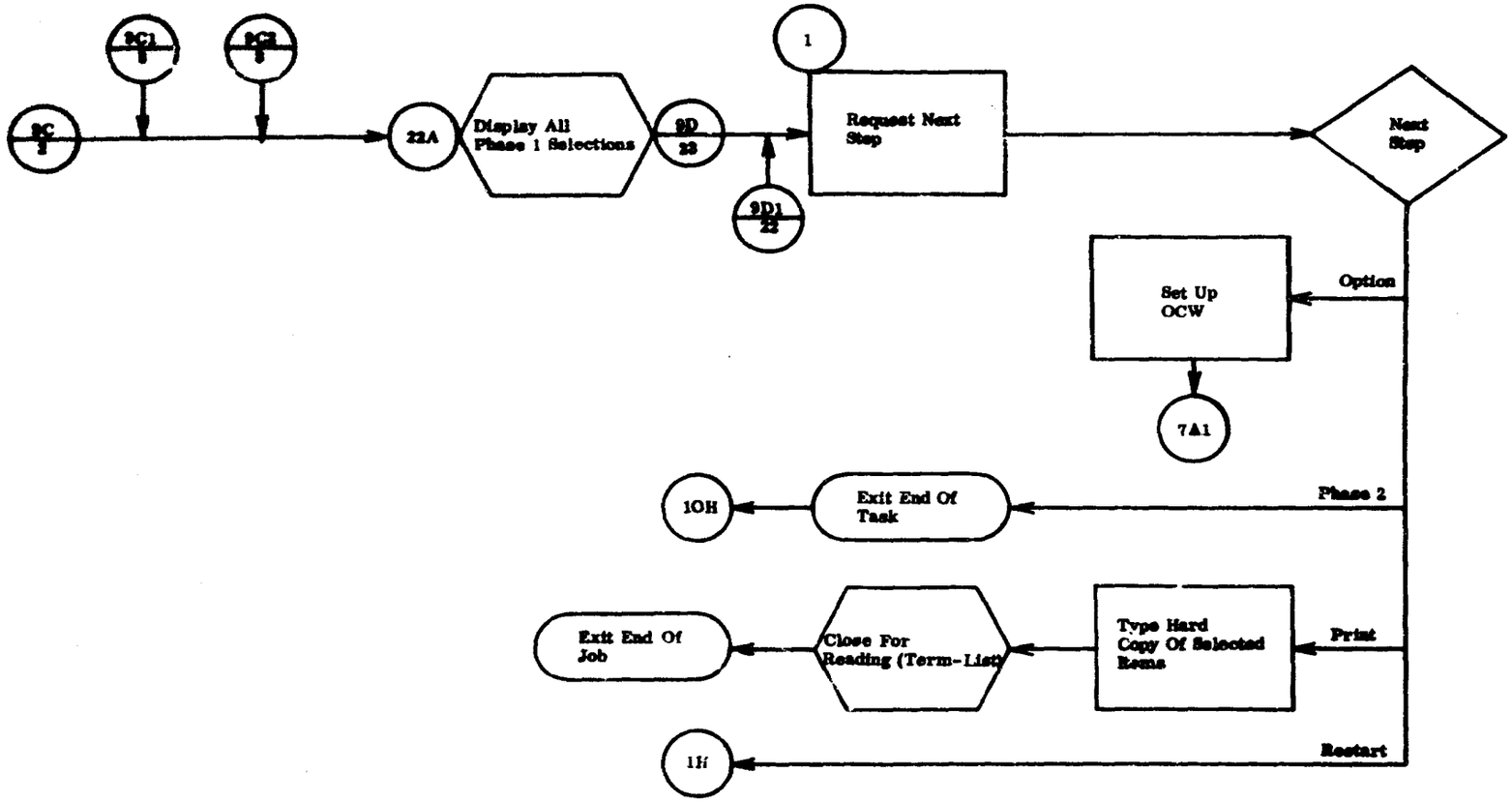


Figure 9-16. Phase 1: Termination

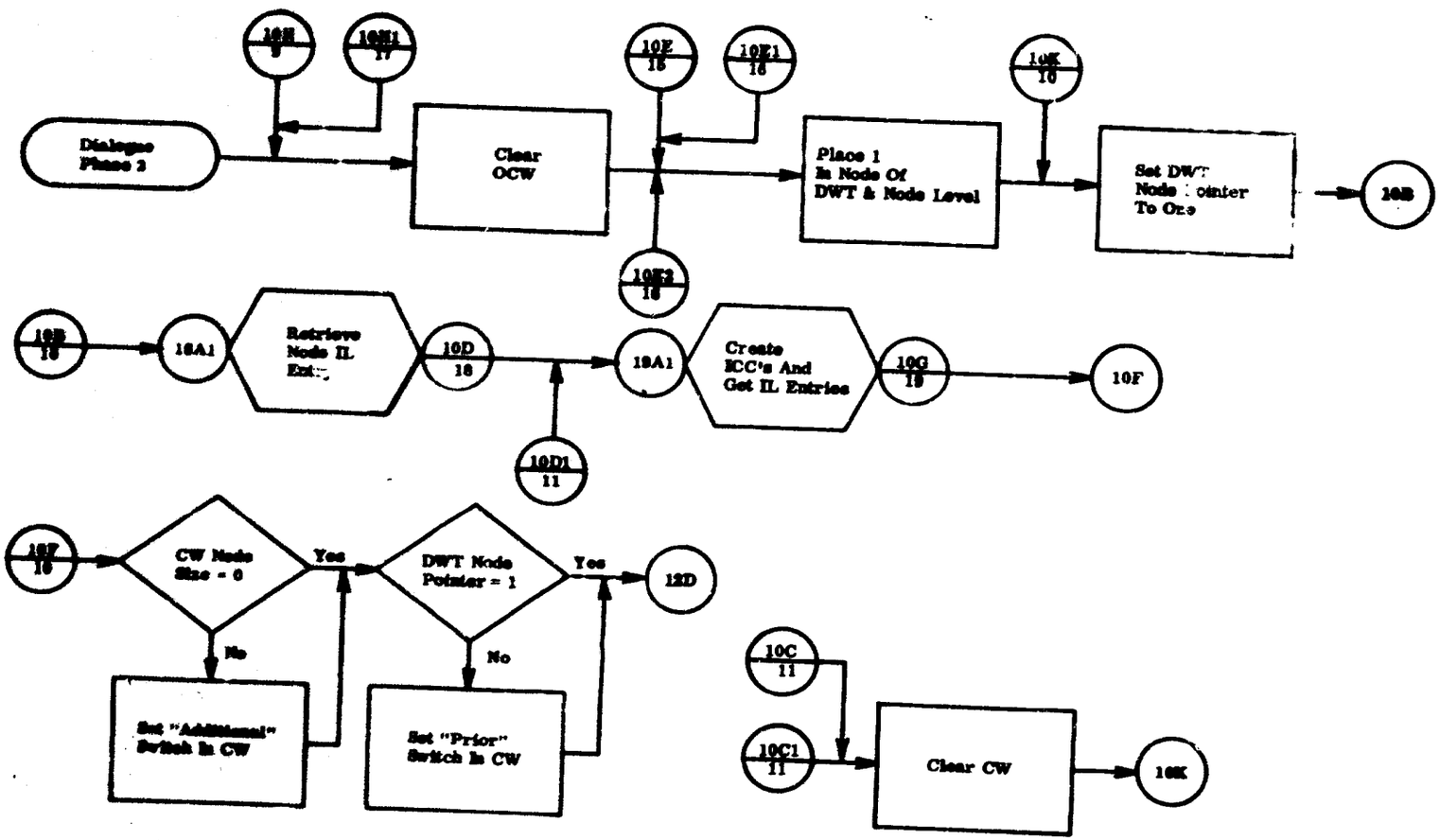


Figure 9-17. Phase 2: Initialization and DWT Setup

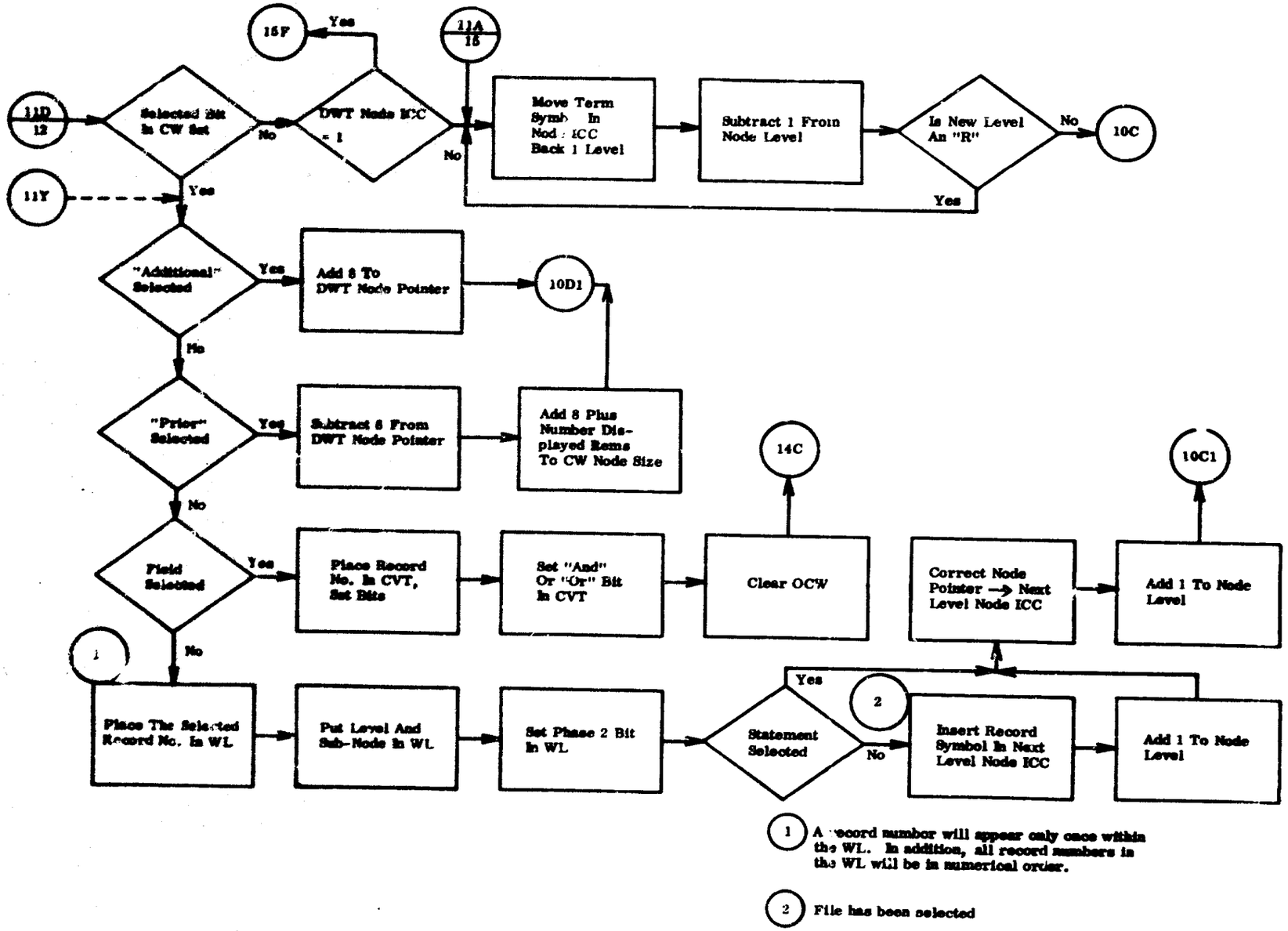


Figure 9-18. Phase 2: Selection Diagnosis

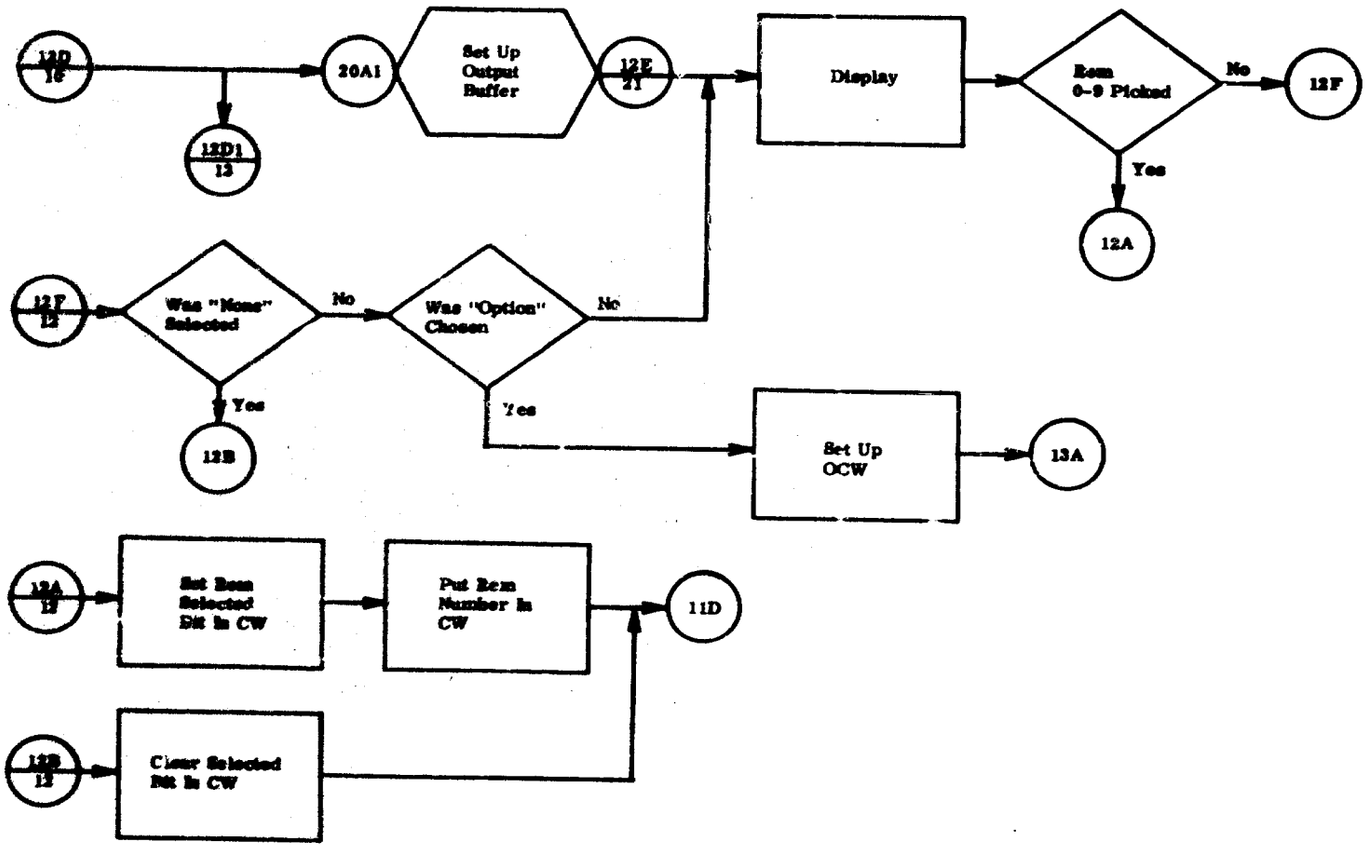


Figure 9-19. Phase 2: Console Display

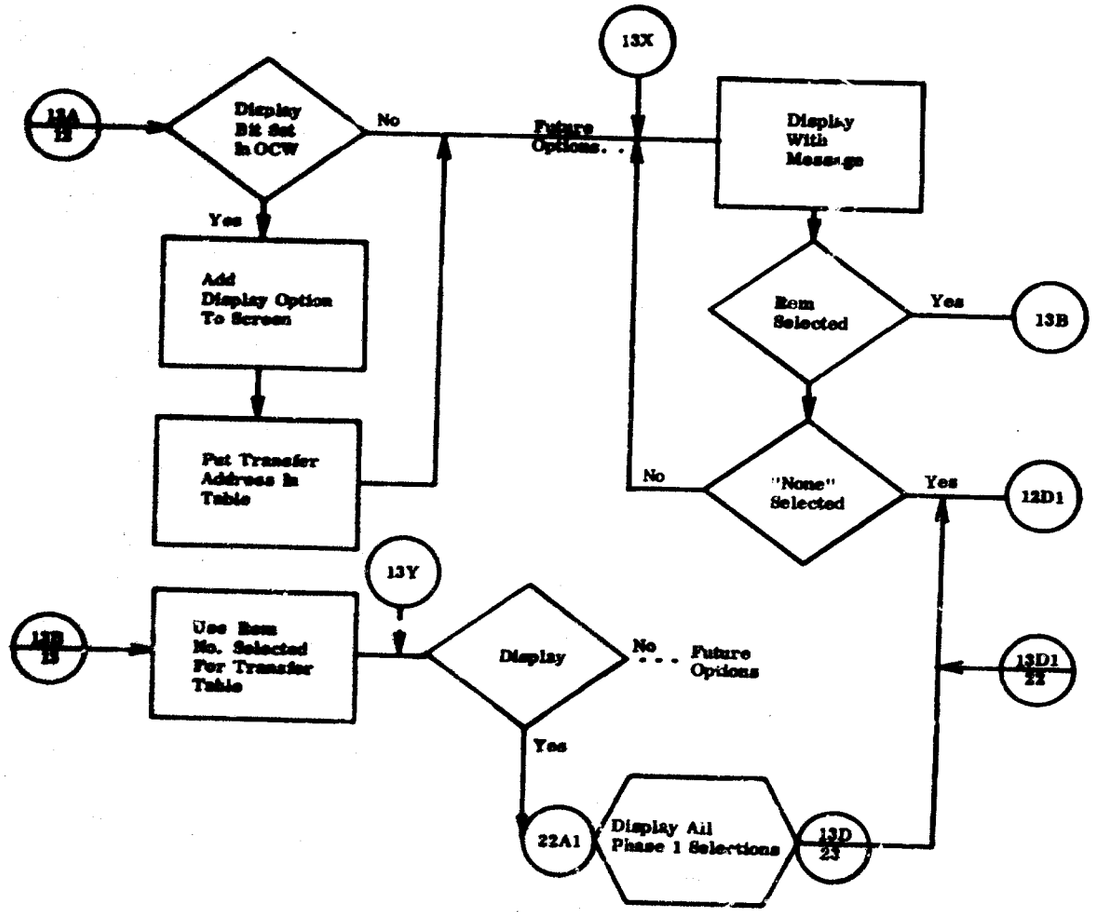


Figure 9-20. Phase 2: Option

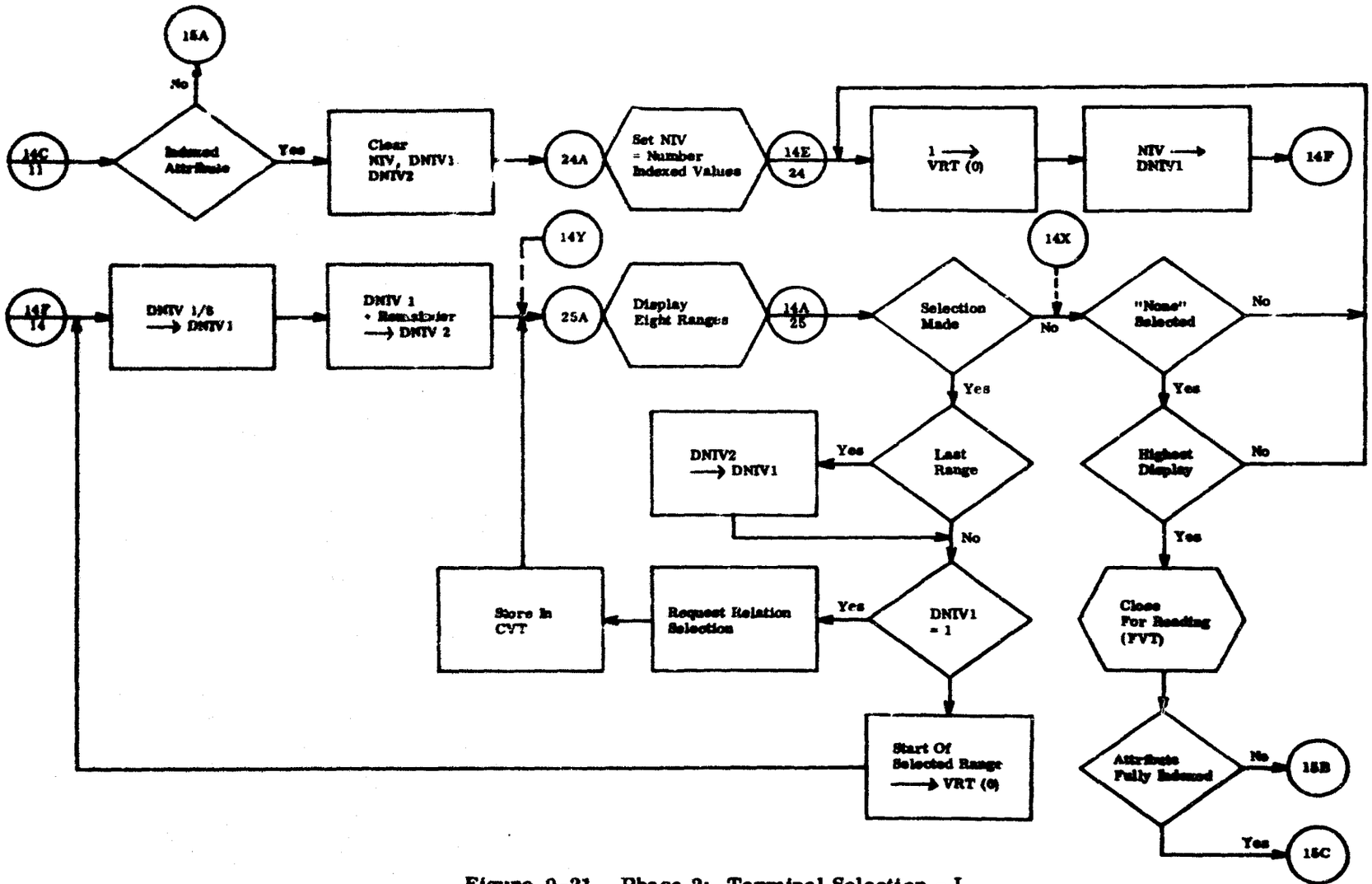


Figure 9-21. Phase 2: Terminal Selection - I

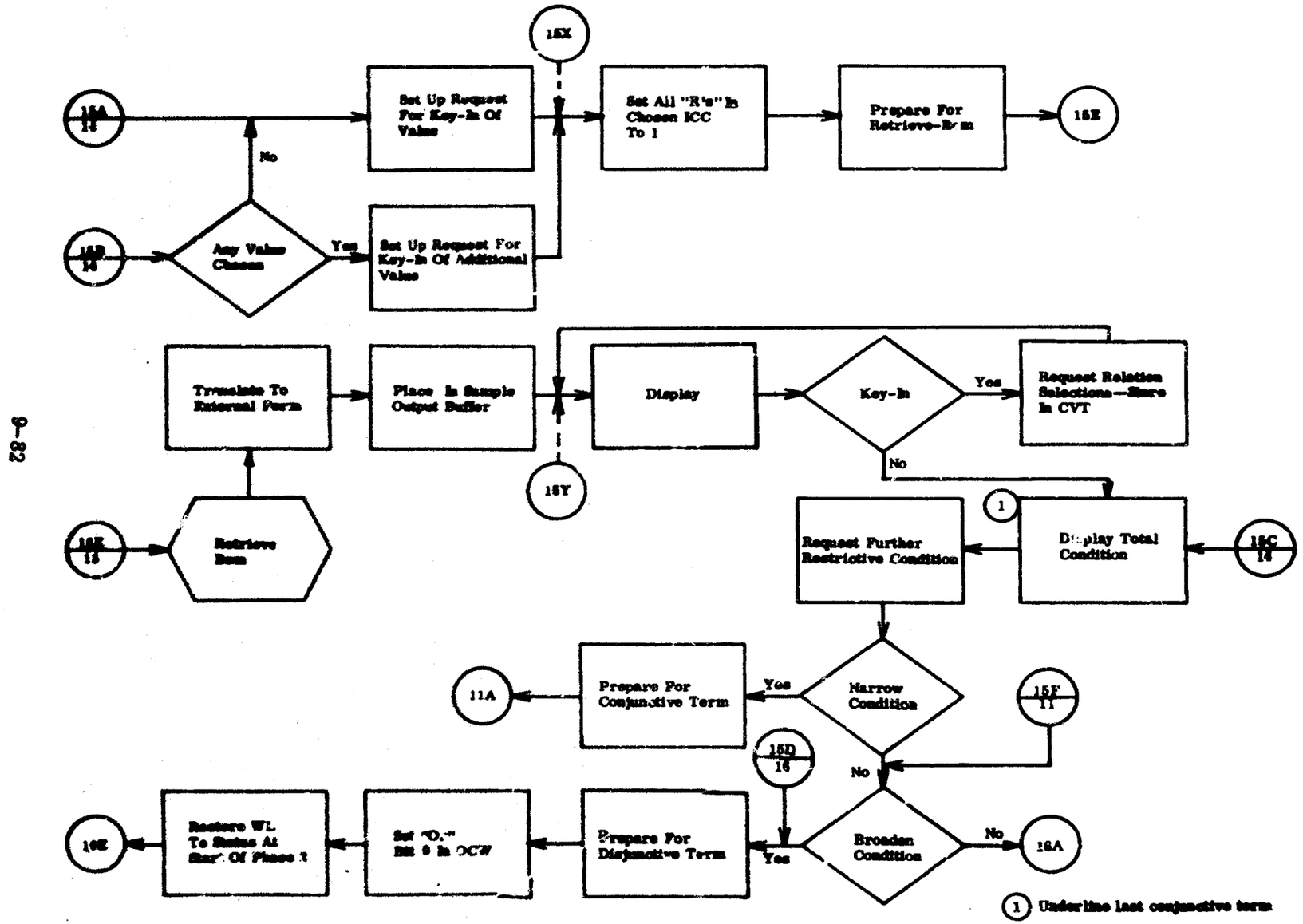
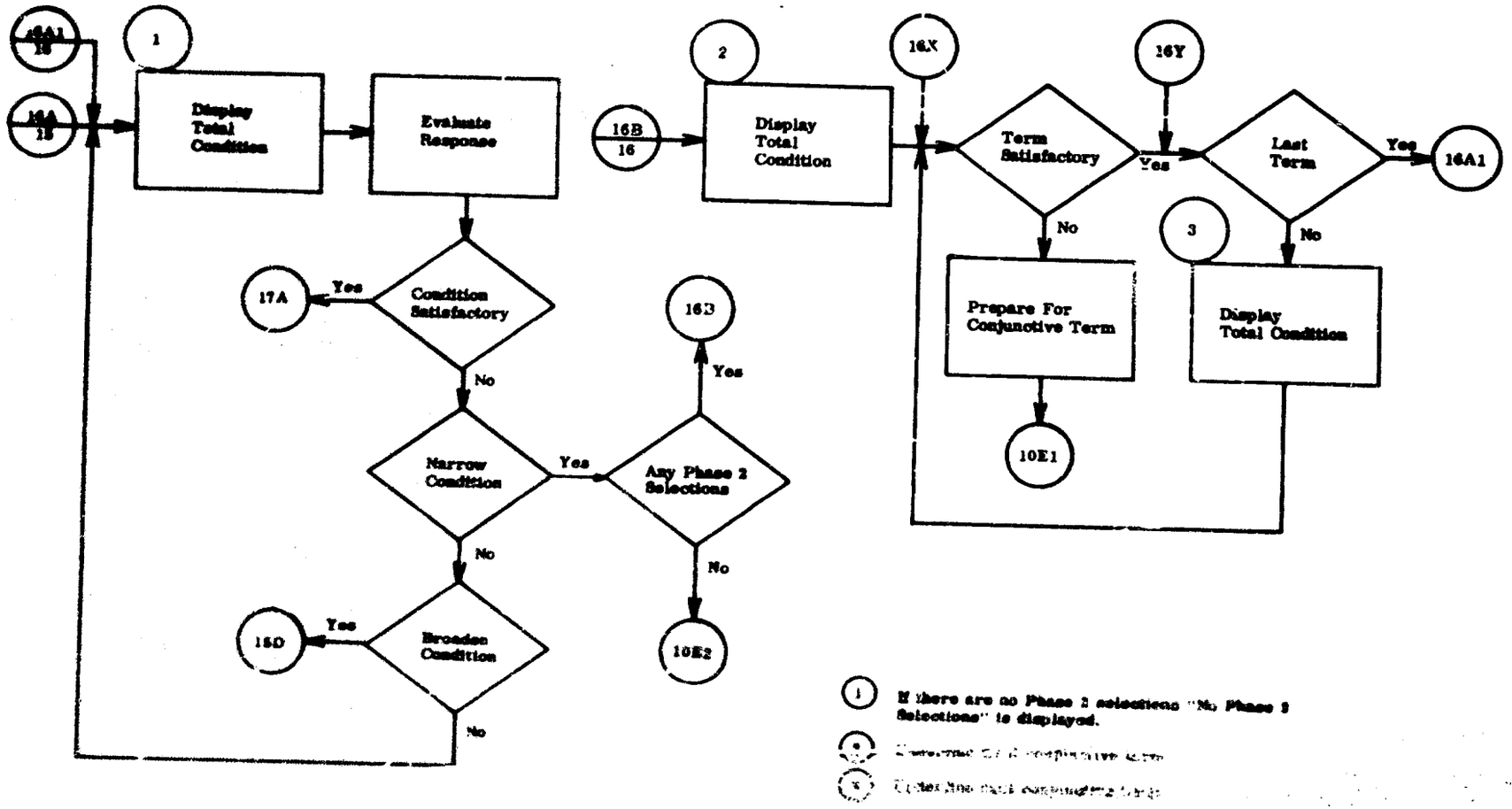


Figure 9-22. Phase 2: Terminal Selection - II


16A1
15
16A
15
1
Display Total Condition
Evaluate Response
16B
16
2
Display Total Condition
16X
16Y
Term Satisfactory
Yes
No
Last Term
Yes
No
16A1
3
Prepare For Conjunctive Term
Display Total Condition
10E1
17A
Yes
Condition Satisfactory
No
16D
Yes
Broaden Condition
No
Narrow Condition
Yes
Any Phase 2 Selections
Yes
No
16D
10E2
1 If there are no Phase 2 selections "No Phase 2 Selections" is displayed.

(1) a) If there were no Phase 1 selections, store query, query, and conditional reformat will not be available.

b) If there were no Phase 2 selections, type copy and store condition will not be available.

c) If there were no Phase 1 or Phase 2 selections only terminate is available, and it is therefore performed.

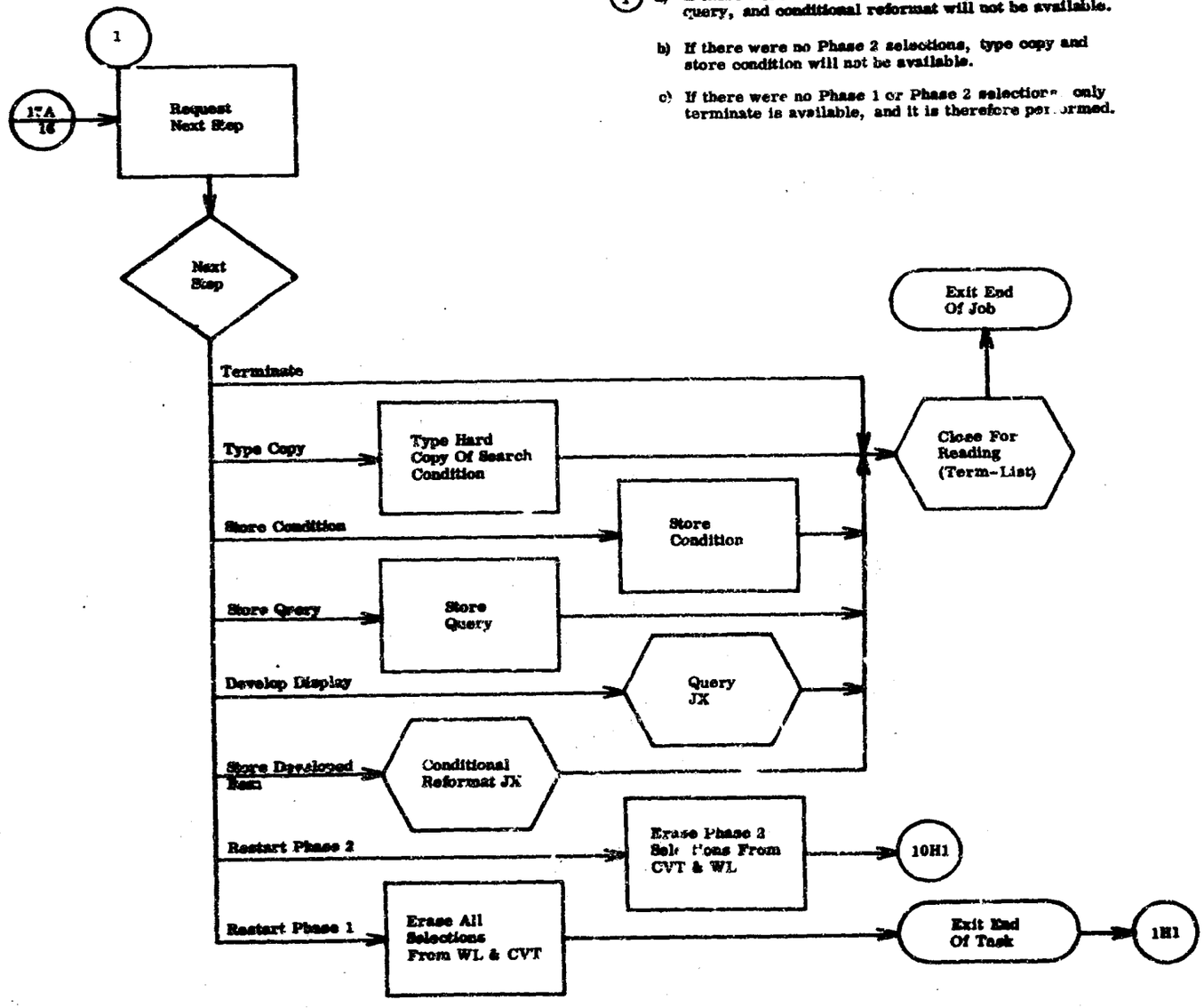


Figure 9-24. Phase 2: Termination - II

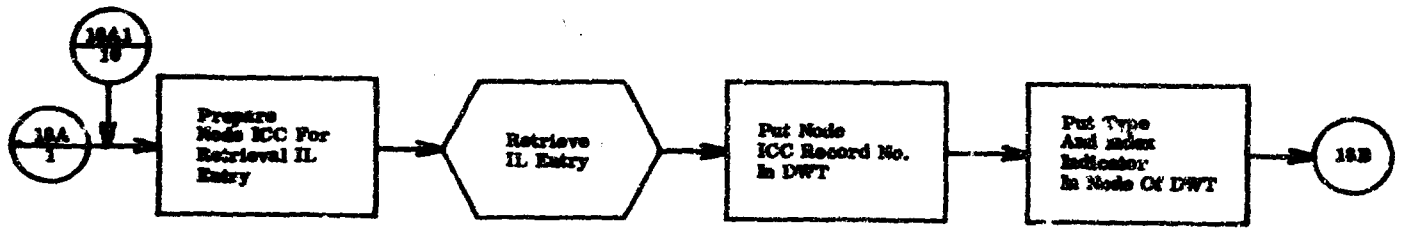


Figure 9-25. Retrieve Node IL Entry

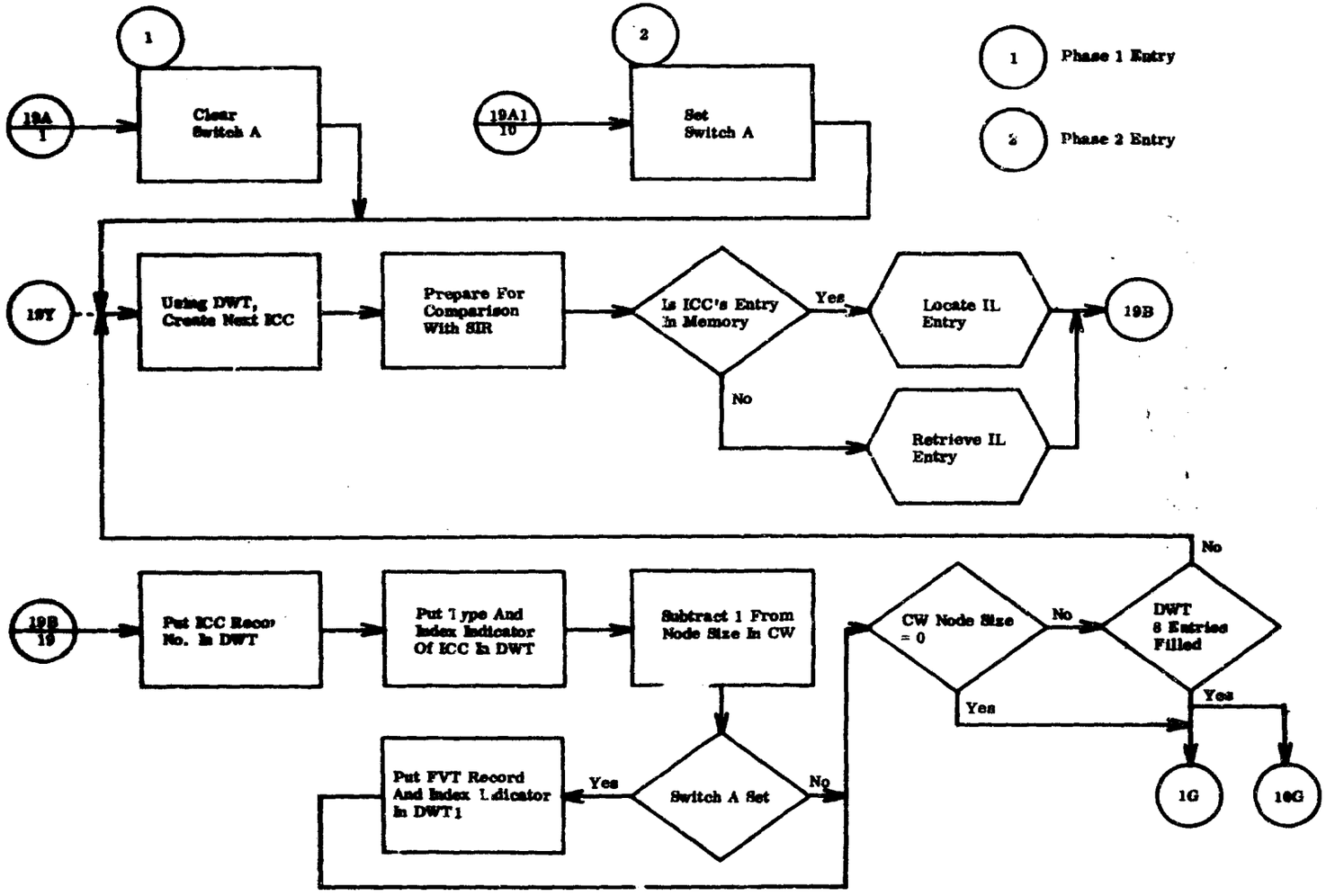


Figure 9-26. Create ICC's and Obtain IL Entries

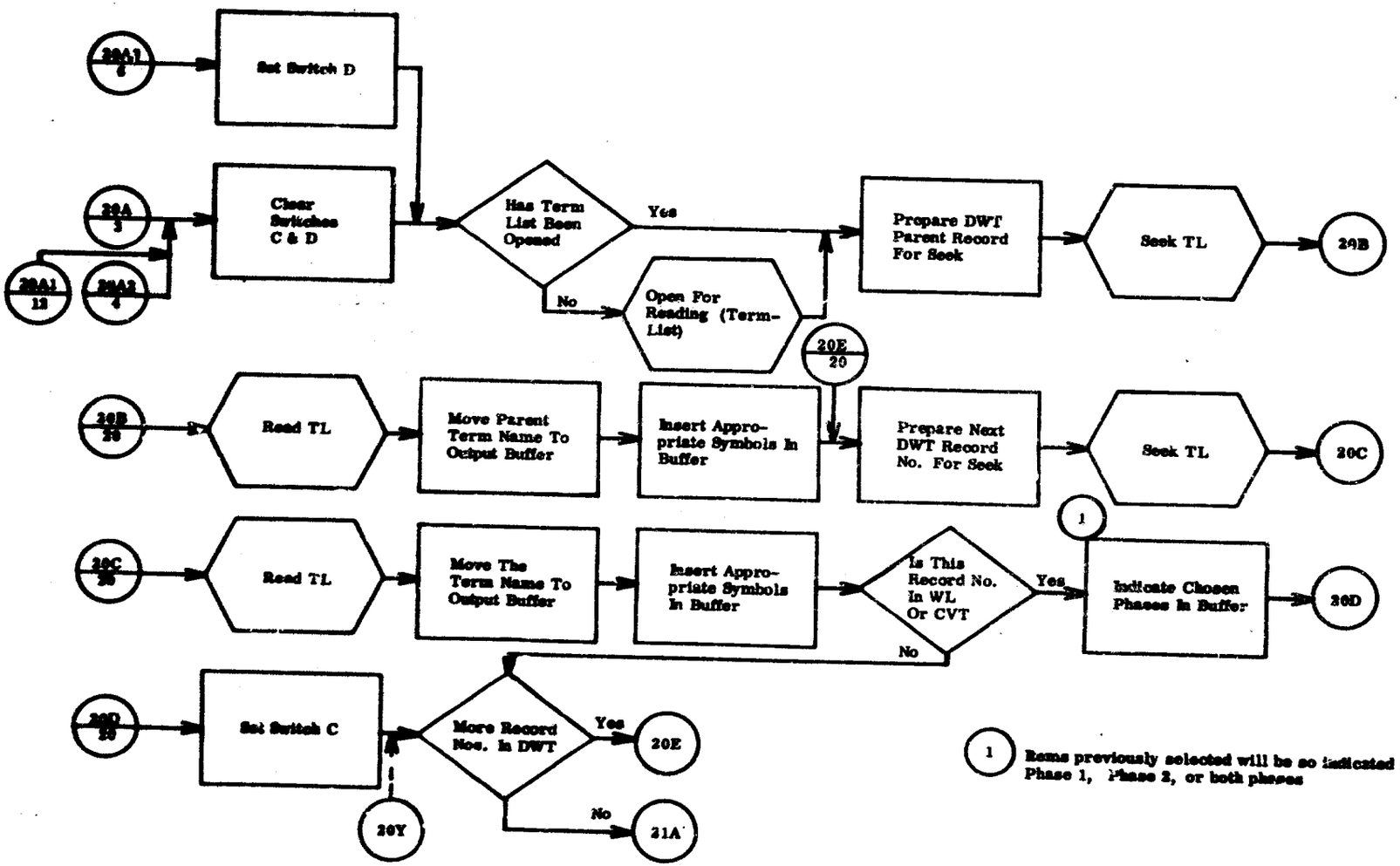
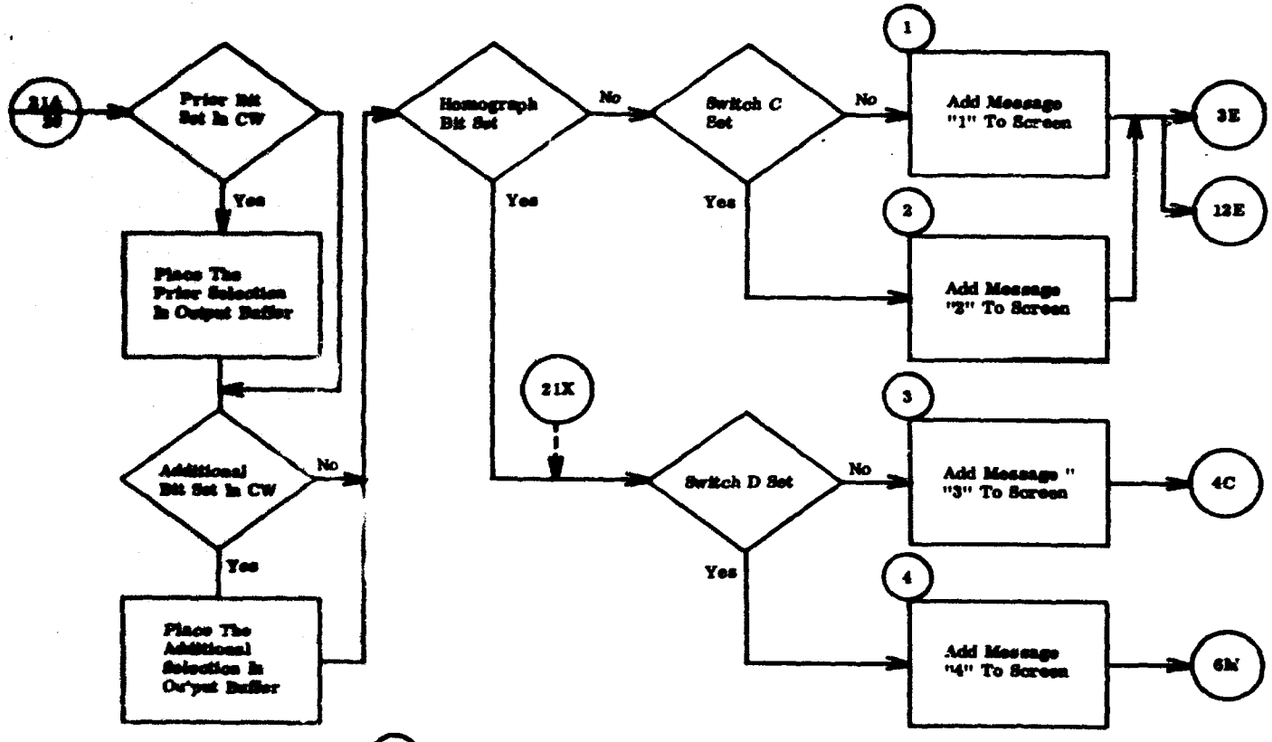


Figure 9-27. Set Up Output Buffer - I



1. Message will request a selectio...
2. Message will request an additional selection.
3. Message will request an item for homograph option.
4. Message will request a homograph item selection.

Figure 9-28. Set Up Output Buffer - II

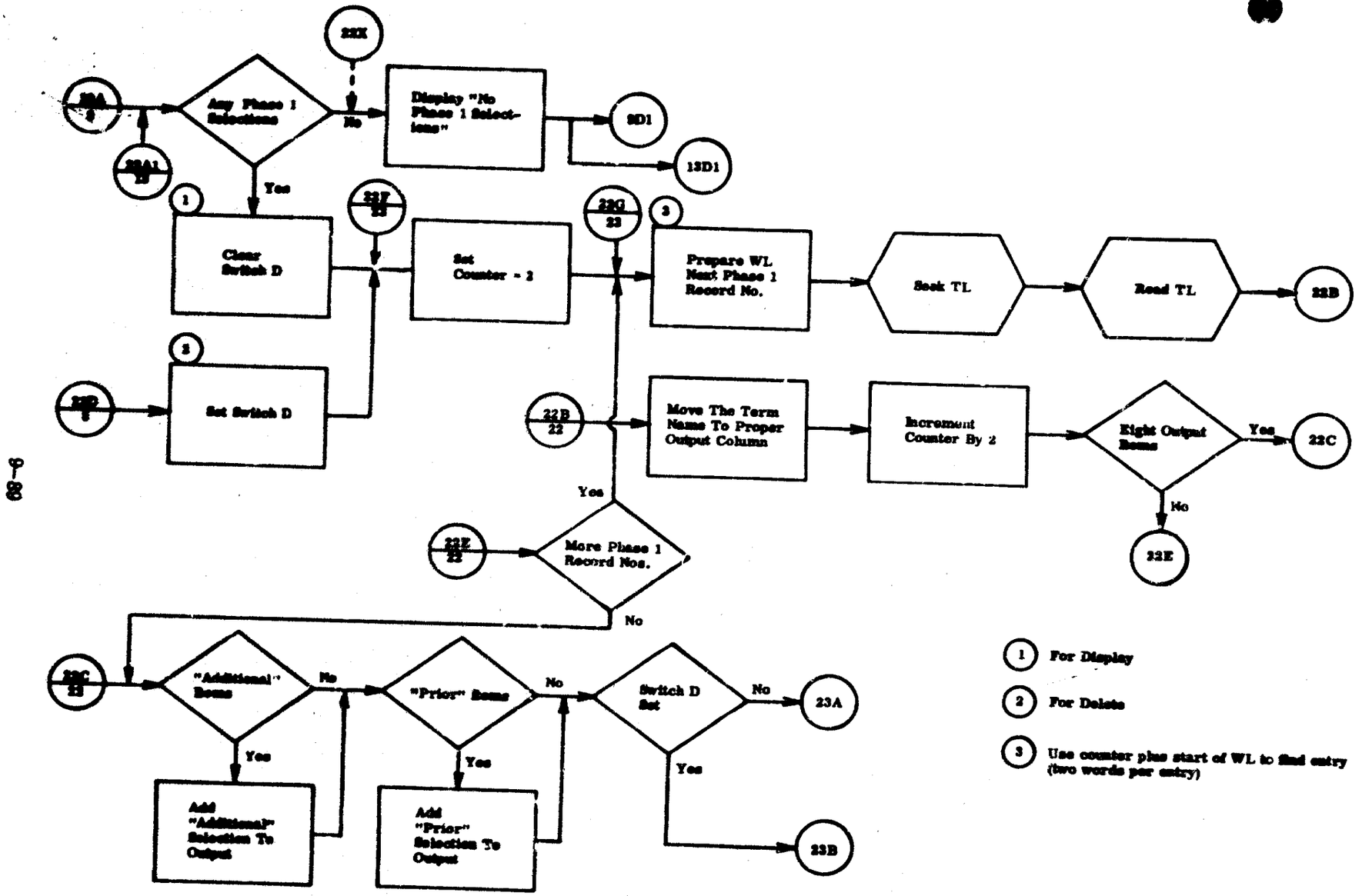


Figure 9-29. Display (Delete) Selected Items - I

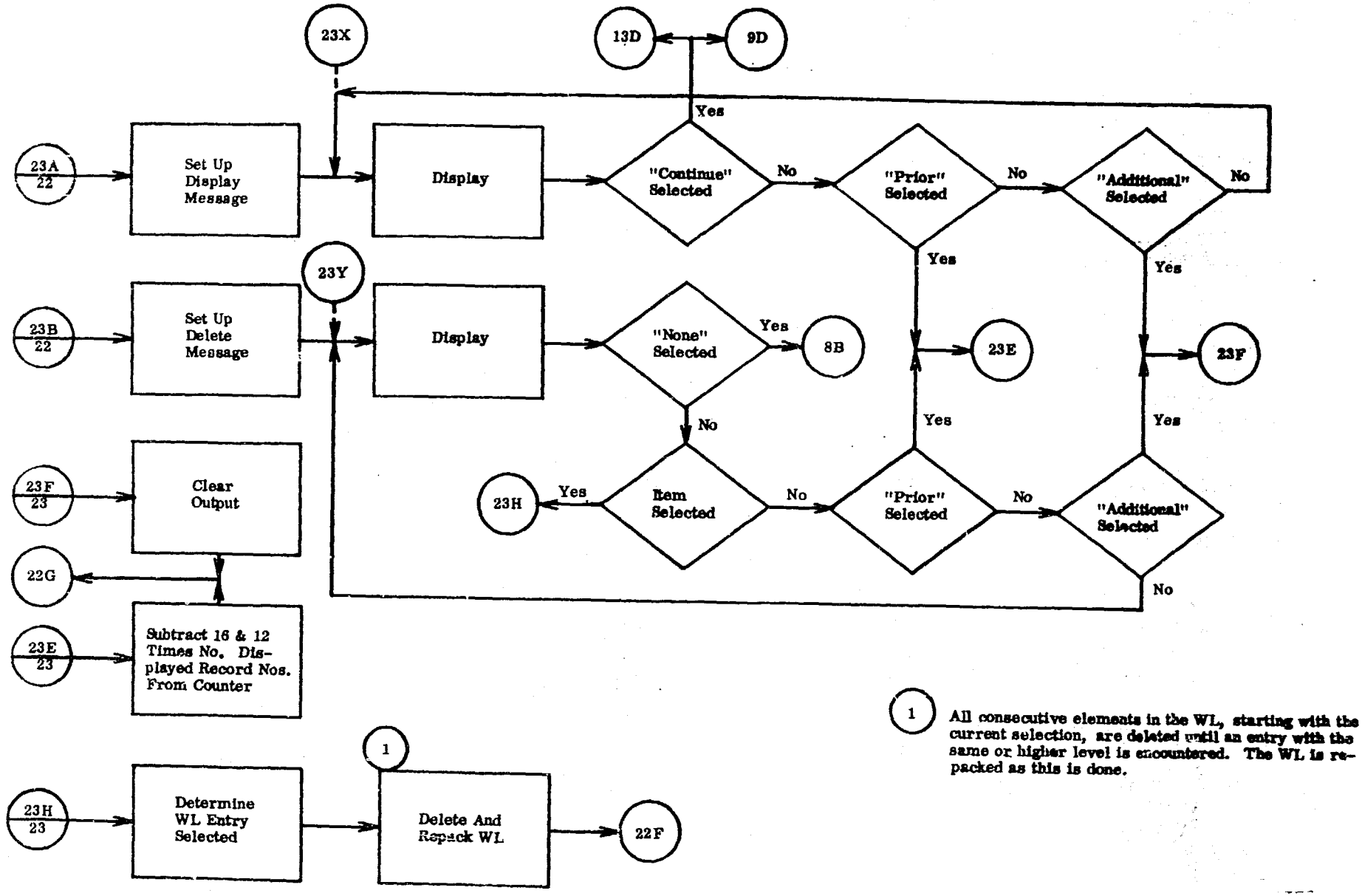


Figure 9-30. Display (Delete) Selected Items - II

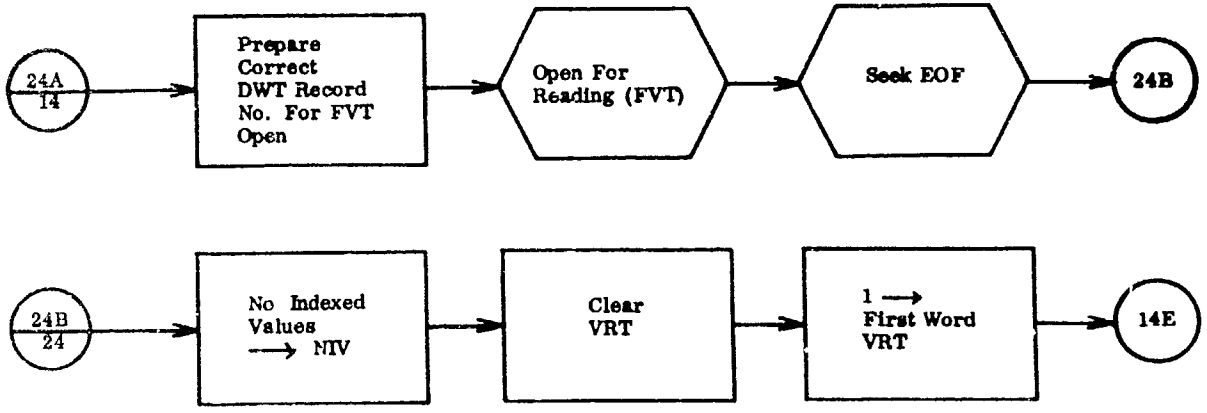


Figure 9-31. Set NIV = Number Indexed Values

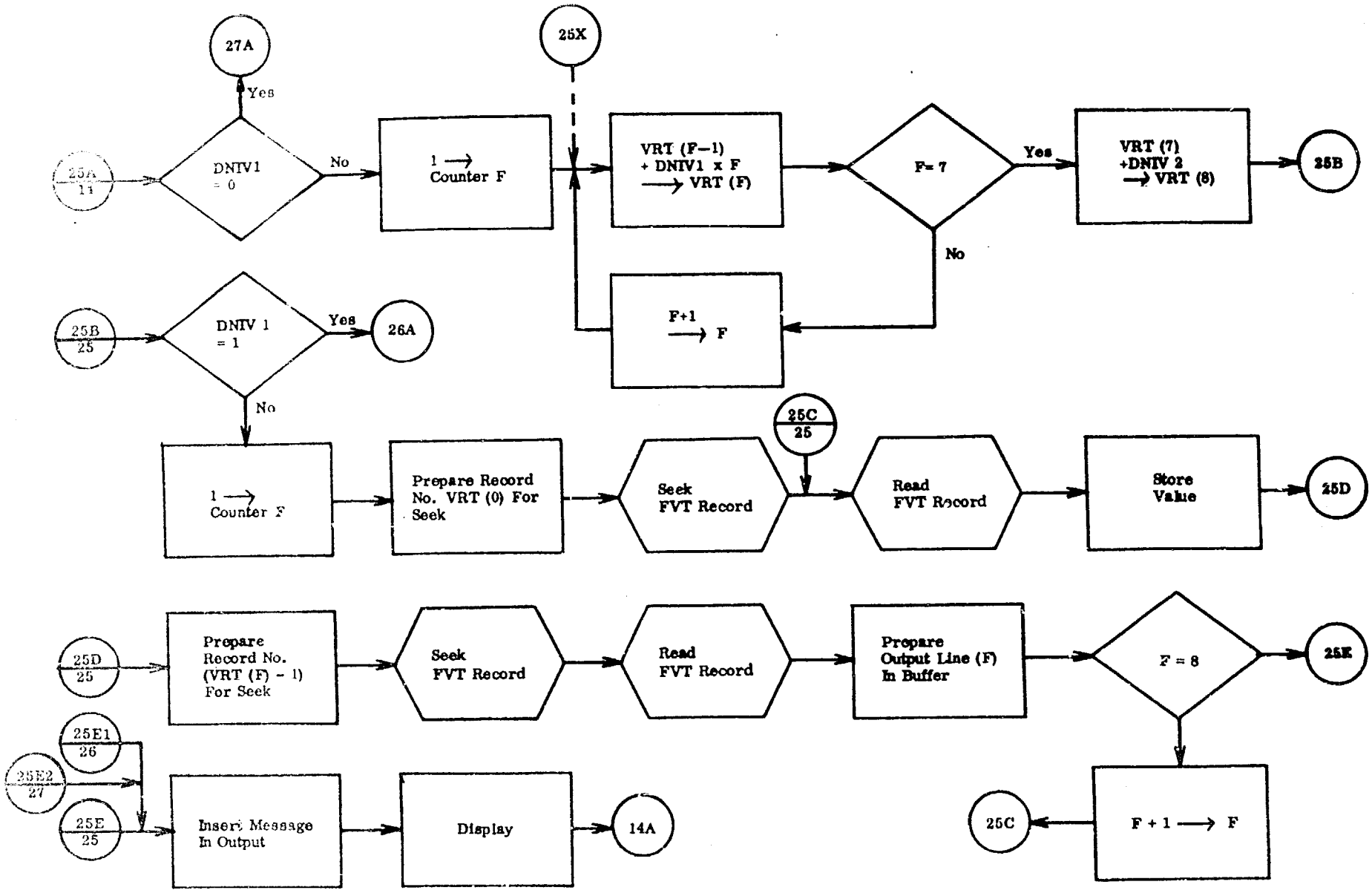


Figure 9-32. Display Eight Ranges - I

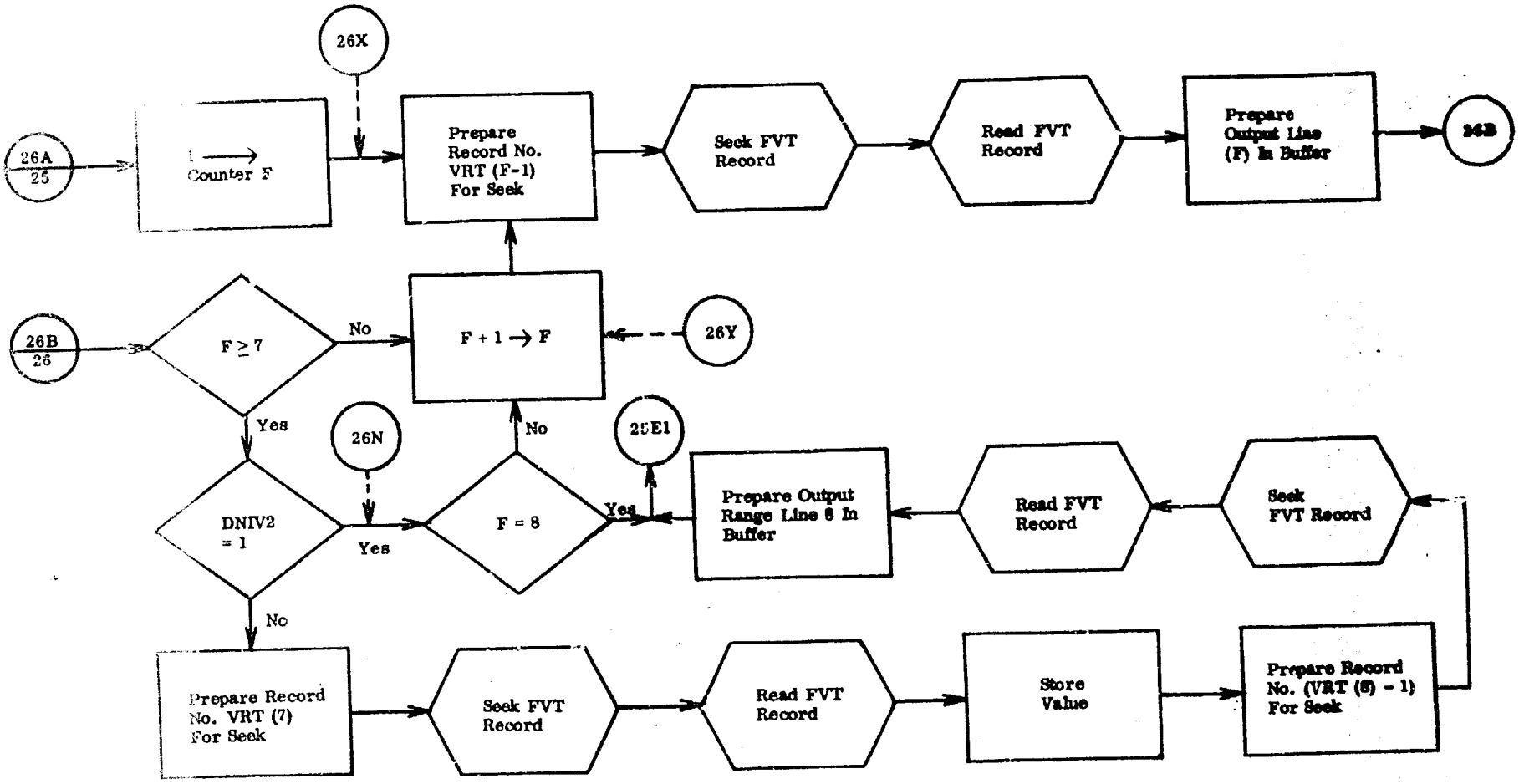


Figure 9-33. Display Eight Ranges - II

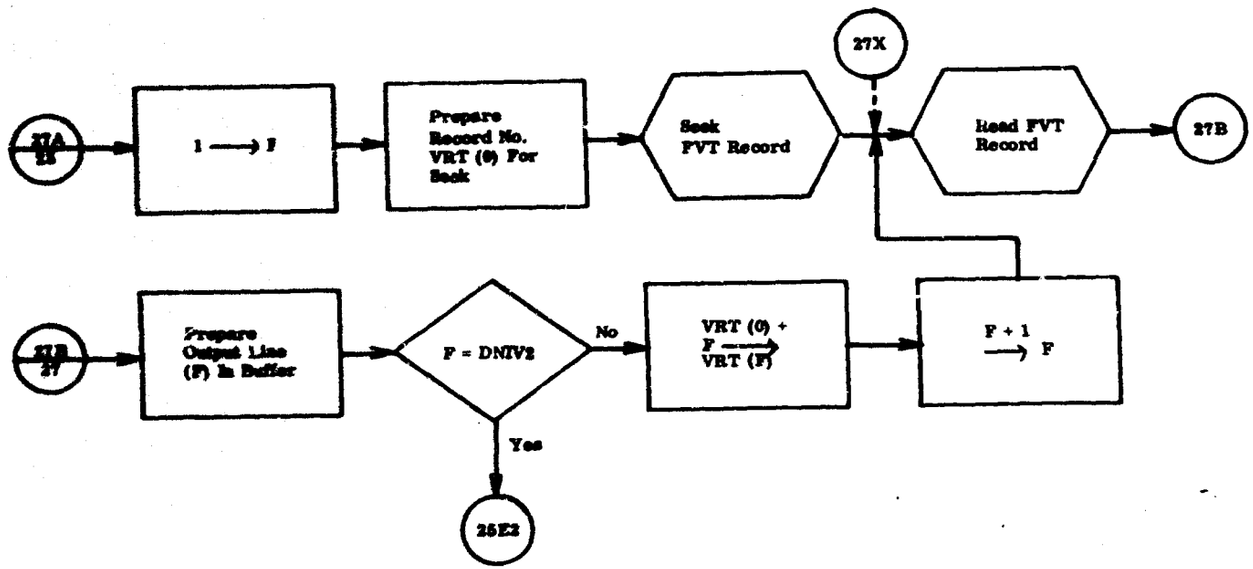


Figure 9-34. Display Eight Ranges - III

9.6 EXPLANATION OF TERMS AND PARAMETERS

9.6.1 General

This section explains the DM-1 directory elements and system terms and the key tables and parameters used in the flow charts for the Dialogue Query. Table 9-3 gives the full name for each abbreviation used and shows the paragraph number of the explanation where appropriate. Tables 9-4 through 9-11 show the formats for the tables and parameters described in Paragraphs 9.6.2 through 9.6.9.

9.6.2 Display Work Table (DWT)

The DWT is a table used in the Dialogue Query to specify the items to be displayed. Table 9-4 shows the structure of the DWT. Each display created by using the DWT, except the homograph display, represents the subitems of some parent node. The homograph routine uses the DWT in a slightly different manner, namely, to display the parents of items with the same TET name.

9.6.3 Display work Table 1 (DWT1)

The DWT1 is a table used in the Dialogue Query Phase 2 (Condition Specification Phase). At the time that the DWT is setup during Phase 2, the DWT1

TABLE 9-3. ABBREVIATIONS/NAMES USED IN FLOW CHARTS

Abbreviation	Name	Reference
CVT	Condition Value Table	Paragraph 9.6.5
CW	Control Word	Paragraph 9.6.6
-	Directory Schematic	Table 9-1
DWT	Display Work Table	Paragraph 9.6.2
DWT1	Display Work Table 1	Paragraph 9.6.3
FVT	Field Value Table	Paragraph 9.6.12
-	Homograph	Paragraph 9.3.1.1.1(4)
ICC	Item Class Code	Table 9-1
IPC	Item Position Code	-
IL	Item List	Paragraph 9.6.10 Table 9-1
JX	Job Extension	-
-	Node Level	Table 9-4
-	Node Pointer	Table 9-4
OCW	Option Control Word	Paragraph 9.6.7
QRCC	Query Response Communications Console	-
SIR	Segment Index Right	-
-	Structure of Purchasing Data Base	Figure 9-1
TET	Term Encoding Table	Paragraph 9.6.9
TL	Term List	Paragraph 9.6.11 Table 9-1
VRT	Value Range Table	Paragraph 9.6.8
WL	Want List	Paragraph 9.6.4

is also setup. The DWT1, Table 9-5, contains eight entries which correspond to the eight subsumed items in the DWT. Each entry of the DWT1 contains information about indexing of the attribute, and the FVT record number of the attribute. This information may be necessary for listing indexed values of attributes.

9.6.4 Want List (WL)

The WL (Table 9-6) is a table used to retain Phase 1 selections and Phase 2 nonfield selections. An entry in this table is made for each Phase 1 item selection, and for non-terminal Phase 2 selections. It is used in conjunction with the CVT (refer to Paragraph 9.6.5) which contains Phase 2 terminal selections. Each entry in the WL consists of two words containing the level, subnode, terminal item indicator, previous selection indicator, and record number of the item selected.

9.6.5 Condition Value Table (CVT)

The CVT (Table 9-7) is a table used to retain Phase 2 terminal (field) selections, with the specified value and conditions. The length of an entry is variable. The entries are linked via a count of the number of words to the start of the next entry. New entries will, however, always begin with a full word. The stored information consists of a field name, value, relation, and end-of-range (optional). Other indicators concern the "NOT" possibility and a boolean "AND/OR" indicator. The CVT is filled after a value has been selected and completed when a relation has been specified. The boolean "OR" is set if bit 0 in the Option Control word (OCW) is set.

9.6.6 Control Word (CW)

The Control Word (CW), Table 9-8, is a vital part of the display routine. Bits 0 through 9 contain the size of the current node ICC of the DWT as of the last display of subitems of that particular node. After a selection, bits 10 through 13 may contain the number of the selection. Bit 14 will indicate whether (set to 0) or not (set to 1) a selection of a displayed choice was made. When the dialogue is in the homograph option, bit 15 is set to alert the display routine. Bits 16 and 17 are used to inform the display routine of additional and prior items of the current node, so that corresponding choices may be added to the screen setup.

9.6.7 Option Control Word (OCW)

The Option Control Word (OCW), Table 9-9, is primarily a link to the Option routine. When calling on the Option routine, the bits in the OCW are examined for desired options. If bit 17 is set, the homograph option is available; if bit 16 is set, the delete-item option is available; if bit 15 is set, the add-item option is available; if bit 14 is set, the Display All Phase 1 Items option is available. Another use of the OCW is the specification of the boolean operator with bit 0. If this is set at the time of a condition specification, a boolean "OR" versus "AND" condition is desired.

9.6.8 Value Range Table (VRT)

The Value Range Table (VRT), Table 9-10, is used for displaying a range of values if an indexed attribute is selected in Phase 2. The list of values of the attribute is broken into eight ranges, with the last range possibly longer because of a remainder after division of the full range by eight. The table contains nine words; the first contains the start of the first range, and the remainder contain the start of successive ranges.

9.6.9 Term Encoding Table (TET)

The Term Encoding Table (TET), Table 9-11, is a file containing a record for each unique item name in alphabetical order. Each record subsumes a file of ICC's corresponding to the name. This is used by the homograph option to obtain the ICC entries for all items with the same name.

9.6.10 Item List (IL)

The Item List (IL) is the source of structure information for the dialogue. It is used to obtain the type, size, record number, and level of a particular ICC.

The Item List is a file with a record for each item (Node) in the data-pool structure. The records are in order by the Item Class Code (ICC) of that item.

9.6.11 Term List (TL)

The Term List (TL) is used by the dialogue to translate the IL record number previously obtained into a name.

The item names and units are maintained in a Term List file which is parallel to the Item List file. For each record in the Item List, there is a record in the Term List, and the corresponding record numbers contain information about the same item.

9.6.12 Field Value Table (FVT)

The Field Value Table is used by the dialogue to retrieve values of an indexed attribute.

The FVT is a file that contains records relating to certain attributes. These records contain actual values for the desired item.

TABLE 9-4. DWT (DISPLAY WORK TABLE)

PARENT . . . . . (IN B-S Format)					
. ICC					
— Node Level — Level of the Current Node ICC					— Node Pointer — Lowest Node Number Being Displayed This Time
I	Unused	I_A	P_S	Parent Type	Parent Record Number
I	Unused	I_A	P_S	Type	Record Number
I	Unused	I_A	P_S	Type	Record Number
I	Unused	I_A	P_S	Type	Record Number
I	Unused	I_A	P_S	Type	Record Number
I	Unused	I_A	P_S	Type	Record Number
I	Unused	I_A	P_S	Type	Record Number
I	Unused	I_A	P_S	Type	Record Number
I	Unused	I_A	P_S	Type	Record Number
0	1 7	8 9	10 11	12 17	18 35

BITS		
	18 - 35	Record Number
	12 - 17	Type
	10 - 11	Previous Selection
		00 = Unused
		01 = Phase 1
		10 = Phase 2
		11 = Both Phases
	8 - 9	Indexed Attribute Indicator
	1 - 7	Unused
	0	If Indicator Set, No Record Number

NOTE: DWT may have from one to eight Record Numbers.

TABLE 9-5. DWT 1 (DISPLAY WORK TABLE 1)

012		17
I_A	FVT Rec. No.	
I_A	FVT Rec. No.	
I_A	FVT Rec. No.	
I_A	FVT Rec. No.	
I_A	FVT Rec. No.	
I_A	FVT Rec. No.	
I_A	FVT Rec. No.	
I_A	FVT Rec. No.	

BITS 0 - 1 Indexed Attribute Indicator

2 - 17 FVT Record Number

NOTE: Only used in Phase 2.

TABLE 9-6. WL (WANT LIST)

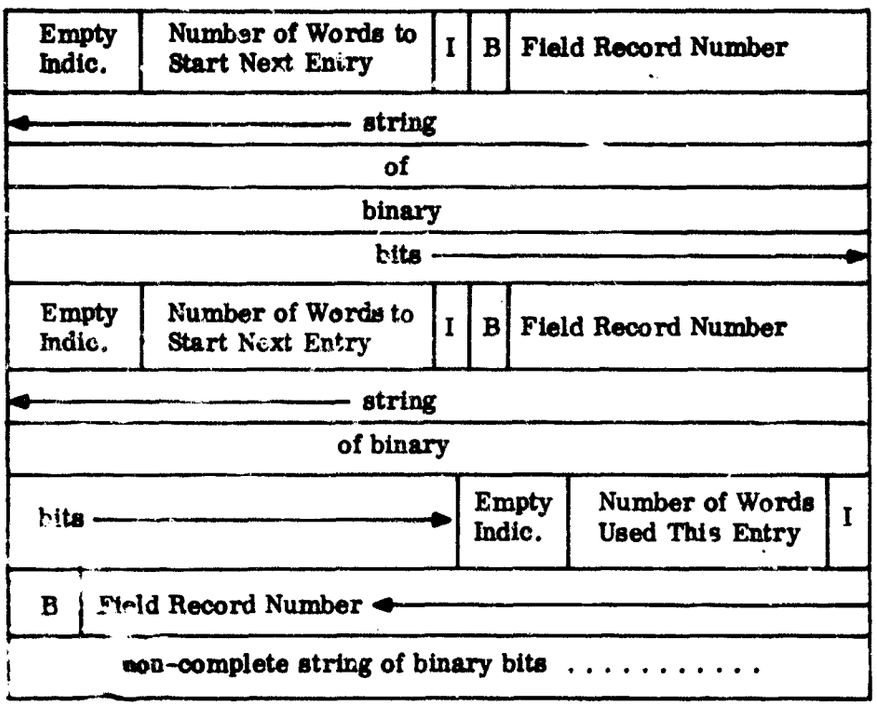
			# Of Words in Table
Level	Subnode	T_I	P_S Record Number
Level	Subnode	T_I	P_S Record Number
Level	Subnode	T_I	P_S Record Number
Level	Subnode	T_I	P_S Record Number
.	.	.	.
.	.	.	.
.	.	.	.
Level	Subnode	T_I	P_S Record Number
Level	Subnode	T_I	P_S Record Number
Level	Subnode	T_I	P_S Record Number
Level	Subnode	T_I	P_S Record Number

0 5 6 16 17 18 19 20 35

BITS
- 20 - 35 Record Number
- 18 - 19 Previous Selection
  - 00 = Unused
  - 01 = Phase 1
  - 10 = Phase 2
  - 11 = Phases 1 & 2
- 17 Terminal Item
  - 00 = NO
  - 01 = YES
- 6 - 16 Subnode Number
- 0 - 5 Level Number (Last Digit in ICC)

NOTE: WL is maintained by having all Record Numbers in consecutive order.

TABLE 9-7. CVT (CONDITION VALUE TABLE)



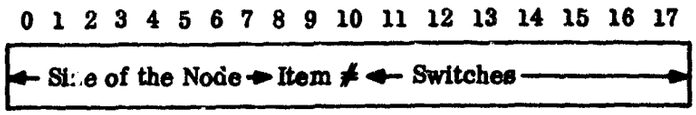
Empty Indic. = No. of unused bits in last word used for binary stream of bits

I = If 1, complete string of Binary Bits
If 0, non-complete string of Binary Bits

String of Binary Bits:
... Field Name (Alpha-Variable Length) ...
Value (Binary-Variable Length) ... Relation
(2 Bits: 00 is =, 01 is $\geq$, 10 is $\leq$, 11 is $\leq$Name$\leq$)
... End of Range (Binary-Variable Length - Optional) ... I (1 Bit: 0 is =, 1 is $\neq$) ...

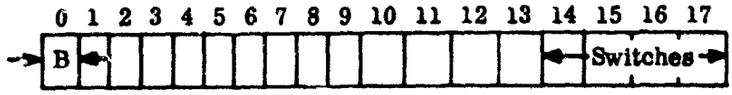
B = Boolean Connector; 0 = AND; 1 = OR

TABLE 9-8. CONTROL WORD



BIT	0 - 9	Size of the Node
BIT	10 - 13	Item Selected { 0 = Prior; 1-8 = Actual; 9 = Additional }
BIT	14	Item Has Been Selected
BIT	15	Homograph Option Selected
BIT	16	Additional Items This Node
BIT	17	Prior Items This Node

TABLE 9-9. OPTION CONTROL WORD



BIT	17	Homograph Option Available
BIT	16	Delete Item Option Available
BIT	15	Add Item Option Available
BIT	14	Display Items Option Available
BIT	1 - 13	Future Options
BIT	0	Boolean AND (0)/OR (1) Connector

TABLE 9-10. VRT (Value Range Table)

VRT			
	(0)	Start of First Range	
	(1)	Start of Next Range	Range 1
	(2)	Start of Next Range	Range 2
	(3)	Start of Next Range	Range 3
	(4)	Start of Next Range	Range 4
	(5)	Start of Next Range	Range 5
	(6)	Start of Next Range	Range 6
	(7)	Start of Next Range	Range 7
	(8)	Start of Next Range	Range 8

TABLE 9-11. TERM ENCODING TABLE (TET)

NAME	ICC FILE
BUYER	1.1.2.R.5
COST	1.1.2.R.6.R.3
DESCRIPTION	1.1.1.R.4
DUE DATE	1.1.2.R.2
ORDER LIST	1.1.3.R.4
P. O. NO.	1.1.2.R.1 1.1.3.R.4.R.1
P. O. VALUE	1.1.2.R.3
PART	1.1.1
PART LIST	1.1.2.R.6
PART NO.	1.1.1.R.1 1.1.2.R.6.R.1
PRICE	1.1.1.R.3
PRODUCTION INFORMATION	1.8
PURCHASE ORDER	1.1.2
PURCHASING INFORMATION	1.1
QUANTITY	1.1.2.R.6.R.2
RELIABILITY INFORMATION	1.5
VENDOR	1.1.2.R.4 1.1.3
VENDOR ADDRESS	1.1.3.R.3
VENDOR NAME	1.1.3.R.2
VENDOR NO.	1.1.1.R.2 1.1.3.R.1

9.7 TIMING ESTIMATES

The following paragraphs demonstrate the time required to get from one display to the next. In order to create a display, the following steps are required:

(1) Retrieve Node IL Entry. This step is only required for the first display of subitems of a node item. This subroutine will retrieve the IL entry of the DWT Node ICC. It will then store the record number in the DWT, place the type and index indicator in the DWT, and put the size of the node into the CW.

(2) Create ICC's and Get IL Entries. This subroutine will use the DWT to create the next ICC, and, by checking a certain indicator, determine whether the segment of the IL containing the record number of the desired ICC is in memory. If so, a locate of the IL entry is performed; if not, a retrieve of the IL entry is performed. In either case, such items as the record number, type, and index indicator of the ICC are placed in the DWT. The node size is decremented by one. This is done for up to eight ICC's.

(3) Set Up Output Buffer. This subroutine searches the Term List for up to nine entries of the DWT (record numbers), and it sets up the screen display with actual names, inserts appropriate symbols, indicates previous choices, adds additional and prior selections, if available, and adds the appropriate message.

The timing estimates for these three routines are:

(1) Retrieve Node IL Entry (Figure 9-25). This routine puts information about the parent node ICC of the DWT into the DWT.

One Retrieval = R

One Locate IL = L; L = 3 msec.

One Retrieve IL = 2R + L

Table Update = 1 msec.

Time = (2R + L) + 1; L = 3

= 2R + 4

= 2 (R + 2).

Example:

If R = 250 msec.

Time = 2 (R + 2)

= 2 (252)

= 504 msec.

(2) Create ICC's and Get IL Entries (Figure 9-26). This routine puts the next set of record numbers into the DWT, by creating the appropriate ICC's.

One Retrieval = R

One Locate IL = L; L = 3 msec.

One Retrieve IL = 2R + L

Table Updates = 1 msec/entry

No. of Entries in Table (Maximum of 8) = N

Maximum Time = N(2R + L) + N(1)

= N(2R + L + 1); L = 3

= N(2R + 4)

= 2N(R + 2)

Example:

If N = 8, and R = 250 msec;

Maximum Time = $2N(R + 2)$

$= 16(252)$

$= 4032$ msec.

Average Time:

1/4 of accesses = Retrieve IL

3/4 of accesses = Locate IL

$= \frac{N}{4}(2R + L) + \frac{3N}{4}(L) + N(1)$

$= \frac{N}{4}(2R + L + 3L + 4)$

$= \frac{N}{4}(2R + 4L + 4)$

$= \frac{N}{2}(R + 2L + 2);\ L = 3$

$= \frac{N}{2}(R + 8).$

Example:

If N = 8, and R = 250 msec;

Average Time = $\frac{N}{2}(R + 8)$

$= 4(250 + 8)$

$= 4(258)$

$= 1032$ msec.

(3) Set Up Output Buffer (Figures 9-27 and 9-28). This routine sets up the actual names in the output buffer.

One Internal Seek TL = S; S = 5 msec.

One Read TL = [illegible]

One External Seek TL = 2R + S

Buffer Set-up = 1 msec/entry

Number of Entries = N, $3 \leq N \leq 8$

Maximum Time = $N(2R + S) + Nr + N(1)$

= $N(2R + S + r + 1)$; $S = 3$

= $N(2R + 3 + 1 + 1)$

= $N(2R + 5)$.

Example:

If N = 9, and R = 250 msec;

Maximum Time = $N(2R + 5)$

= $9(500 + 5)$

= $9(505)$

= 4545 msec.

Average Time

1/2 of Seeks = $2R + S$

1/2 of Seeks = S

= $\frac{N}{2}(2R + S) + \frac{N}{2}(S) + N(r) + N(1)$

= $\frac{N}{2}(2R + S + S + 2r + 2)$

= $\frac{N}{2}(2R + 2S + 2r + 2)$

= $N(R + S + r + 1)$.

Example:

If N = 9, and R = 250 msec;

Average Time = $N(R + S + r + 1)$

= $9(250 + 3 + 1 + 1)$

= $9(255)$

= 2295 msec.

After a selection by the inquirer, the dialogue must perform the following steps so as to produce the next display:

(1) Retrieve Node IL Entry (Optional).

(2) Create ICC's and Get IL Entries.

(3) Set Up Output Buffer.

As can be seen in Table 9-12, the range of time required for a new screen display, after a selection has been made, is approximately three seconds to nine seconds.

TABLE 9-12. TIMING ESTIMATES

MAXIMUM TIME TO NEXT DISPLAY			
NAME	EQUATION	TIME/SEC WITH STEP 1	TIME/SEC WITHOUT STEP 1
Retrieve Node IL Entry	2 (R + 2)	0.504	—
Create ICC's and Get IL Entries	2N (R + 2)	4.032	4.032
Set Up Output Buffer	N (2R + 5)	4.545	4.545
TOTAL		9.081	8.577

AVERAGE TIME TO NEXT DISPLAY			
NAME	EQUATION	TIME/SEC WITH STEP 1	TIME/SEC WITHOUT STEP 1
Retrieve Node IL Entry	2 (R + 2)	0.504	—
Create ICC's and Get IL Entries	$\frac{N}{2}$ (R + 8)	1.032	1.032
Set Up Output Buffer	N (R + s + r + 1)	2.295	2.295
TOTAL		3.831	3.327

9.8 TECHNICAL NOTE ON POPULATION ESTIMATES

Plans for the dialogue procedure call for a system estimate of the probable number of items which satisfy the condition developed in Phase 2. In the course of the design, it became clear that an estimate based on the information available in the DM-1 directory would be likely to grossly overstate the probable population. Current plans for the implementation of the Dialogue Query job include the display of an upper bound for the number of items which meet the condition. The value of this upper bound is questionable, since it is frequently orders of magnitude greater than the actual number of items which meet the condition.

A reliable estimate of the probable population is a valuable piece of information for the inquirer. A small estimate might save the user the extra work of exhaustive specification of the restrictions. A large estimate might warn him that further restrictions are required. The upper bound, which can actually be presented to the inquirer, is not nearly so useful, because the inquirer will be justified in having little confidence in it.

9.8.1 Population Estimate With a One-Level File

A mistake was made in viewing the DM-1 data pool as a single-level file when the plans for developing a population estimate were established. With such a file, a reliable estimate based on the information in the DM-1 directories can be developed.

If the fields used in a condition are all indexed by value, the number of records meeting each primitive condition is available in the index tables. A primitive condition consists of a field name, a relation, and a test value. For indexed fields, each of the field's values is maintained in an index table with a list of the record numbers where the value occurs. The number of records which meet a primitive condition can be obtained directly from the index tables.

In general, the condition combines primitive conditions with the logical operators AND and OR. An estimate of the number of records which meet the total condition can be obtained by the following steps:

(1) Convert the number of records for each primitive condition into a percentage by dividing by the total number of records in the data pool.

(2) Convert each AND in the condition to the operation multiply, convert each OR to the operation add, and replace each primitive term with the percentage obtained in (1). This converts the condition from a parenthetic logical expression to an arithmetic expression.

(3) Calculate the value of the arithmetic expression developed from the condition in Step (2).

(4) Multiply this result by the total number of records in the data pool. The resulting number is an estimate of the number of records which meet the total condition.

This estimate is based on the assumption that the probability that any primitive term is met by a record is equal to the percentage of records in the file which meet the condition. Also, the probability of meeting a primitive condition is assumed to be independent of the probability that any other condition is met. These assumptions are invalid, since the values for one attribute of a record are usually related to the values for other attributes. For example, suppose that one-third of

the records of a personnel file are for engineers and one-half of the records are for people without college degrees. It does not necessarily follow that one-sixth (1/3 times 1/2) of the records are for engineers without college degrees. However, the assumptions are useful because they lead to a decent ballpark estimate most of the time.

A reasonable upper limit for the number of records which meet the total condition can be obtained for the single-level file. When two terms of a condition are connected by AND, the maximum number of records which could meet the combined term is the sum of the numbers met by the individual terms. If condition A is met in 50 records and condition B is met in 40 records, at most 90 records meet one condition or the other (some might meet both conditions). These rules can be used in developing an upper limit for the number of records of the single-level file which meet the condition.

9.8.2 The Situation With Multilevel Structures

The rules for calculating an estimated population and an upper bound break down when they are applied to a multilevel structure. When files may be embedded within the records of higher level files, the number of records meeting the conditions of the primitive terms may apply to various files. These numbers cannot be manipulated as in a single-level file.

Several alternatives were investigated in an attempt to overcome this difficulty. Any approach which uses the information available in the DM-1 index tables, without a severe amount of processing, results in an estimate which misses the mark by a wide margin for most situations. A suitable estimate can be developed for indexed fields only by actually performing the list processing of the search-strategy algorithm in the Conditional Search routine. This requires an undesirable amount of computer time for an inquirer who is performing a dialogue.

The search-strategy algorithm will be incorporated as an option in the Dialogue Query job. It will take much less time to develop the list of the records which meet the condition than to retrieve and display them all. In general, however, this operation will take more time than is suitable for man-machine interaction in a dialogue.

APPENDIX A. RELIABILITY CENTRAL TEST OPERATION

A.1 TEST OPERATION FUNCTIONAL REQUIREMENTS

A.1.1 Definition

The Reliability Central will be a centralized clearing house which will collect, organize, analyze, and store parts reliability information and serve as a source of reliability knowledge for users. The primary objective of the Reliability Central will be to improve the reliability of Air Force equipment.

The Rome Air Development Center proposes to demonstrate the validity of the Reliability Central concept and the feasibility and potential of its full-scale operation by implementing a Test Operation of sufficient scope to exercise all functions to be involved in the fully operational system.

A.1.2 General Requirements

The requirements of the Test Operation for equipment and software support are based on the need to develop, manage, and exploit a large and complex data base.

The DM-1 system provides the manipulative facilities required for the Test Operation. It must be used in a suitable hardware and software environment so that its facilities can be exploited in support of the Test Operation.

The equipment to be used for the Test Operation is to be supplied by the Rome Air Development Center (RADC). The major components are an M1218 computer and a large capacity bulk storage system. The output devices required for the products of the Test Operation are a line printer and a plotter. The characteristics of the equipment which is made available by RADC will influence the formats of the products of the Test Operation. The Query-Response Communications Console (QRCC) will be used for job initiation and communication between DM-1 and the system operators.

RADC will also supply the software background required by DM-1 in its role as the Test Operation data management system. The major software elements are the M1218 operating system, a storage system for the management of auxiliary storage, and programming language processors. The operating system is the Mobile Wing Executive Control Program (MW ECP). The programming language processors required are a JOVIAL compiler and a TRIM assembler. In addition, the utility programs required to deal with peripheral devices will be supplied by RADC. These routines provide for the transfer of data blocks among storage devices and for the display, printing, and plotting of system outputs prepared according to the conventions of the utilities.

A.1.3 Storage Requirement

The DM-1 data pool is segmented into many blocks of 512 eighteen-bit words. These blocks are stored in auxiliary bulk storage. Part of the software background supplied by RADC is a storage system which manages the storage and retrieval of data blocks in the auxiliary bulk storage. It is estimated that the storage system will have to accommodate between 35,000 and 50,000 data blocks for the Test Operation.

Each data block is assigned a symbolic name which is used by DM-1 when it requests the block through the storage system. The symbolic names are arbitrarily assigned nine character symbols.

The DM-1 system reads and writes segments through the facilities of the storage systems. The following operations are required:

(1) Write a segment. A segment in memory is transferred to some permanent storage outside memory. The DM-1 system specifies the location of the segment in memory. A segment name is assigned by DM-1 or returned by the storage system. The segment name is maintained by DM-1 in an index, the Segment Name List (SNL), which translates DM-1 data identifiers (IPC) to segment names.

(2) Read a segment. A previously written segment is transferred to memory from external storage. The DM-1 system specifies the segment name and the location in memory.

(3) Delete a segment. A segment name is released by the DM-1 system. This should release storage space in the bulk memory to be used for some other segment with some other name at the discretion of the storage system. This is the concern of the storage system. However, the DM-1 system assumes no responsibility for conservative use of segment names; names should be arbitrary and independent of bulk storage locations.

(4) Replace a segment. A segment from main memory replaces a segment in bulk storage. The segment name remains unchanged. The DM-1 system supplies the location of the new segment and the segment name.

A.2 ADPS INTERFACE WITH THE M1218 COMPUTER AND OPERATING SYSTEM

The Reliability Central test operation will use the M1218 computer as its data processing facility. The Reliability Central ADPS will operate on the M1218 through an interface with the RADC Mobile Wing Executive Control Program (MW ECP).

A.2.1 The M1218 Computer

The M1218 is a medium-scale, binary computer. It is a modified UNIVAC 1218 Digital Data Computer. The M1218 has 32,768 eighteen-bit words of addressable memory and a four microsecond core memory cycle. It has a limited instruction repertoire, but combinations of basic instructions provide programming flexibility for logical and mathematical computations. Floating point operations can be accommodated through subroutines. There are eight input-output channels and an interleaving technique provides for buffered input-output operations.

The major modification which differentiates the M1218 from the 1218 is the division of the M1218 memory into eight 8,192-word sectors called interlaces. One interlace is designated as the executive interlace and programs operating there can

perform such operating system functions as input-output initiation, interrupt answering, and special register setting and modification. Programs in the executive interlace and in other interlaces designated by the executive program can read and write all areas of the M1218 memory. Programs in all other interlaces are prohibited by a hardware register, the Jam Address Register (JAR), from reading or writing memory areas outside their own interlace. This memory protection is accomplished by the jammer, a computer circuit which constrains all memory references to the interlace from which they are made by jamming the three bits that represent the interlace into all addresses.

Since the memory of the standard 1218 is divided into eight banks, the further division of the M1218 memory into eight interlaces produces 64 segments of 512 words each. Each interlace consists of seven full segments and half of an eighth segment.

A.2.2 The MW ECP

Figure A-1 is a diagram of the flow of control through the MW ECP. When an operator depresses a parameter button at the QRCC, the Console Interrupt Control Program receives control through a hardware interrupt of the running program. The depression of the parameter button is interpreted as a job parameter and the interrupt word is stored for the job's interpretation. If the operator at the QRCC has pressed the "reset" button before depressing a parameter button, the depression of the parameter button is interpreted as a job request, and the interrupt word, specifying the console and the button, is stored in the request table. Then the Switcher selects the function or job which is to take control.

Every two seconds, a real-time clock (RTC) interrupt occurs. The RTC Interrupt Control Program places the Request Processor into the Switcher's queue and exits to the Switcher. When the Request Processor gets control, it checks the request table. If any job requests have been made, the Request Processor retrieves the program select switch table for the requesting console to determine which job was requested. It builds the Processing Priority Table for the job, enters the Scheduling Processor into the Switcher's queue, and exits to the Switcher.

When the Scheduling Processor gets control, it determines whether memory is available. If not, no job can be scheduled. When a job terminates, the Scheduling

Processor will get control again. When memory is available, the Scheduling Processor determines which job in the Processing Priority Table should be loaded, based on its scheduling algorithm. The algorithm attempts to advance each requested operation sequence by one step before selecting another step from the same operation sequence. The chosen job is developed into the scheduling table, assigned its requested disc areas, and loaded by the loader. It is then an eligible member of the Switcher's queue, and it receives control when conditions permit.

During its operation, the user job makes requests of the ECP through a Return-to-Executive interrupt after placing the appropriate operation code in the accumulator. When the service routine takes control, it responds to the job's request. An I/O request causes an entry into the I/O queue for the appropriate channel. If the channel is not currently busy, the I/O Processor initiates the request immediately. If the channel is busy, the I/O operation will be initiated when its I/O queue entry is pushed up by the I/O Interrupt Control Processor, after an I/O completion interrupt on the appropriate channel.

The job remains eligible for the processor in the Switcher's queue, even after it has requested an executive service. When it must have the results of the request to continue (e.g., the data from an I/O request), it issues a completion check request. If the requested operation has not been completed, the job is marked as blocked, pending completion. It remains in that state until the requested event occurs and its eligibility code is changed by the appropriate executive routine.

A.2.3 The ADPS Job Environment in the MW ECP

A key factor in defining the relationship of the ADPS to the MW ECP is the way the M1218 memory is interlaced. The characteristics of a job within the MW ECP hinge on this aspect of the hardware.

Figure A-2(b) represents the M1218's memory as a large square. The square is divided into eight horizontal slices (rows) by the eight memory banks. Memory bank 0 contains memory locations 00000 through 07777; bank 1 contains 10000-17777; bank 2 contains 20000-27777; etc. The memory bank is defined by the high-order octal digit of the address. The square in Figure A-2 is divided into eight vertical slices (columns)

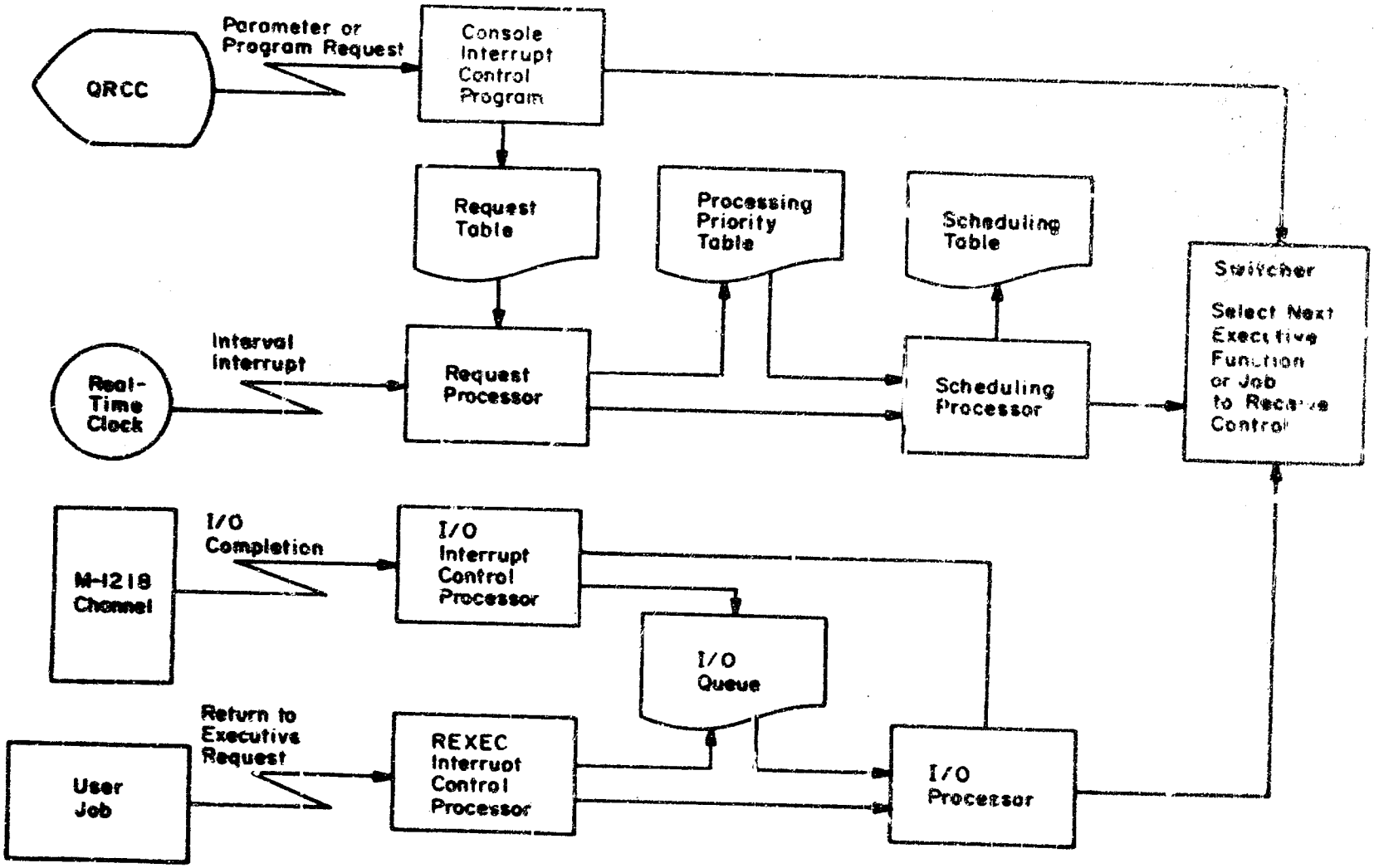
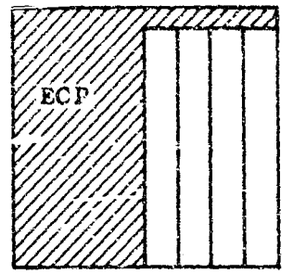


Figure A-1. Schematic Flow of Executive Control



(a) ECP Memory Residence

0xxx0	0xxx1	0xxx2	0xxx3	0xxx4	0xxx5	0xxx6	0xxx7	0
1xxx0	1xxx1	1xxx2	1xxx3	1xxx4	1xxx5	1xxx6	1xxx7	1
2xxx0	2xxx1	2xxx2	2xxx3	2xxx4	2xxx5	2xxx6	2xxx7	2
3xxx0	3xxx1	3xxx2	3xxx3	3xxx4	3xxx5	3xxx6	3xxx7	3
4xxx0	4xxx1	4xxx2	4xxx3	4xxx4	4xxx5	4xxx6	4xxx7	4
5xxx0	5xxx1	5xxx2	5xxx3	5xxx4	5xxx5	5xxx6	5xxx7	5
6xxx0	6xxx1	6xxx2	6xxx3	6xxx4	6xxx5	6xxx6	6xxx7	6
7xxx0	7xxx1	7xxx2	7xxx3	7xxx4	7xxx5	7xxx6	7xxx7	7
0	1	2	3	4	5	6	7	

MEMORY BANKS

INTERLACES

(b) 64 Memory Segments

Figure A-2. M1218 Memory Segmentation

by the eight interlaces. Interlace ø contains all addresses ending in ø; interlace 1 contains addresses ending in 1; etc. This two dimensional division of memory creates 64 memory segments of 512 words each. As shown in Figure A-2(b), each segment contains addresses with a fixed high-order digit which corresponds to the memory bank and a fixed low order digit which corresponds to the interlace. The other three digits range from øøø to 777.

Each of the eight interlaces contains eight segments and is separated functionally from the other interlaces by hardware constraints.

The MW ECP occupies four interlaces and a portion of memory bank ø in each of the other four. Figure A-2(a) shows the region of memory occupied by the MW ECP by cross-hatching. The narrow strip across the top of the figure corresponds to the portion of memory bank ø which is reserved for the executive. Memory bank ø differs from the other memory banks because it is not completely interlaced. Figure A-3 shows the configuration of memory bank ø.

Addresses ø - 777 Addresses 1øøø - 1777								Not Interlaced
2xxø	2xx1	2xx2	2xx3	2xx4	2xx5	2xx6	2xx7	Reserved
3xxø	3xx1	3xx2	3xx3	3xx4	3xx5	3xx6	3xx7	Area
4xxø	4xx1	4xx2	4xx3	4xx4	4xx5	4xx6	4xx7	
5xxø	5xx1	5xx2	5xx3	5xx4	5xx5	5xx6	5xx7	Scratch Pad
6xxø	6xx1	6xx2	6xx3	6xx4	6xx5	6xx6	6xx7	Area
7xxø	7xx1	7xx2	7xx3	7xx4	7xx5	7xx6	7xx7	
ø	1	2	3	4	5	6	7	

← INTERLACES →

Figure A-3. Memory Bank ø

The first 1024 memory locations (addresses ø through 1777) are not interlaced. They are used for operating system purposes. The remainder of memory bank ø is interlaced. However, the first 128 words (addresses 24øi through 377i) are reserved for executive purposes; the eight index registers for each interlace are in this area. The next 256 words (addresses 4øøi through 777i) are called the scratch pad area.

The MW ECP is designed to control jobs which can function in one interlace. Provision is made for multi-interlace jobs, but there are several restrictions on the

methods of communication across interlaces. The object code of a job may be loaded into the seven segments of the interlace which correspond to memory banks 1 through 7. Programs are segmented into 512-word floats which fit into a segment. Data is transmitted between programs and the bulk, random-access storage in 512-word segments. Thus, a typical job uses a 512-word segment for each input-output element and the remaining segments for program code. The scratch pad area (256 words of memory bank ∅) may be used for temporary working space.

The programming language processors provide facilities for squeezing large programs into a single interlace. They generate a Local Control Routine (LCR) which occupies one segment and manages the swapping of floatable code and data between the other six segments and auxiliary storage. This technique permits large programs to be written without concern for the limited memory space, but it results in a very inefficient execution in cases where large tables are accessed randomly or a large number of data-dependent paths exist in the logic of the program.

A second key factor in defining the relationship of the ADPS to the MW ECP is the mechanism of control of random-access storage used by the MW ECP. The disc file is divided into 1024 symbolically-addressable segments and 2048 segments called Database. Two kinds of symbolically-addressable segments provide for the random access requirements of jobs. They are: permanent areas and temporary areas.

Permanent areas are assigned to designated jobs in named blocks of eight 512-word segments. The assignments are made by an executive action taken by the system supervisor who plays a vital role in setting many system parameters. The jobs reference the segments in their permanent areas by the symbolic name and a subscript which identifies one of the eight segments. This means that the symbolic name must be negotiated when the program is written.

Temporary areas are also assigned in blocks of eight 512-word segments. Names for these areas are assigned by the programmer. The language processors convert all temporary area names to logical numbers and specify the number of temporary areas required by a program in a control block which precedes the object code. When a program is loaded, the MW ECP assigns disc areas to serve as the program's temporary areas.

The database area is treated like a large permanent area. The system supervisor designates the jobs which may read and write the database area. Individual segments are referenced by a number which designates the logical location of the segment in the database area. This approach treats the database area as a physical storage device with addresses 0 through 2047. The management of this area is the responsibility of the jobs which use it.

The third major factor which affects the ADPS job environment in the MW ECP is the operating system's conventions regarding jobs. These conventions are described in the following statements:

(1) A job is the basic unit of work. The MW ECP maintains a library of the source language for each job and another library of the generated language produced by the language processors. The basic job is a set of programs designed to operate in one interlace with data references to temporary and permanent disc areas.

(2) Jobs may be linked into operation sequences. The sequence is a list of jobs to be executed serially. The jobs in the sequence are normally written to operate together, since the MW ECP makes no provision for relating the inputs and outputs of different jobs in the sequence. All jobs in the sequence share common temporary areas.

(3) Operation sequences are defined by the system supervisor and associated with a program selection switch on a specified QRCC console. A request to execute the operation sequence is issued from the QRCC by depressing the program selection switch immediately after depressing the reset button.

A.2.4 The Requirements of DM-1 As the Reliability Central ADPS

The DM-1 system, which is to function as the Reliability Central ADPS, imposes a set of requirements on the operating environment. These requirements are the result of design choices, made under the direction of RADC technical representatives, which relate to the operating environment to be provided by RADC.

The DM-1 requirements are to be met by features of the MW ECP and by operational features of the DM-1 system itself. Most of the services required of the MW ECP are already part of that system. However, some services dealing with the management of the physical store for data segments must be added to the MW ECP to

make it compatible with the operating environment guidelines defined for the DM-1 system by RADC. Also, minor changes to the MW ECP are required to provide for the incorporation of the operational features which are part of the DM-1 system itself.

The following list specifies the requirements of DM-1 in its operation under the MW ECP.

(1) Object Code Storage. The programs of the DM-1 system and the jobs in the system library will be written in the JOVIAL or TRIM languages, compiled by the appropriate language processors, and stored in the Generated Language Library (GLL) of the MW ECP. The libraries maintained by the DM-1 system contain descriptions of jobs and input-output parameters. The DM-1 libraries use the MW ECP identifiers to reference the object code for the programs.

(2) Job Initiation. DM-1 will be a job within the MW ECP. When the QRCC button associated with the DM-1 job is depressed, the MW ECP will load the DM-1 Request Processor to initiate a DM-1 job.

(3) Program Loading. In the course of execution of a DM-1 job, the system will call on the MW ECP to load programs. This will be done through the Segment Load feature of the MW ECP.

(4) Data Segment Reading and Writing. All segments of the DM-1 data pool, including the directory and library segments, will be stored under the logical and physical control of the MW ECP. The DM-1 system will assign a nine character name to each segment when writing, and use that name when reading. The segment names will be assigned at the discretion of the DM-1 system. They should bear no direct relation to the segments' locations on the disc, and inactive segment names should not correspond to storage locations. These features are not included in the current version of the MW ECP.

(5) Job Key Storage. In managing the flow of control among the tasks of a DM-1 job, the Supervisor needs a small block of memory in the scratch pad area for job key storage. This area contains a set of numbers which specify the DM-1 request which occupies the interlace and the current status of the DM-1 job. Nothing in the MW ECP documentation prohibits this.

(6) Float Management. Most DM-1 jobs require more memory than the 3584 words available in one interlace. Some of the jobs can operate with job-controlled memory overlays when this technique does not violate the logic of the process. Others will use the Local Control Routine (LCR) generated by the JOVIAL compiler. The LCR occupies one memory bank of the interlace and manages the exchange of

program floats between the disc and the other six memory banks. The use of the LCR creates two possible problems. First, the use of scratch pad area for job key storage might conflict with the design of the LCR. Second, references made to program floats by system components which are outside the program cannot be made through the LCR. Some means must be devised to guarantee that the parameter tables and buffer areas are in memory, when a program calls a routine of the DM-1 Service Package.

(7) Service Package Loading. The DM-1 Service Package requires one interlace for the system routines which serve programs in the storage and access of data in the data pool. Some provision must be made to ensure that the Service Package is loaded whenever a DM-1 job is active. Several alternatives exist for checking the presence of the Service Package. The MW ECP could make the check and load the Service Package, if necessary. The DM-1 Request Processor could make the check and call on the MW ECP to load the Service Package, if necessary. The loading of the Service Package could be part of system initiation. Each of these alternatives requires a special status for the Service Package within the MW ECP.

(8) Service Routine Calling. The program of DM-1 jobs must be able to call routines of the Service Package. This will be done through the normal service calls of the MW ECP. In making a normal service call in the MW ECP, a program loads the accumulators with a code and a pointer to a parameter packet, and executes the Return to Executive instruction. The Executive Control Coordinator (ECC) of the MW ECP responds to service requests and directs control to the appropriate routine. To accommodate the Service Package, a block of codes will be assigned to the DM-1 system. A program will call for a Service Package routine by loading the accumulators and executing the Return to Executive instruction, as for normal service calls. The ECC will recognize the code as a member of the Service Package block and direct control to the Service Package in the special DM-1 interlace. The Service Package will determine which of its routines is needed and transfer control to the appropriate routine.

(9) Cross-Interlace Reading and Writing. DM-1 jobs may be active in many interlaces at once. The routines of the DM-1 Service Package must be reentrant so that a single copy of the Service Package can serve all active DM-1 jobs. All storage modifications required in the operation of a service routine are made in the interlace of the program which called the routine. The MW ECP normally prohibits any communication across interlace boundaries but it contains provisions for permitting such communication. The DM-1 Service Package must be granted this permission.

(10) Service Package Interruption. After responding to a hardware interrupt, the MW ECP Switcher may give control to any program

which is ready. In particular, the program which was interrupted may not regain control directly after the interrupt is processed. When the DM-1 Service Package is interrupted and another program gains control, the second program may call the Service Package. Even though the Service Package has not completed its operation for a previous program, it can operate for another program. No confusion results because all parameter storage is in the interlace of the calling program. Reference to these parameters is made through index registers. As long as the index registers and other computer registers are saved when an interrupt occurs and restored when control is returned to the interrupted program, the program can resume its operation as though the interrupt never occurred. The MW ECP saves the common computer registers with each interrupt and restores them before passing control to a program. However, the index registers are not saved by the present version of the MW ECP, because each interlace has its own set of eight index registers. When the Service Package is interrupted, the index registers will have to be saved. Storage space in the scratch area of the interlace being served by the Service Package may be used for this purpose.

(11) Service Package Blocking. When a program requests the status of a previously initiated input-output operation, the MW ECP places the program into a blocked state which makes it ineligible to regain control until the operation is completed. Since the Service Package acts as an agent for other programs, it should never be in a blocked state. The program for which the Service Package is operating should be blocked instead.

A.2.5 DM-1 Jobs in the MW ECP

The DM-1 system consists of the Supervisor, the Service Package, and a series of jobs. The Supervisor responds to console-based requests and manages the transitions of control among the executive, the task programs of the jobs, and the parts of the Supervisor itself. The Service Package acts as an agent for job programs in dealing with the data pool. The jobs perform the work requested by users at consoles.

During the execution of a job, the routines of the Service Package must be available to the job. They must be resident in the main memory of the M1218. A single reentrant copy of the Service Package can serve the several jobs which share the M1218 at once. The Service Package occupies one interlace and job programs occupy the other interlaces.

When a user wants to execute a job in the DM-1 library, he types a job run request on the QRCC. When he is satisfied with the request, he depresses the one

program select button which is assigned to DM-1. The following sequence of operations takes place during the execution of the job:

(1) The ECP places the DM-1 job (a single job from the ECP's viewpoint) into its Processing Priority Table.

(2) When an interlace is available, the ECP assigns it to DM-1, loads the job and gives it control. The code loaded is actually a bootstrap program which is part of the DM-1 Supervisor.

(3) After some initialization, the bootstrap program calls on the ECP to load the Request Processor, another part of the Supervisor.

(4) The Request Processor reads the console message (a job run request), decodes it, and builds a request record containing a Task List, which defines the sequence of programs from the DM-1 library, to be executed in response to the user's request.

(5) When the Request Processor Terminates, the Task Terminate routine, another part of the Supervisor, reads the first entry in the Task List to determine which program should be executed. It calls the ECP to load this program and gives it control.

(6) When the Task Terminate routine reaches the end of the Task List, it terminates the job and returns to the ECP.

These steps are taken in one interlace. The Supervisor and job programs use the Service Package routines which reside in another interlace. Meantime, similar sequences of events can take place in the remaining interlaces in response to other requests issued from consoles. With four interlaces, three DM-1 jobs can share the M1218 at once.

A.3 DATA CONVERSION

Data for the Reliability Central data base has been collected by the Illinois Institute of Technology Research Institute (IITRI) under Contract AF30(602)-3621. In addition, a large volume of reliability data has been collected by Autonetics as part of the Minuteman Project. These two sets of data must be converted into the Internal Data Language (IDL) of DM-1 and entered into the Test Operation data base.

A.3.1 The Reliability Central Data Entry and Conversion System

The Data Entry and Conversion System is an interim system for the preparation of a data base for the Test Operation of the Reliability Central. It prepares the data

for the Reliability Central Information Management System to be implemented by AUERBACH Corporation under a subcontract with IITRI

The system is designed for large-scale collection, error detection and correction, and conversion of the data assembled, structured and transcribed to punched cards by IITRI. Its goal is the development of an initial data base in the Internal Data Language (IDL) of the Information Management System. IDL is a machine-independent string of binary bits whose structure is stated in an independent directory. It is segmented so that it can be stored on random-access devices in a flexible and accessible form.

A.3.1.1 System Operation. The system consists of four programs. Figure A-4 is a simplified chart of the relationships among the programs.

The input card deck contains definition cards, data cards, and correction cards. Each definition describes the structure for some class of data. It expresses the relationships among items of data in the class. Definitions are developed by analysis of hard copy source documents to prepare data for the predicted processing. Data cards contain items of data transcribed from the source documents. They belong to some class of data and are prepared to fit the definition for that class. Correction cards specify changes to be made to cards on the data base tape. They are prepared to correct errors detected by the Batch Update program and reported on the error listing.

The 1401 Card-to-Tape program transfers the cards to magnetic tape, produces a reference listing, and detects syntactic errors. The emphasis in error detection is on definition cards and identity errors which could cause major problems if permitted to pass to later states in the process. The valid cards are written onto the unsorted batch tape.

The 1604-B Sort produces the sorted batch tape, placing the card images in order by class and item identifier. The program is the 1604 Sort supplied by the vendor.

The Batch Update program maintains the card image data base file in order by class and item identifier. It merges the new data from the sorted batch tape with the existing data on the data base tape to produce an updated data base tape. In merging, it performs the corrections to the data base tape requested on the batch tape. All new

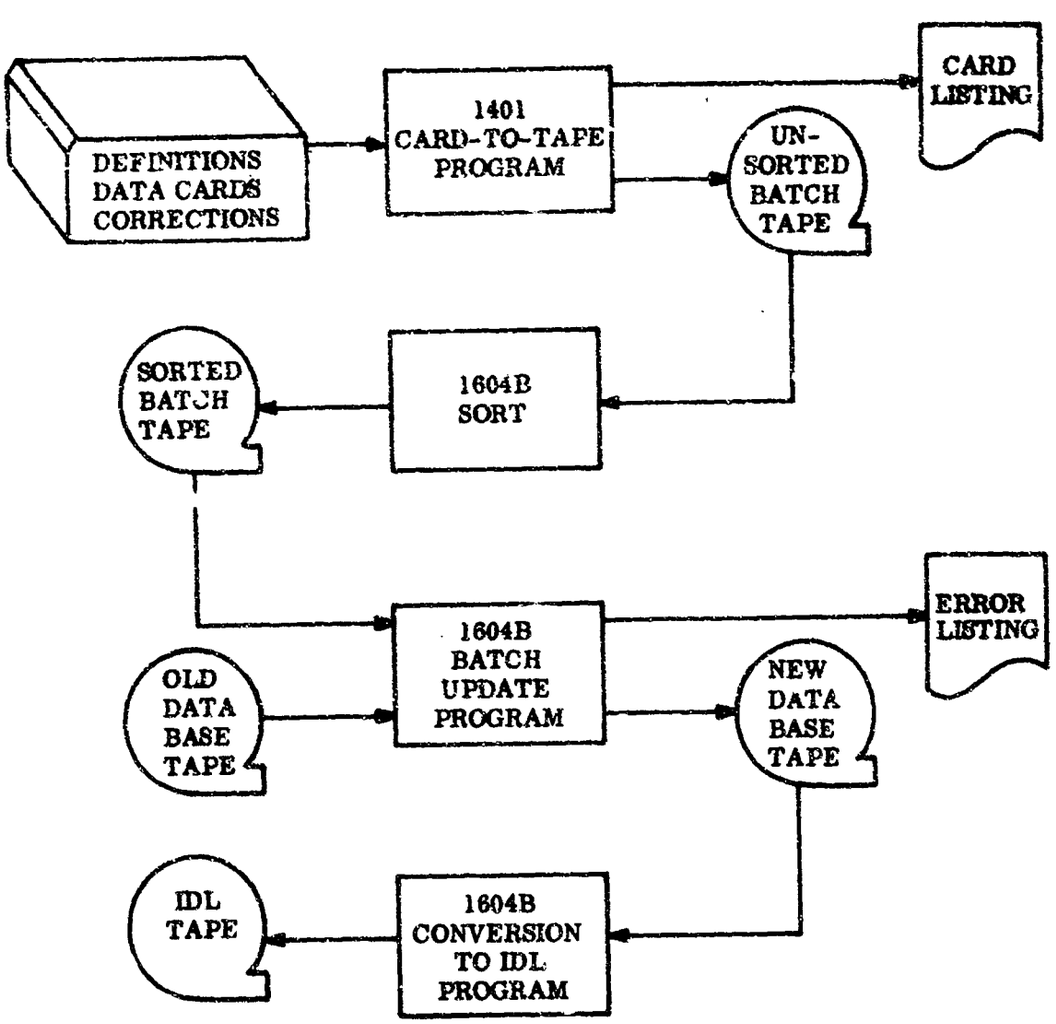


Figure A-4. Data Entry and Conversion System

data cards and any old cards affected by changes are checked against the definition for the appropriate class. Errors detected are reported on the error listing.

Any class of data containing no errors may be converted to Internal Data Language by the Conversion to IDL program. The program selects the requested class from the data base file, converts the definition to an IDL directory, and converts the data from the card image form to the IDL segmented stream of bits. The directory and the IDL segments are written on the IDL tape.

A.3.1.2 Collection. The first process in the system is the collection of definition and data cards by the 1401 Card-to-Tape program. Figure A-5 shows the input and output items associated with the program.

The input card deck consists of several elements. The batch/run card specifies the batch number and the run number for the run. A batch is a unit of data to be passed to the Batch Update program. It normally consists of several collection runs. Definition decks, if there are any, follow the batch/run card. Each definition deck is the complete definition for one class of data. The cards within a deck must be in order by identifier. Data cards are individual unit records. They may be in any order. For all runs except the last run in a batch, the input deck ends with the EOF card following the data cards. For the last run of a batch, the batch end deck and another EOF card are appended.

The previous run tape is a tape of the current batch produced by a previous run of the Card-to-Tape program. A program option provides for the addition of this run's cards to the previous cards of the batch.

The new run tape contains the card images copied from any previous run tape followed by the cards from the input deck. Only cards which pass the syntactic checks are written onto the tape.

The new batch/run card is the card to be used as the batch/run card for the next run of the Card-to-Tape program. If this run is the closing run of a batch, the new card has an incremented batch number and a run number of 1. Otherwise, the new card has the current batch number and an incremented run number.

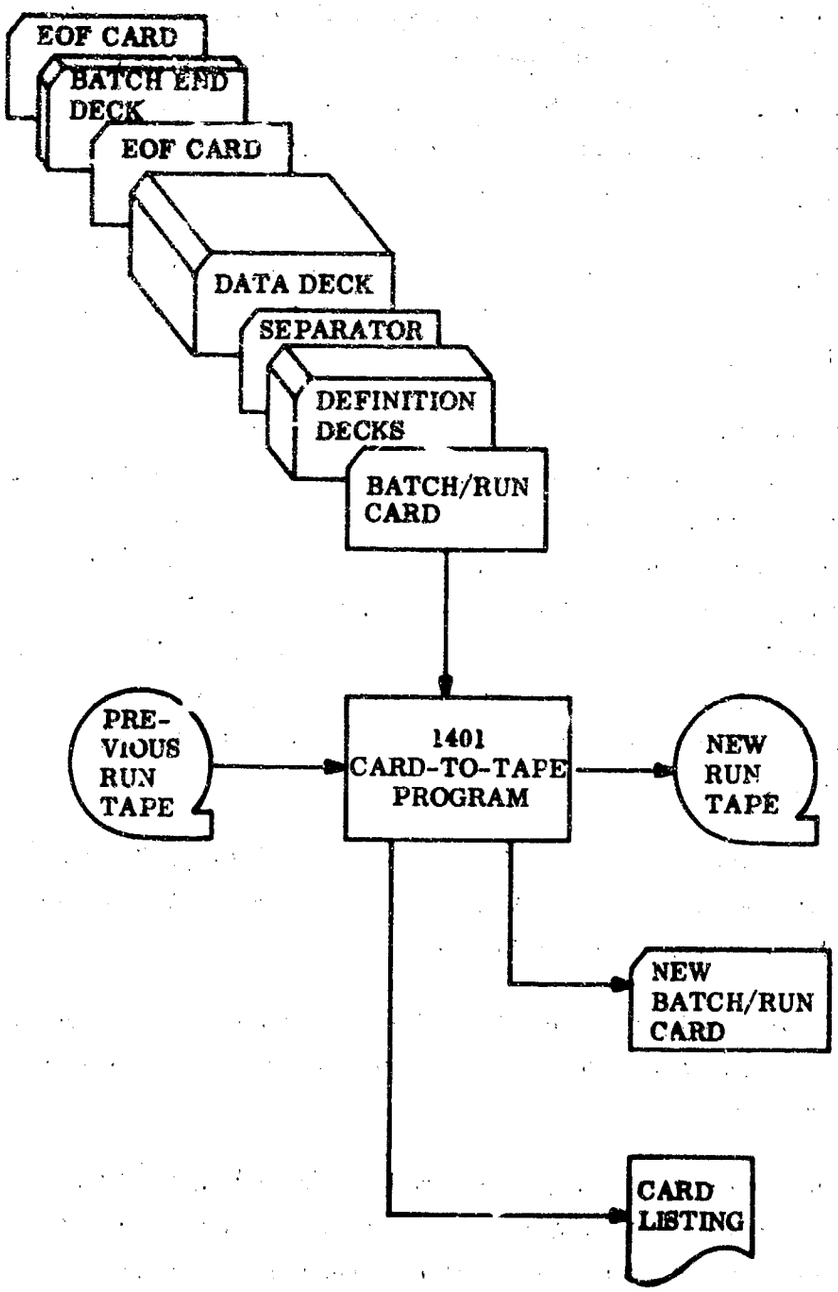


Figure A-5. Card Collection in the Card-to-Tape Program

The batch/run cards from each run of the batch, except the last, form the batch end deck for the last run of the batch. This permits an accounting to take place in the Batch Update program to ensure that the cards from all runs of the batch are included on the sorted batch tape. The Card-to-Tape program writes an accounting card with each run. It writes the batch accounting cards, derived from the batch end deck, with the last run. When the run tapes are sorted, the accounting cards sort to the front. The Batch Update program checks the run accounting cards against the batch accounting cards to be sure that all runs are included on the sorted batch tape.

The card listing contains all definition and data cards which were collected by the run. The listing is identified by the batch and run numbers. The cards which failed to pass the syntactic checks applied by the program are flagged on the listing and rejected by the program. They are not written on the tape.

The program treats definition decks as a unit. It applies extensive logical and syntactic checks to the cards of a definition. The discovery of any discrepancy causes the rejection of the entire definition deck for that class. The checking of data cards consists primarily of a check of the identifier.

A.3.1.3 Validation and Correction. The set of run tapes produced by the Card-to-Tape program contains new class definitions, new data cards and corrections to previously-entered data cards. They are processed through a validation and correction cycle by the Batch Update program. Figure A-6 is a logical data flow diagram of the process.

The set of run tapes from the same batch are sorted by the 1604 Sort program to produce the sorted batch tape. This tape is in the same order as the data base tape, by identifier within class.

The old data base tape is the one produced by the Batch Update program when the last batch was run. It contains all previously entered cards. Each class consists of a class definition followed by the data cards.

The Batch Update program merges the batch tape with the data base tape, performing the actions specified by the cards on the batch tape. This correction process is handled by the Process Batch File routine.

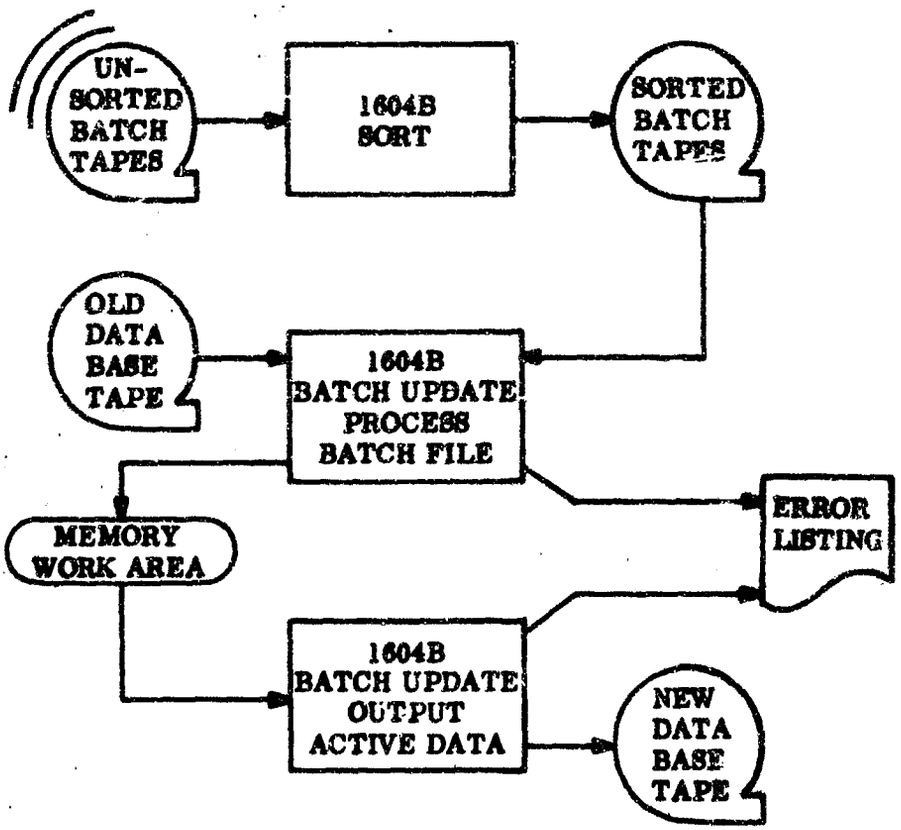


Figure A-6. Validation and Correction in the Batch Update Program

In parallel with the correction process, the program checks each new data card and those old data cards whose content was changed by the new cards. This validation process is handled by the Output Active Data routine.

The Process Batch file routine reads a card from the batch tape and copies the old data base tape until it finds a card of the same class whose identifier is equal to or greater than the batch card's identifier. In copying the data base tape, the program enters the definition for the batch card's class into a definition table. This will be used as a guide in checking the data affected by the batch card. The data base tape is positioned so that the batch card may be operated on. If the batch card is an entry card (new data), and no card with its identifier already exists in the data base, the new card is placed into the work area, validated by the Output Active Data routine and written on the new data base tape. If the batch card is a delete card and the data base contains the card identified, the data base is moved forward one card to delete the card. The data base cards whose context is changed by the deletion will be checked while positioning the data base for the next batch card. The other card types, file header entry and deleted cards and range delete cards are processed similarly.

The Output Active Data routine checks the new and affected cards against the definition. It uses the identifiers to keep track of the data items in the definition table as they are passed to the output data base. When a card is to be checked, it profiles the data to develop a table giving the size and location of each field. Each field is checked to see that its size and character composition agree with the definition.

Any errors detected in either the correction or the validation process are recorded on the error list. In the correction phase, entry of a card that already exists in the data base, deletion of a card that does not exist or an unidentified card are errors. The card image is printed with codes specifying the errors detected.

A.3.1.4 Data Conversion. Cut/classes that passed through the Validation and Correction program and contain no detected errors may be converted to the Internal Data Language (IDL). Refer to Figure A-7.

The Conversion Program searches serially through the data base tape for the first requested cut/class. The definition card entries are placed into a table in memory.

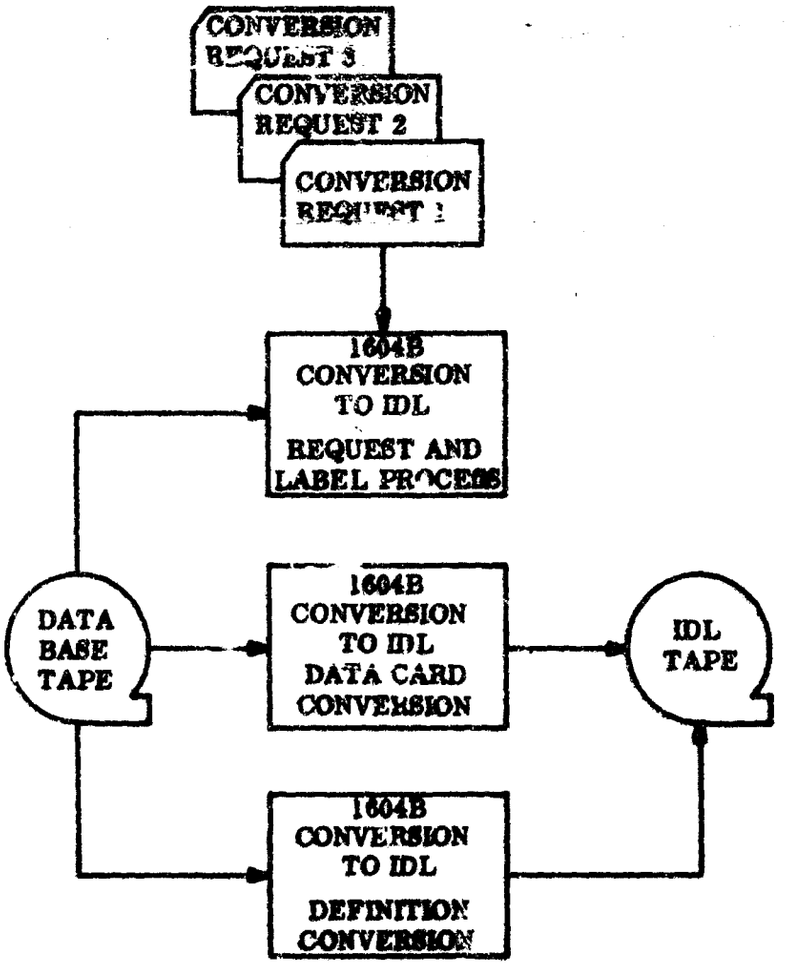


Figure A-7. Data Conversion in the Conversion to IDL Program

The entire definition table is then converted sequentially. Item name is not changed, but all other parts are converted to binary values.

Each output segment contains 127 definition entries except the last one; that segment contains the residue.

A data card is read and tested for a cut/class match with the request. The item position code (IPC) of the card is converted to binary and compared to the identity processed last. Gaps are filled with null entries.

The card is submitted to a routine which creates parameters for accession of each field in the string. Using the definition tables as a master template, the data strings are converted to the specified radix. The converted data is shifted and adjusted to conform to the definition specification in size and value.

The output routines arrange the data from the temporary hold tables to form segments. No reference is made across segments except for the definition block, which is a map of the data segments.

The only error testing performed by the conversion program is the testing of the data base error flag area. A data entry with this flag initiates a delete routine which clears the output tape of all references to the deleted cut/class.

A.3.2 Autonetics Minuteman Data

The Autonetics Minuteman data consists of 125 magnetic tapes which contain reliability data. Since the data is in machine retrievable form, it can be converted directly to IDL by a special purpose program.

Each magnetic tape is a history file which is composed of sections. Each section contains data on a single group of specimens tested under one set of stress conditions. Within each section there are three subsections:

(1) The Key Word Record. This is a group of sixteen 36-bit words which identify the data in the section.

(2) The Time-Parameter Record. This is a 21 x 21 matrix (441 words) which identifies the part parameters that were measured during

the test and the readout times at which they were measured. The first row of the matrix specifies the readout times. The first column identifies the parameters which were measured. The other positions in the matrix contain a zero if the row parameter was measured at the column readout time or a one if the parameter was not measured. Unused positions in the matrix are filled with dummy indicators.

(3) The Data Record. This subsection contains the actual data values in a large matrix whose rows contain the matrix defined by the Time-Parameter Record. There is one such matrix for each specimen tested.

A.3.3 Conversion of Autonetics Minuteman Data

Each section of the Autonetics Minuteman data corresponds to a test file in the Reliability Central data base. There is an appropriate file for each type of test in the data structure defined by IITRI. A special-purpose program will be written during the Test Operation implementation to convert selected sections to IDL under the direction of the existing definitions.

Header information corresponding to the tests represented in the Autonetics Minuteman data sections will be entered manually through the Reliability Central Data Entry and Conversion System. At the points in the manually entered data where a test file fits, the IDL Conversion program will provide for the merging of the IDL segments which were produced by the Autonetics Minuteman Conversion program. With this approach, the conversion of the manual data is independent of the conversion of the Autonetics Minuteman data. Each part is related to the other by the IPC of the IDL segments. When the IDL data from the two sources is entered into the M1218 Reliability Central data base, an integrated data base will exist.

APPENDIX B. RELIABILITY CENTRAL SCHEDULED OUTPUTS

Eleven basic outputs of Reliability Central are planned for the Test Operation. Although they are separate products, the eleven outputs are not independent. They share common routines, and some outputs require summary information produced by others. The key components of the scheduled outputs are a set of general-purpose analytic programs. These are combined with data processing programs to produce the job descriptions for the eleven outputs. The system facilities of DM-1 provide the framework for the selection of data for these jobs and control over their execution.

B.1 OUTPUTS FOR TEST OPERATION

The eleven scheduled outputs planned for the Test Operation are defined in RADC-TDR-65, Development of a Detailed Plan for a Reliability Central, February, 1965, prepared by IIT Research Institute on Contract AF30(602)-3426. The following sections contain a brief definition of each of the outputs.

B.1.1 Part Failure Rates

Data on tests performed under given stress conditions or performance data under operational stress is used to determine the failure rates for specified components. The regression coefficients for median failure rate versus stress are developed whenever there is sufficient data.

B.1.2 Part Failure Distribution Analysis

Laboratory life test data on component parts is used to determine the parametric distribution of component failures. This shows whether part failure rates increase, remain constant, or decrease with time.

B.1.3 Part Parameter Drift and Stability Versus Stress and Time

A series of plots and histograms are developed to show how a component parameter varies with time. The plots are developed from the lift test data and environmental test data available in the Data Base. Gradual changes in a parameter under a given stress are a measure of drift. More abrupt changes, brought about by high-stress applications, are a measure of stability.

B.1.4 Part Failure Mode Summary

Data on the applied stress conditions under which failures were observed is used to produce summaries defining the number of failures which occurred under each observed mode and set of stress conditions.

B.1.5 Documented Reliability Parts List

A list is periodically produced of those parts on which Reliability Central has a sufficient volume of documented reliability test data to warrant a reasonable level of confidence in the results reported in the reliability analyses performed.

B.1.6 Valid Data Parts List

A list is periodically produced of those parts for which properly validated test data is on file in the Reliability Central Data Base.

B.1.7 Part Application Data Summaries

These summaries contain data on the capabilities and limitations of parts. The application experience data and test data is used to develop summaries on major

Best Available Copy

electrical characteristics with tolerance limits, specified environmental capabilities, maximum use, storage ratings, etc.

B.1.8 Production Test Data Against Specification Requirements

Data from Qualification and Quality Conformance inspection tests is used to tabulate approved vendors and the accept/reject history of component lots. Output reports contain plots giving a running record of lot quality (percent defective) and part parameter distributions for electrical parameters measured as part of group A tests.

B.1.9 Part Reliability Improvement Rate Report

As a scheduled output, Reliability Central plans to produce annual summaries showing failure rates, year by year, for each component category and generic class. The information may be obtained, on a demand basis, for selected part groupings and different time increments.

B.1.10 Summaries of Test Programs Planned and Under way

Technical information regarding the nature and design of test programs will be maintained in the Reliability Central Data Base. Periodic summaries on these test programs will be distributed to Reliability Central users.

B.1.11 Specification Reviews

Specification reviews to maintain military specifications at a highly current level are performed manually by Reliability Central specialists. The facilities of the DM-1 system will be used as needed to support these reviews with information from the Reliability Central Data Base and analyses on available data.

B.2 THE GENERAL REQUIREMENTS OF AN OUTPUT JOB

The jobs which produce the scheduled outputs make various demands on the DM-1 system. Some of the planned outputs represent typical data processing applications (selection, ordering, summarizing, and reporting). Others involve the application of statistical and analytic processes. All of the jobs must be capable of producing useful results under a variety of conditions that relate to the information available in the Data Base.

In general, the scheduled outputs can be developed into jobs by combining a series of data processing functions, analytic functions, and display functions with the data selection and restructuring capabilities of DM-1. General-purpose data processing, analytic, and display routines can be incorporated in a number of output jobs in various combinations. For any given job, some of the parameters of the general-purpose components can be specified in the job description to tailor the component to the particular output, while other parameters can be left open until a run of the job is requested. In fact, the same job may exist in several states of parameter specification in the DM-1 library. This provides a convenient means of executing the job in some standard way, while retaining the flexibility to execute the same job with various other sets of parameters.

The parameter-binding features of DM-1, with the ability to perform conditional selection of data and structure transformations, provide for the requirements of the output jobs to a great degree. The program entry and job description features of DM-1 provide the flexibility required in combining programs and jobs to produce the scheduled outputs.

The basic requirements of the scheduled outputs and the features of the DM-1 system which support them are discussed in the following paragraphs.

B.2.1 Selection of Data for Processing

The scheduled outputs must be capable of operating on data selected on specific criteria to meet information needs as they arise. For example, the Part Failure Rates process must be capable of operating on selected data that describes tests for specified components, generic classes, military part numbers, process families, reporting agencies, etc.

In specifying the data to be bound to the input of any job, the DM-1 system permits the user to issue an arbitrary condition clause, which relates items in the Data Base to given values. The condition defines a specific subset of data which is to be selected from the Data Base to serve as the input to the job. The condition might specify, for example, that only test data on transistors, reported after a given date and associated with a certain validation code, should be processed. Any other reasonable set of qualifiers could be used.

When the data to be processed is not formatted in the Data Base according to the specifications of the job to be executed, a reformat clause may be included in the data binding specification. This notifies the system that the data is to be presented to the programs in a format different from its Data Base structure.

These two qualifiers may also be used in describing the job when entering it into the DM-1 library. The job can be stored with various conditions under different names, so that the appropriate selection criteria are applied automatically when the appropriate name is used in the job-run request. These powers of selecting and reformatting can also be used to relate the output of one process to the input of another within the definition of the job, so that independent, general-purpose components can be related in various combinations to produce different jobs. With these features of DM-1, Reliability Central specialists can develop descriptions for many useful jobs without doing any programming.

The same conditional selection and restructuring capabilities can be used to select data from the Data Base to create a work item with specified characteristics. This item can be used as input to analytic and data reduction processes in the course of special studies conducted by Reliability Central.

B.2.2 Data Summarization and Organization

In the course of developing the jobs to produce the scheduled outputs, several data processing functions are required. Little work has been done, so far, in defining the precise character of these functions, because such processes are normally a standard part of the software support associated with a computer system. The lack of a comprehensive set of data processing tools for the planned environment of the Test Operation presents an opportunity to develop a set of general-purpose functions to serve as components of many jobs for the Test Operation and far beyond.

A general-purpose summary program can be developed to operate with any set of items in the data pool. In general, it would be capable of creating an output consisting of the sum of the values across a file for specified fields. Identifying information could be carried from the source data to the results. This program would be useful in several of the scheduled outputs. For example, in Output No. 1, it could be used to calculate the total number of failures and total part test hours across many tests run under similar conditions.

A general-purpose sort program is needed to organize the data into the appropriate groupings for processing and for output. It could be used in a number of the scheduled outputs. Other data processing programs would also be useful. It is expected, however, that the specifications for such programs will be developed only after some operational experience with Reliability Central has been gained. In the early stages, it is essential that the independent, special-purpose character of certain Reliability Central operations be recognized.

B.2.3 Analytic Processes

The reduction, correlation, and analysis of the data in the reliability Data Base requires a repertoire of analytic programs. A fundamental set of analyses will be available during the Test Operation. These are primarily the ones required to produce the eleven scheduled outputs. As the analysis procedures become more clearly defined and experience is gained with the Data Base and reliability requirements, more analytic programs can be added to the library.

The value of an analytic program is greatly enhanced if it is written as a general-purpose program that is independent of the specific data structure on which it must operate. These programs are particularly amenable to the general-purpose approach. The nature of the algorithm predicates a structure for the input and output items. These structures should be defined as the formal parameters of the analytic program; then, when the program is used as a component in a job and applied to a particular data set, the DM-1 restructure capability can be used to transform the actual data format to the format required by the program.

A list of the statistical routines, distributions and statistical tests required for reliability data analysis is given in Appendix B of RADC-TDR-65, Development of the Detailed Plan for a Reliability Central. The list is repeated in Table B-1 for reference. Some of these routines will be needed in producing the eleven scheduled outputs. Others are useful analytic processes for which specific requirements are not yet defined. However, all the analytic programs can be implemented as general-purpose functions to be used on any reasonable data from the Reliability Data Base.

B.2.4 Output Presentation

The products developed by the scheduled outputs are listings and plots. To produce these will require the application of display programs. The DM-1 system includes a display capability which produces console displays from structured items

TABLE B-1. LIST OF STATISTICAL ROUTINES, DISTRIBUTIONS, AND STATISTICAL TESTS

1	Normal Distribution Model
2	Exponential Distribution Model
3	Weibull Distribution Model
4	Gamma Distribution Model
5	Truncated Normal Distribution Model
6	Logarithmic Normal Distribution Model
7	Pearson Family of Distribution Functions
8	X^2 (Chi-squared) Distribution
9	F-Distribution
10	Students' "t" Distribution
11	Analysis of Variance
12	Analysis of Covariance
13	Linear Regression (Two Variables)
14	Multiple Linear Regression (Up to Six Variables)
15	Non-Linear Regression
16	Goodness of Fit and Contingency Tests (X^2 test)
17	Tests of Regression Linearity
18	Tests for Significance of Correlation
19	Tests for Independence of Samples
20	Test for Equality of Means (Variance and Covariance Analysis)
21	Test for Equivalence of Variances
22	Determination of Confidence Regions for Computed Distribution Parameter and Regression Coefficients

in the Data Base. This can be easily converted to operate with a line printer. However, the Test Operation will require a more extensive report preparation capability. This can be provided by special-purpose programs which produce the appropriate report for each of the required outputs. This might be a satisfactory interim solution for the Test Operation. A more desirable solution, however, is the design and implementation of a general-purpose report generator. This approach will be evaluated during the forthcoming implementation of the programs required for the Test Operation. It would provide a better support tool for the long-range requirements of Reliability Central.

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)
Auerbach Corporation
Philadelphia 3, Pennsylvania 19107

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b. GROUP

3. REPORT TITLE
Reliability Central Automatic Data Processing Subsystem

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
Final Report

5. AUTHOR(S) (Last name, first name, initial)
Dr. J. Sable, W. Crowley, M. Rosenthal, S. Forst, P. Harper

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
August 1966	780	

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
AF 30(602)-3820 b. PROJECT NO. 5519	1280-TR
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) RADC-TR-66-474 (3 Vols)
d.	

10. AVAILABILITY/LIMITATION NOTICES
This document is subject to special export controls and each transmittal to foreign governments or foreign nationals may be made only with prior approval of RADC (EMLI), GAFB, NY 13440.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Rome Air Development Center (EMIID) Griffiss Air Force Base, New York 13440

13. ABSTRACT

This is a three-volume final report produced for the Rome Air Development Center (RADC) under Contract AF 30(602)-3820. Volumes I and II are the Design Specification Report for the Automatic Data Processing Subsystem (ADPS) of Reliability Central, known as Data Manager-1 (DM-1). Volume III is a survey of major, computer-oriented on-line information and fact retrieval systems.

The system design specification will be used for the implementation of the computer programs required to operate the RADC Reliability Central. The work reported in these volumes is an extension and detailing of the functional system design developed by Auerbach Corp. under Contract AF 30(602)-3433 and reported in RADC-TR-65-189, Design of Reliability Central Data Management Subsystem, July 1965. The DM-1 design provides for the incorporation of the reliability data collected by the Illinois Institute of Technology Research Institute (IITRI) under Contract AF 30(602)-3621 with Auerbach Corp. as subcontractor.

DD FORM 1473

UNCLASSIFIED
Security Classification

UNCLASSIFIED
Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
File Structures Programming Languages Data Processing Storage and Retrieval						

INSTRUCTIONS

1. ORIGINATING ACTIVITY: Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization *(corporate author)* issuing the report.

2a. REPORT SECURITY CLASSIFICATION: Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.

2b. GROUP: Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

3. REPORT TITLE: Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.

4. DESCRIPTIVE NOTES: If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.

5. AUTHOR(S): Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

6. REPORT DATE: Enter the date of the report as day, month, year; or month, year. If more than one date appears on the report, use date of publication.

7a. TOTAL NUMBER OF PAGES: The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7b. NUMBER OF REFERENCES: Enter the total number of references cited in the report.

8a. CONTRACT OR GRANT NUMBER: If appropriate, enter the applicable number of the contract or grant under which the report was written.

8b, 8c, & 8d. PROJECT NUMBER: Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

9a. ORIGINATOR'S REPORT NUMBER(S): Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.

9b. OTHER REPORT NUMBER(S): If the report has been assigned any other report numbers *(either by the originator or by the sponsor)*, also enter this number(s).

10. AVAILABILITY/LIMITATION NOTICES: Enter any limitations on further dissemination of the report, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this report from DDC."

(2) "Foreign announcement and dissemination of this report by DDC is not authorized."

(3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through _____."

(4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through _____."

(5) "All distribution of this report is controlled. Qualified DDC users shall request through _____."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. SUPPLEMENTARY NOTES: Use for additional explanatory notes.

12. SPONSORING MILITARY ACTIVITY: Enter the name of the departmental project office or laboratory sponsoring *(paying for)* the research and development. Include address.

13. ABSTRACT: Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as *(TS)*, *(S)*, *(C)*, or *(U)*.

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. KEY WORDS: Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, roles, and weights is optional.

UNCLASSIFIED
Security Classification

SUPPLEMENTARY

INFORMATION

AD-489 667

UNCLASSIFIED

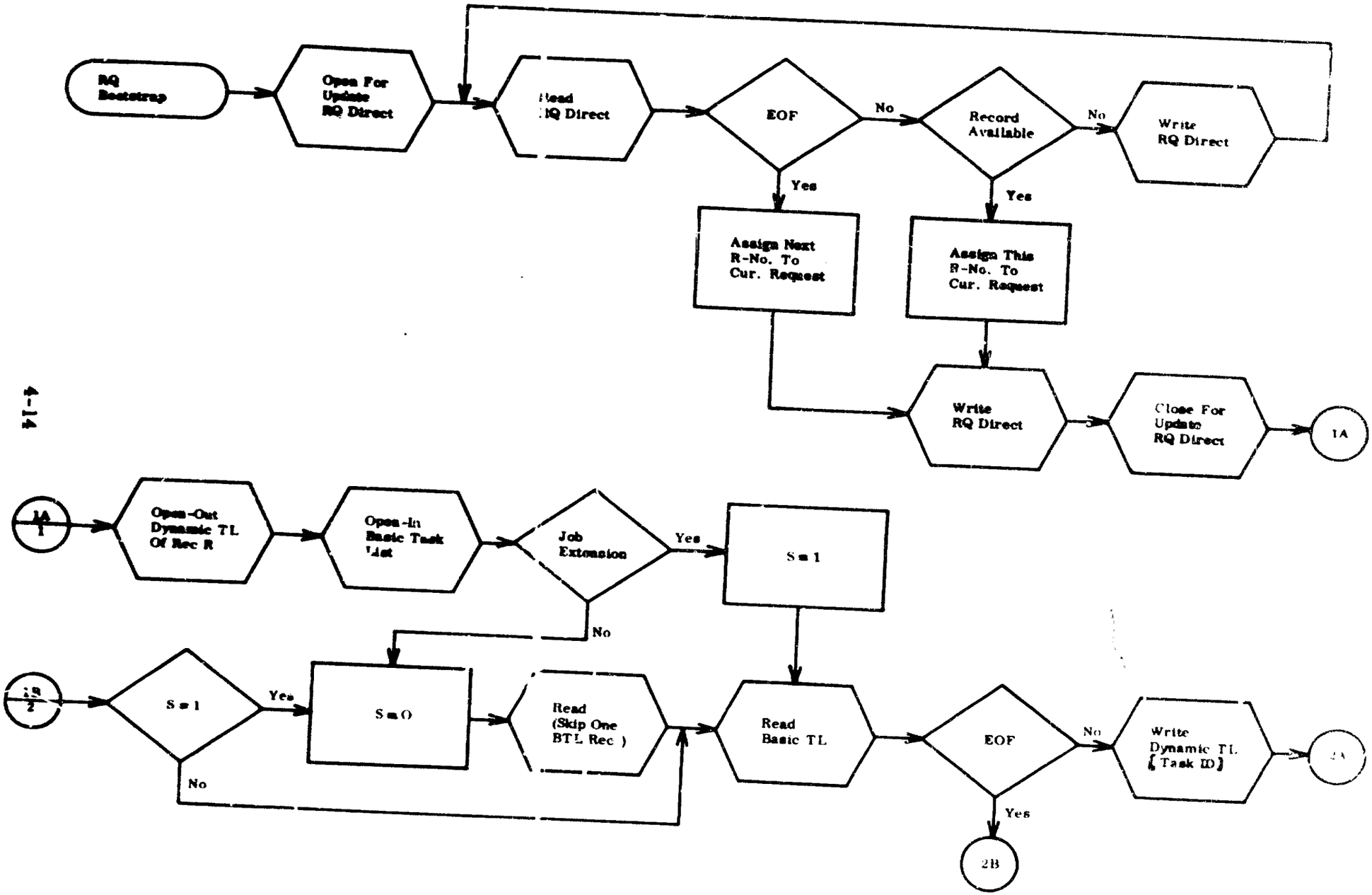
RADC-TR-66-474, Volume II August 1966

ERRATA - October 1966

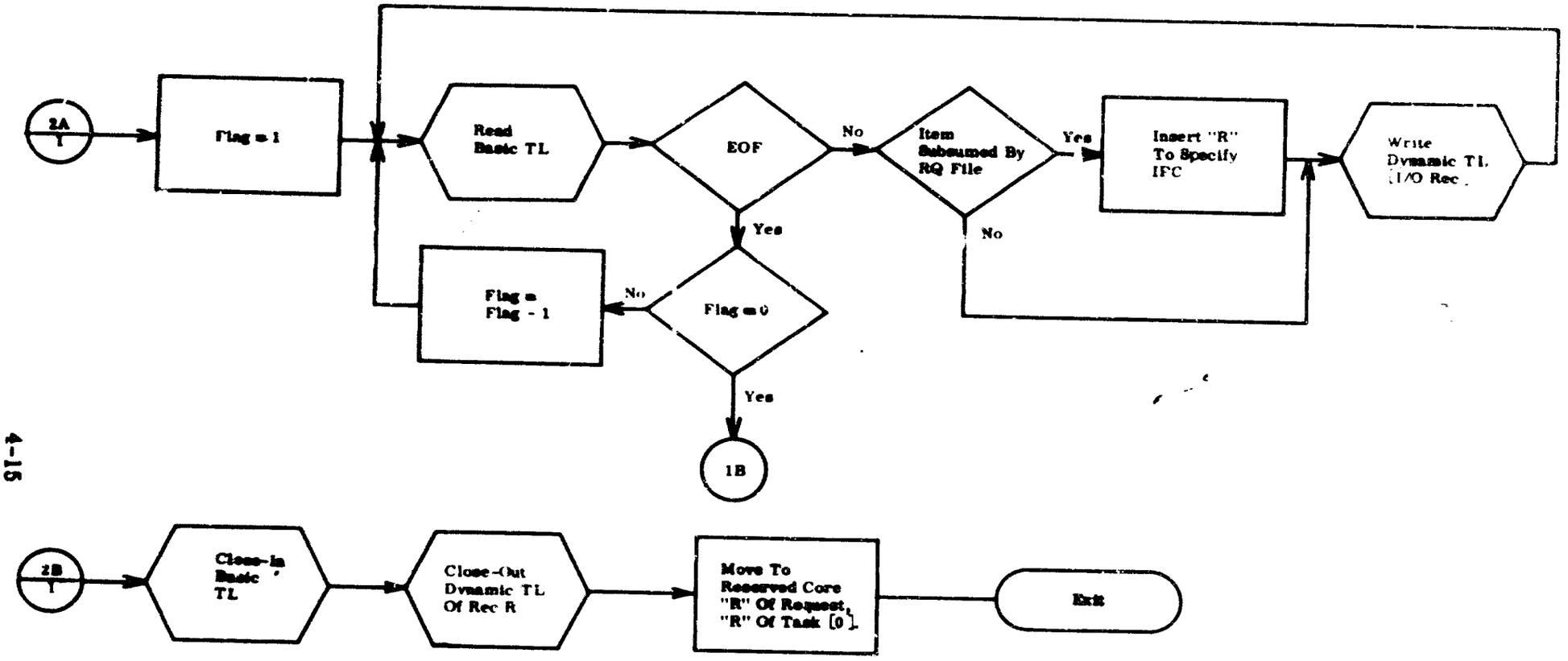
The following correction is applicable to RADC-TR-66-474, Volume II entitled "Reliability Central Automatic Data Processing Subsystem" Unclassified Report, dated August 1966.

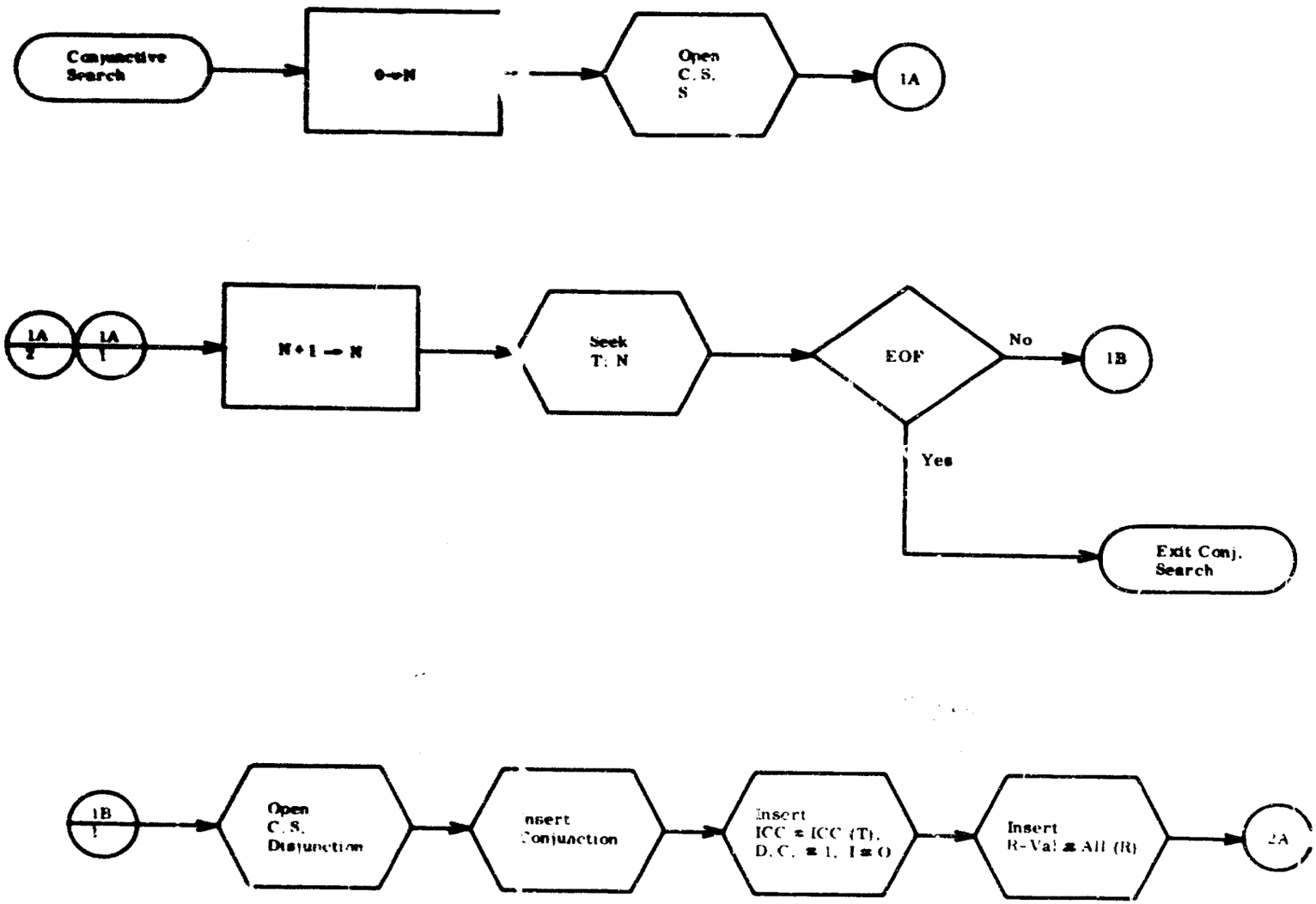
Remove pages 4-14, 4-15, 7-26, 7-27 and 7-77, insert revised pages 4-14, 4-15, 7-26, 7-27 and 7-77.

Information Processing Branch
Rome Air Development Center
Research and Technology Division
Air Force Systems Command
Griffiss Air Force Base, New York

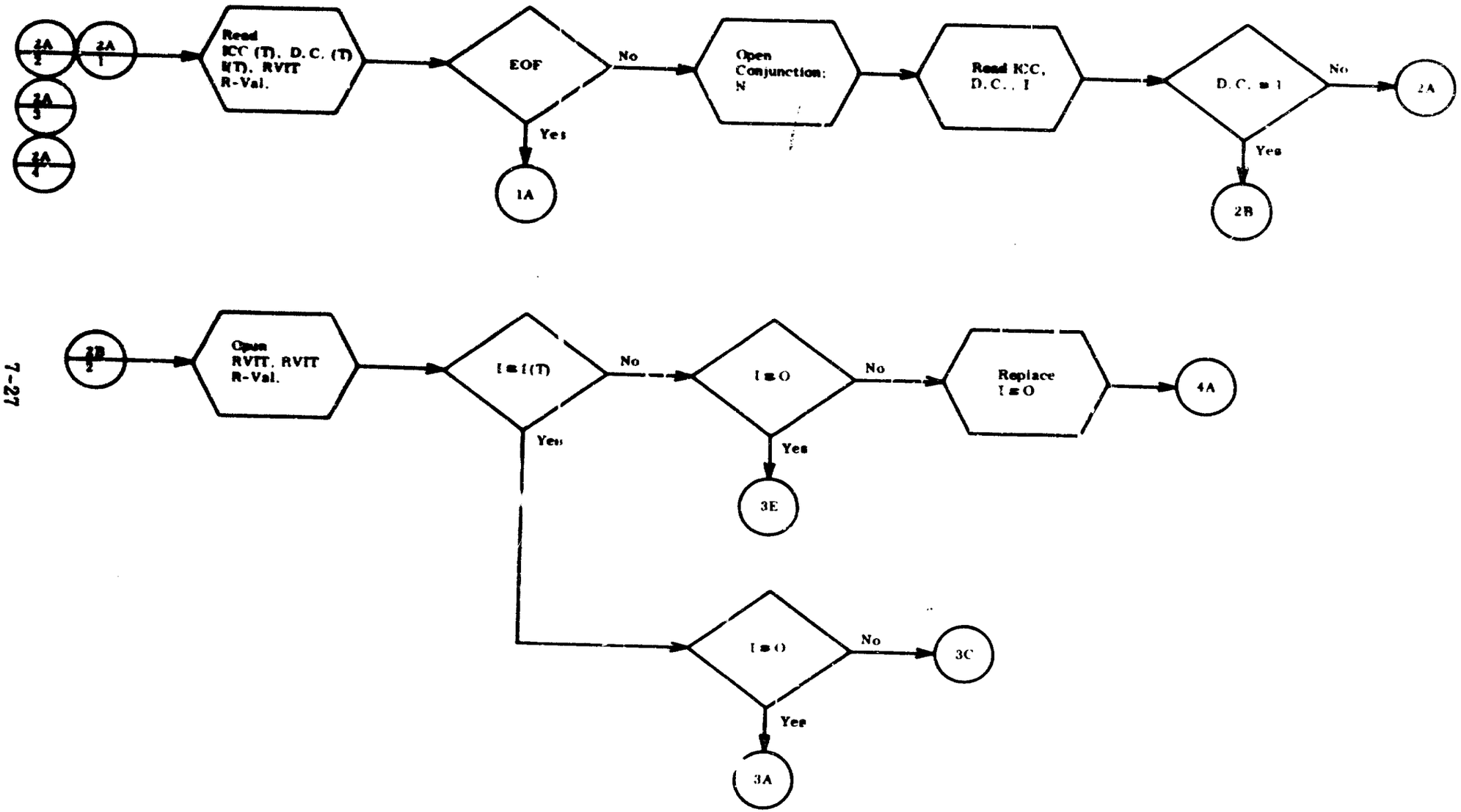


RQ Bootstrap
Sheet 1 of 2

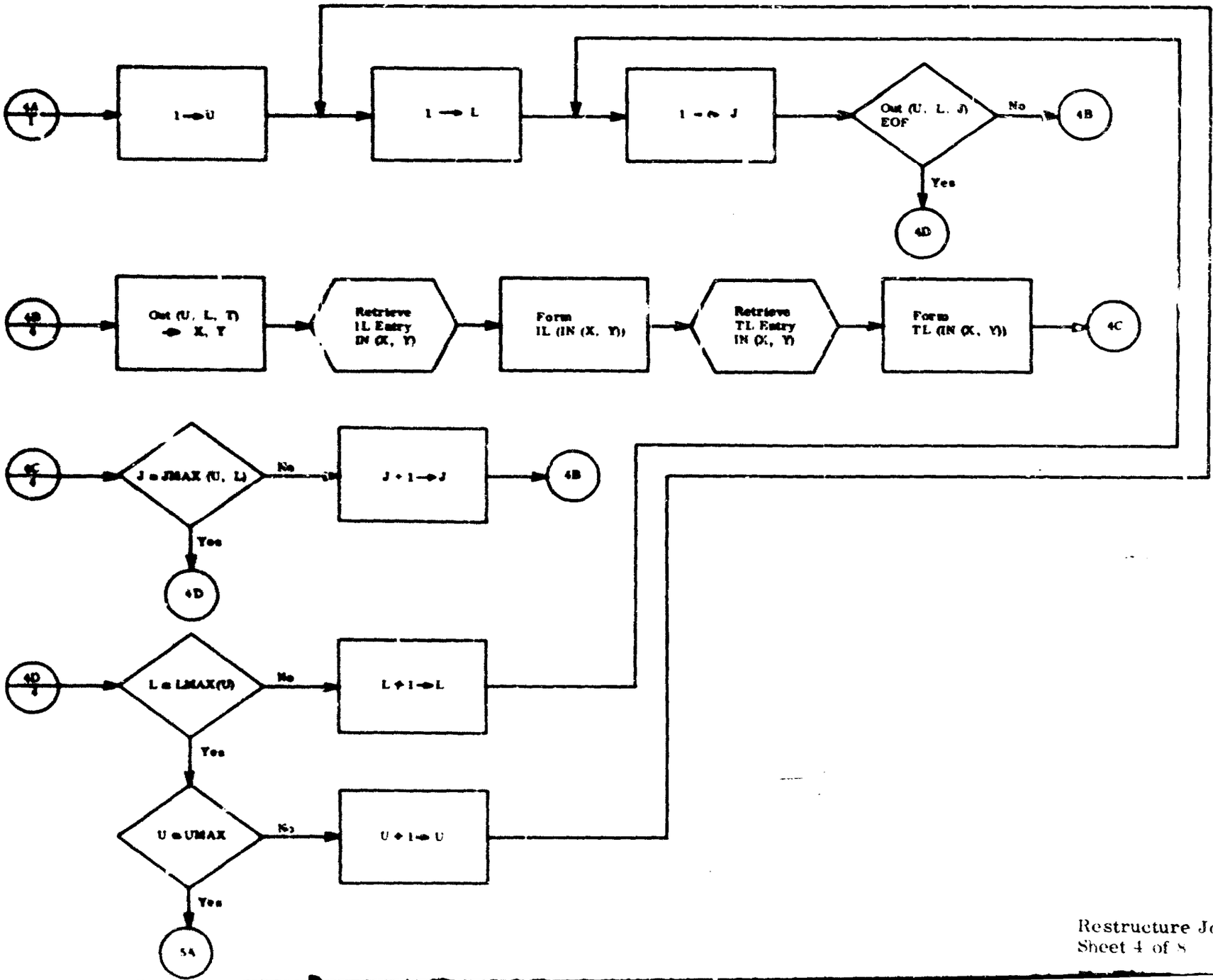

2A
Flag = 1
Read Basic TL
EOF
No
Item Subsumed By RQ File
Yes
Insert "R" To Specify IFC
Write Dynamic TL (I/O Rec.)
No
Yes
Flag = Flag - 1
No
Flag = 0
Yes
1B
2B
Close-In Basic TL
Close-Out Dynamic TL Of Rec R
Move To Reserved Core "R" Of Request, "R" Of Task [0]
Exit



Conjunctive Search Job
Sheet 1 of 9



Conjunctive Search Job
Sheet 2 of 9



Restructure Job
Sheet 4 of 8