IBM System/3
Overlay Linkage Editor and Checkpoint/Restart Programs Logic Manual

Program Number 5702-SC1
Features 6026 and 6027

SY21-0530-1

Second Edition (September 1972)

This is a major revision of, and obsoletes, the previous edition SY21-0530-0 and Technical Newsletter SN21-7643. This new edition applies to revision 07, modification 00 of the IBM System/3 Model 10 Disk System, Program Number 5702-SC1 and to all subsequent revisions and modifications unless otherwise indicated in new editions or Technical Newsletters.

Changes to text and illustrations are indicated by a vertical line at the left of the change; new or extensively revised illustrations are denoted by a bullet at the left of the figure title.

Changes are continually made to the specifications herein; before using this publication in connection with the operation of IBM Systems, consult the latest IBM System/3 Newsletter, GN20-2228, for the editions that are applicable and current.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A Reader's Comment Form is at the back of this publication. If there isn't a form, address your comments to IBM Corporation, Publications, Department 245, Rochester, Minnesota 55901.

PREFACE

This manual is a guide to program listings for people who maintain the Overlay Linkage Editor or Checkpoint/Restart features for the IBM System/3 Model 10 Disk System and the Overlay Linkage Editor feature for the IBM System/3 Model 6.

Before using this manual, the reader should be familiar with the operating procedures contained in the *IBM System/3 Overlay Linkage Editor Reference Manual,* GC21-7561, and the *IBM System/3 Model 10 Disk System Control Programming Reference Manual,* GC21-7512.

SYSTEM/3 MODEL 8

The System/3 Model 8 is supported by System/3 Model 10 Disk System control programming and program products. The facilities described in this publication for the Model 10 are also applicable to the Model 8, although the Model 8 is not referenced. It should be noted that not all devices and features which are available on the Model 10 are available on the Model 8. Therefore, Model 8 users should be familiar with the contents of *IBM System/3 Model 8 Introduction,* GC21-5114.

RELATED PUBLICATIONS

- *IBM System/3 Card and Disk System Components Reference Manual,* GA21-9103.
- *IBM System/3 Disk Systems System Control Program Logic Manual,* SY21-0502.
- *IBM System/3 Disk Systems Data Management and Input/Output Supervisor Program Logic Manual,* SY21-0512.
- *IBM System/3 Model 10 Disk System Operator's Guide,* GC21-7508.

HOW THIS MANUAL IS ORGANIZED

This manual has two parts: Overlay Linkage Editor and Checkpoint/Restart.

Part I includes:

Introduction: general information about the characteristics of the program, functions of the program, and input descriptions.

Program Organization: an operational diagram, storage maps, and flowcharts.

Data Areas: contents and formats of data areas used by two or more routines.

Diagnostic Aids: various aids that help diagnose problems.

Part II includes:

Introduction: general information about the program.

Program Organization: functional considerations, operational diagrams, storage maps, linkage considerations, and flowcharts for both the Checkpoint and Restart programs.

Data Areas: contents and formats of data areas used by Checkpoint and Restart.

Diagnostic Aids: various aids that help diagnose problems.

CONTENTS

PART I – OVERLAY LINKAGE EDITOR 1

SECTION 1. INTRODUCTION 3
Minimum Machine Requirements 3
Operating Characteristics 3
Compiler Entry 3
User Entry 3

SECTION 2. PROGRAM ORGANIZATION . . . 9

SECTION 3. DATA AREAS 33
Overlay Linkage Editor Common (LOMMON) . . . 33
Segment List Entries 33

SECTION 4. DIAGNOSTIC AIDS 43
APAR Submission 43
Overlay Fetch Routine 43
Overlay Fetch Table 44
How to Find an Overlay 44
Messages 49

PART II – CHECKPOINT/RESTART 51

SECTION 1. INTRODUCTION 53
Minimum Machine Requirements 53
Operating Characteristics 53

SECTION 2. PROGRAM ORGANIZATION . . . 55
Checkpoint 55
Checkpoint Linkage 55
Restart 68
Restart Linkage 68

SECTION 3. DATA AREA FORMATS 77
Table of Entries 77

SECTION 4. DIAGNOSTIC AIDS 79
APAR Submission 79
Data Saved at Checkpoint 79

INDEX 81

PART I

OVERLAY LINKAGE EDITOR

SECTION 1. INTRODUCTION

The Overlay Linkage Editor enables the user to influence the determination of overlays for his programs. An automatic determination of overlays is also provided.

OPERATING CHARACTERISTICS

The Overlay Linkage Editor can be entered two ways: directly from a language processor (compiler), or as a user-called program. The functions and method of operation are different depending on whether the entry is via the compiler entry or the user entry.

Compiler Entry

When entered directly from a language processor (Figure 1), the Overlay Linkage Editor can perform any or all of the following functions:

1. Catalog an R module into an object library on disk.
2. Punch an R module into cards.
3. Link R modules into an object program and catalog the program into an object library on disk and/or punch it into cards.

Input for Compiler Entry

Input to the Overlay Linkage Editor is in the $WORK file on disk. Each record in $WORK is 64 bytes long (Figure 2). The first record must be the options record; R modules follow the options record.

Each R module consists of External Symbol List (ESL) fields (packed five to a 64-byte S-type record) and text records. An END record follows the R modules. A /* record must be the last record in the compiler output.

Options Record: The options record tells the Overlay Linkage Editor what functions to perform. The options record must be the first record in $WORK. Figure 3 shows the format of the options record.

R module: The R module consists of ESL fields packed into S-type records, text records, and an END record. Each 64-byte S-type record can contain up to five, 12-byte ESL fields. The S-type record must be hex '00's after the ESL fields.

R modules are described in the *IBM System/3 Overlay Linkage Editor Reference Manual*, GC21-7561.

Output From Compiler Entry

Output from the Overlay Linkage Editor is specified by the options record in $WORK. The R module in $WORK can be punched into cards and/or cataloged into the object library. If link edit is specified, an O module is built from the input R module. The O module is then punched into cards and/or cataloged into the object library.

A storage map and cross reference list is printed unless the options card specifies otherwise.

User Entry

The Overlay Linkage Editor can be loaded by using a // LOAD $OLINK OCL statement. The user must supply control statements.

Input for User Entry

Input for the user entry is described in the *IBM System/3 Overlay Linkage Editor Reference Manual*, GC21-7561.

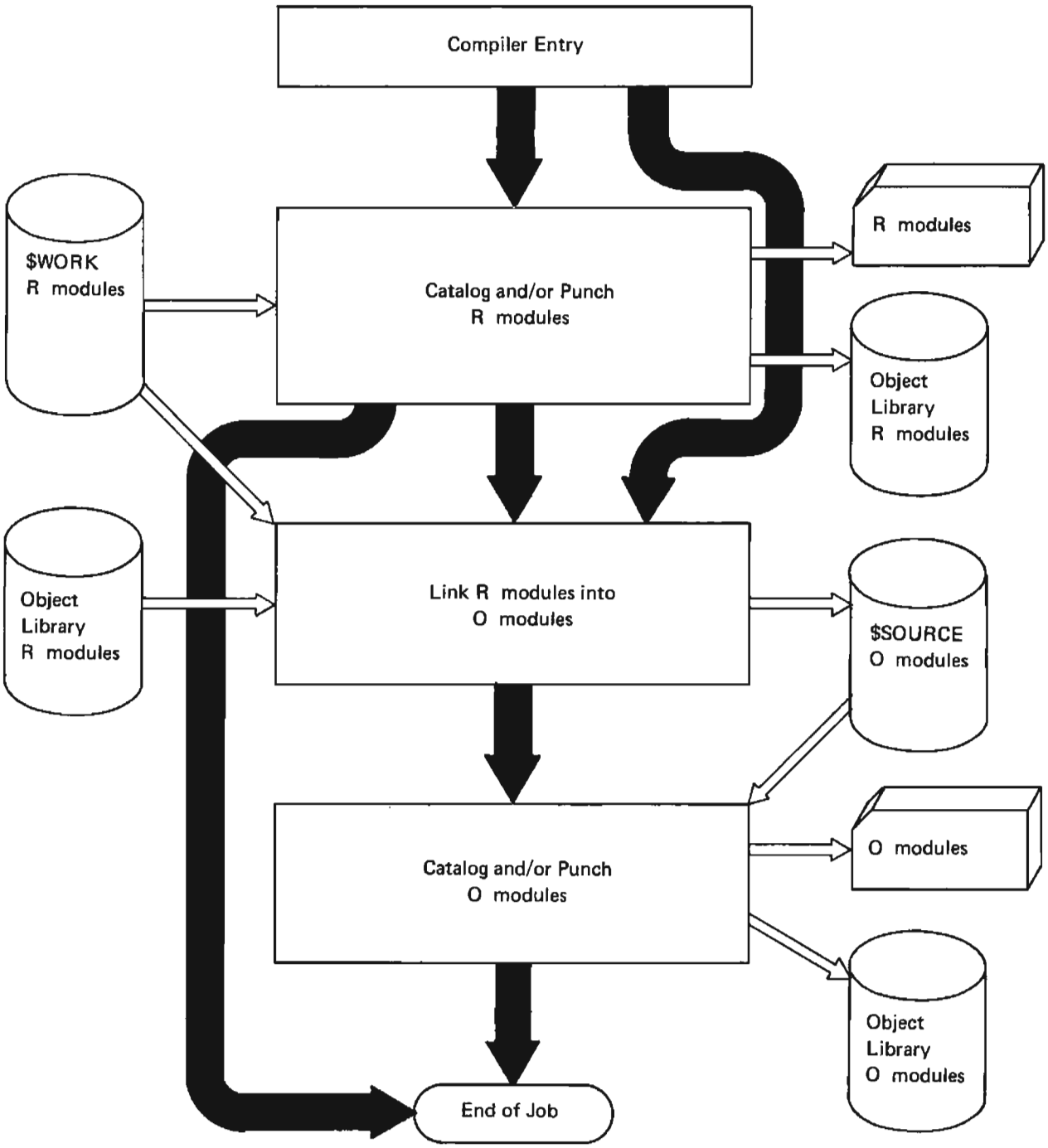


Figure 1. Overview of Overlay Linkage Editor Compiler Entry

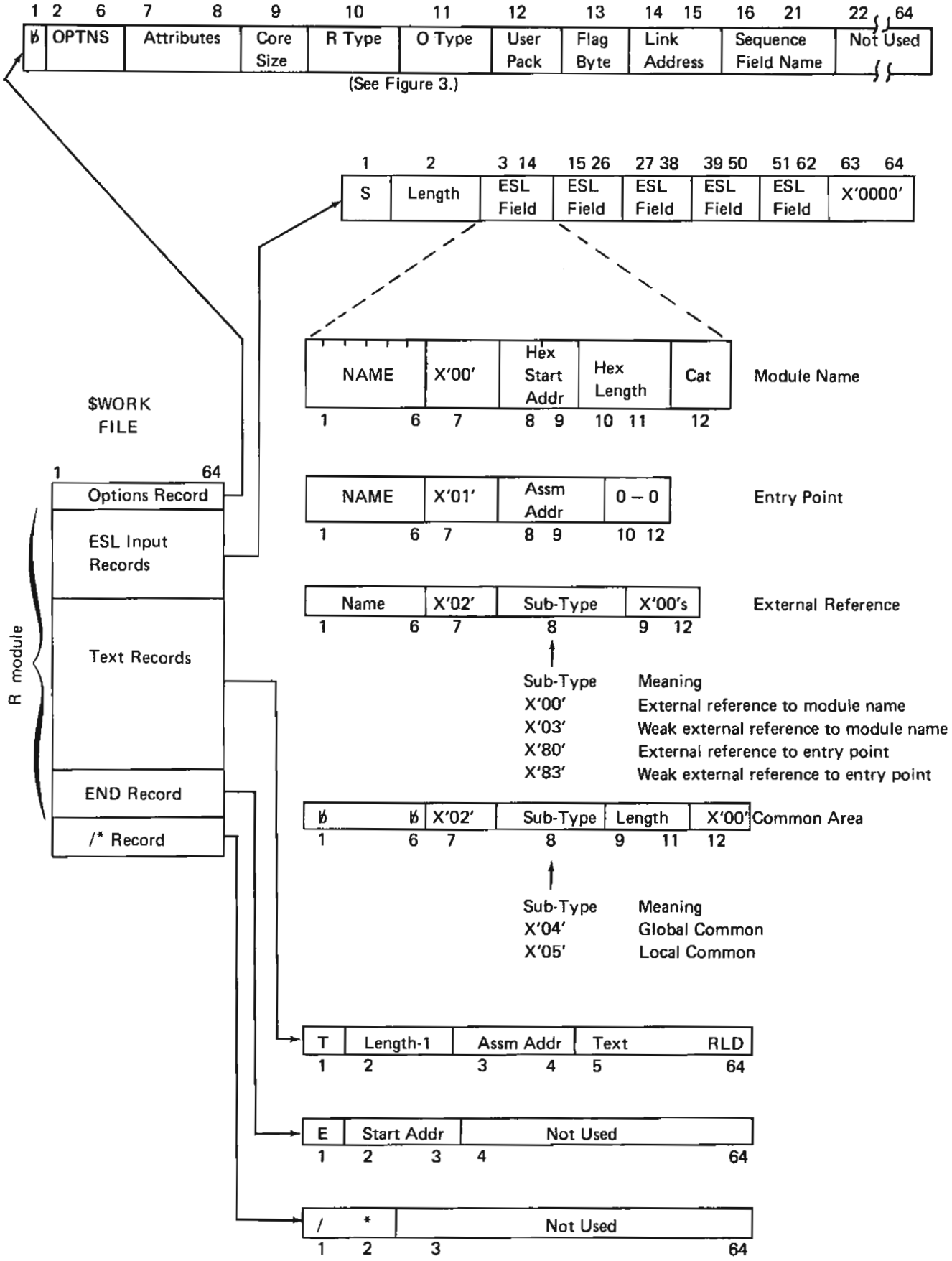


Figure 2. Input to Compiler Entry

1	6	7	8	9	10	11	12	13	14	15	16	21	22	64
b	OPTNS	Attributes		Core Size	R Type	O Type	User Pack	Flag byte	Link Address		Sequence Field Name		Not Used	

Attributes: This two-byte field describes the linked object program built by the Overlay Linkage Editor.

	Byte 1	Byte 2	
Bit	01234567	01234567	
	1xxxxxxx	xxxxxxxx	Permanent entry
	0xxxxxxx	xxxxxxxx	Temporary entry
	x1xxxxxx	xxxxxxxx	Inquiry
	xx1xxxxx	xxxxxxxx	Inquiry Evoking
	xxx1xxxx	xxxxxxxx	Must run dedicated
	xxxx1xxx	xxxxxxxx	Requires source
	xxxxx1xx	xxxxxxxx	Deferred mounting
	xxxxxx1x	xxxxxxxx	PTF applied (not used by Overlay Linkage Editor)
	xxxxxxx1	xxxxxxxx	Overlay program
	xxxxxxxx	1xxxxxxx	System input dedication
	xxxxxxxx	x1xxxxxx	Checkpoint program
	xxxxxxxx	xx1xxxxx	Direct source read
	xxxxxxxx	xxx1xxxx	Macro processor allowed
	xxxxxxxx	xxxx1xxx	Reserved
	xxxxxxxx	xxxxx1xx	FORTRAN Common
	xxxxxxxx	xxxxxx1x	Reserved
	xxxxxxxx	xxxxxxx1	Reserved

Main Storage Size: This is the amount of storage (in 1/4K increments) necessary for execution of the object program.

Example = Hex'12' = 18(Hex'12') X 256 (1/4K) = 4608 Bytes

R-Type: This byte specifies the disposition of the R. module in $WORK.

Bit	01234567	
	1xxxxxxx	Punch into cards
	x1xxxxxx	Catalog into object library on R1
	xx1xxxxx	Catalog into object library on R2
	xxx1xxxx	Catalog into object library on F1
	xxxx1xxx	Catalog into object library on F2
	xxxxx1xx	Catalog as Retain-R in library
	xxxxxx1x	Catalog as permanent entry in library
	xxxxxxx1	Catalog into object library on program pack
	00000000	No R module

Figure 3 (Part 1 of 2). Options Record

O-Type: This byte specifies the disposition of the linked object program and the type of printed output from the Overlay Linkage Editor.

Bit	01234567	
	1xxxxxxx	Punch object program into cards
	x1xxxxxx	Catalog object program into object library on R1
	xx1xxxxx	Catalog object program into object library on R2
	xxx1xxxx	Catalog object program into object library on F1
	xxxx1xxx	Catalog object program into object library on F2
	xxxxx1xx	Do not print core map
	xxxxxx1x	Do not print cross-reference list
	xxxxxxx1	Catalog object program into object library on program pack
	00000000	No linked output. (If neither an R or an O module is specified, an O is cataloged on the program pack.)

User Pack: This byte specifies the pack where user routines are stored. The Overlay Linkage Editor will search this pack first when resolving EXTRNs to modules whose names do not begin with $. If the EXTRN name is not found on this pack, the program pack is searched.

Bit	01234567	
	1xxxxxxx	Reserved
	x1xxxxxx	Search R1
	xx1xxxxx	Search R2
	xxx1xxxx	Search F1
	xxxx1xxx	Search F2
	xxxxx111	Reserved

Flag Byte: This byte passes information to the Overlay Linkage Editor.

Bit	01234567	
	11110000	Reserved
	000000x1	Link edit address in bytes 14 and 15
	0000001x	Catalog Load Module as Retain-R in library
	000001xx	Sequence field name given in bytes 16 through 21
	00001xxx	Print messages

Link Address: These two bytes specify a link edit address. If bit 7 of the flag byte is not on, the Overlay Linkage Editor links the load module to the end of the supervisor.

Sequence Field Name: Name put in sequence field. If not specified, ESL will be used.

Figure 3 (Part 2 of 2). Options Record

Output From User Entry

Output of the Overlay Linkage Editor for user entry is an object program cataloged into an object library and/or punched into cards (Figure 4).

A storage map and cross reference list are printed depending on the MAP parameter of the // OPTIONS card.

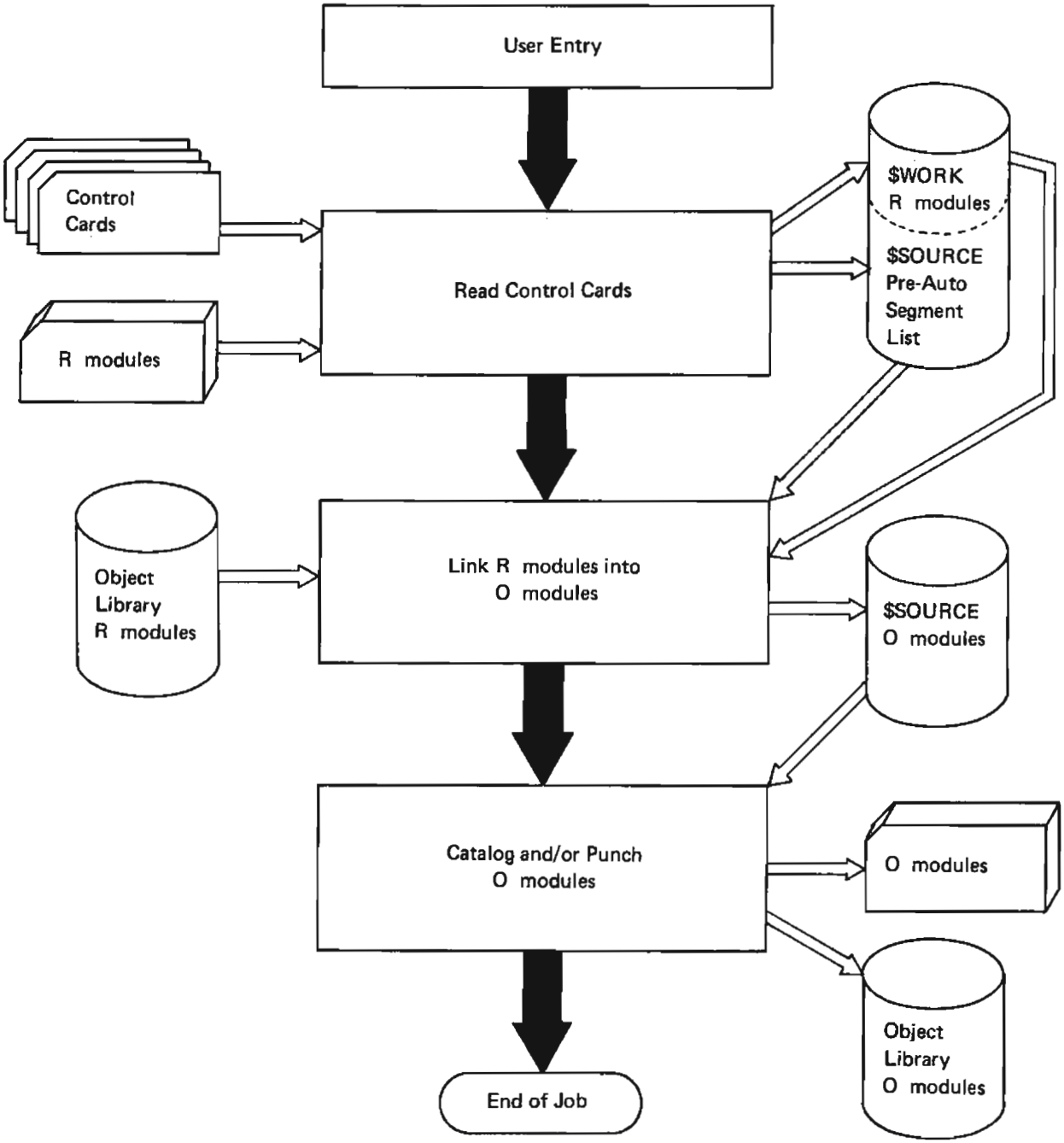


Figure 4. Overview of Overlay Linkage Editor User Entry

The Overlay Linkage Editor is divided into self-overlaying routines. The sequence in which routines are loaded and which routines are used depends on whether the compiler entry or the user entry is used and which functions are requested. Figure 5 shows an operational diagram of the Overlay Linkage Editor program. Storage maps of the compiler interface and the user interface are shown in Figures 6 and 7.

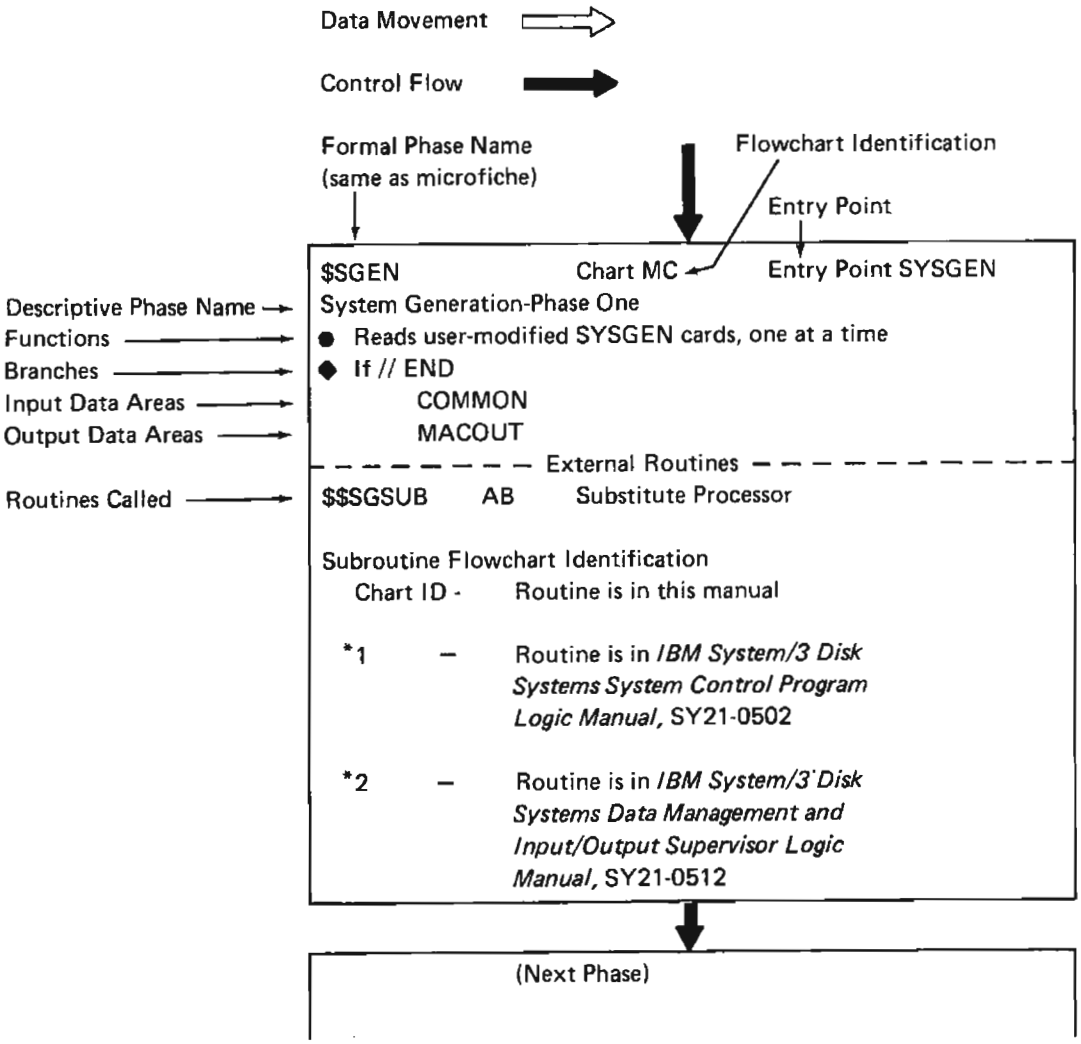
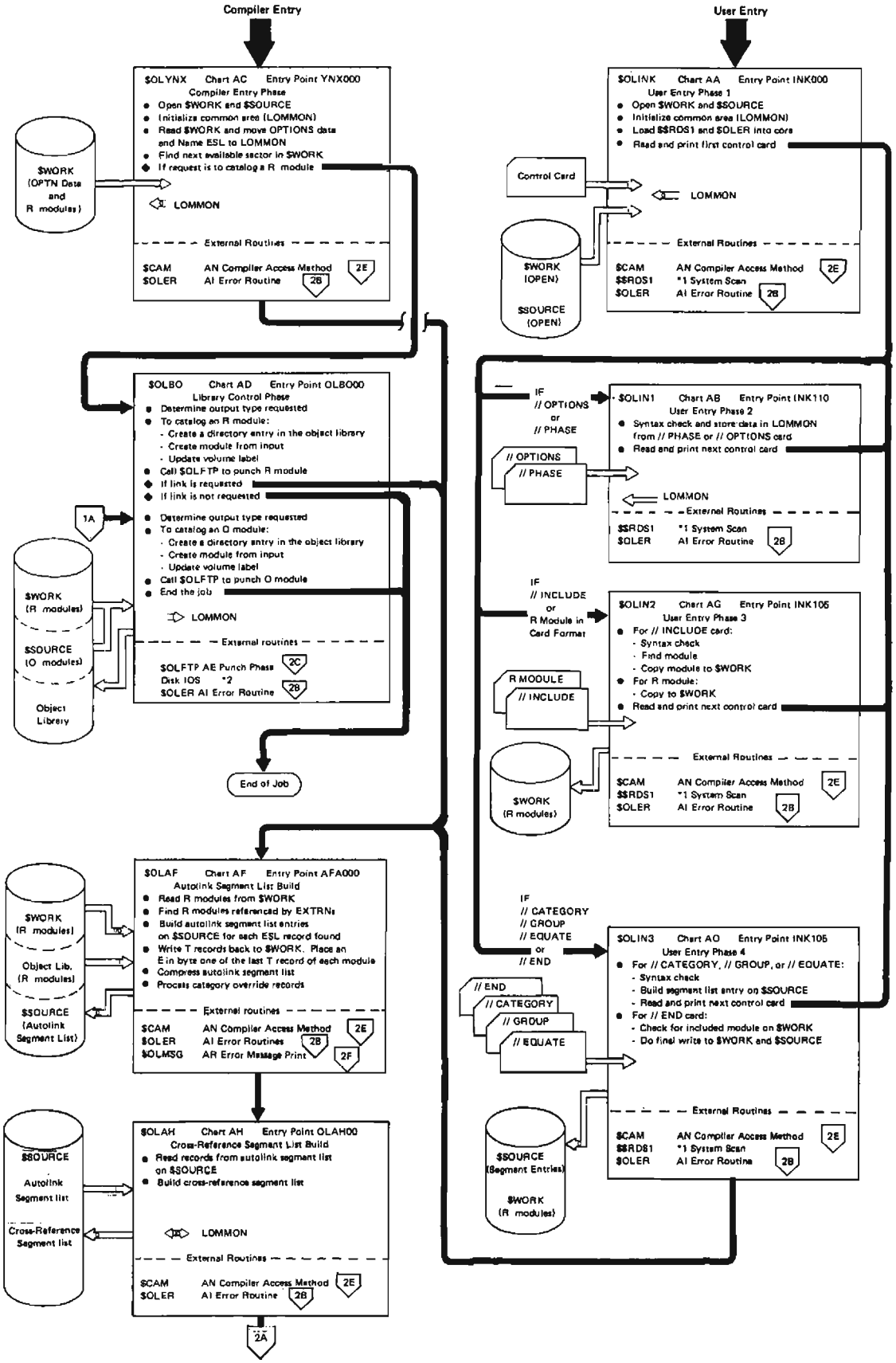
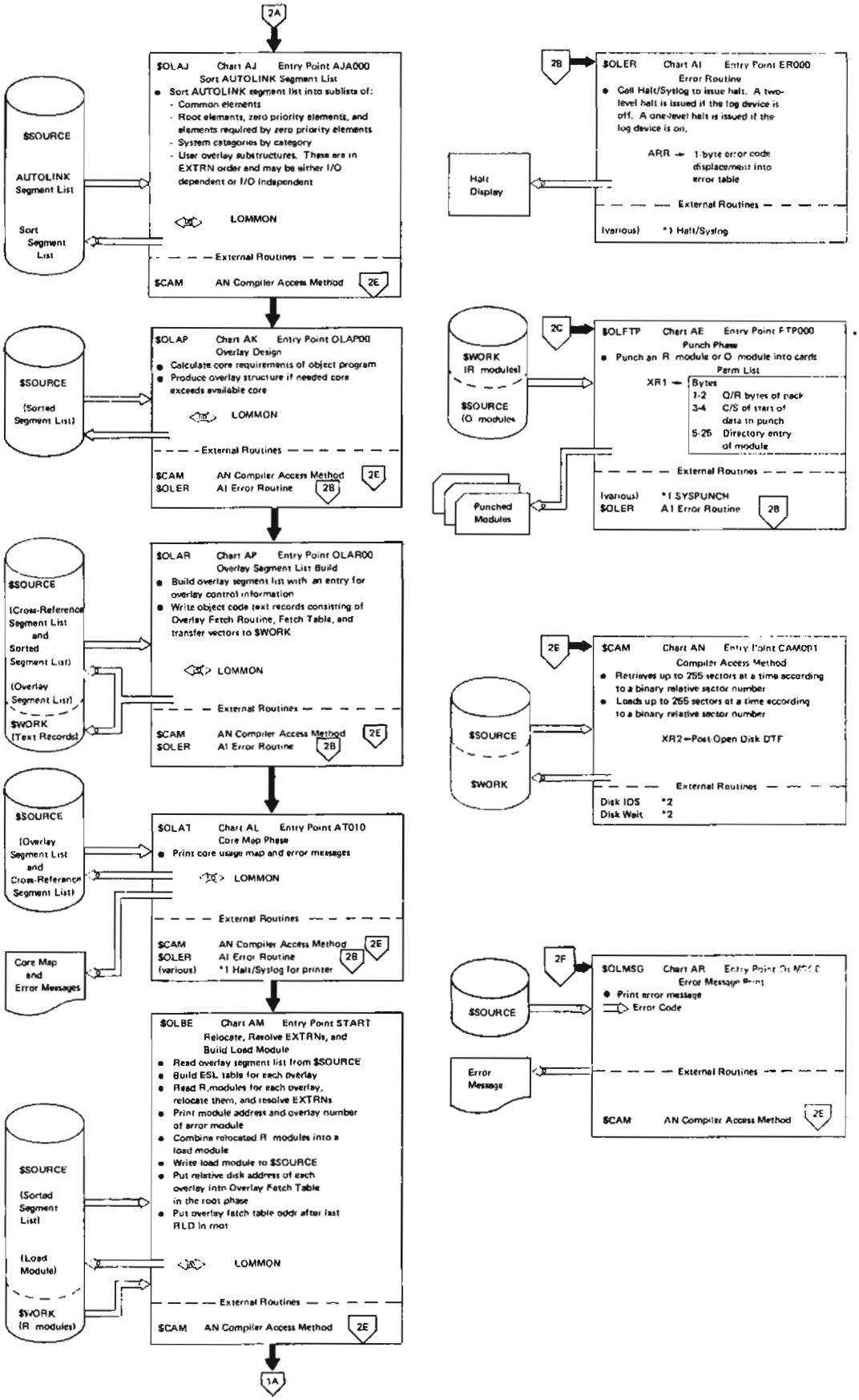


Figure 5 (Part 1 of 3). Operational Diagram Legend



- Figure 5 (Part 2 of 3). Operational Diagram



● Figure 5 (Part 3 of 3). Operational Diagram

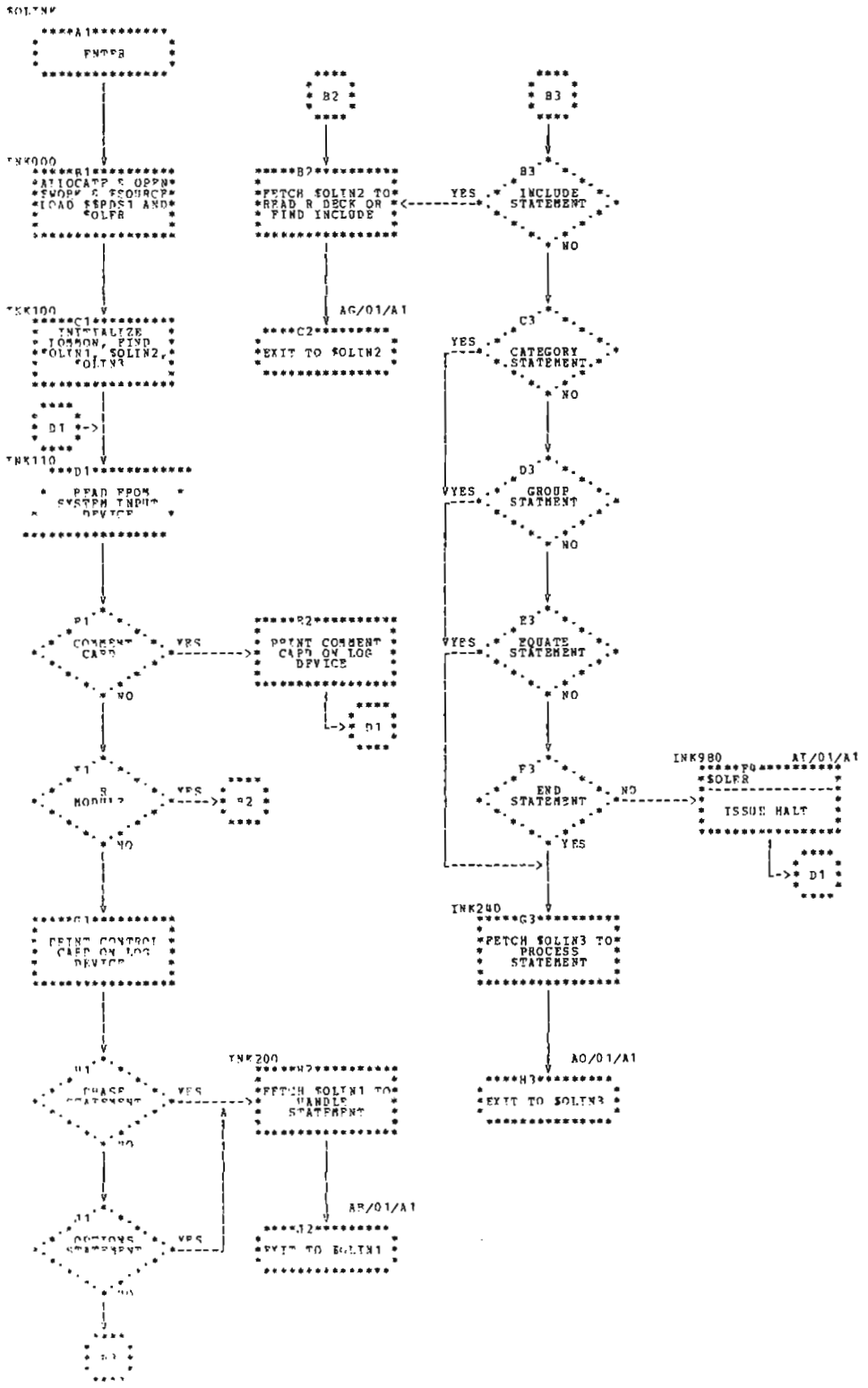


Chart AA. User Entry Phase 1 ($OLINK)

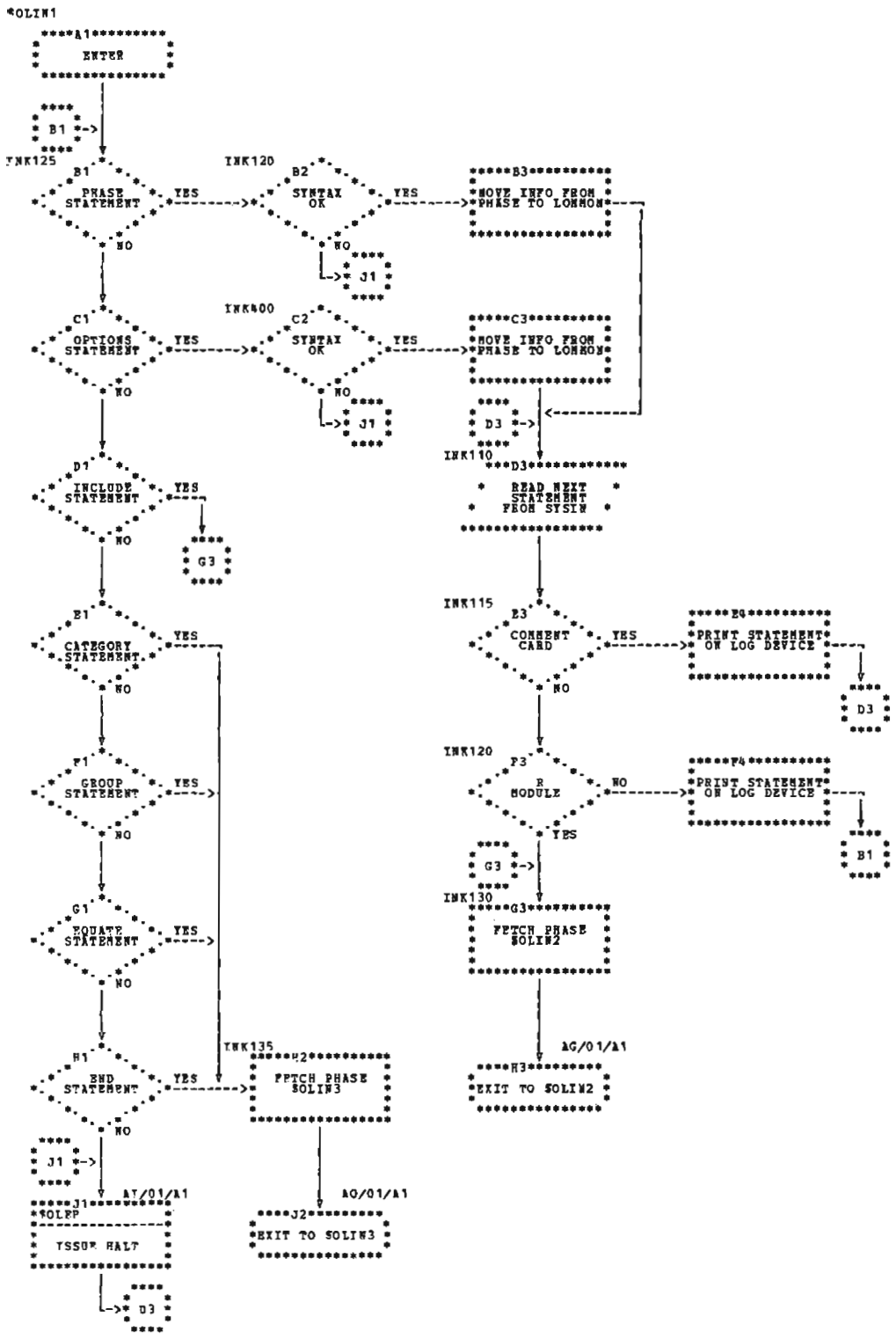


Chart AB. User Entry Phase 2 ($OLIN1)

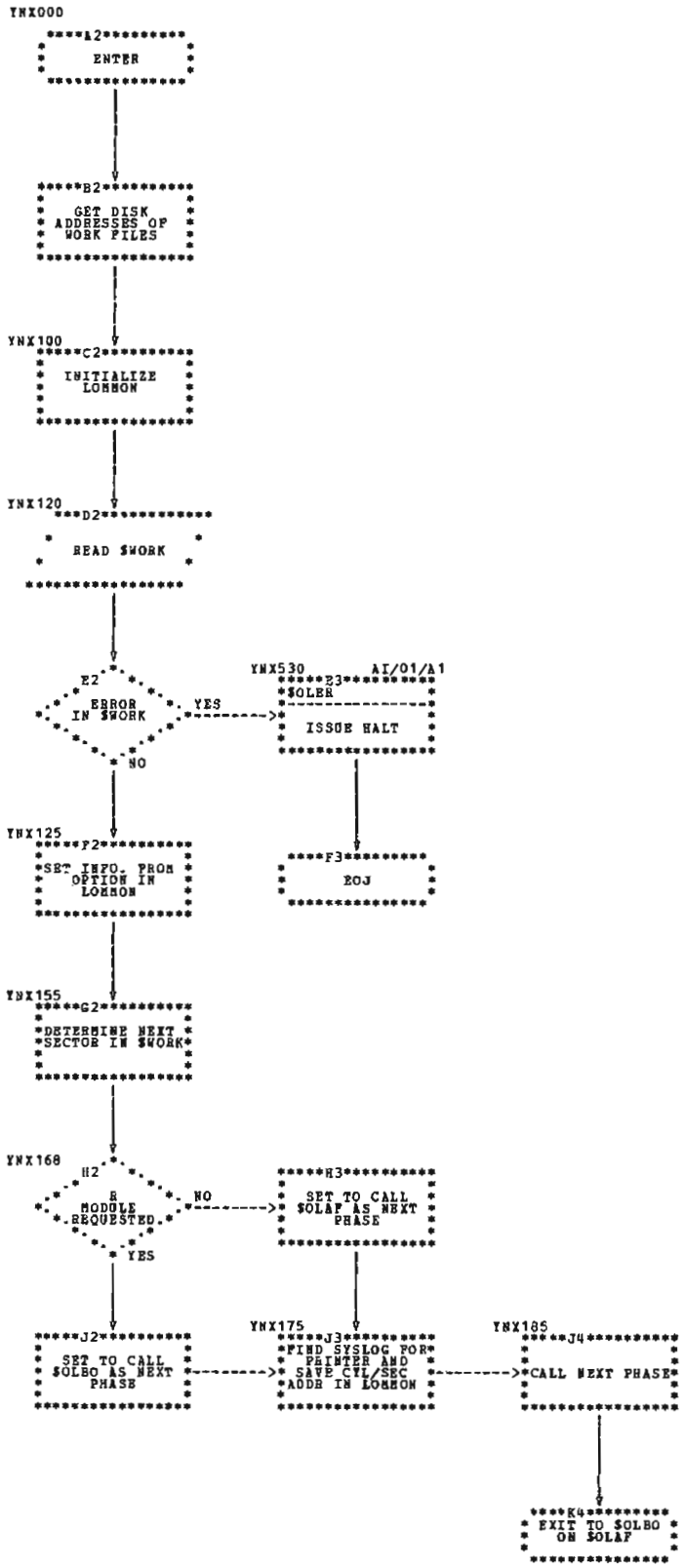


Chart AC. Compiler Entry Phase ($OLYNX)

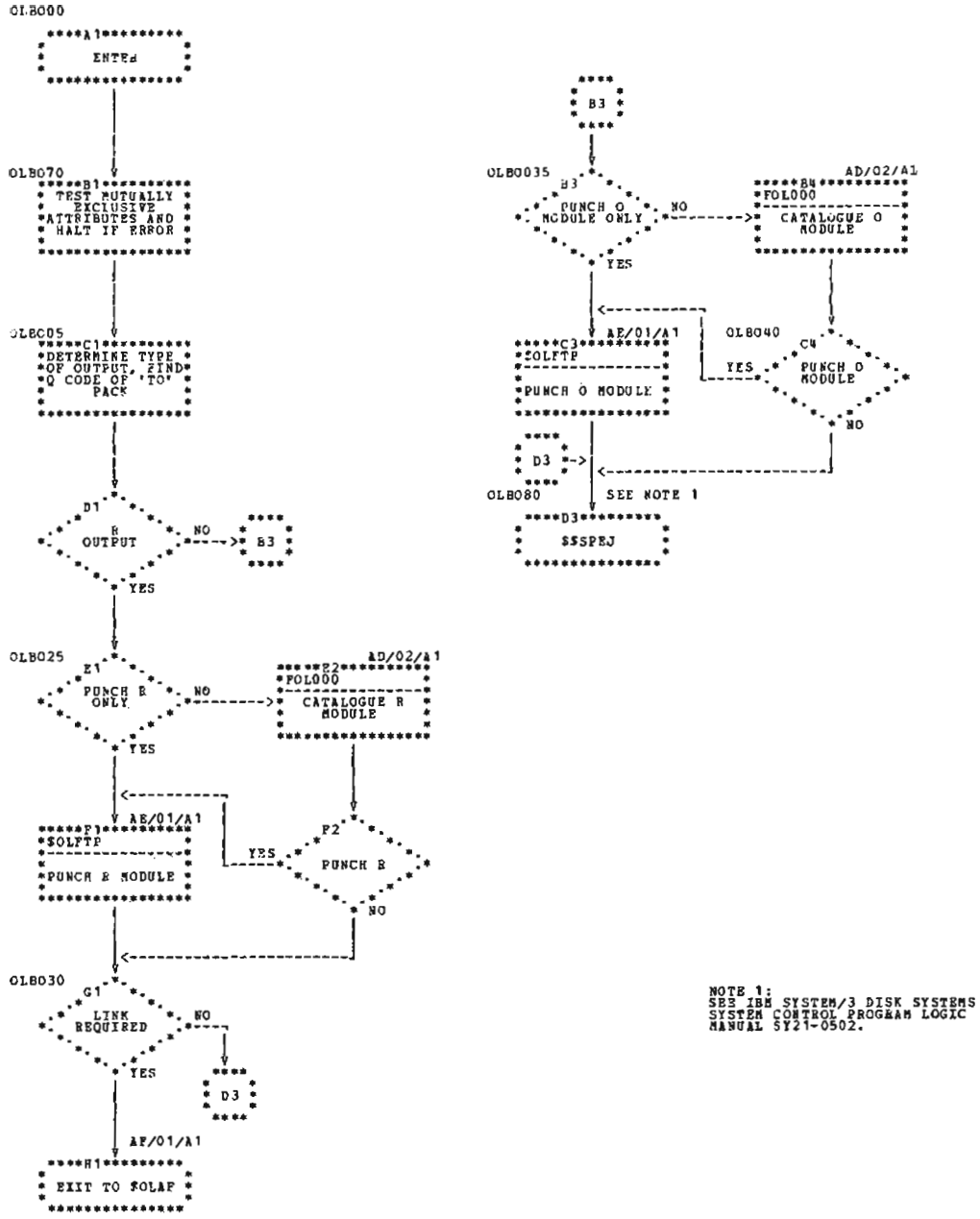


Chart AD (Part 1 of 2). Library Control Phase ($OLBO)

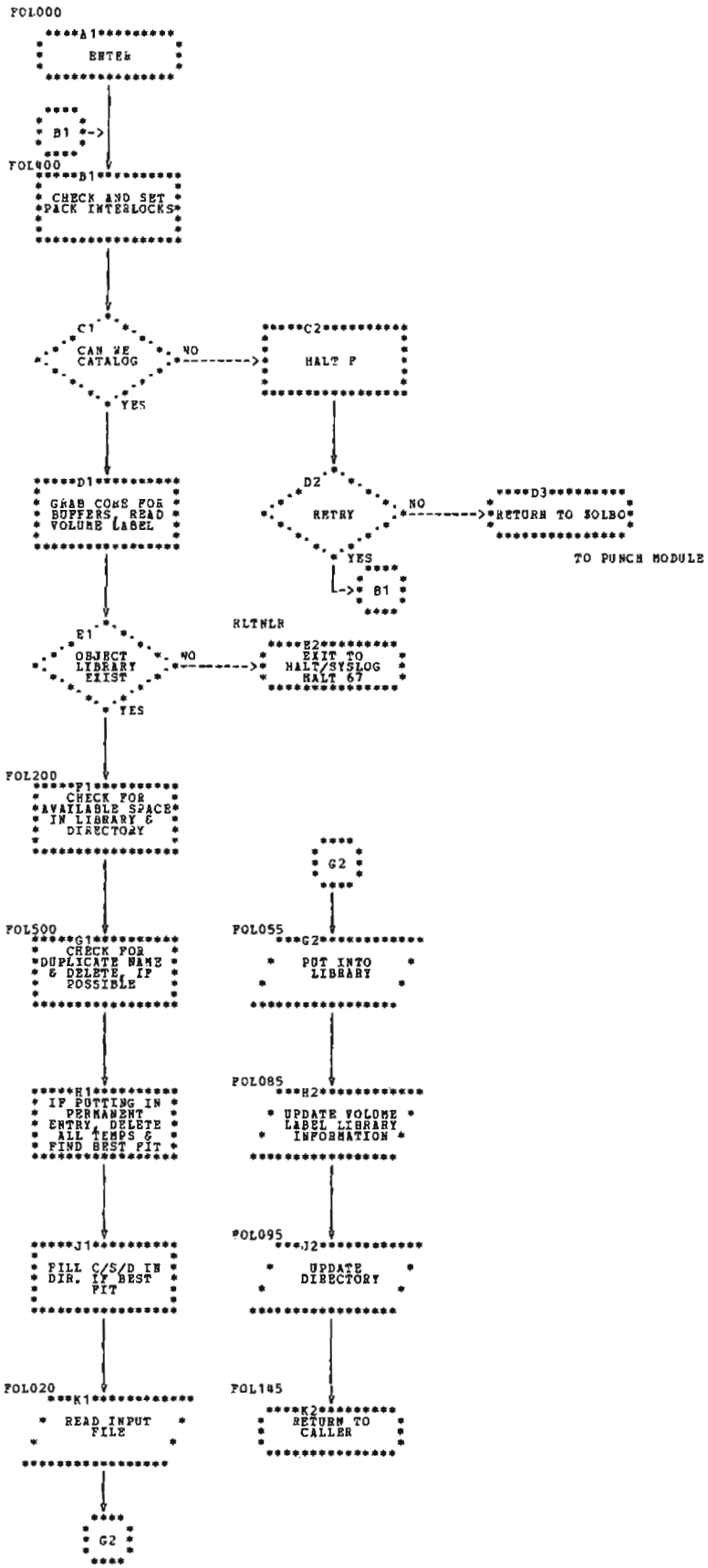


Chart AD (Part 2 of 2). Library Control Phase ($OLBO)

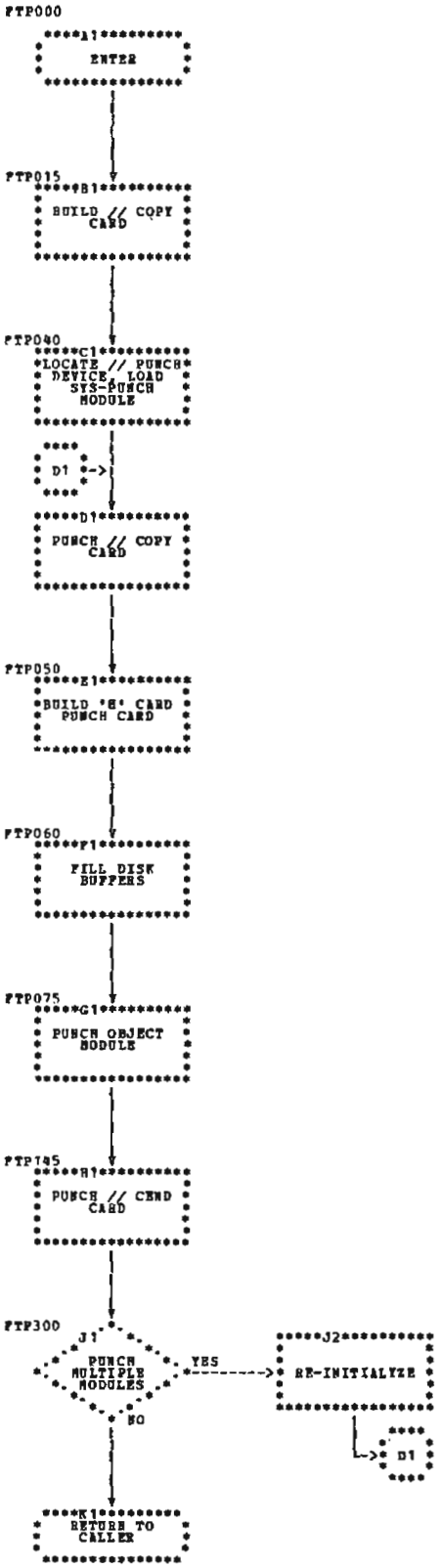
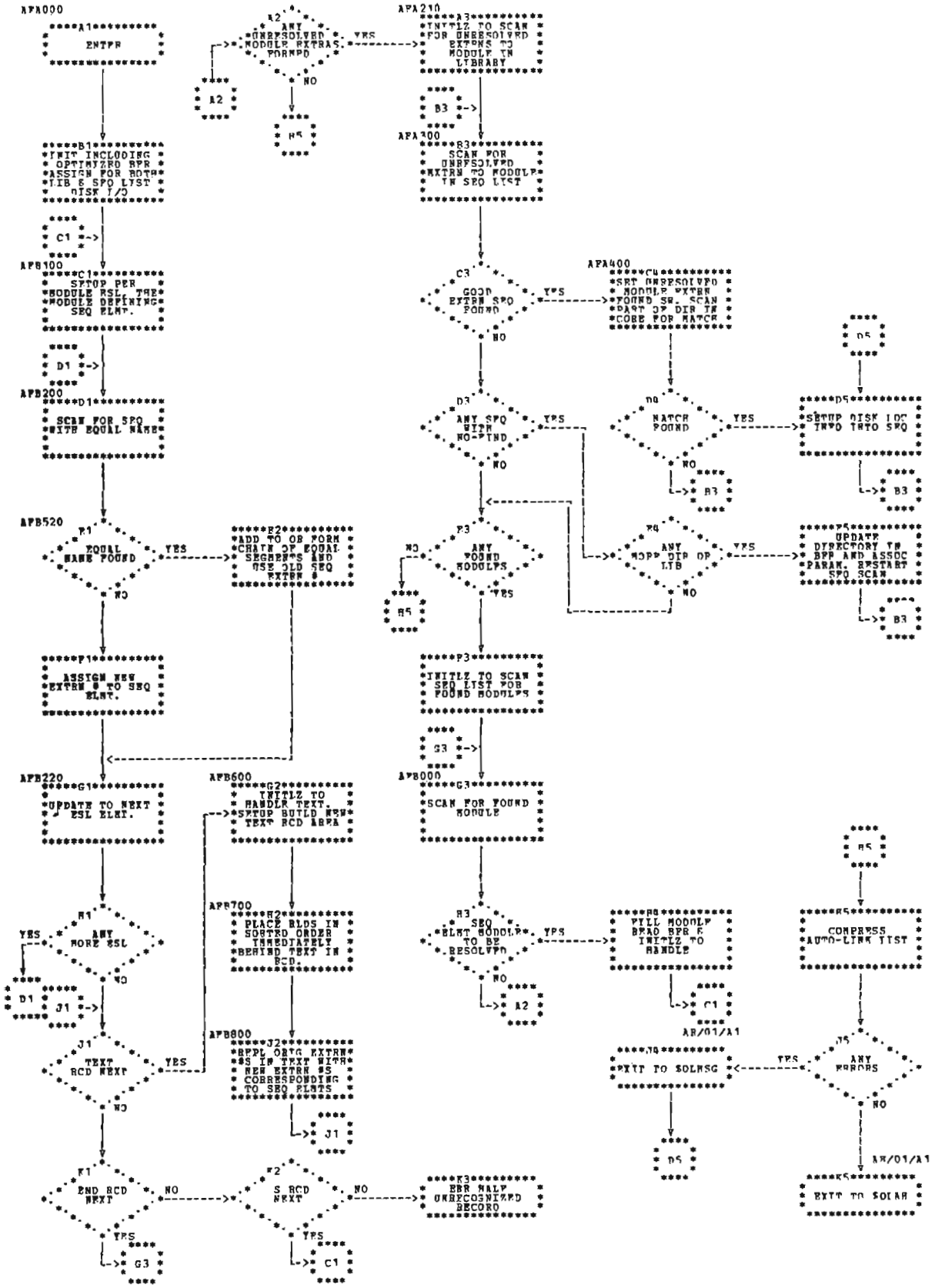
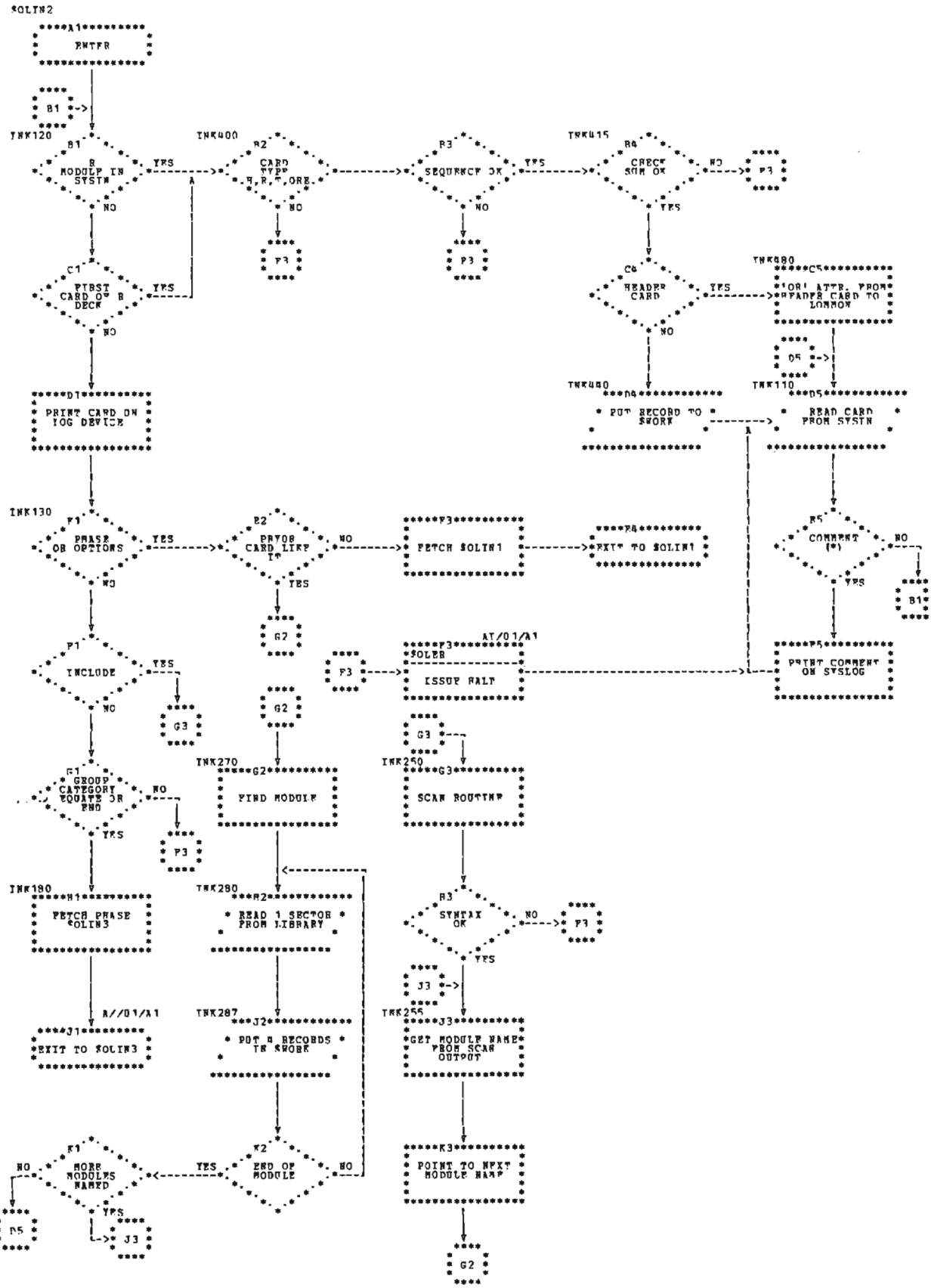


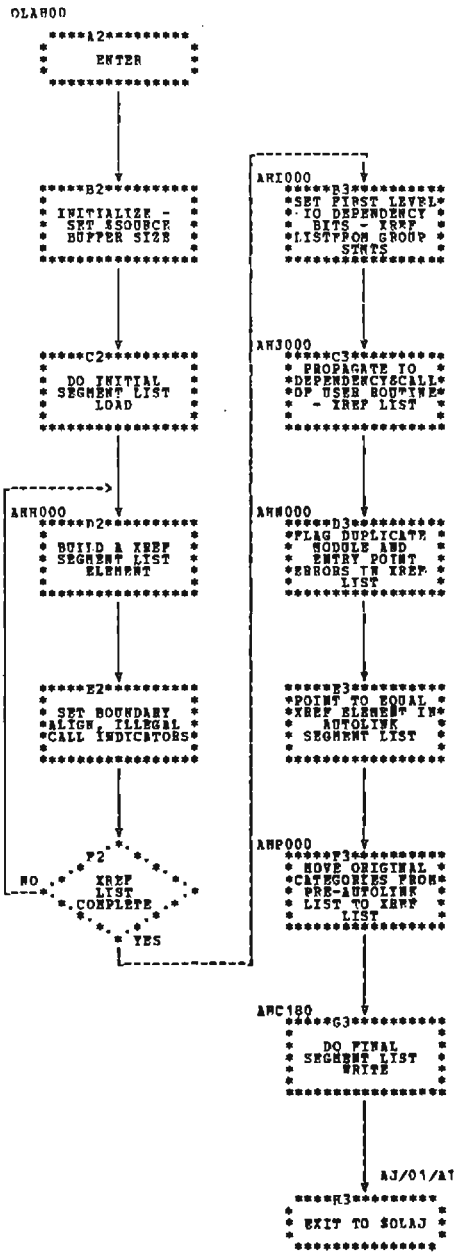
Chart AE. Punch Phase ($OLFTP)



● Chart AF. Auto-Link Segment List Build ($OLAF)



● Chart AG. User Entry Phase 3 ($OLIN2)



● Chart AH. Cross-Reference Segment List Build ($OLAH)

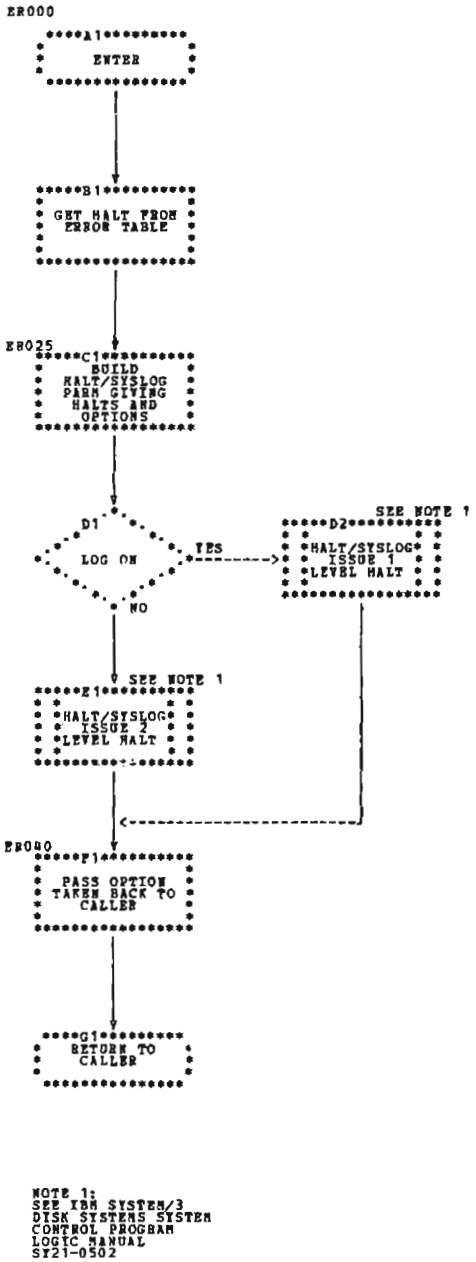
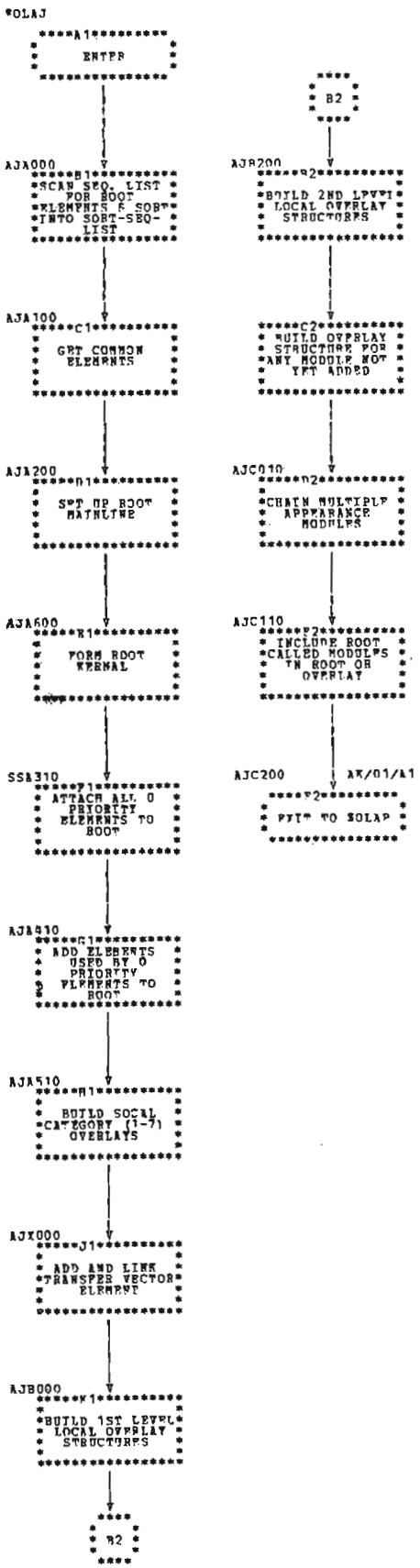


Chart A1. Error Routing ($OLER)



● Chart AJ. Sort Auto-Link Segment List ($OLAJ)

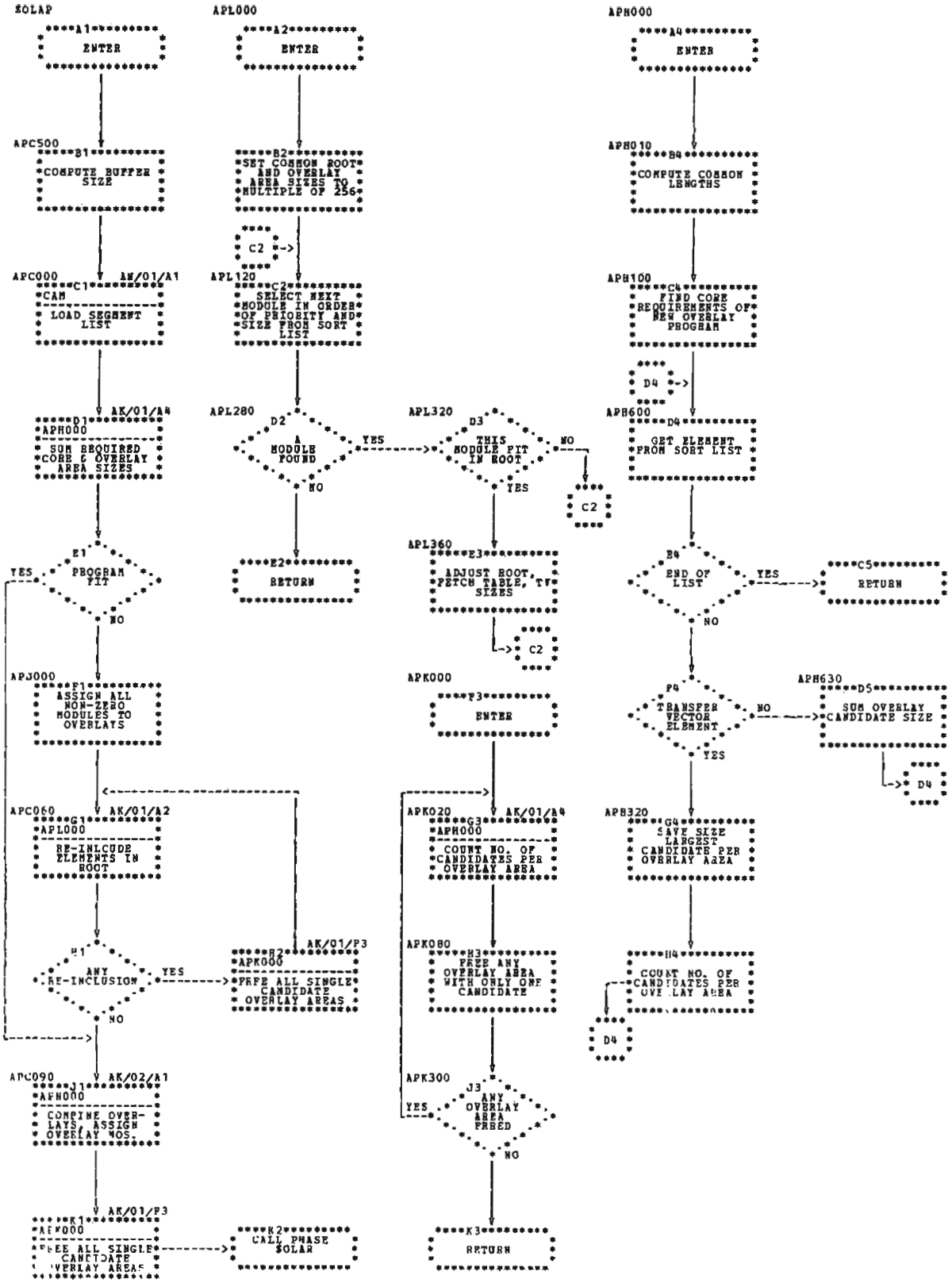


Chart AK (Part 1 of 2). Overlay Design ($OLAP)

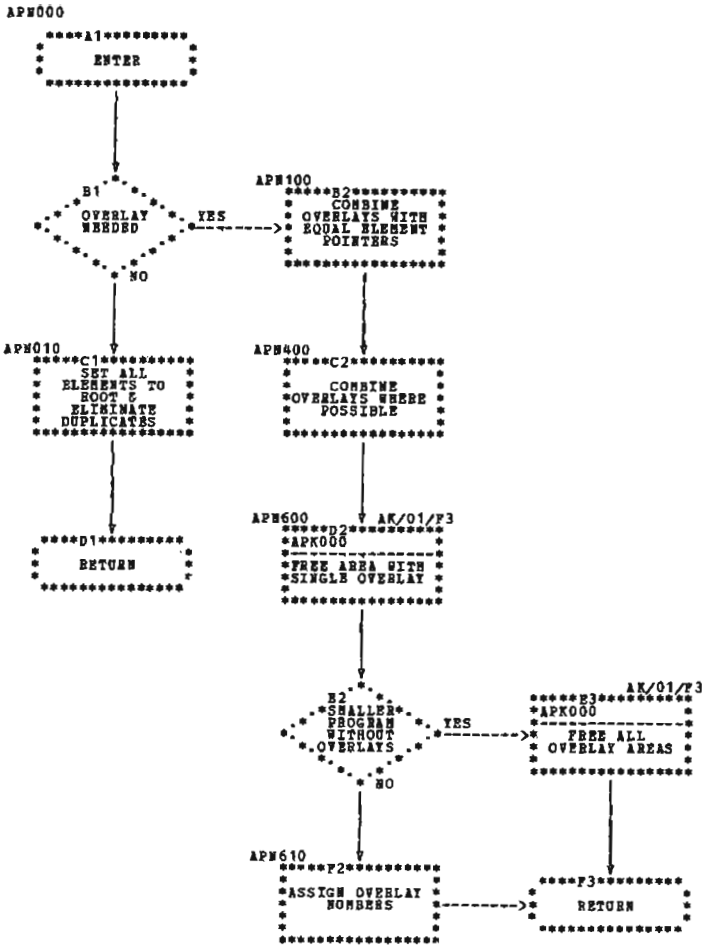


Chart AK (Part 2 of 2). Overlay Design ($OLAP)

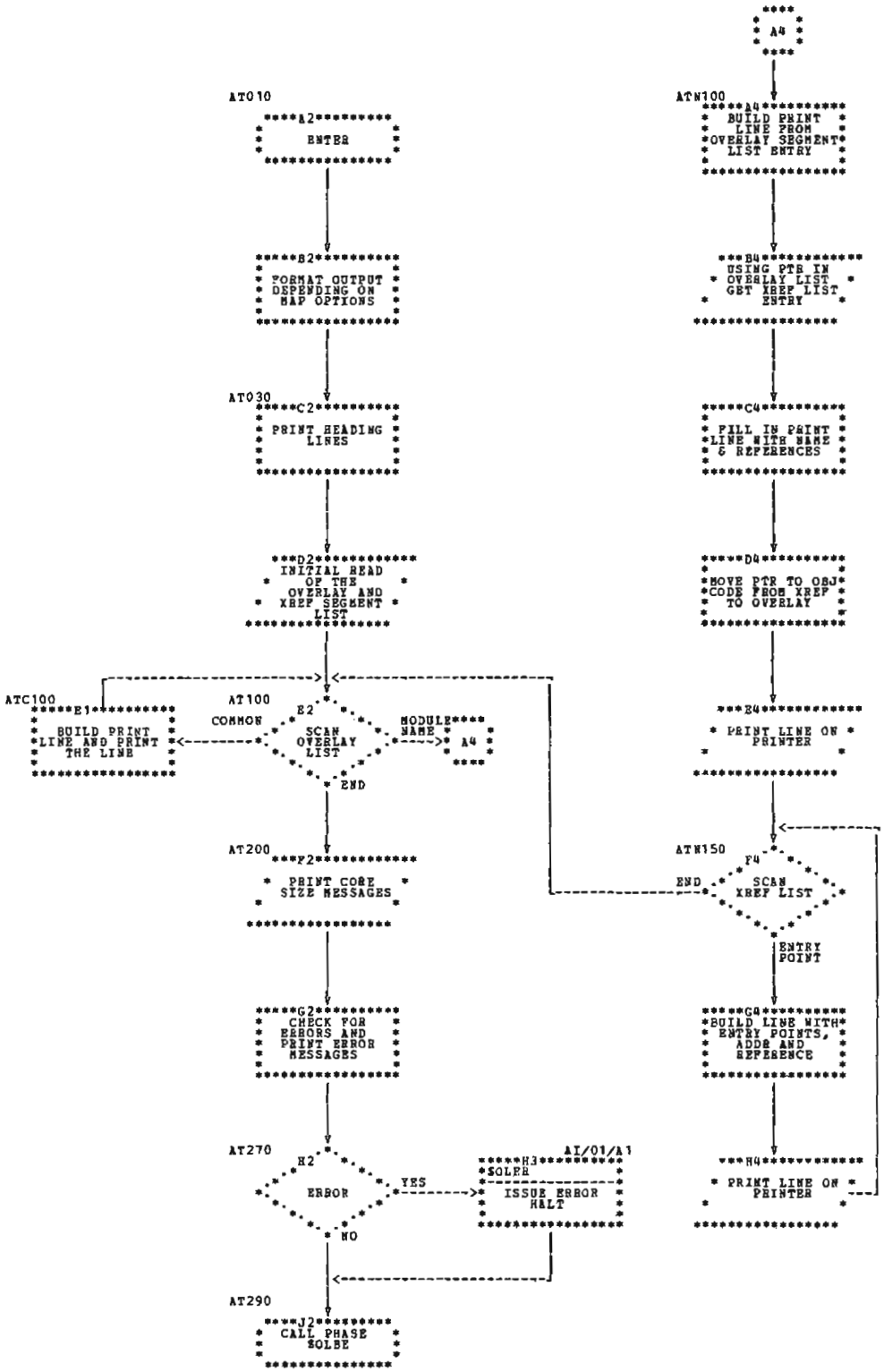


Chart AL. Core Map Phase ($OLAT)

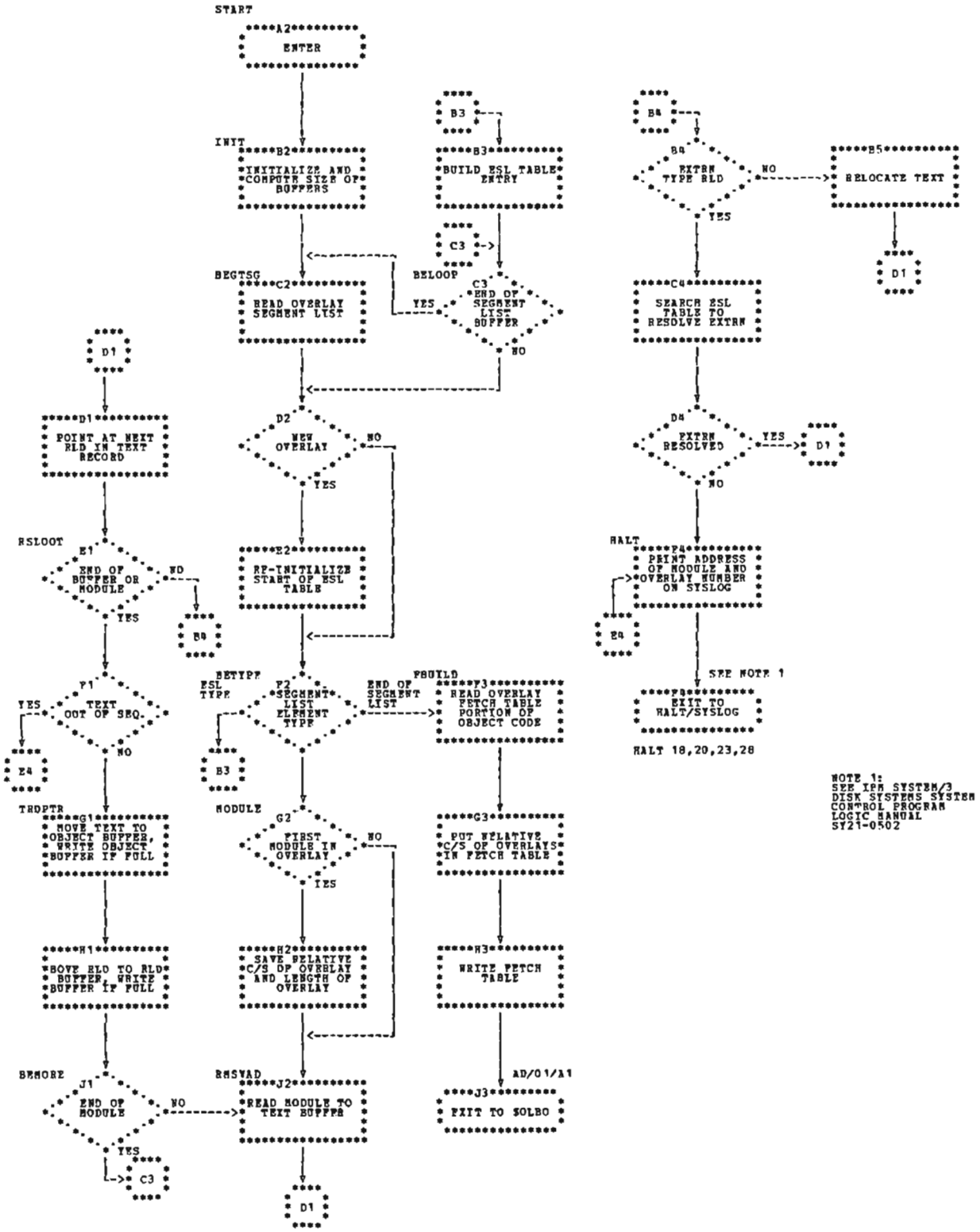


Chart AM. Relacate, Resolve EXTRNs, and Build Load Module ($OLBE)

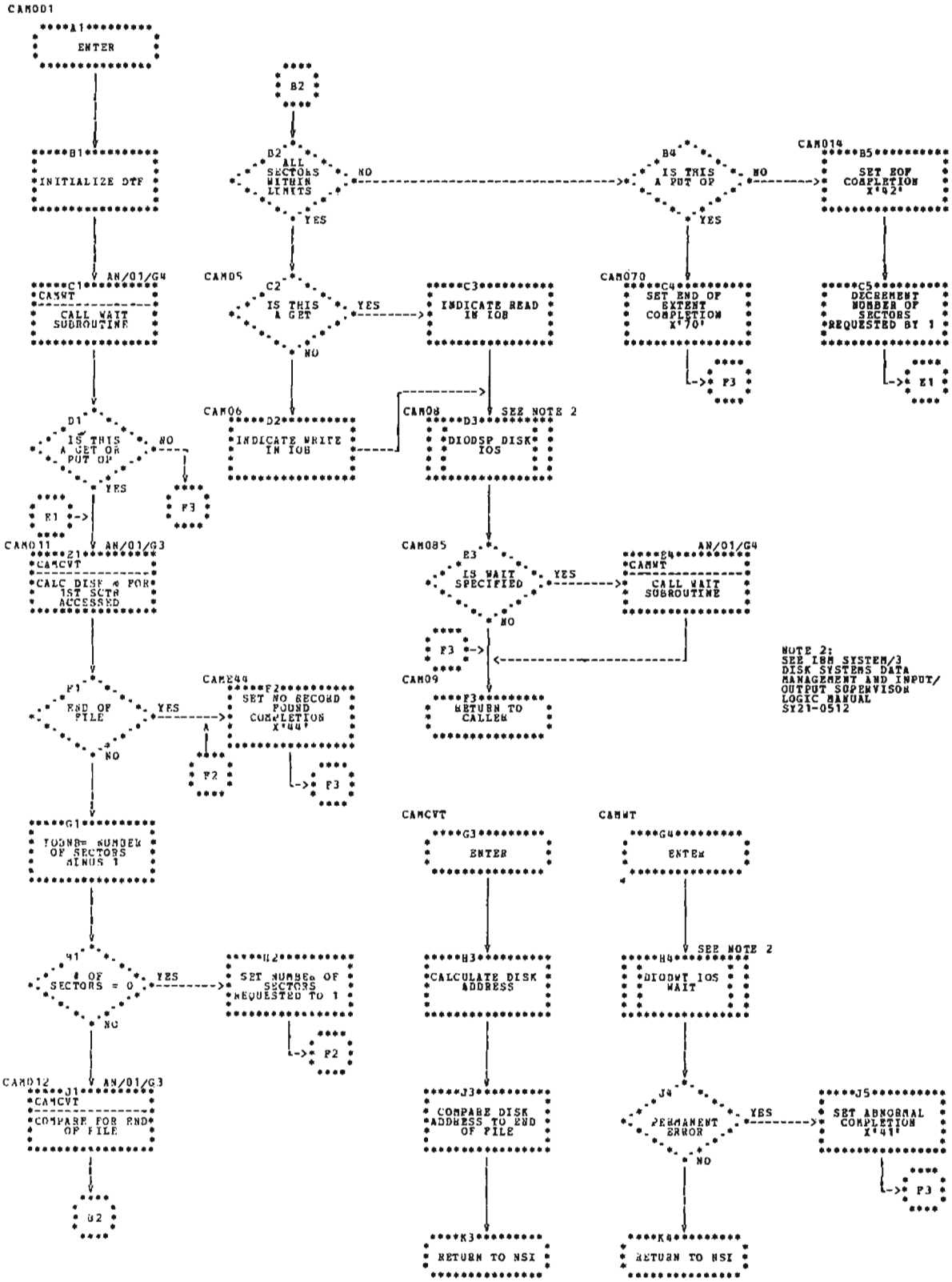


Chart AN. Complier Access Method ($CAM)

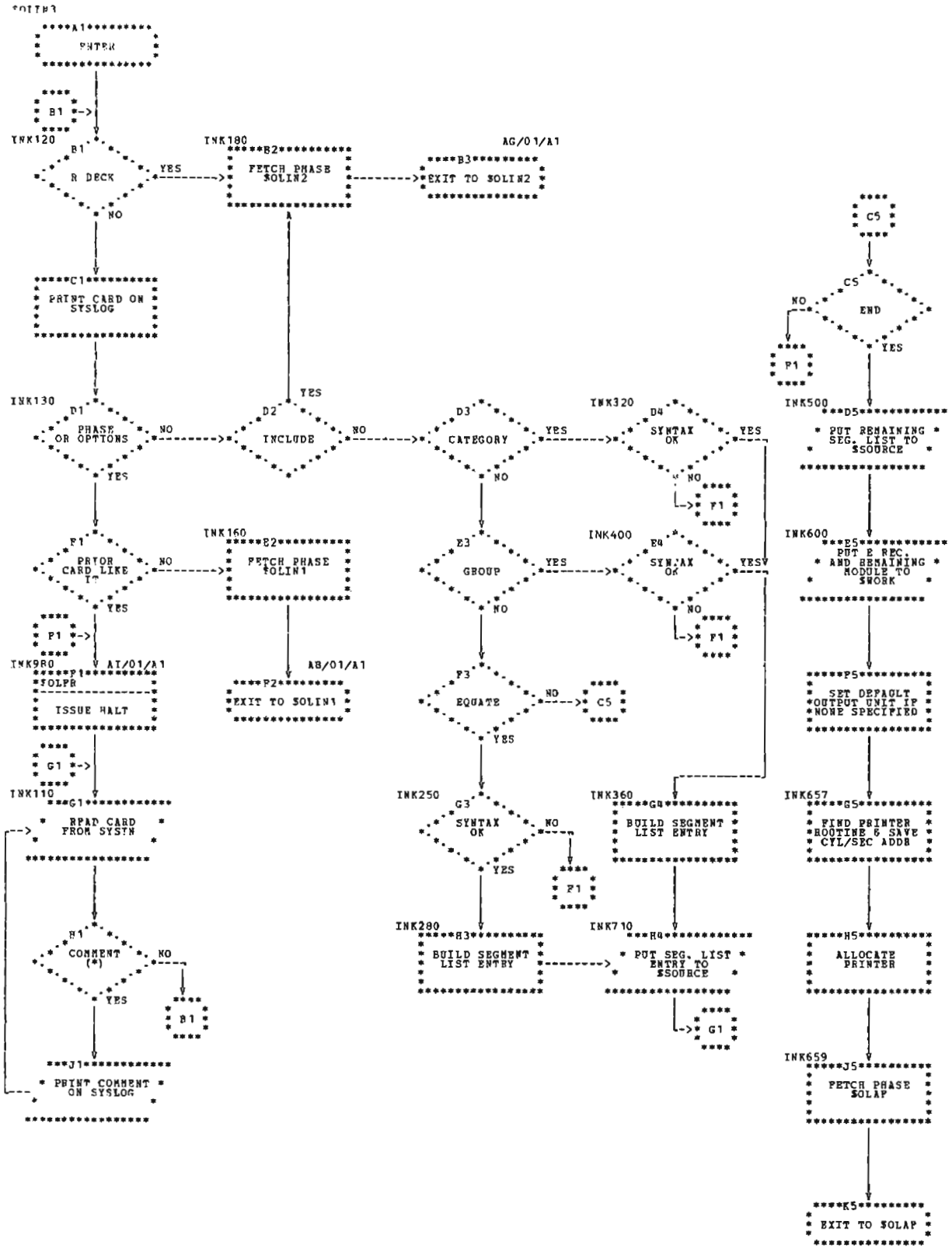


Chart AO. User Entry Phase 4 ($OLIN3)

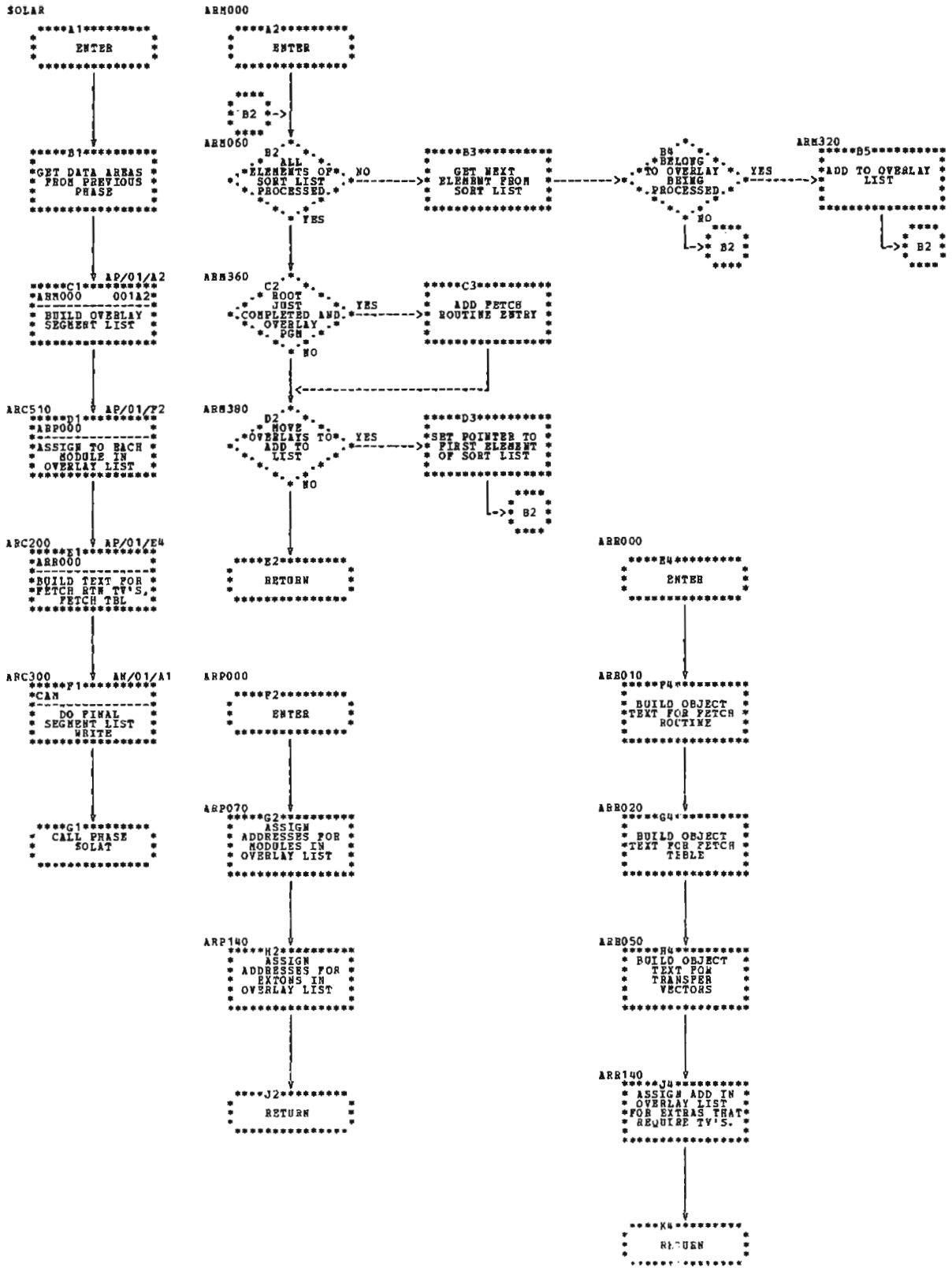
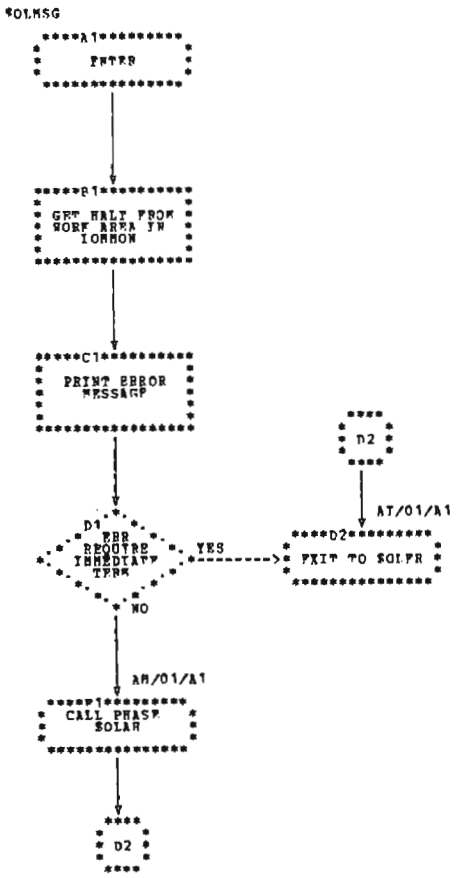


Chart AP. Overlay Segment List Build ($OLAR)



● Chart AR. Error Message Print Phase ($OLMSG)

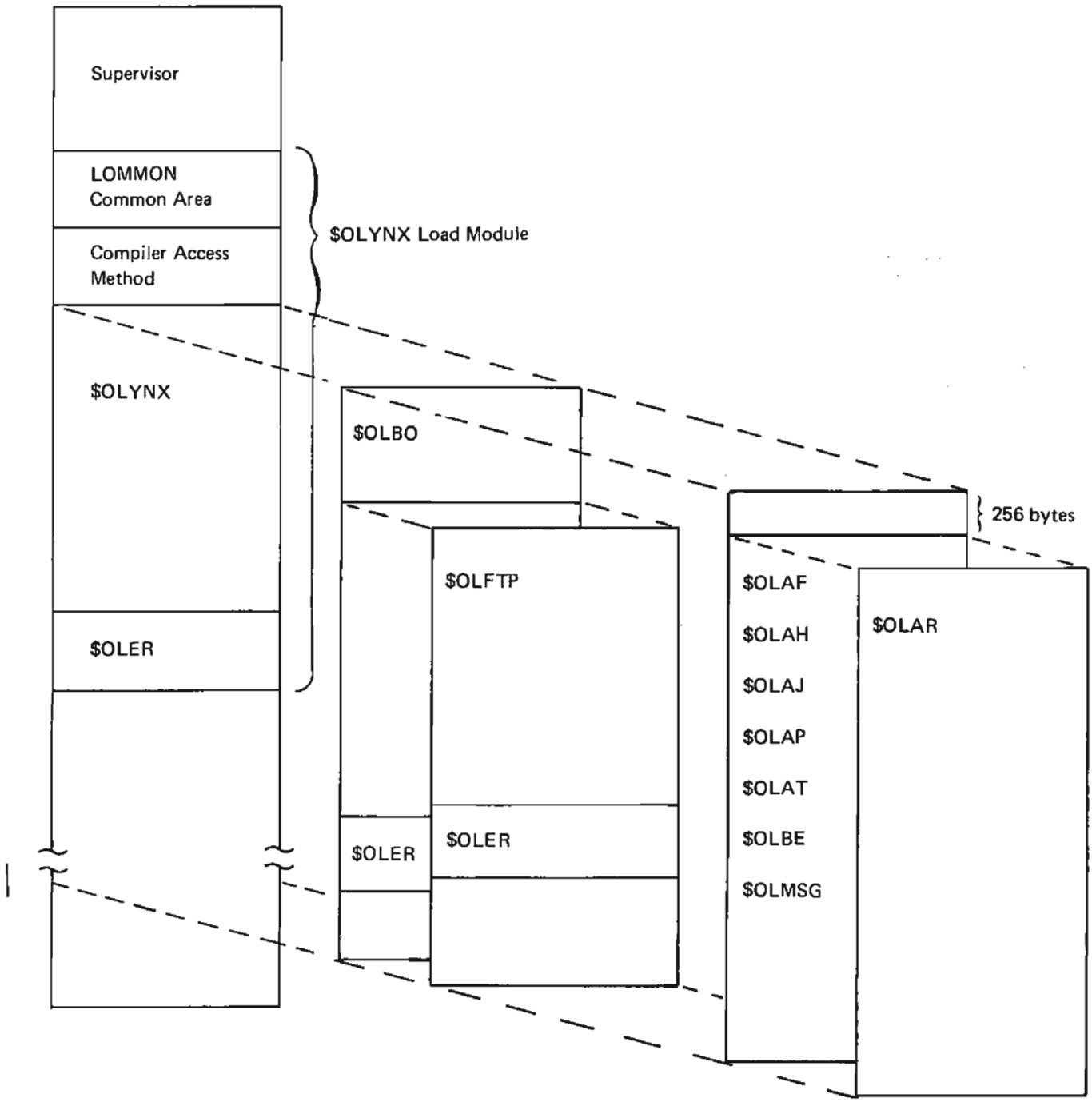


Figure 6. Compiler Entry Storage Map

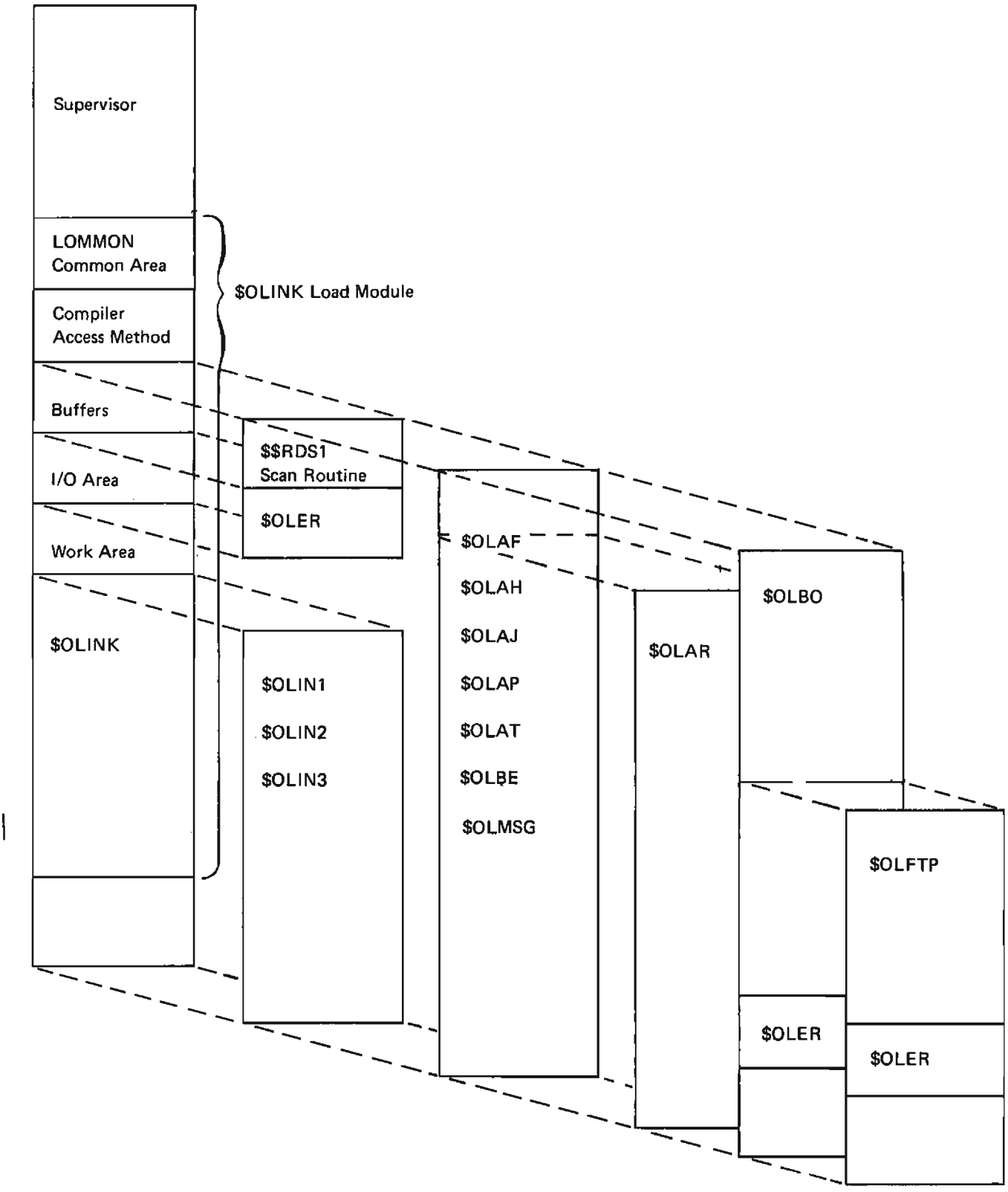


Figure 7. User Entry Storage Map

This section describes the data areas that pass information between routines of the Overlay Linkage Editor.

OVERLAY LINKAGE EDITOR COMMON (LOMMON)

The Overlay Linkage Editor common area (Figure 8) passes control information between the various routines. All of LOMMON, except for DTFs and IOBs, is initially set to zero by $OLINK or $OLYNX.

SEGMENT LIST ENTRIES

The various routines of the Overlay Linkage Edition build a series of segment lists. These segment lists are built in the $SOURCE work file (Figure 9). Each entry is 16 bytes long. The format of entries varies between and within segment lists depending on the type of entry. See Figures 10 through 14. Because all data fields in the segment list entries are not used for all types of entries, the column headed 'Applies to Segment Type' indicates which types of segment list entry will contain the data. Figure 11 lists all the segment types.

Displacement Hex	Displacement Decimal	Label	Length in Bytes	Routines that Change Data ($OLxxx)	Description
0	0	LODTFW	52	All	DTF for $WORK (see note)
34	52	LOIOBW	23	All	IOB for $WORK
4B	75	LODTFS	52	All	DTF for $SOURCE (see note)
7F	127	LOIOBS	23	All	IOB for $SOURCE
96	150	LORTYP	1	YNX, B0	R module information *Value* *Description* X'80' Punch R module X'40' R module to R1 X'20' R module to R2 X'10' R module to F1 X'08' R module to F2 X'04' Replace as a permanent entry X'02' Permanent entry X'01' R to program pack X'00' No R module
97	151	LOOTYP	1	INK, YNX, IN1, IN3, AT	O module information *Value* *Description* X'80' Punch O module X'40' O module to R1 X'20' O module to R2 X'10' O module to F1 X'08' O module to F2 X'04' No core map X'02' No cross-reference entry X'01' O module to program pack X'00' No O module
98	152	LOSWT1	1	INK, YNX, AF, IN1, IN2, IN3	Overlay Link Switch 1 *Value* *Description* X'80' Segment list in $SOURCE X'40' User call X'20' User specified overlays X'10' Start control addr set X'08' Groups in segment list X'04' Reserved X'02' Print messages X'01' Override 6E halt, Retain-R specified

Figure 8 (Part 1 of 3). Common Area (LOMMON)

Displacement Hex	Displacement Decimal	Label	Length in Bytes	Routines that Change Data ($OLxxx)	Description
99	153	LOSWT2	1	AF, AH, AJ, AR, AT, IN1, YNX	Overlay Link Error Switch *Value* *Description* X'80' System Cat called by User X'40' Not used X'20' An element in group is category 0 through 7 X'10' Not used X'08' Entry point not 0 in program with common X'04' No find on start control entry label X'02' Core size in OPTIONS record or program will not fit X'01' Terminal error
9A	154	LOUSPK	1	INK, YNX, IN1	User pack Q byte
9B	155	LONOVL	1	AP	Number of overlays
9C	156	LOPRPK	1	INK, YNX	Program pack Q byte
9D	157	LOEND@	2	INK, YNX	Addr of partition end
9F	159	LOCRSZ	2	AP, YNX, B0	Actual exec core size
A1	161	LORSV1	2		Reserved
A3	163	LOFTBL	2	AR	Displacement of OLFT
A5	165	LOAUTO	2	IN3	Relative entry number of auto segment list
A7	167	LOXREF	2	AF, BE	Relative entry number of X-ref segment list
A9	169	LOSORT	2	AH, BE	Relative entry number of sort segment list
AB	171	LOPRDA	2	INK, YNX	Directory addr on prog pack
AB	171	LOOVER	2	AJ, AP, BE	Relative entry number of overlay segment list
AD	173	LOUSDA	2	INK, YNX, IN1	Directory addr on user pack
AD	173	LOLIMT	2	AR, BE	Relative entry number of last delimiter
AF	175	LOWKCS	2	IN3, IN2, INK YNX, AF, AR	Relative sector number of next sector in $WORK
B1	177	LOLQBT	1	INK, YNX, BE	Q-byte of data pack
B2	178	LOLCSB	3	INK, YNX, BE	C/S addr of data start
B5	181	LOLHDR	1	INK, YNX, B0	Library type R or O
B6	182	LOLNAM	6	IN1, YNX, IN2, FTP, BO	Module name
BC	188	LOLLCS	2	BO	C/S of library entry
BE	190	LOLCAT	1	YNX	Overlay category of R
BE	190	LOLTXS	1	BE, B0	Number of text records in O
BF	191	LOLLEA	2	INK, YNX, AR, IN1	Link edit address
C1	193	LOLRLD	1	BE	RLD displacement
C2	194	LOLSCA	2	YNX, IN2, AJ, AR	Start control address
C4	196	LOLCSZ	1	INK, YNX, IN1, BO	Execution core size in 1/4K incr

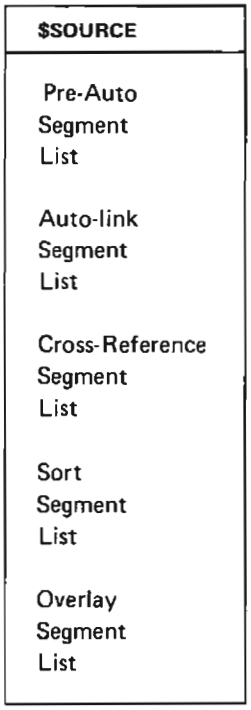
Figure 8 (Part 2 of 3). Common Area (LOMMON)

Displacement Hex	Displacement Decimal	Label	Length in Bytes	Routines that Change Data ($OLxxx)	Description
C5	197	LOATB1	2	IN1, IN2, AF, BE, B0, YNX, AJ	Attributes *Value* *Description* X'8000' Permanent entry X'4000' Inquiry X'2000' Inquiry invoking X'1000' Must run dedicated X'0800' Requires source X'0400' Deferred mounting allowed X'0200' Reserved, must be OFF X'0100' Reserved, must be OFF X'0080' System input dedication X'0040' Checkpoint restart program X'0020' Direct source read X'0010' Reserved X'0008' Reserved X'0004' Common X'0002' Reserved X'0001' Reserved
C7	199	LOLLVL	1	IN1, INK, YNX	Release level
C8	200	LOLTSC	2	YNX, BE	Total sector count
CA	202	LOWORK	31	Any	Phase work area
E9	233	LOCZER	2	INK, YNX	Constant of zero
EB	235	LOCONE	1	INK, YNX	Constant of one
EC	236	LOCHFF	2	INK, YNX	Constant X'FFFF'
EC	236	LOCM1	2	INK, YNX	Constant of minus one
EE	238	LOSCAT	1	AH	System category
EF	239	LOPTCS	2	YNX, IN3	C/S addr of printer routine
F1	241	LOERCD	1	AT, AF	Error code
F2	242	LOENTR	6	INK, IN1, AH, YNX	Entry point name save area

Note: Unused portions of the DTFs are used to store load lists of the linkage editor modules. These areas are:

Displacement Hex	Displacement Decimal	Label	Length in Bytes	Routines that Change Data ($OLxxx)	Description
02	02	LOAJ	6	AF	Load list $OLAJ
23	35	LOAP	6	AF	Load list $OLAP
29	41	LOAR	6	AF	Load list $OLAR
40	64	LOAT	6	AF	Load list $OLAT
6E	110	LOBE	6	AF	Load list $OLBE
74	116	LOBO	6	AF	Load list $OLBO *Load List Format* 2-byte Cyl/Sec Addr 1-byte No. of Sectors 1-byte RLD Displacement 2-byte Start Control Address

Figure 8 (Part 3 of 3). Common Area (LOMMON)



Segment List Entry Types

00	Module name
01	Entry point
02	EXTRN
03	Weak EXTRN
04	Global common
05	Local common
0B	EQUATE entry
0C	Transfer vector
0D	Reference a previous name or entry point
0E	GROUP entry
0F	CATEGORY entry
FE	Nulled entry
FF	End of segment list

Figure 9. Segment Lists in $SOURCE

Byte(s)	Bit(s)	Routine that Sets Data ($OLxxx)	Applies to Segment Type	Description
0	0-3	IN3	0E, 0F	Reserved
0	4-7	IN3	0E, 0F, 0B	Segment type (see Figure 9)
1		IN3	0E	Group number
1		IN3	0E	Category override number
2		AH	0E	Work area—original category
2-5		IN3	0E, 0F	Reserved
3	7	IN3	0E	User area specified for module
6-7		AF	0E, 0F	Reference number—pointer to module element in Auto-Link Segment List*
6-7		AJ	0E	Reference number—pointer to lead element in last overlay*
8-9		AF	0E, 0F	ESL Sequence Number
A-F		IN3	0E, 0F, 0B	Module name
A-B		AF	0E	Reserved
C-D		AJ	0E	Module element pointer (moved from bytes 6 through 7)
E-F		AF	0E	Reserved

* Segment displacement within $SOURCE.

Figure 10. Pre-Auto Segment List

Byte(s)	Bit(s)	Routine that Sets Data ($OLxxx)	Applies to Segment Type	Description
0	0	AF	00	COBOL entry
0	0	AF	01, 02, 03, 04, 05	Entry point or references an entry point
0	1	AF	02, 03	Resolved to module and/or entry
0	1	AF	00, 01	This module or entry point has an EXTRN referencing it
0	2	AF	00, 01, 02, 03	Work area must be OFF at phase end
0	2	AJ	00	Used—do not place in tree
0	3	AF	00	Calls a user routine or requires a transfer vector
0	3	AF	02, 04, 05	Delete this element when compressing list.
0	3	AJ	00	Module already placed in root
0	4-7	AF	All	Segment type (see Figure 9)
1		AF, AH	00, 01, 02, 03	Category
2-3		AF	00	$WORK address of object code
2-3		AF	01	Entry displacement from start of module
2		AH	00	Number of entry points
3	0	AH	00	Module requires boundary alignment
3	1	AH	00	Module calls a user routine
3	2	AH	00	Module has I/O dependency
3	3	AJ	00	Module already in an overlay
3	3	AJ	00	Substructure pointer already built
4-5		AF	00	Object code length
6-7		AF	00, 01, 02, 03, 04	Reference number—pointer to equal ESL number in Auto-Link Segment List
8-9		AF	00, 01, 02, 03, 04, 05	ESL number
A-F		AF, INK	00, 01, 02, 03, 04, 05	ESL name
A-B		AH	00	Reference number—pointer to equal O type in X-ref segment list
C-D		AJ	00	Work area
E-F		AH	00	Reserved

The Pre-Auto Segment List is included in this list and will appear as the first elements in the list.

Figure 11. Auto-Link Segment List

Byte(s)	Bit(s)	Routine that sets data ($OLxxx)	Applies to Segment Type	Description
0	0	AH	00, 01, 0D	Reserved
0	1	AH	00, 01	Reserved
0	2	AH	00, 01, 0D	Reserved
0	3	AH	00, 01,0D	Reserved
0	4-7	AH	00,01,0D	Segment type (see Figure 9)
1		AH	00,01,0D	Category
2-3		AH	00,01	Entry point displacement from start of module
4		AH	00	Work area = number of entry points on original category
5	0	AH	00	Module requires boundary alignment
5	0	AH	0D	Categories make this call a potential program failure
5	1	AH	00	Module calls a user routine
5	2	AH	00	Module has I/O dependency
5	3	AH	00	Work area - OFF at end of phase
5	4	AH	00	Work area - OFF at end of phase
5	5	AH	00	No reference made to this module
5	5	AH	01	Same name as module name
5	6	AH	00,01	Duplicate name
5	7	AH	00,01	Start control label
6-7		AH	00	Location of object text in $WORK (X'FFFF'=null text)
8-9		AH	00,01,0D	ESL number
A-F		AH	00,01,0D	Module or entry point name

Figure 12. Cross-Reference Segment List

Byte(s)	Bit(s)	Routine that Sets Data ($OLxxx)	Applies to Segment Type	Description
0	0-3	AP	0C	Set of modules already summed
0	1	AP	0C	Set of modules contains a boundry alignment module
0	0-3	AJ	00,02,04,05,0C	Reserved
0	4-7	AJ	00,02,04,05,0C	Segment type (see Figure 9)
1		AJ	00,02	Category
1		AP	00,0C	Overlay number
2		AJ	00	Number of entry points this module
2		AP	0C	Number of entry points this overlay
3	0	AJ	00	Module requires boundary alignment
3	1	AJ	00	Module calls a user routine
3	2	AJ	00	Module has I/O dependency
3	3	AP	00	Work area
3	4-7	AJ	00	Reserved
3		AJ	0C	Overlay area used by this set of modules at execution time
4-5		AJ	00,04,05	Length of object area associated with this ESL
4-5		AP	0C	Length of object area this overlay candidate
6-7		AJ	00	Reference number - pointer to equal module
6-7		AJ	02	Pointer to module
8-9		AJ	00,02,04,05	ESL number
A-B		AJ	00	Pointer to module name element in X-ref list
A-B		AP	0C	Pointer to next set of modules in same overlay
C-D		AJ	00	Chain to substructure referencing this module
C-D		AJ	02	Chain to other substructure and module
C-D		AJ	0C	Chain to last previous transfer vector element
E-F		AJ	02,04,05,0C	Reserved
E-F		AP	00	Boundry alignment adjustment factor

Figure 13. Sort Segment List

Byte(s)	Bit(s)	Routine that Sets Data ($OLxxx)	Applies to Segment Type	Description
0	0	AR	02	Work area = resolve to transfer vector
0	4-7	AR	00,02,04,05	Segment type (see Figure 9)
1		AR	00,02,04,05	Overlay Number
2-3		AR	00,02,04,05	Object time address for this ESL
4-5		AR	00,04,05	Object time length for this ESL
4-5		AR	02	Corresponding module type ESL number
6-7		AR	00	Address of module's first transfer vector
6-7		AT	00	$WORK location of object text
6-7		AR	02	3-byte RLD object time addr for this ESL
8-9		AR	00,02,04,05	ESL number
A-B		AR	00	Pointer to equal 0-type in X-ref segment list. FFFF designates overlay fetch routine
B		AR	02	Relative entry point position
C-D		AR	00	Overlay size - first 0 type of overlay only
C-D		AR	02	Pointer to 0 type entry in sort list
E-F		AR	00,02,04,05	Reserved

Figure 14. Overlay Segment List

APAR SUBMISSION

For APAR submission, the following information is necessary:

- Disk dumps of $WORK and $SOURCE
- Core dump
- Library directory dump of all routines used by the object program

OVERLAY FETCH ROUTINE

The Overlay Fetch routine is added to the root segment of every program that has overlays. It is built by routine $OLAR. When an overlay segment is needed during program execution, the Overlay Fetch routine is called. It fetches overlay segments from access devices and places them in the overlay regions in main storage. Bits are set in the overlay fetch table (Figure 15) telling which overlay region is used.

The Overlay Fetch routine requires three parameters as input:

1. Overlay number (one byte)
2. Entry address of the overlay (two bytes)
3. Return address from the overlay (two bytes)

A transfer vector is built for each overlay in an object program. Transfer vectors provide input parameters for the Overlay Fetch routine. Overlay Linkage Editor routine $OLAR builds transfer vectors. Figure 16 shows the format of transfer vectors.

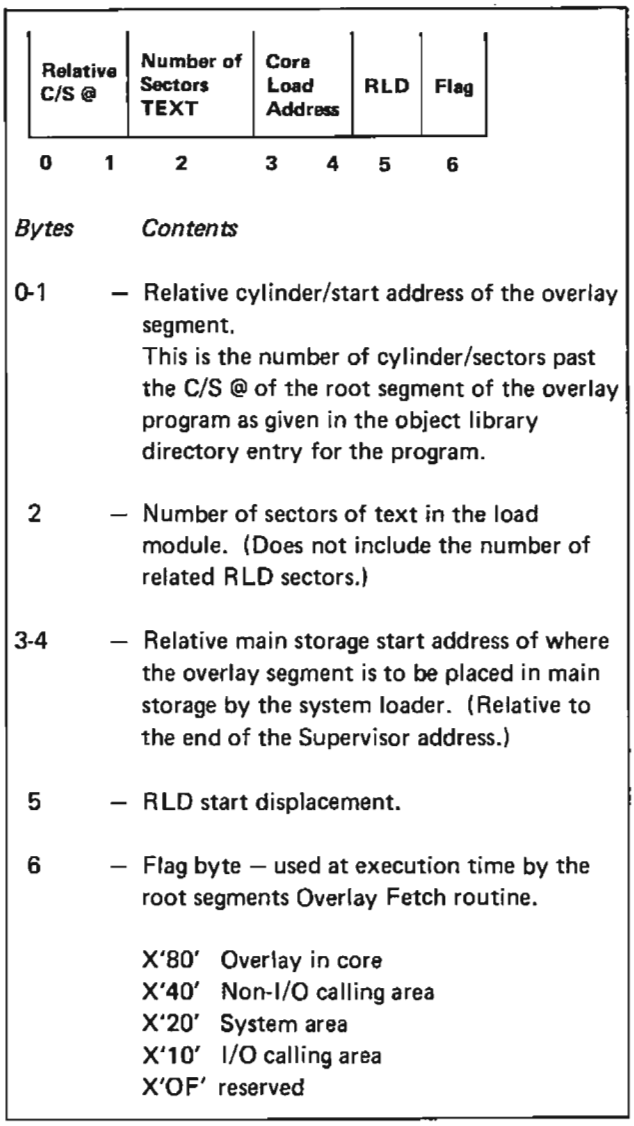


Figure 15. Overlay Fetch Table Entry Format

ST	OVFRS1,ARR	Save the return address
B	OVFR	Call the Overlay Fetch routine
DC	XL1 'NN'	One byte containing the overlay number
DC	AL2 (entry)	Two-byte entry address

Figure 16. Transfer Vector Format

The Overlay Fetch routine checks to see if the requested overlay segment is already in main storage. If it is, the routine branches to the entry address of the overlay; if not, the overlay fetch table entries are checked to see if they use the same main storage. If they do, the overlay is flagged as not being in main storage.

After the Overlay Fetch routine checks all entries in the overlay fetch table, it sets the 'overlay in core' bit in the overlay fetch table entry for the requested overlay. The Overlay Fetch routine then loads the overlay segment and branches to its entry address. Figure 17 describes the Overlay Fetch routine.

Overlay Fetch Table

The overlay fetch table is built by routine $OLAR. It contains one 7-byte entry for each overlay in the program. Figure 15 shows the format of an overlay fetch table entry.

How to Find an Overlay

When a process check occurs, the following steps can determine which overlays are in main storage and where to find them.

1. Locate the address of the Overlay Fetch routine on the core usage map of the source listing (Figure 18).

2. Locate the overlay fetch table in the dump. The overlay fetch table is 115 bytes past the start address of the Overlay Fetch routine. It can be obtained by this hex formula: Address of Overlay Fetch routine +X'73'=overlay fetch table (Figure 19).

3. Mark off every 7-byte entry in the overlay fetch table until the last entry is reached. The last entry is X'FF' (Figure 19).

4. Number each entry left to right, starting with number 1. Each entry refers to an overlay (Figure 18).

5. Look at the seventh byte in each entry. This is the flag byte. The first bit will be on for every overlay in storage at the time of the dump (Figure 19).

6. Compare the numbers you gave the overlays in storage at the time of the dump with the number of the overlays in the core usage map (Figure 18). This gives the names and addresses of the segments within the overlays which were in storage at the time of the dump (Figure 19).

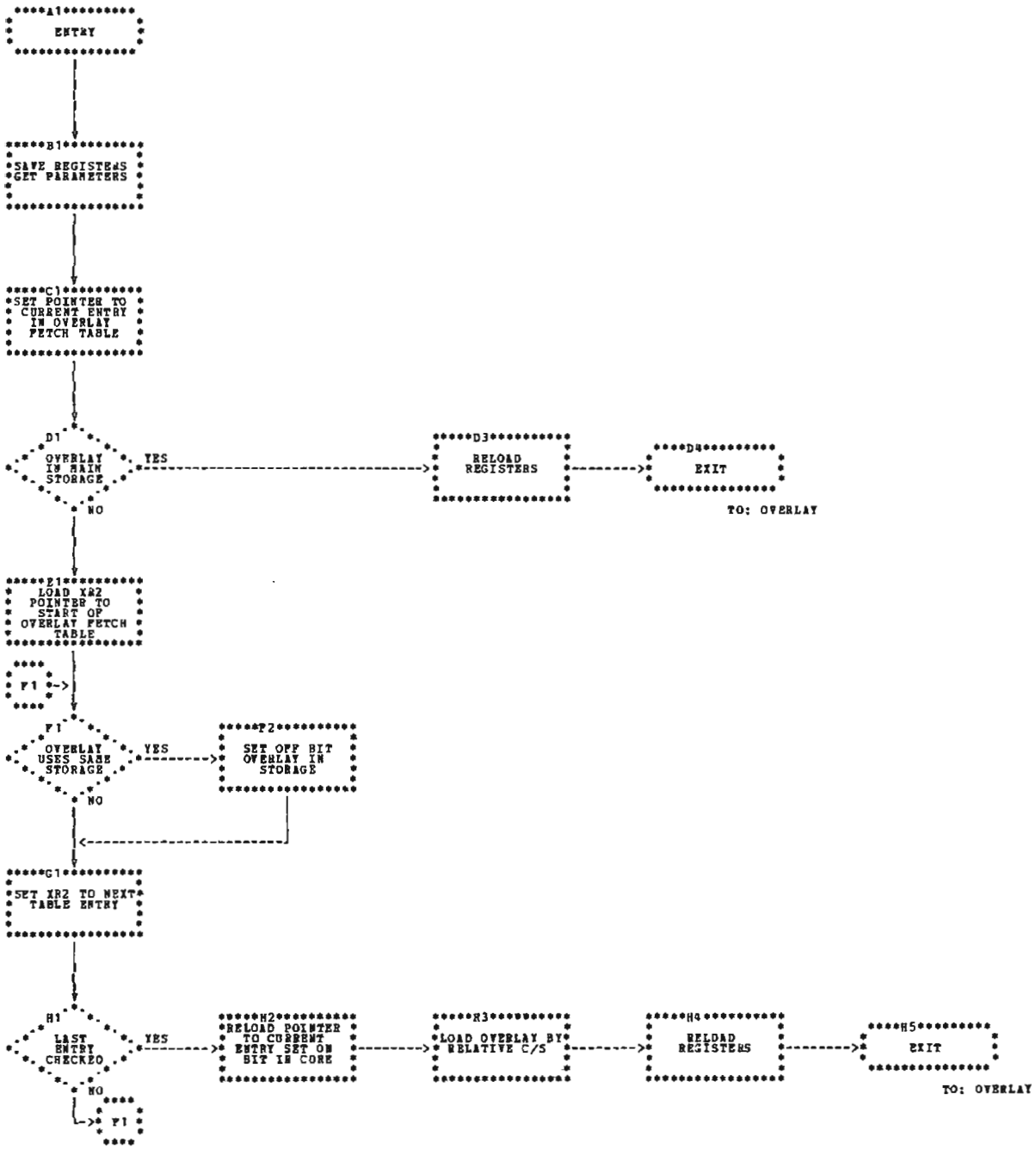


Figure 17. Overlay Fetch Routine

```
                OVERLAY LINKAGE EDITOR CORE USAGE MAP AND CROSS REFERENCE LIST     01/25/72

 START     OVERLAY   CATEGORY  NAME AND       CODE LENGTH      REFERENCED
 ADDRESS   NUMBER AREA         ENTRY     HEXADECIMAL DECIMAL    BY

 1300                   127    MAIN           11FF     4607    MAIN50 MAIN60
 24FF                   2780   C800           0067      103    MAIN   MAIN50 MAIN60
 2566                          OVLFRTN        0009      217
 2700      1    U        8    MAIN50         0866     2150    MAIN   MAIN60
 2700      2    U        9    MAIN60         0842     2114    MAIN   MAIN50
 3000      3    S        2    $$CSOP         0010       29    MAIN   MAIN50 MAIN60 C800
 3010      3    S        2    $$SRBR         0079      121    $$CSOP
 3096      3    S        2    $$SRUA         0026       38    $$CSOP
 30BC      3    S        2    $$SRDP         001C       28    $$CSOP
 30D8      3    S        2    $$SRTC         001C       28    $$CSOP $$SRDI
 30D8                        DMSRLO
 30E9                        DMSRTC                           $$CSOP
 30E6                        DMSRER                           $$CSOP $$SRDI
 30F4      3    S        2    $$SRMO         0081      129    $$SRBR
 3175      3    S        2    $$SRSB         0043       67    $$SRBR
 3188      3    S        2    $$SRDI         0038       56    $$SRBR $$SRSB
 3107                        DMSRPD                           $$SRSB
 3100                        DMSRRD                           $$SRSB
 31F0      3    S        2    $$SRBP         002F       47    $$SRUA $$SRSB
 3000      4    S        3    CB01           01C2      450    MAIN   MAIN50 MAIN60
 304F                        $CB002                           MAIN   MAIN50 MAIN60
 3000      5    S        4    SUBRTN         00FA      250    MAIN   MAIN50 MAIN60

OL100 I  THE TOTAL CORE USED BY CB1    IS  8192 DECIMAL
OL101 I  THE START CONTROL ADDRESS OF THIS MODULE IS 148C.
OL102 I  THE NON-OVERLAY CORE SIZE IS    10217 DECIMAL
OL033 W  ALL THE ELEMENTS IN A GROUP ARE CATEGORY 0-7
OL027 W  PROGRAM WILL NOT FIT IN THE CORE SIZE SPECIFIED


OL104 I  TOTAL NUMBER OF LIBRARY SECTORS REQUIRED IS   44
         NAME-CB1    ,PACK-F1F1F1,UNIT-F1,RETAIN-T,LIBRARY-O
```

Start address of Overlay Fetch routine

Length of Overlay Fetch routine including Overlay Fetch table and transfer vectors

Overlays in main storage at time of dump (Figure 19)

Figure 18. Core Usage Map and Cross-Reference List

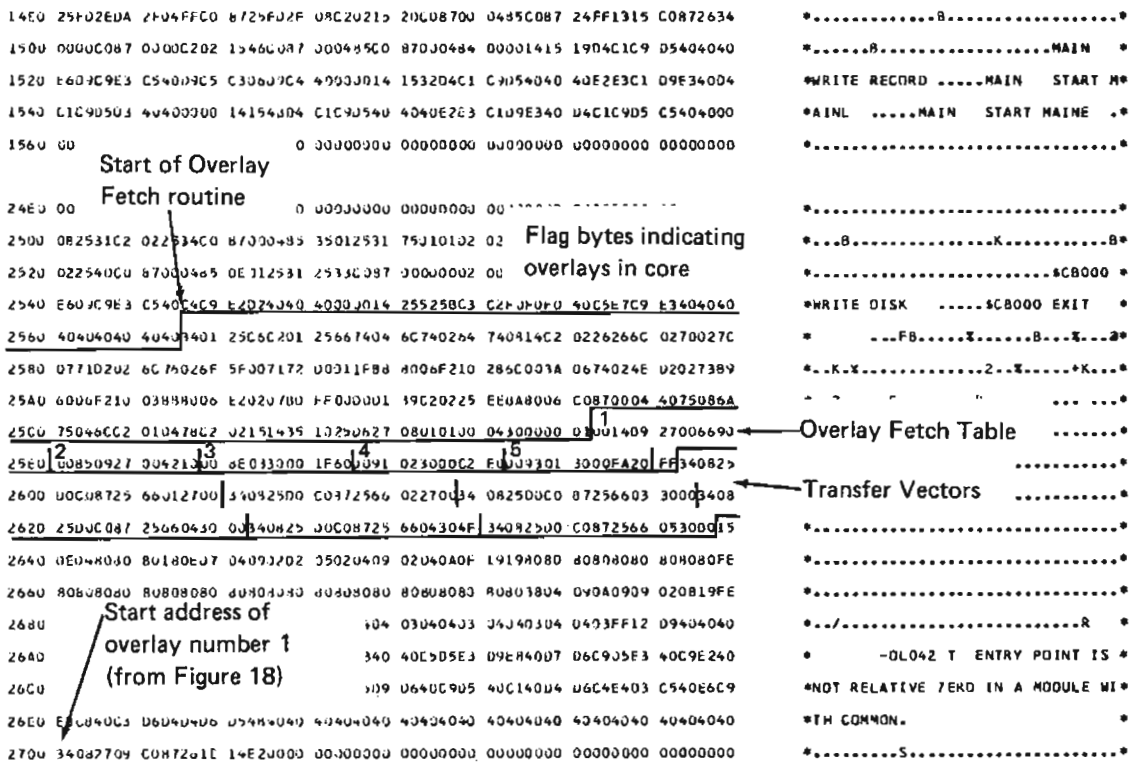


Figure 19 (Part 1 of 2). Sample Core Dump

```
2EC0 00000000 00000000 00000000 00000000 00000000 00000000 00000202 2F18C087   *.............................B.....*
2EE0 00048562 012EEF34 01290500 8728E0C0 A724FF27 15C08726 342F14C2 022F34C0   *...B.........................B....*
2F00 87000485 C087146E7 C2022F40 C0870004 85C08726 29260826 0A000000 00142F20   *........X8.........................*
2F20 D4C1C9D5 F5F040E2 E3C1D9E3 40C1F5F0 40404040 00001420 3934C1C9 D5F5F040   *MAIN50 START A50     .....MAIN50 *
2F40 C5D5C440 C1F5F040 40404040 40000014 2F5204C1 C9D5F5F0 40C505E3 C5D940C3   *END A50       .....MAIN50 ENTER C*
2F60 F5F04040 40400304 80808DFE 80808080 80808080 80808080 56090404 04020402   *50    ............................*
2F80 040[illegible]04 04030404 03040403 04040304 0403FF12 D9404040   *...............................R   *
2FA0 404[illegible]F4 F240E340 40C5D5E3 D9E84007 D0C9D5E3 40C9E240   *       -QL042 T  ENTRY POINT IS *
2FC0 D5[illegible]C5 40E4C5D9 D640C9D5 [illegible]0E6C9   *NOT RELATIVE ZERO IN A MODULE WI*
2FE0 E3[illegible]40 40404040 40404040 [illegible]04040   *TH COMMON.                    *
3000 F2870656 03C2F0F0 F1340830 76020230 77C0870C [illegible]50200   *2..$CB001...B.............K......*
3020 05020134 02304E75 02030BFF 04F20112 [illegible]0E202D3 02010284   *.......*.......2.....K.....S..K...*
3040 01C0C202 30A9C087 0004B5C0 872E0434 08307602 02309DC0 87000485 35013076   *..B.................8.............*
3060 02020375 01013401 307220C1 307400CC 87000000 00270800 00143D7C 5BC3C2F0   *K.............................a$CB0*
3080 F0F140C5 D5E3C5D9 40404040 40404040 03001430 955BC3C2 F0F0F140 C5D5C3C5   *01 ENTER         .....$CB001 ENTE*
30A0 D940D5C3 C2F3F0F2 40000014 00AE5BC3 C2F0F0F1 4040C5E7 C9E34040 40404040   *R $CB002 .....$CB001  EXIT      *
30C0 40400000 00000000 00000000 00000000 00000000 00000000 00000000 00000000   *  ..............................*
30E0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000   *................................*

XR1   XR2   PSR   LPIAR LPDAR MPTAR MRDAR MPCAR DFCR  DFDR  IAR2  ARR2  XR1-2 XR2-2 PSR2
2EFF  2F14  0001  0428  037F  1E60  1340  1E20  0F73  3200  5F04  5F1A  5F10  0563  0004
TRANSIENT AREA
0100 F28705E2 E3D6D7D6 34010208 34080208 34020358 C0870004   *2..STOP.....................*
0118 00080458 35020358 88400083 80000010 02888800 00030100   *.......... ................*
0130 F2100235 01001189 C0930820 00F2100A 78001379 8016C090   *2..B..........2...........*
0148 02037A00 1B3C4003 FF0C4203 FE03FFC0 8702D035 01001170   *.......  ...............1*
```

Start address of overlay number 4 (from Figure 18)

Entry point of overlay number 4 (from Figure 18)

Figure 19 (Part 2 of 2). Sample Core Dump

MESSAGES

The Overlay Linkage Editor issues informational messages and error messages. Figure 20 lists these message numbers with the routines that issue them. For a full explanation of these messages, see *IBM System/3 Overlay Linkage Editor Reference Manual*, GC21-7561.

Message Number	Issued by Routine
OL016 W	$OLAT
OL021 T	$OLAF
OL022 T	$OLAF
OL025 T	$OLAT
OL026 T	$OLAT
OL027 W	$OLAT
OL029 W	$OLAT
OL031 W	$OLAT
OL032 W	$OLAF
OL033 W	$OLAT
OL034 W	$OLAF
OL035 T	$OLAT
OL036 W	$OLAT
OL038 T	$OLAF
OL042 T	$OLAT
OL100 I	$OLAT
OL101 I	$OLAT
OL102 I	$OLAT
OL103 I	$OLBO
OL104 I	$OLBO

Figure 20. Message Numbers with Issuing Routine

PART II

CHECKPOINT/RESTART

Checkpoint/Restart enables the user to restart a checkpointed program from the last checkpoint taken, provided no intervening program executions take place.

MINIMUM MACHINE REQUIREMENTS

- IBM 5410 Processing Unit with 12K storage
- IBM 5444 Disk Storage Drive
- IBM 5424 Multi-Function Card Unit or IBM 1442 Card Read Punch
- IBM 5203 Printer

OPERATING CHARACTERISTICS

Checkpoint/Restart depends on the System/3 Model 10 Disk System SCP programming support and the System/3 Model 10 Disk System Overlay Linkage Editor support of the OPTNS statement.

Checkpoint/Restart requires that the system IPL pack scheduler workarea contain the additional tracks for a Roll-in/Roll-out area. This area is available if either the inquiry capability or the Checkpoint/Restart feature is included at system generation time.

Checkpoint/Restart enables the user to restart a checkpointed program. Checkpoint is a means of recording the status of a problem program at certain intervals (checkpoints). After an error occurs, Restart can resume execution of a checkpointed program from the last checkpoint before the error. Restart is not allowed after a controlled cancel or normal end of job.

CHECKPOINT

Figure 21 shows an overview of Checkpoint. (An operational diagram legend is included.) Figure 22 is the storage map.

$$STKP (Checkpoint - Main Load) is entered from the supervisor as a result of a find and fetch request. Before a checkpoint can be made, $$STKP awaits completion of all pending non-tape I/O operations. Upon completion, $$STKP then gives control to $$STKQ (Checkpoint-Quiesce Magnetic Tape I/O) if tapes are used; otherwise, control goes to $$STKR (Checkpoint-Problem Program and SWA Load) to de-activate the checkpoint area and save the SWA and checkpointed program on disk. $$STKT (Checkpoint-Final Load) can be called to store the checkpoint information, then restore the checkpointed program and re-activate the checkpoint area. At this time the checkpointed program is given control to continue processing.

If errors occur, control returns to the checkpointed program with a completion code of X'41'. If an unrecoverable disk error occurs, control passes to the end-of-job transient, $$SPEJ.

The position of tapes can be saved only if the tapes were opened and allocated by SCP support. Checkpoint modules use fields NPSCHA (set by ALLOCATE) and NPDTF@ (set by OPEN) in the program level 1 communication region (N1COMN) to determine if tapes are used by the program and to save the following information from the DTFs:

- Q-code
- SWA Format 1 (F1) numbers
- Current block count

This information is used by RESTART to reposition the tapes. Checkpoint/Restart cannot reposition BTAM tape files or files accessed directly via tape IOS.

CHECKPOINT LINKAGE

To use Checkpoint, the checkpoint attribute (bit 1 of the second attribute byte) must be set in the object library directory entry for the program to be checkpointed. This attribute is set from the information in the object deck header card if the program is put in the object library by the Library Maintenance program. The checkpoint attribute is mutually exclusive with the inquiry invoking attribute.

To call Checkpoint, the following linkage convention is required:

```
         LA    FDPARM,2        Pointer to find parameter list
         B     NCENTR          General entry
         DC    XL1'81'         Find
         CLI   0(,2),C'O'      $$STKP FOUND?
         BE    ERRTN           NO, GO TO USER ERROR RTN
         MVC   CS,1(2,2)       Set up C/S for fetch
         B     NCENTR          General entry
         DC    XL1'80'         Fetch
CS       DC    CL2'00'         C/S of $$STKP from object library into transient area
         DC    XL1'02'         (three sectors)
         B     CONTIN          Return from $$STKP
         DS    XL1             Completion code
RESTRT   EQU   *               Restart entry point
           .                   (Any setup/repositioning necessary to
           .                   continue after
           .                   RESTART)
FDPARM   EQU   *               Find parameter list
         DC    CL7'0$$STKP'    Object module $$STKP
         DC    CL1'S'          On system pack
         DS    CL4
```

Linkage to Checkpoint is achieved by branching to the resident general entry routine, NENTRY, through the resident communication vector NCENTR. (NENTRY is documented in the *IBM System/3 Disk Systems System Control Program Logic Manual,* SY21-0502.) This branch must be followed by a constant value, called the request indicator byte (RIB), identifying the Find transient, $$SPFN. $$SPFN finds Checkpoint, using the find parameter list (FDPARM). The C/S address is set up for a fetch, and another branch is made to NENTRY with a RIB for requesting $$STKP. This is followed by the C/S address of the $$STKP and the module size (three sectors).

The completion code should be checked to ensure that a valid checkpoint occurred. The checkpoint is ignored if an I/O error is outstanding on any of the supported and allocated devices. Valid return codes are X'40' (checkpoint taken) and X'41' (checkpoint ignored).

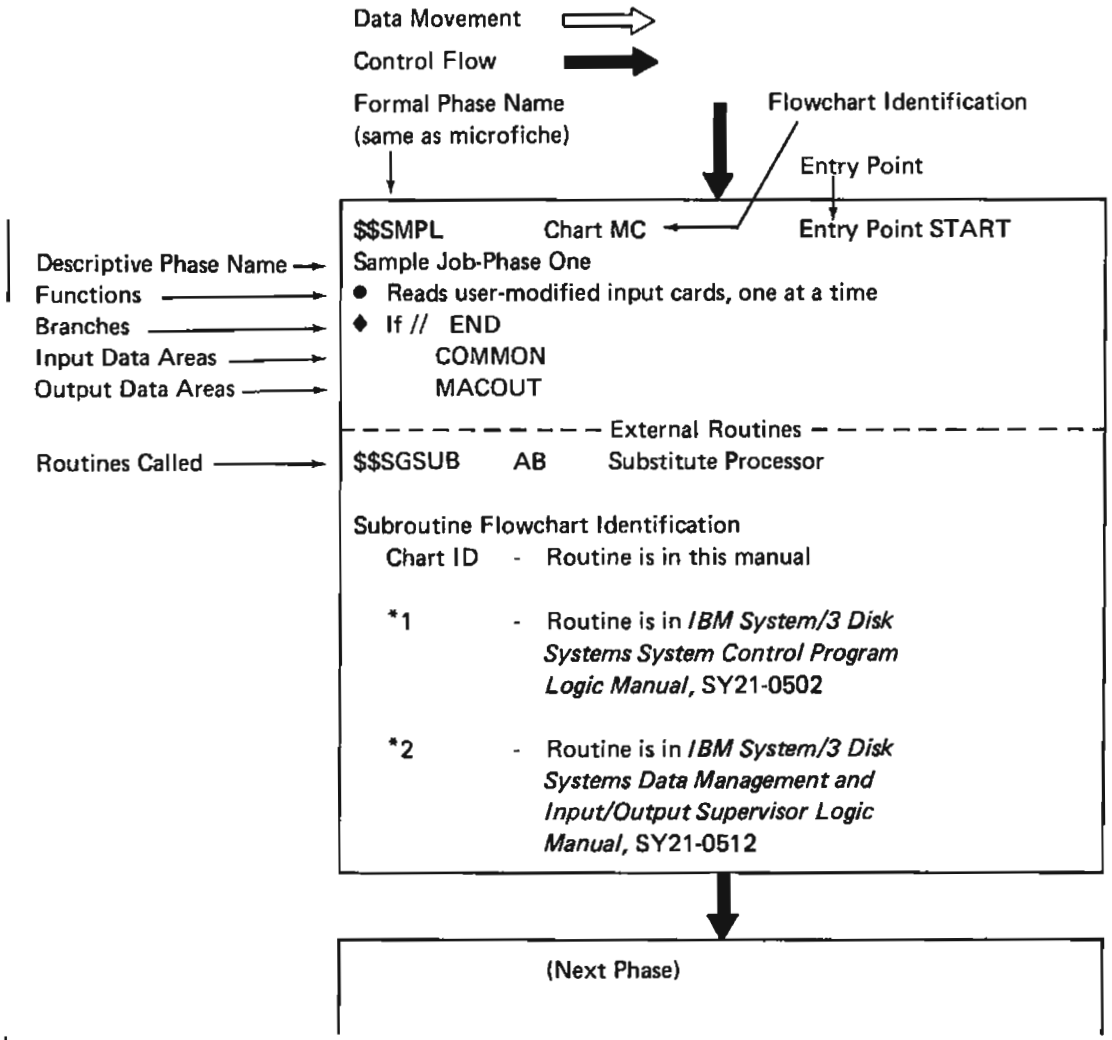


Figure 21 (Part 1 of 5). Checkpoint Operational Diagram Legend

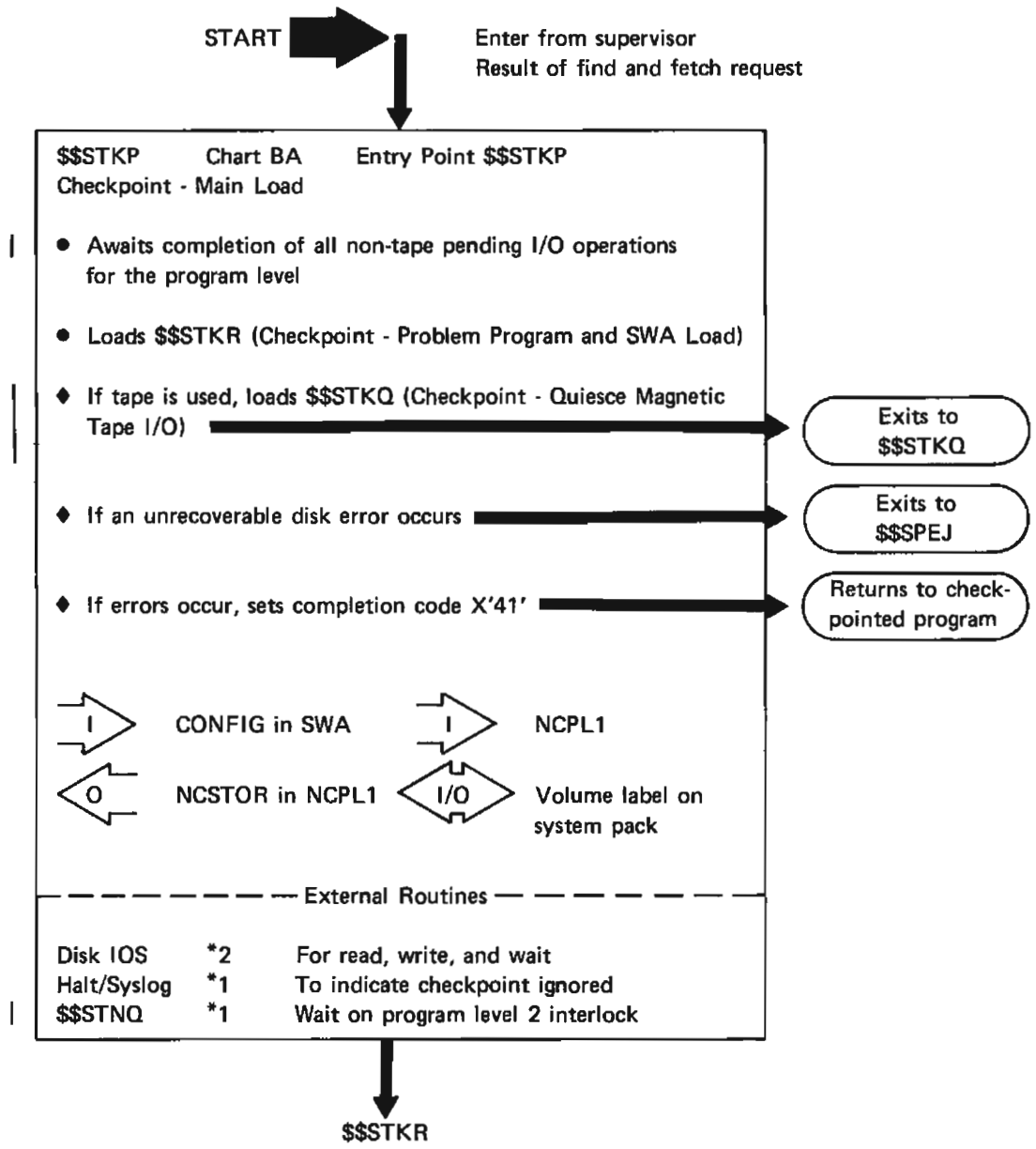
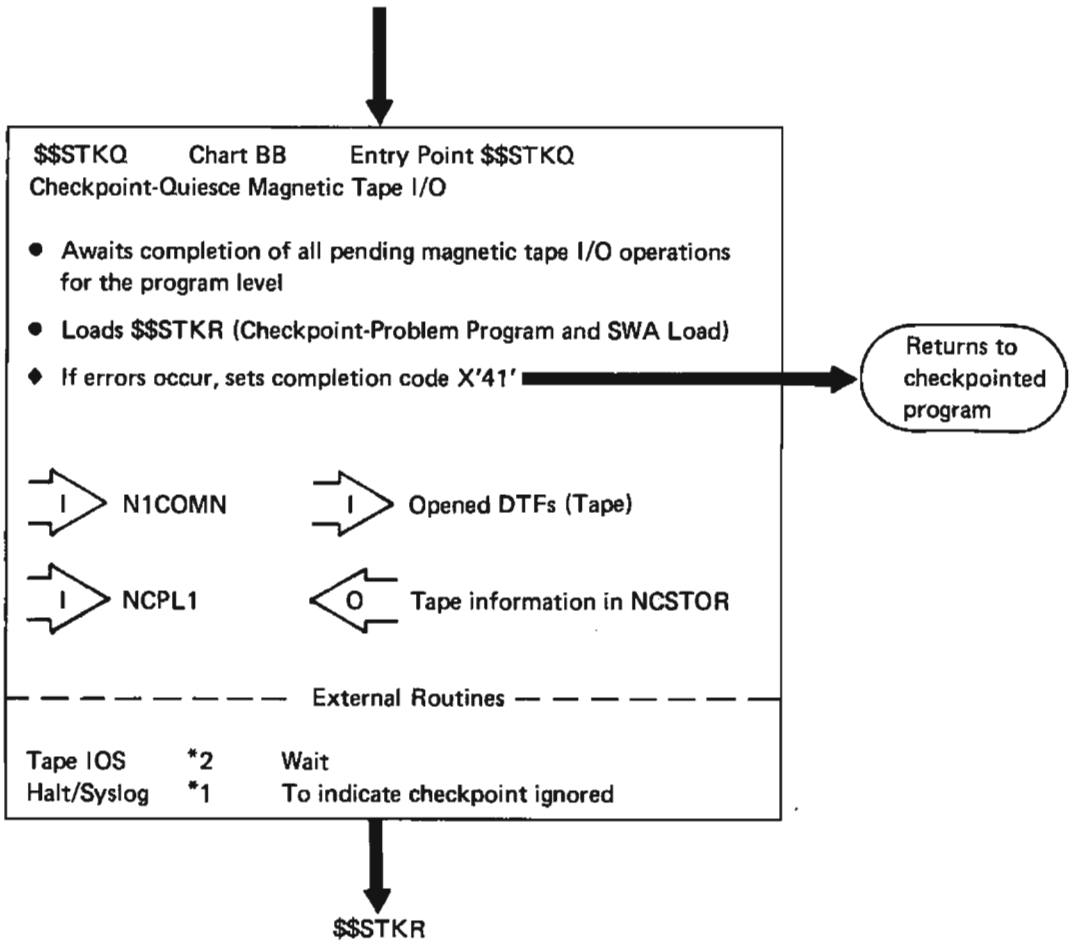


Figure 21 (Part 2 of 5). Checkpoint Operational Diagram Legend



● Figure 21 (Part 3 of 5). Checkpoint Operational Diagram Legend

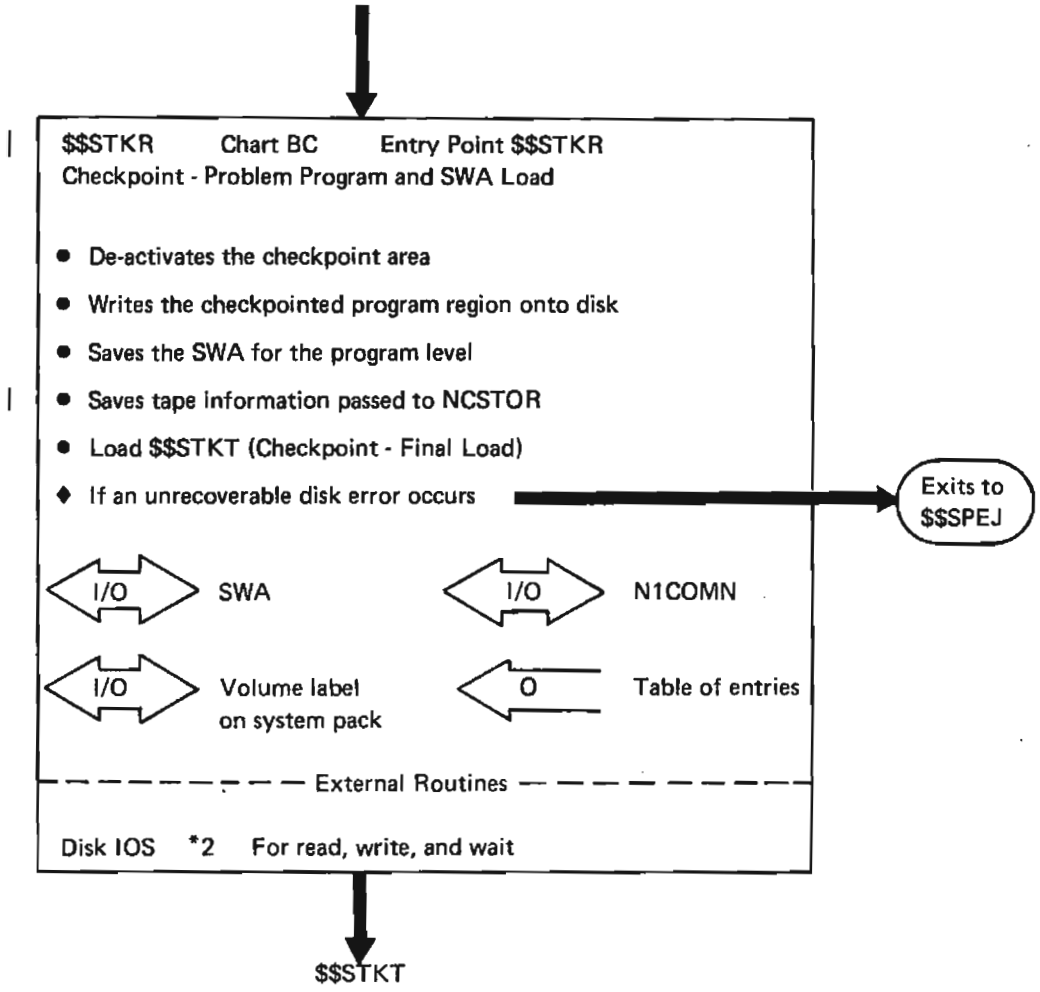


Figure 21 (Part 4 of 5). Checkpoint Operational Diagram Legend

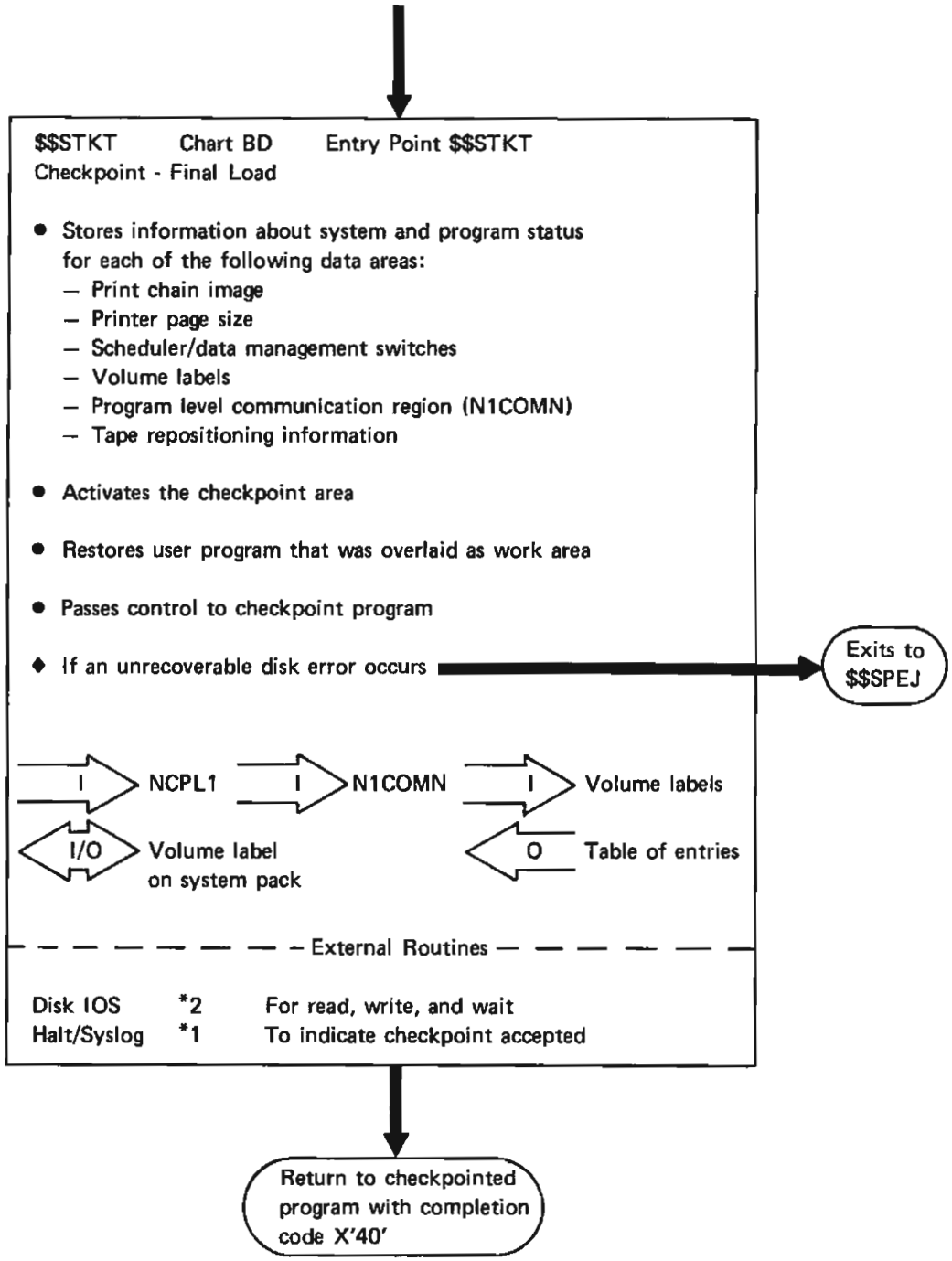
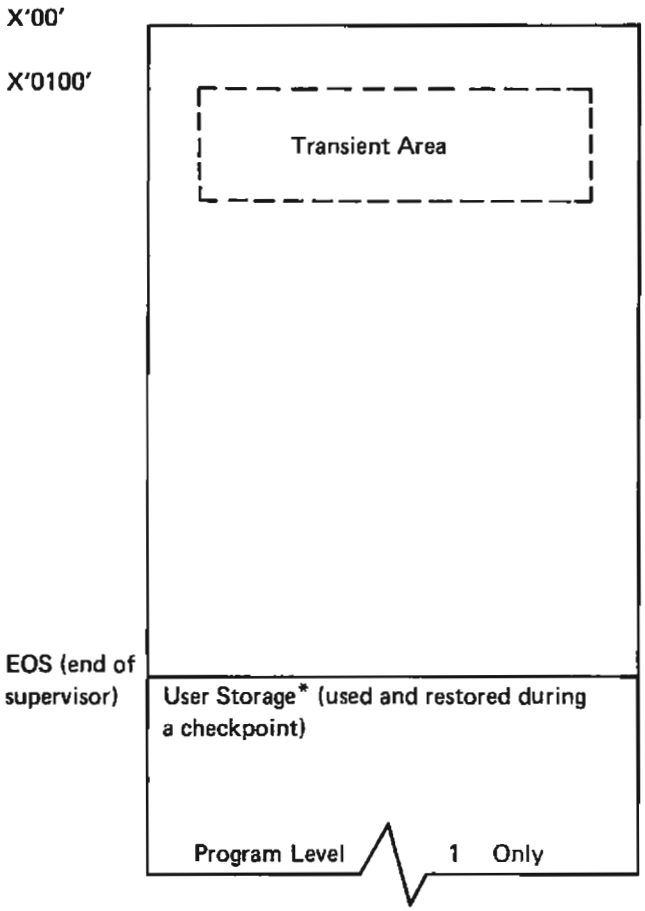
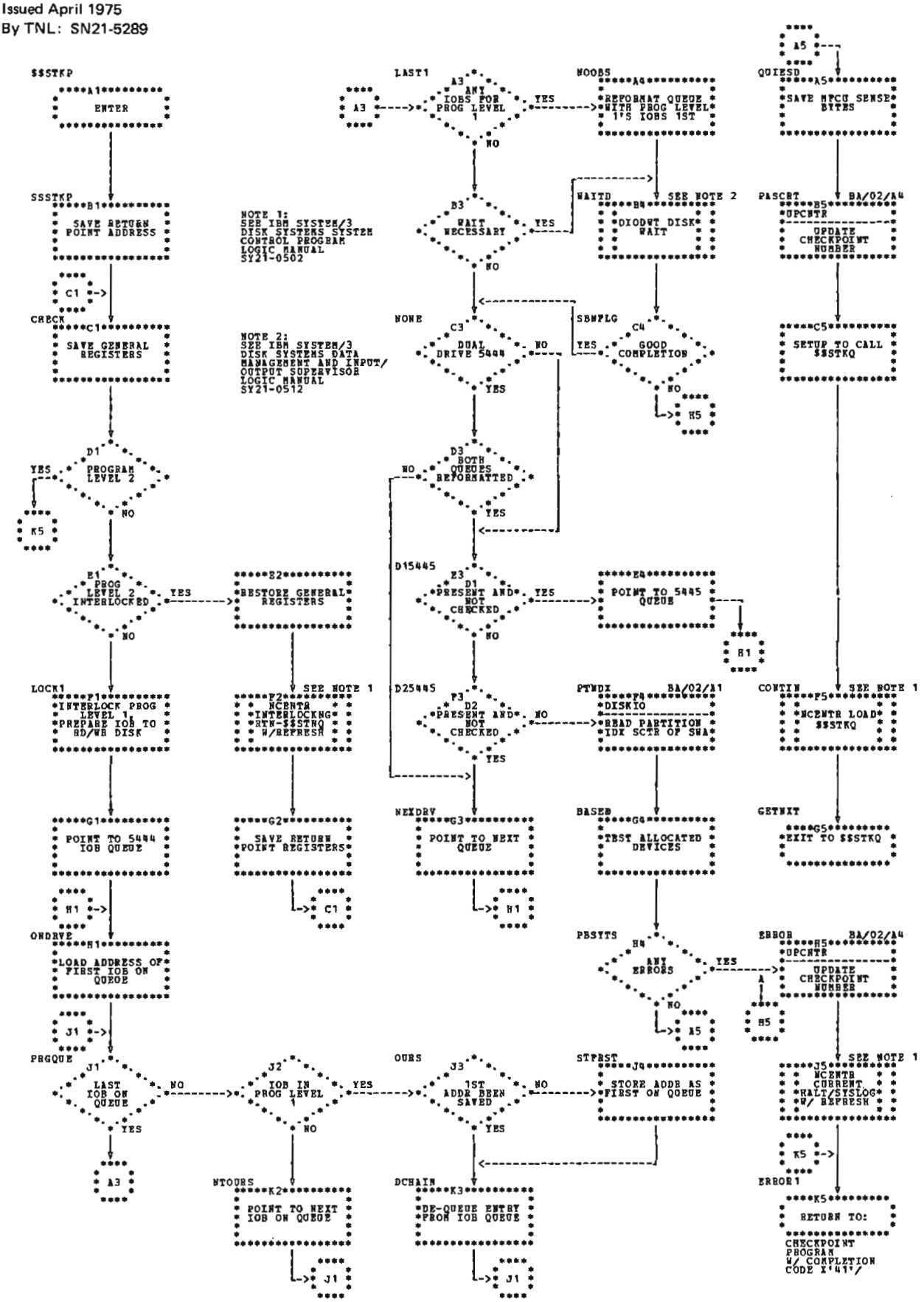


Figure 21 (Part 5 of 5). Checkpoint Operational Diagram Legend



*Note: User's program must be at least 1K since this is the minimum work area required.

Figure 22. Main Storage Map Showing Transient Area and User Storage Needed by Checkpoint



● Chart BA (Part 1 of 2). Checkpoint – Main Load ($$STKP)

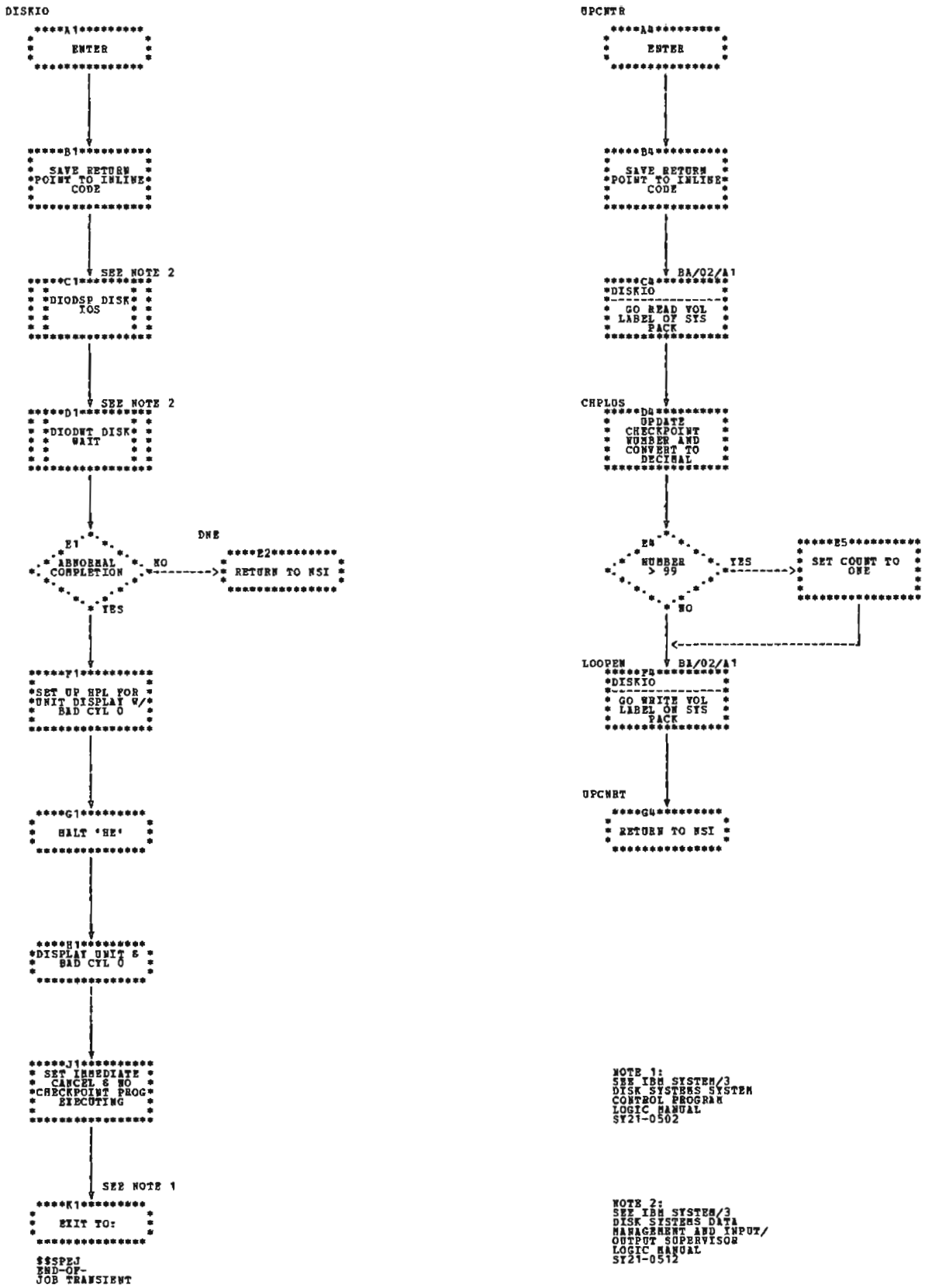
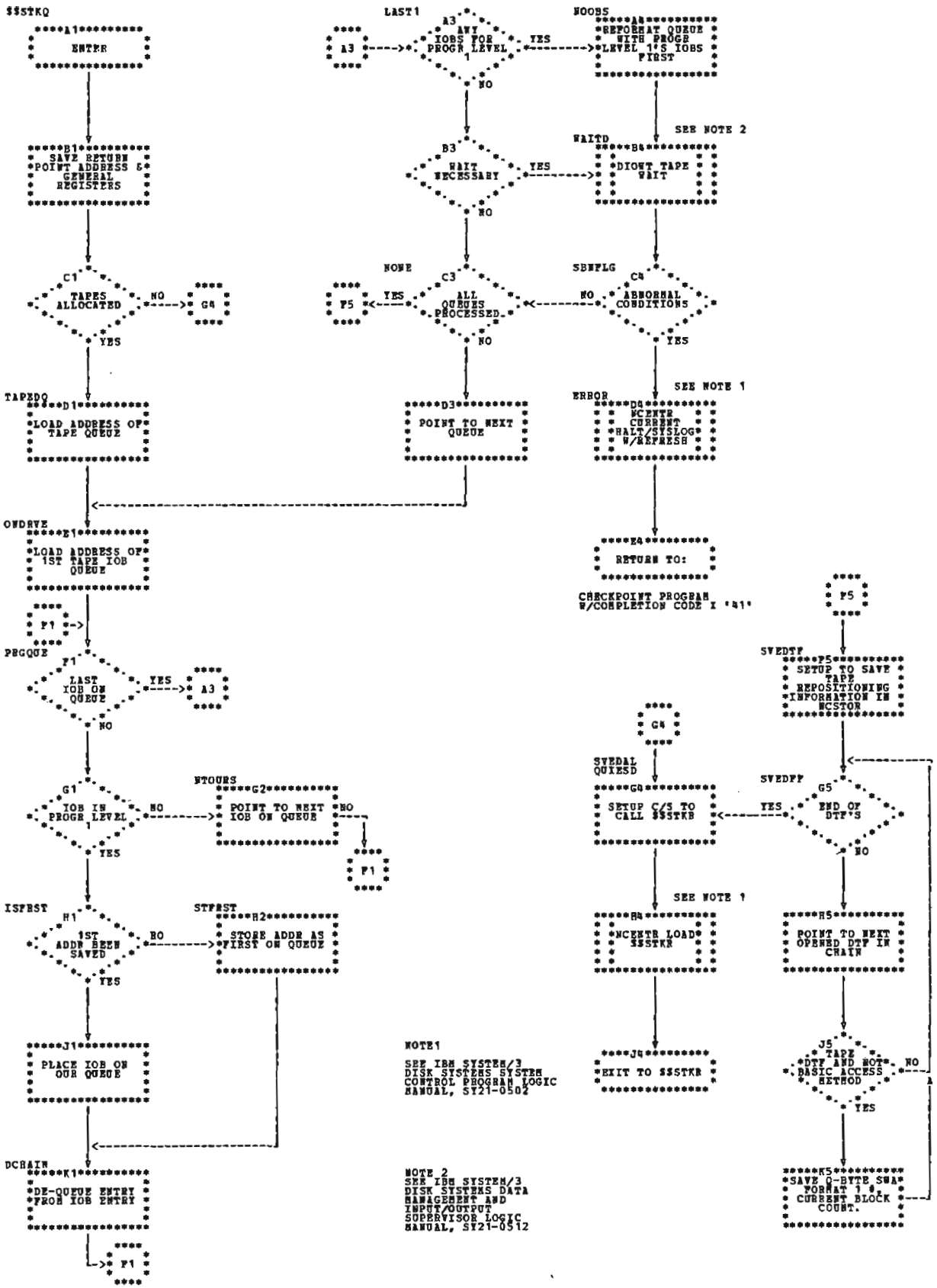


Chart BA (Part 2 of 2). Checkpoint — Main Load ($$STKP)



● Chart BB. Checkpoint — Quiesce Magnetic Tape I/O ($$STKQ)

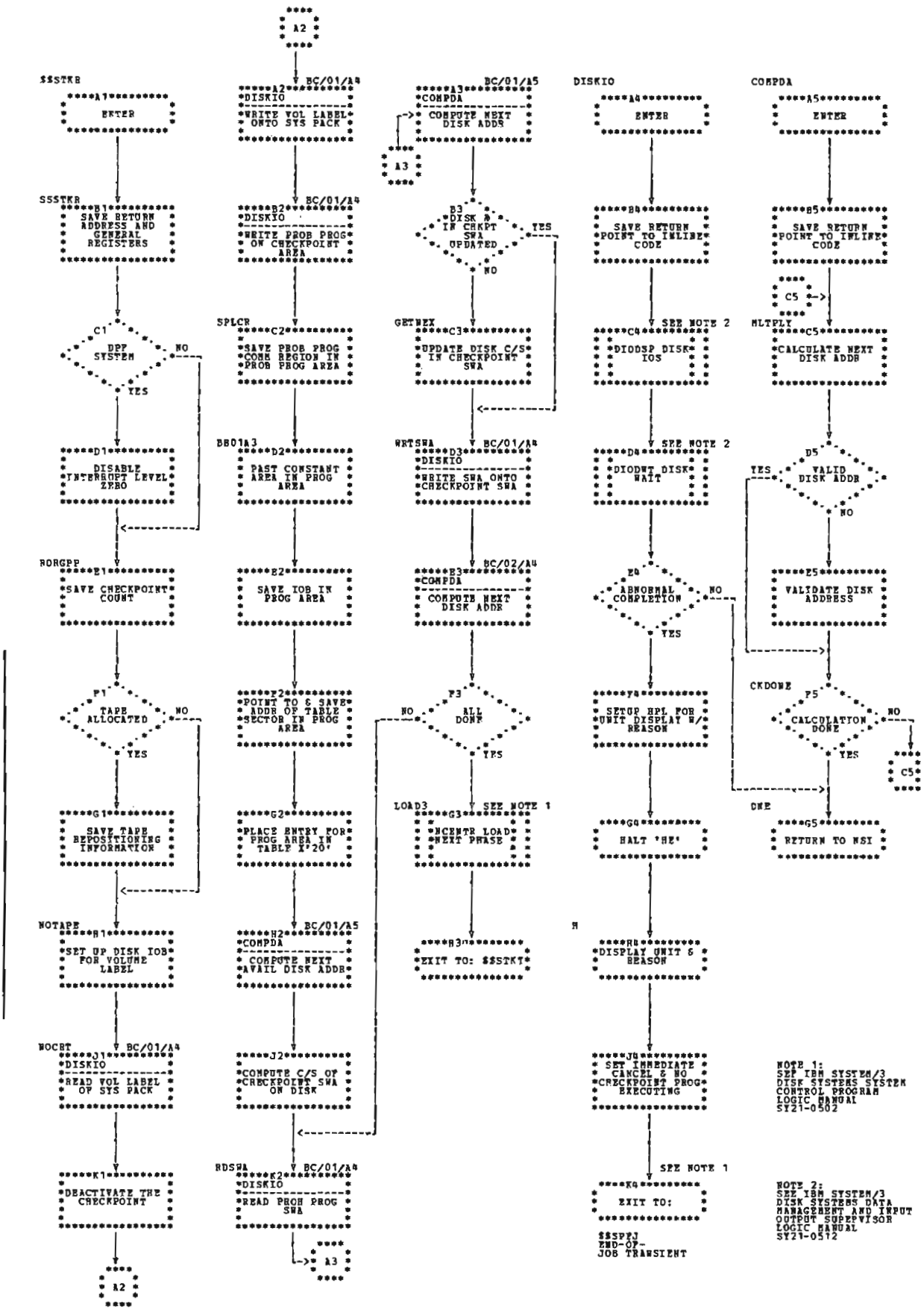


Chart BC. Checkpoint — Problem Program and SWA Load ($$STKR)

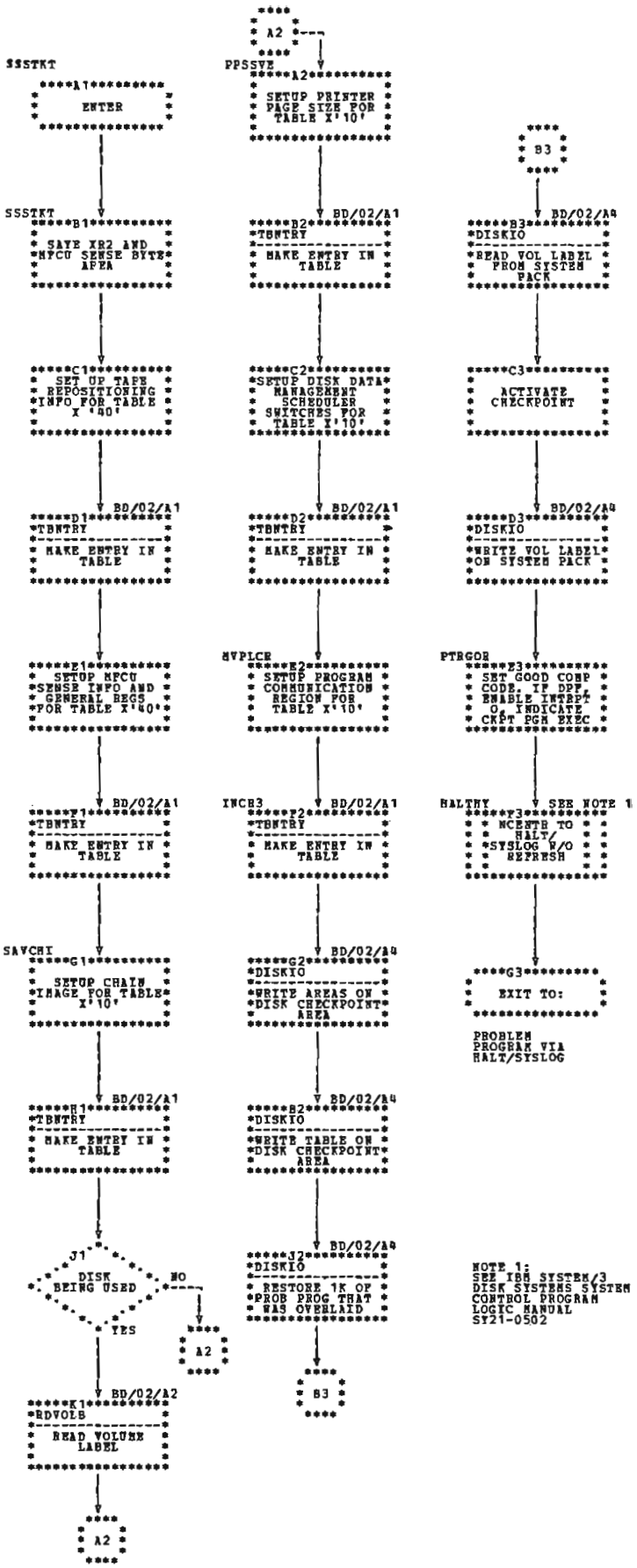


Chart BD (Part 1 of 2). Checkpoint — Final Load ($$STKT)

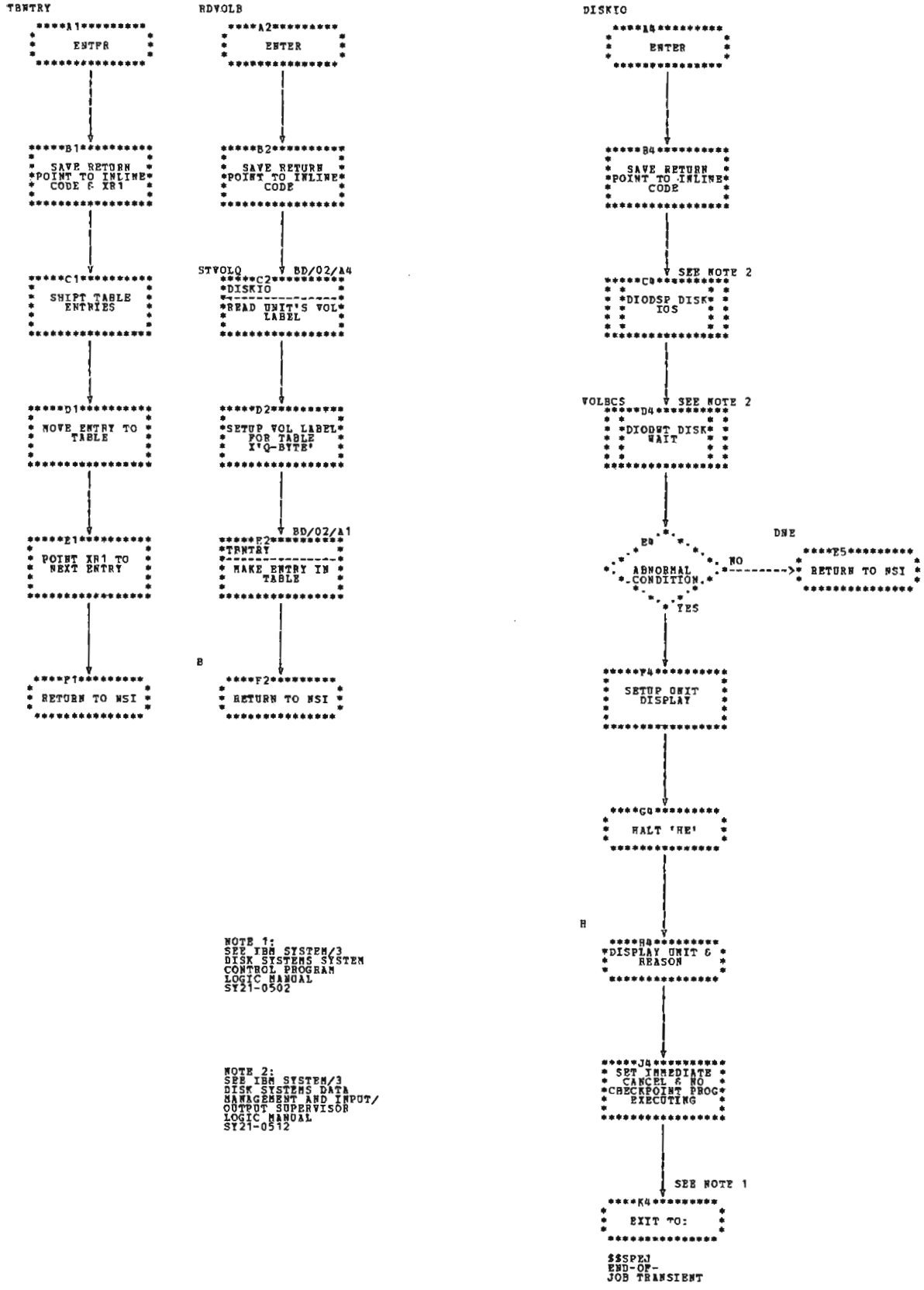


Chart BD (Part 2 of 2). Checkpoint – Final Load ($$STKT)

RESTART

Figure 23 shows an overview of Restart. (An operational diagram legend is shown in Figure 21.) A storage map is given in Figure 24.

$$RSTR (Restart - Main Load) is entered from the Supervisor as a result of // LOAD $$RSTR OCL statements. It checks for an active checkpoint and available storage. $$RSTR restores the SWA from the checkpoint disk area back to the system disk area. It also assures that the correct disks or tapes are mounted and cards or tapes are repositioned for any allocated and supported card devices (except 1442) or tape devices.

If tapes are used, the files are repositioned to the block following the last block processed at the last checkpoint, provided the same reel is mounted. The filename and the tape drive are logged for each tape drive being processed to give the operator the opportunity to verify that the correct reel is mounted. Standard labeled tapes are checked to verify that the file label and volume sequence number match the reels being processed at the last accepted checkpoint. The nonstandard or unlabeled tapes are not verified.

Basic access method files or direct calls to tape IOS are the responsibility of the user and no repositioning or label checking is done.

Once this is done, $$STKV (Restart - Problem Program and Final Load) is loaded.

$$STKV restores N1COMN and passes control to the checkpointed program. The restart entry point is at the last checkpoint taken.

If an immediate cancel or an unrecoverable disk error should occur, control is passed to the end-of-job transient, $$SPEJ.

RESTART LINKAGE

To continue execution of the interrupted job at the last checkpoint the user must submit the following OCL statements to load Restart:

```
// LOAD $$RSTR,unit
// RUN
```

The LOAD statement identifies the program to be run and indicates the disk on which it is located. For Restart, the unit must be the system IPL pack containing the checkpoint file to be restarted. The RUN statement indicates the end of the OCL statements, and the system runs the program. To guarantee the required minimum size for program level 2 (which must allow 5K for Restart in program level 1), a PARTITION statement may be required. Also, to reestablish the log device, a LOG statement may be required. For more information on these OCL statements, see *IBM System/3 Model 10 Disk System Operation Control Language and Disk Utilities Reference Manual*, GC21-7512.

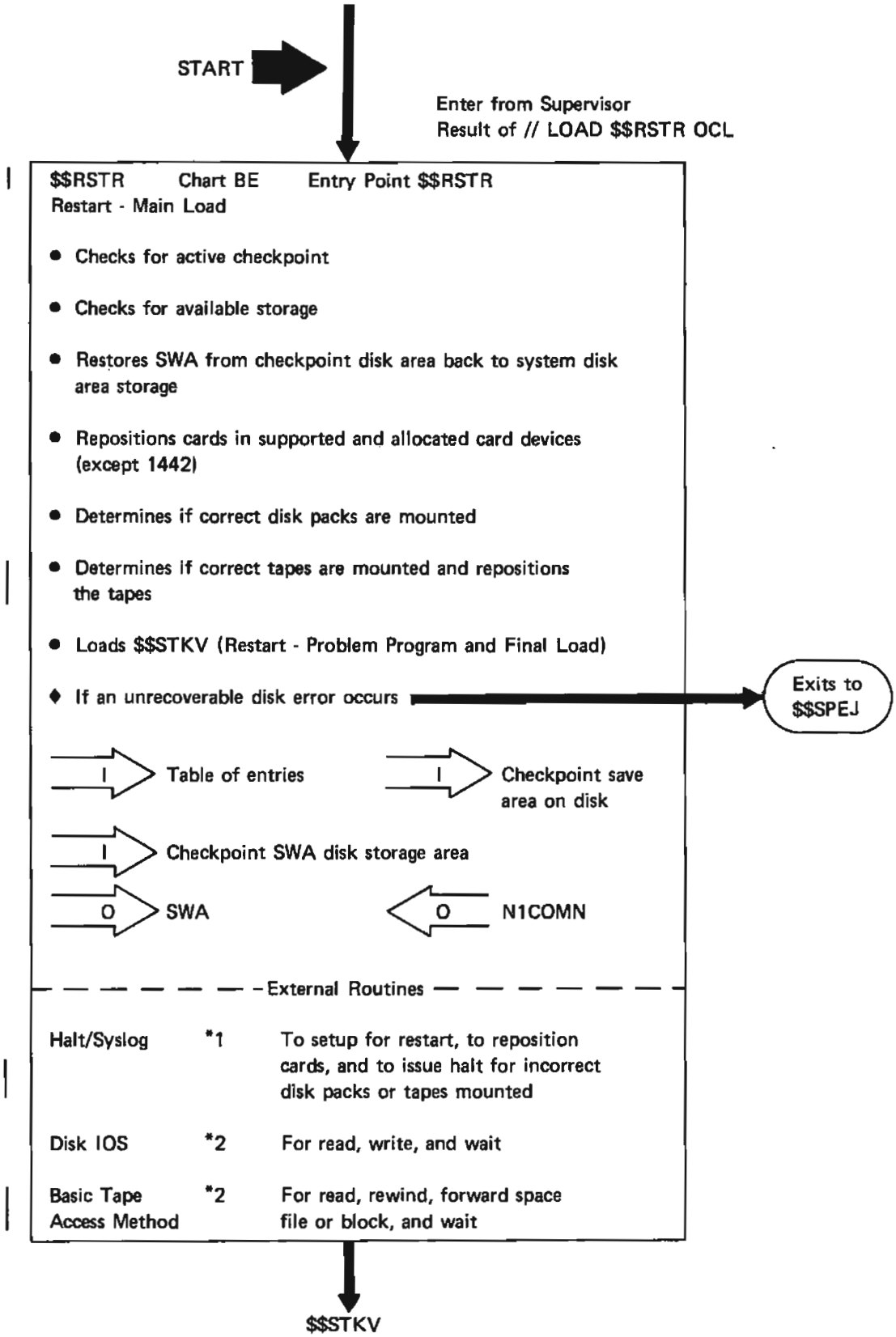


Figure 23 (Part 1 of 2). Restart Operational Diagram

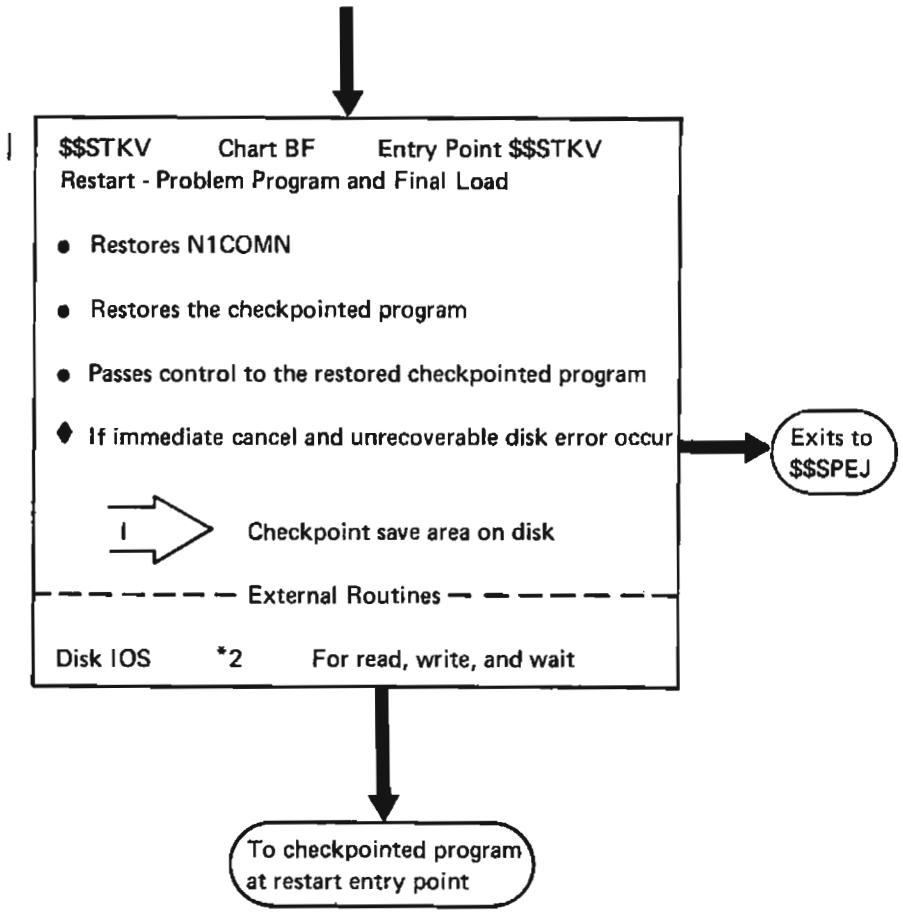


Figure 23 (Part 2 of 2). Restart Operational Diagram

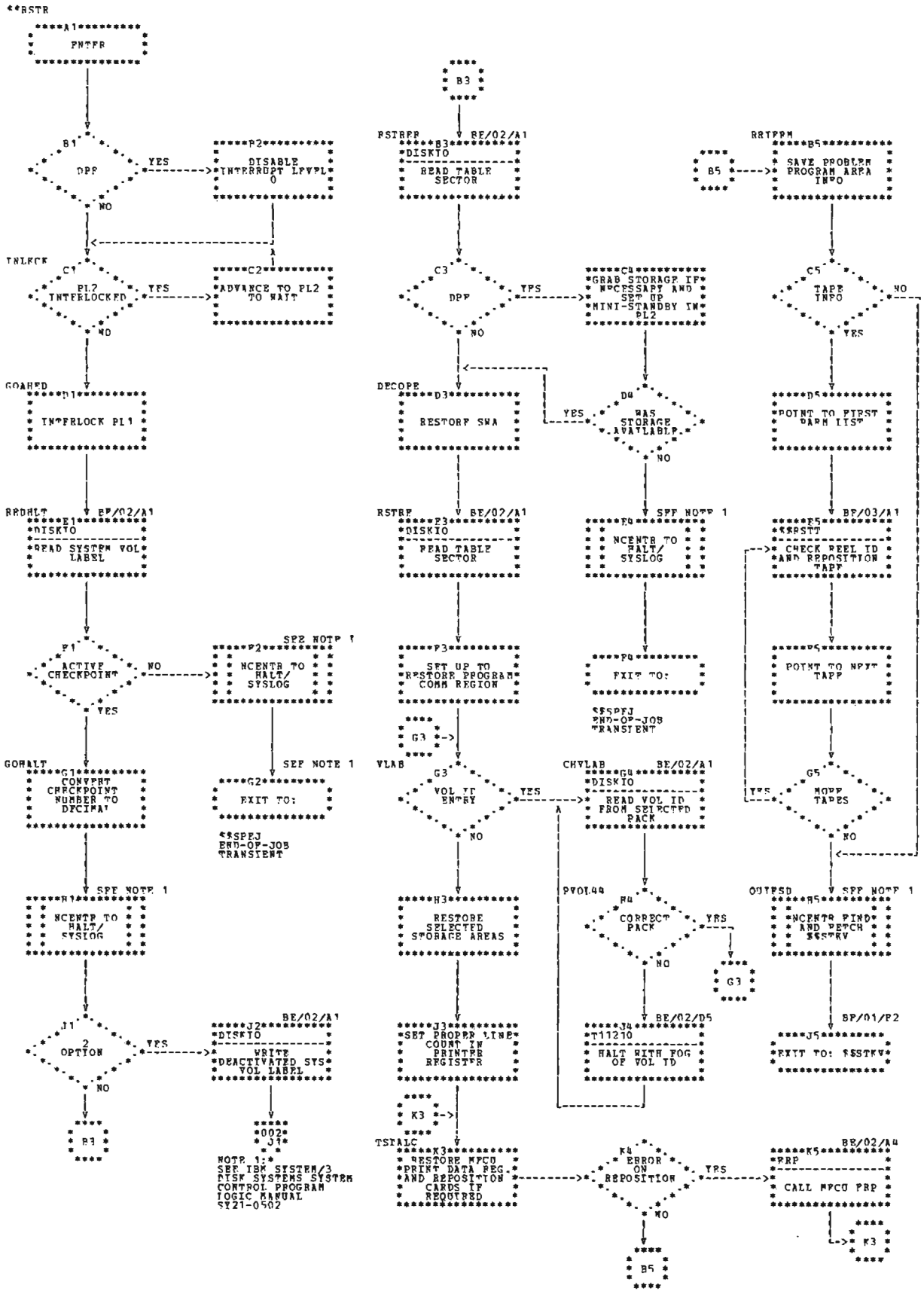
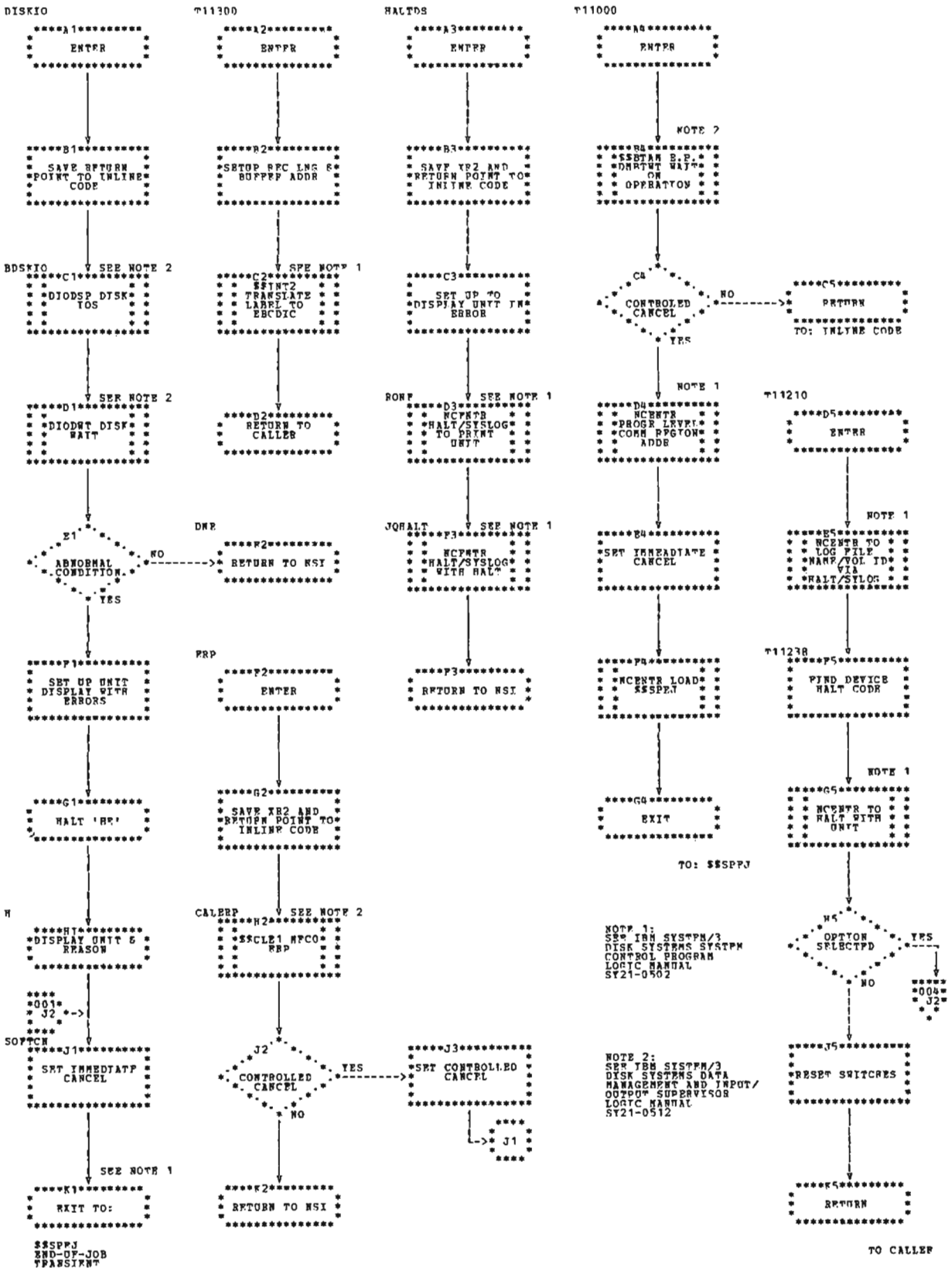
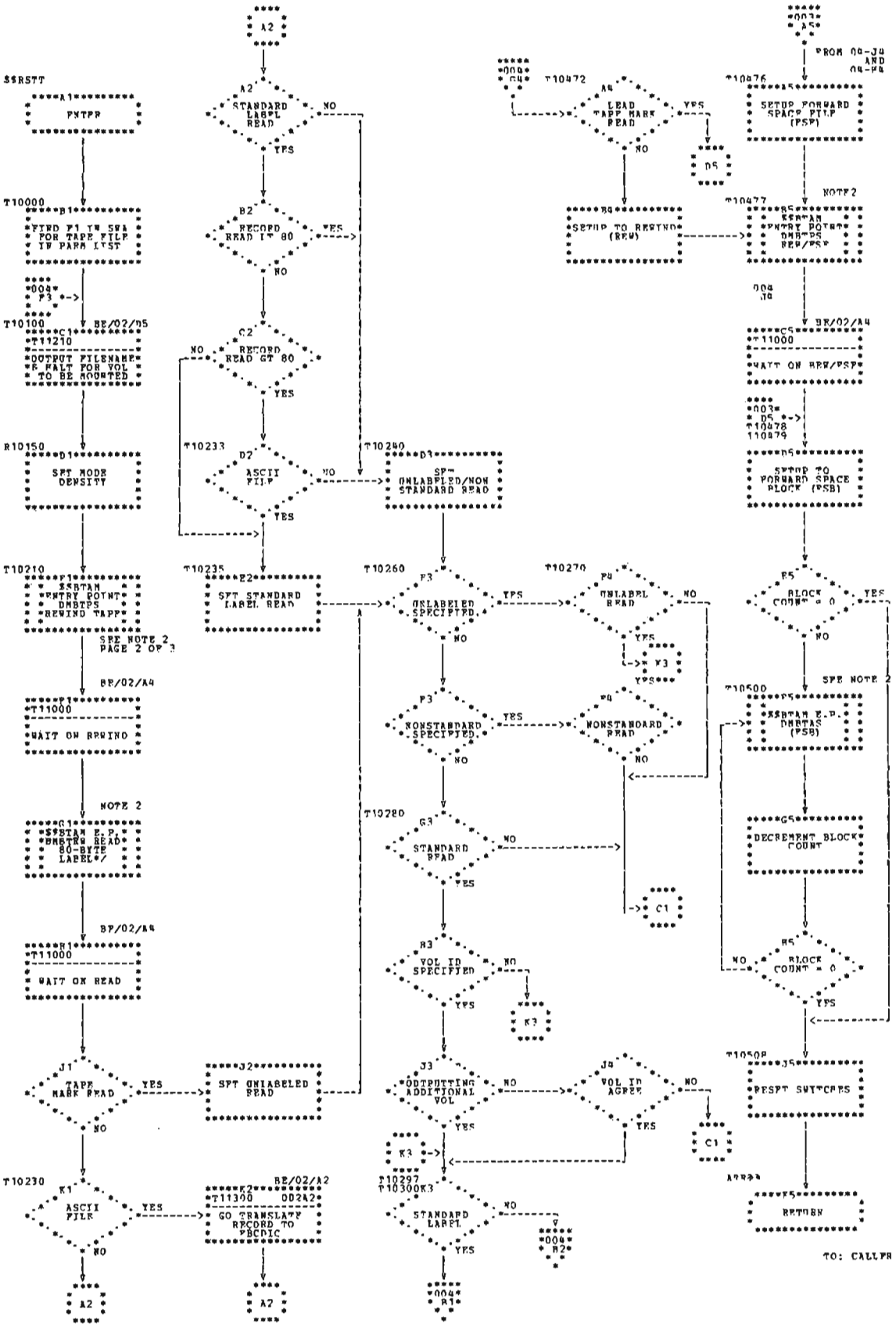


Chart BE (Part 1 of 4). Restart — Main Load ($$RSTR)



- Chart BE (Part 2 of 4). Restart — Main Load ($$RSTR)



● Chart BE (Part 3 of 4). Restart — Main Load ($$RSTR)

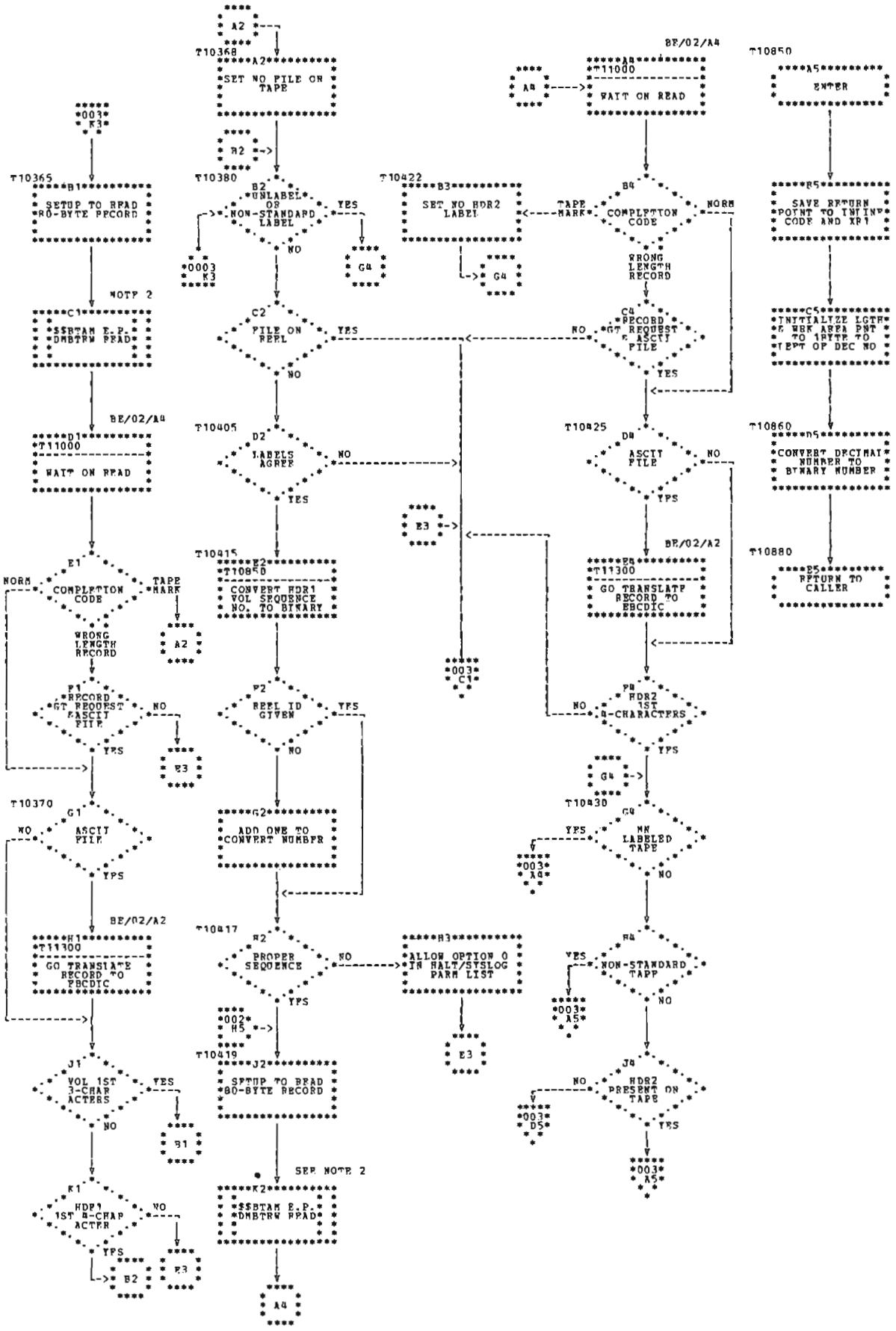


Chart BE (Part 4 of 4). Restart — Main Load ($$RSTR)

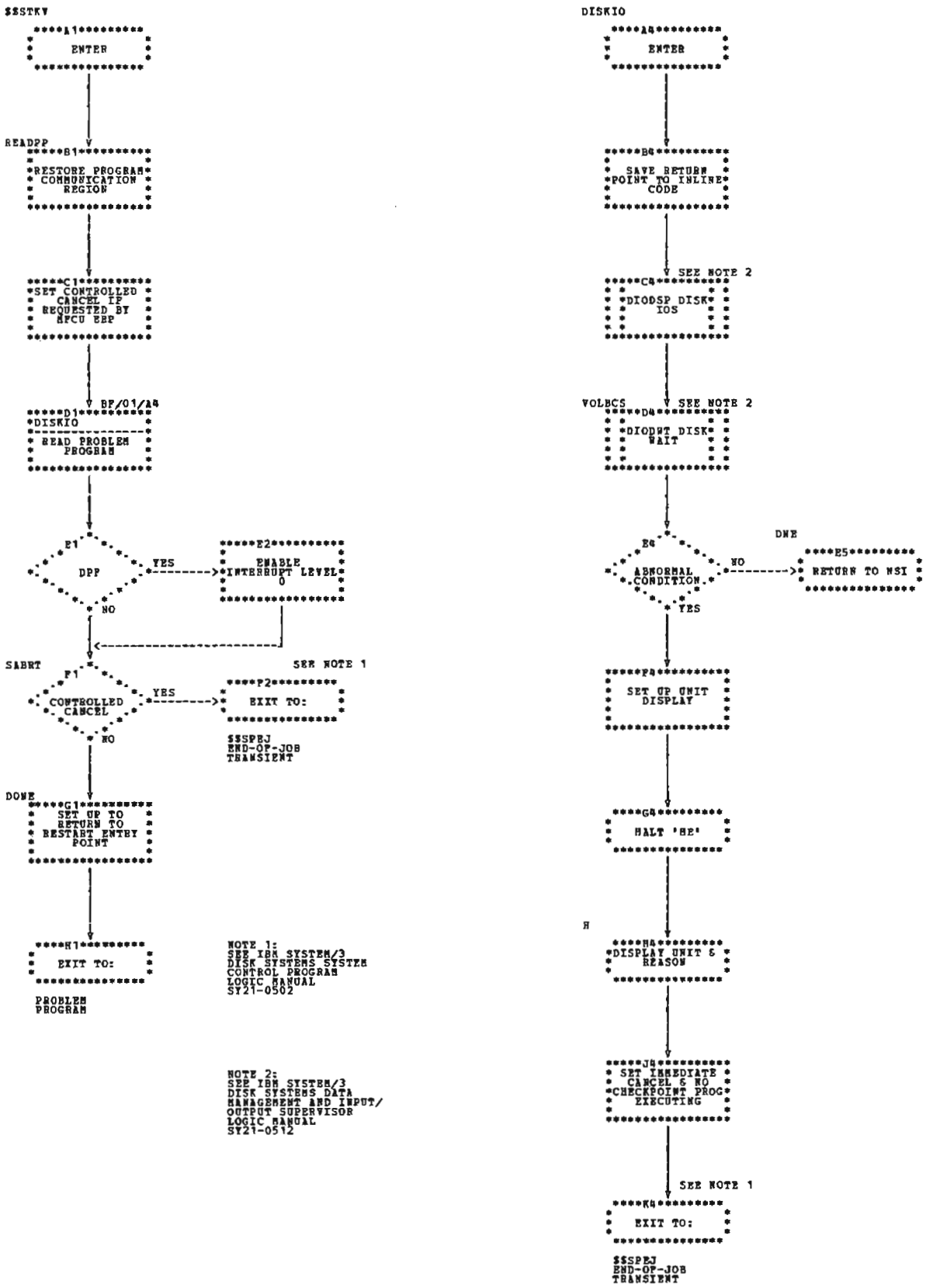


Chart BF. Restart – Problem Program and Final Load ($$STKV)

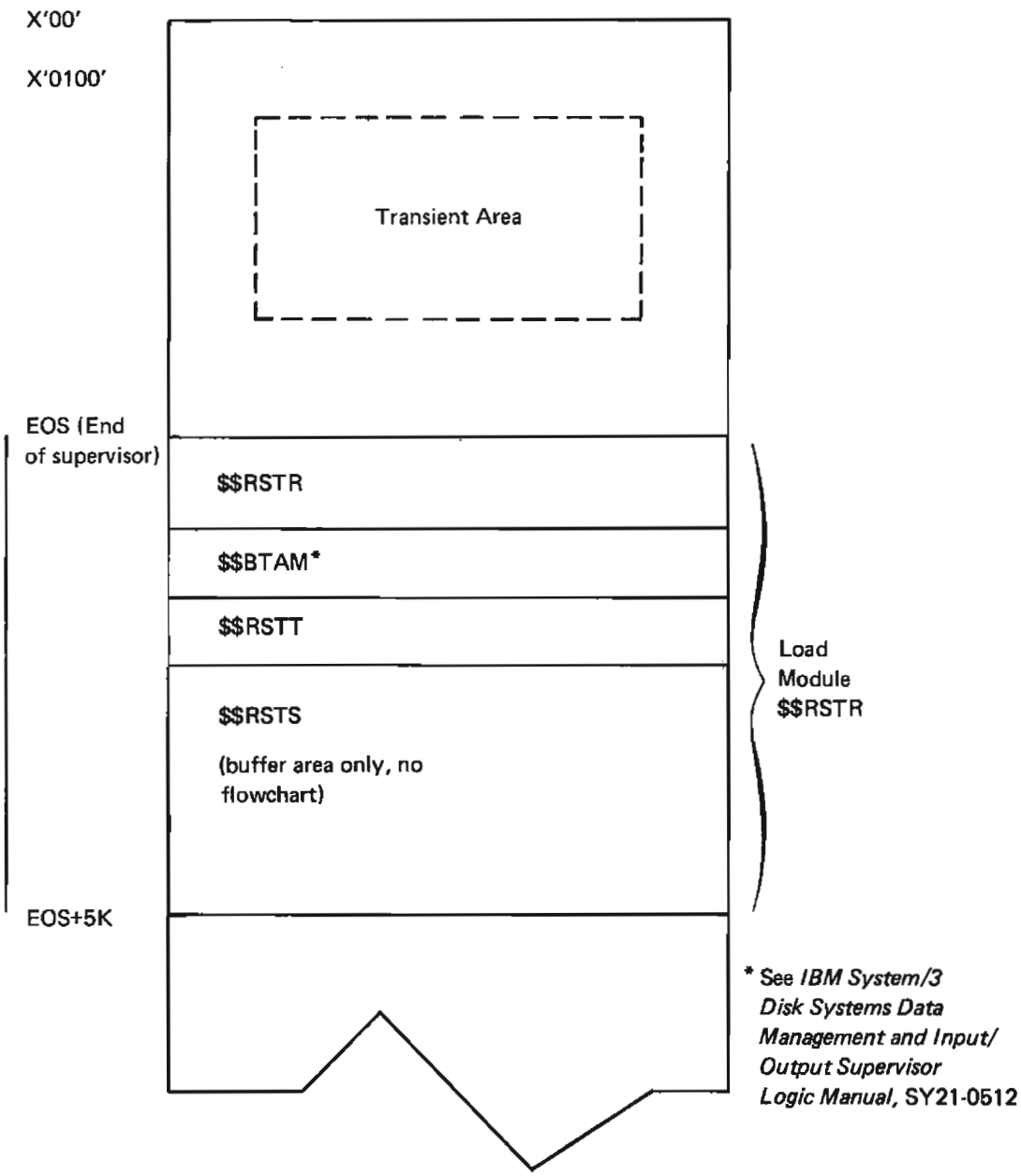


Figure 24. Main Storage Map Showing Transient Area and User Storage Needed by Restart

For a description of the following data areas, refer to *IBM System/3 Disk Systems System Control Program Logic Manual,* SY21-0502:

- Configuration record (CONFIG) in the scheduler work area
- Program level communications region (N1COMN)
- System communication region (NCPL1)
- Volume labels

Volume Label

The volume label of the system pack contains information about the checkpoint stored on the pack. The data is saved at:

Decimal Disp	*Meaning*
185	Bit 0 = 1 Active checkpoint
186	Number of last checkpoint request (01-99)

TABLE OF ENTRIES

Checkpoint saves pointers to the saved data at each checkpoint taken in a table of entries. Restart uses this saved data to resume execution of a program at the last checkpoint. The address of the disk area in which the table of entries is located is contained in field NCRCSS in the system communication region. The format of the table of entries is shown in Figure 25. The possible entries are shown in Figure 26.

	Entry ID*	Disk Address Where Stored (C/S/#)	Displacement into Sector	Length -1 of Data	Storage Location for Restore
Number of Bytes	1	3	2	1	2

*The following entry IDs may be used:

X'10' – Storage information
X'20' – Object program
X'40' – Checkpoint parameter list
X'C1' – 5445 Drive 1
X'C9' – 5445 Drive 2
X'A1' – 5444 Removable 1
X'B1' – 5444 Removable 2

Figure 25. Table of Entries

The entries *not* required *if missing* are not in sector and other entries are shifted left and appear in order given.

TABLE OF ENTRIES

Required	1	3	2	1	2	
YES	10	C S #	Displacement into sector	LNG-1 of Prog Comm Region	Program Communication @ (NCCOMN)	LNG depends on dedicated or DPF System
YES	10	C S #	Displacement into sector	0	SCH/D.M. 5445 Switches @ (NCSMV3)	
YES	10	C S #	Displacement into sector	0	SCH/D.M. 5444 Switches @ (NCSMV1)	
YES	10	C S #	Displacement into sector	1	Printer Page Size @ (NCRPSZ)	
NO	C1	C S #	Displacement into sector	5	N/A	
NO	C9	C S #	Displacement into sector	5	N/A	
NO	A1	C S #	Displacement into sector	5	N/A	
NO	B1	C S #	Displacement into sector	5	N/A	
NO	10	C S #	Displacement into sector	LNG-1 of Chain Image (119)	Chain Image @ (NCHIMG+119)	
YES	40	C S #	Displacement into sector	09	N/A	MFCU status, ARR, XR1, XR2, MFCU Print Data Register.
YES	40	C S #	Displacement into sector	15	N/A	Tape Repositioning Information *
YES	20	C S #	N/A	N/A	Program Level 1 Start @	

*The saved data for tape is 16 bytes, as follows:

Number of Bytes	1	1	2	1	1	2	1	1	2	1	1	2
	60 T1	SWA F1 Number	Current Block #	68 T2	SWA F1 Number	Current Block #	70 T3	SWA F1 Number	Current Block #	78 T4	SWA F1 Number	Current Block #

The entries are left adjusted and any entries not required are X'00'. The entries are *not* necessarily in this order.

● Figure 26. Possible Table Entries

APAR SUBMISSION

For APAR submission, the following information is necessary:

- Contents of IAR and ARR.
- Full core dump including the transient area.
- Disk dump of the checkpoint area. The C/S/No. of sectors can be obtained from the core dump (Field NCRCSS of the System Communications region).
- Disk dump of the program level 1 scheduler workarea. The C/S can be obtained from the core dump (Field NCSWRK of the System Communications region). The scheduler workarea is 48 sectors (2 tracks).
- Disk dump of the volume label of the system pack.

DATA SAVED AT CHECKPOINT

The table of entries (described in Section 3) can be used to determine what information was saved at the last checkpoint. This information is used by RESTART.

At each checkpoint, $$STKR and $$STKT build three sectors of data in the first three sectors of program level 1. The first sector contains data to be saved and pointers to buffers and workareas. The second sector contains the table of entries (see Section 3. Data Areas) which points to the table information in the third sector.

$CAM (*see* Compiler Access Method)
$OLAF (*see* AUTOLINK Segment List Build)
$OLAH (*see* Cross-Reference Segment List Build)
$OLAJ(*see* Sort AUTOLINK Segment List)
$OLAP (*see* Overlay Design)
$OLAR (*see* Overlay Segment List Build)
$OLAT (*see* Core Map Phase)
$OLBE (*see* Relocate, Resolve EXTRNs, and Build Load Module)
$OLBO (*see* Library Control Phase)
$OLER (*see* Error Routine)
$OLFTP (*see* Punch Phase)
$OLINK (*see* User Entry Phase 1)
$OLIN1 (*see* User Entry Phase 2)
$OLIN2 (*see* User Entry Phase 3)
$OLIN3 (*see* User Entry Phase 4)
$OLYNX (*see* Compiler Entry Phase)
$SOURCE segment list entries 33
$WORK file 3

APAR submission 43
AUTOLINK Segment List Build ($OLAF)
  description 10
  flowchart 19
AUTOLINK segment list entries 38

Compiler Access Method ($CAM)
  description 11
  flowchart 28
compiler entry
  input 3
  operational diagram 9
  output 3
  storage map 31
Compiler Entry Phase ($OLYNX)
  description 10
  flowchart 14
Core Map Phase ($OLAT)
  description 11
  flowchart 26
core usage map 20
cross-reference list 20
Cross-Reference Segment List Build ($OLAH)
  description 10
  flowchart 21
cross-reference segment list entries 39

data areas
  LOMMON 31
  segment list entries (*see* segment list entries)
description of phases
  AUTOLINK Segment List Build ($OLAF) 10
  Catalog Phase ($OLFOL) 11
  Compiler Access Method ($CAM) 11
description of phases (Cont'd)
  Compiler Entry Phase ($OLYNX) 10
  Core Map Phase ($OLAT) 11
  Cross-Reference Segment List Build ($OLAH) 10
  Error Routine ($OLER) 11
  Library Control Phase ($OLBO) 10
  Overlay Design ($OLAP) 11
  Punch Phase ($OLFTP) 11
  Relocate, Resolve EXTRNs, and Build Load Module ($OLBE) 11
  Sort AUTOLINK Segment List ($OLAJ) 11
  User Entry Phase 1 ($OLINK) 10
  User Entry Phase 2 ($OLAB) 10
diagnostic aids
  APAR submission 43
  messages 49
    numbers 49
  Overlay Fetch Routine (*see* Overlay Fetch Routine)

entering the overlay linkage editor 3
  compiler entry 3
  user entry 3
Error Routine ($OLER)
  description 11
  flowchart 22

finding an overlay 44
flowcharts for phases
  AUTOLINK Segment List Build ($OLAF) 19
  Catalog Phase ($OLFOL) 20
  Compiler Access Method ($CAM) 27
  Compiler Entry Phase ($OLYNX) 14
  Core Map Phase ($OLAT) 25
  Cross-Reference Segment List Build ($OLAH) 21
  Error Routine ($OLER) 22
  Library Control Phase ($OLBO) 15
  Overlay Design ($OLAP) 24
  Overlay Fetch Routine 45
  Punch Phase ($OLFTP) 16
  Relocate, Resolve EXTRNs, and Build Load Module ($OLBE) 26
  Sort AUTOLINK Segment List ($OLAJ) 21
  User Entry Phase 1 ($OLINK) 12
  User Entry Phase 2 ($OLAB) 13

input
  compiler entry 3
  user entry 3

Library Control Phase ($OLBO)
  description 10
  flowchart 15
LOMMON 7

machine requirements 3

messages 49
  numbers 49

numbers, messages 49

operational diagram 9
output
  compiler entry 3
  user entry 8
Overlay Design ($OLAP)
  description 11
  flowchart 24
Overlay Fetch Routine 43
  core usage map 20
  cross-reference list 20
  finding an overlay 44
  flowchart 45
  overlay fetch table 44
  sample core dump 47
  transfer vectors 44
overlay fetch table 44
Overlay Segment List Build ($OLAR)
  description 11
  flowchart 30
overlay segment list entries 33

pre-auto segment list entries 37
Punch Phase ($OLFTP)
  description 11
  flowchart 16

Relocate, Resolve EXTRNs, and Build Load Module ($OLBE)
  description 11
  flowchart 27
R module 3

sample core dump 47
segment list entries 33
  $SOURCE 37
  AUTOLINK 38
  cross-reference 39
  overlay 41
  pre-auto 37
  sort 40
Sort AUTOLINK Segment List ($OLAJ)
  description 11
  flowchart 23
sort segment list entries 40
storage map
  compiler entry 31
  user entry 32

transfer vectors 44

user entry
  input 3
  operational diagram 10
  output 8
  storage map 32
User Entry Phase 1 ($OLINK)
  description 10
  flowchart 12
User Entry Phase 2 ($OLIN1)
  description 10
  flowchart 13
User Entry Phase 3 ($OLIN2)
  description 10
  flowchart 20
User Entry Phase 4 ($OLIN3)
  description 10
  flowchart 17

Checkpoint/Restart

$$RSTR (*see* Restart-Main Load)
$$STKP (*see* Checkpoint-Main Load)
$$STKQ (*see* Checkpoint-Quiesce Magnetic Tape I/O)
$$STKR (*see* Checkpoint Problem Program and SWA Load)
$$STKT (*see* Checkpoint-Final Load)
$$STKV (*see* Restart-Problem Program and Final Load)

Checkpoint
  Final Load (*see* Checkpoint-Final Load
  function 55
  linkage 55
  Main Load (*see* Checkpoint-Main Load)
  operational diagram 57
  Problem Program and SWA Load (*see* Checkpoint-Problem Program and SWA Load)
  storage map 61
Checkpoint-Final Load ($$STKT)
  description 60
  flowchart 66
Checkpoint-Main Load ($$STKP)
  description 57
  flowchart 62
Checkpoint-Problem Program and SWA Load ($$STKR)
  description 59
  flowchart 65
Checkpoint-Quiesce Magnetic Tape I/O ($$STKQ)
  description 58
  flowchart 64
CONFIG (configuration record) 57

data areas
  CONFIG (configuration record) 57, 77
  NCPLI (system communication region) 57, 77
  NICOMN (program level communications region) 57, 77
  SWA (scheduler work area) 57
  table of entries 77
  volume labels 57, 77

function
  Checkpoint 55
  Restart 68

linkage
  Checkpoint 55
  Restart 68

machine requirements 53

NCPLI (system communication region) 57, 77
NICOMN (program level communication) 57, 77

operational diagram
Checkpoint 56
Restart 69

Restart
function 68
linkage 68
Main Load (*see* Restart-Main Load)
operational diagram 69
Problem Program and Final Load (*see* Restart-Problem Program and Final Load)
storage map 76
Restart-Main Load ($$RSTR)
description 69
flowchart 71
Restart-Problem Program and Final Load ($$STKV)
description 70
flowchart 75

SWA (scheduler work area) 57
storage map
Checkpoint 61
Restart 76

table of entries 77

volume labels 77

IBM

Technical Newsletter

This Newsletter No. SN21-7668

Date March 15, 1973

Base Publication No. SY21-0530-1

File No. System

Previous Newsletters None

IBM System/3 Overlay Linkage Editor and Checkpoint/Restart Program Logic Manual

This Technical Newsletter, a part of version, 08 modification 00 of the IBM System/3 Model 10 and Model 6 Disk Systems, provides replacement pages for the subject publication. These replacement pages remain in effect for subsequent versions and modifications unless specifically altered. Pages to be inserted and/or removed are:

5, 6
9 through 12
19 through 24
30.1 (added)
31, 32
35 through 40
71 through 74

Changes to text and illustrations are indicated by a vertical line at the left of the change; new or extensively revised illustrations are denoted by the symbol • at the left of the caption.

Summary of Amendments

An error message phase has been added to the Overlay Linkage Editor.

Note: Please file this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Publications, Department 245, Rochester, Minnesota 55901

Printed in U.S.A.

IBM Technical Newsletter

This Newsletter No.	SN21-5289
Date	April 1975
Base Publication No.	SY21-0530-1
File No.	S3-31
Previous Newsletters	SN21-7668

IBM System/3 Overlay Linkage Editor and Checkpoint/Restart Programs Logic Manual

This Technical Newsletter, a part of version 12, modification 00 of IBM System/3 Model 10 Disk System System Control Programming, Program Number 5702-SC1, and version 12, modification 00 of IBM System/3 Model 6 System Control Programming, Program Number 5703-SC1, provides replacement pages for the subject publication. These replacement pages remain in effect for subsequent versions and modifications unless specifically altered. Pages to be inserted and/or removed are:

i, ii
3, 4
61, 62

Changes to text and illustrations are indicated by a vertical line at the left of the change; new or extensively revised illustrations are denoted by the symbol • at the left of the caption.

Summary of Amendments

- Add Model 8 reference to Preface

Note: Please file this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Publications, Department 245, Rochester, Minnesota 55901

Printed in U.S.A.

IBM System/3 Printed in USA SY21-0530-1

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

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