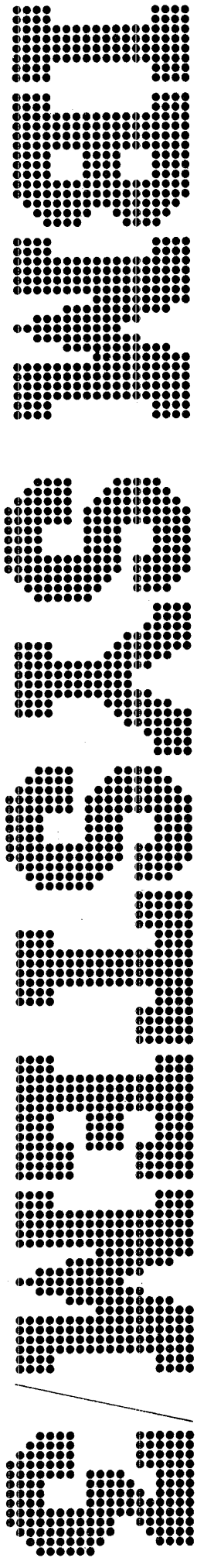


IBM System/3
Models 4, 8, 10, and 12
Communications Control Program
Logic Manual

Program Numbers:
- 5702-SC1 (Models 8 and 10)
- 5703-SC1 (Model 4)
- 5705-SC1 (Model 12)

Feature 6033/6070/6071

SY21-0531-2
File No. S3-36

Third Edition (June 1976)

This is a major revision of, and obsoletes, SY21-0531-1, and Technical Newsletters SN21-5272 and SN21-5348. Numerous miscellaneous changes are added throughout the publication. The manual should be reviewed in its entirety.

This edition applies to the System/3 program versions listed below and to all subsequent versions and modifications until otherwise indicated in new editions or technical newsletters.

SCP Program Number	System/3	Feature Number	Version	Modification
5702-SC1	Models 8 and 10	6033	13	00
5703-SC1	Model 4	6033	13	00
5705-SC1	Model 12	6070/6071	01	00

Changes are made to this information periodically. Before using this publication to operate an IBM system, refer to the latest *IBM System/3 Bibliography,* GC20-8080, for the edition that is applicable and current.

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

A form for readers' comments is provided at the back of this publication. If it has been removed, address your comments to IBM Corporation, Publications, Department 245, Rochester, Minnesota 55901.

Preface

This logic manual is intended for IBM personnel responsible for maintaining the Communications Control Program (CCP), a feature of IBM System/3 Disk System Management, IBM System/3 Models 4, 8, 10, and 12. This feature facilitates the implementation of telecommunications applications on the Models 4, 8, 10, and 12. The manual is a guide to CCP program listing; it does not contain instructions for using CCP.

This manual is divided into six parts:

- Introduction (Chapter 1)
- Generation/Installation (Chapters 2-3)
- Assignment and Offline Programs (Chapters 4-7)
- Operation (Chapters 8-10)
- Service Aids (Chapters 11-14)
- Appendixes

Part 1 introduces the components of CCP. Parts 2 through 5 describe programs within CCP. Each chapter in these parts contains:

- An introduction to the topic of the chapter.
- A method of operation section showing the functional relation of the different parts within the program described. Method of operation diagrams are numbered in the form cM.nnnn, where c is the chapter number, M is method of operation, and nnnn refers to the sequence of the diagrams within the chapter.
- A program organization section showing main storage maps for the program described and either HIPO diagrams or module descriptions and flowcharts describing individual modules. The HIPO diagrams in this manual do not include cross references to listing labels or extended descriptions in text, but do include cross references to other HIPOs and flowcharts. HIPO diagrams in the program organization sections are numbered in the form cP.nnnn, where c is the chapter number, P is program organization, and nnnn refers to the sequence of the diagrams within the chapter.

Appendix A in Part 6 is a system directory for locating information in Parts 2 through 5. Appendix D describes the macros used during CCP generation. Appendix F illustrates the flowchart and HIPO conventions followed in this manual.

Throughout this manual, page numbers are in the form c-n, where c identifies the chapter or appendix and n is the page number within the chapter or appendix. The running feet, which precede the page number on odd-numbered pages, identify the section of the chapter (for example, method of operation or program organization) or, in the appendixes, a major topic of the appendix.

For ease of illustration, many of the examples in this book use card-like figures to represent records. This does not imply that a card device must be used for input or output in these situations. Any of several input/output devices can be used, depending on which System/3 model and configuration you are using.

Integrated communications adapter (ICA) is available only on the Model 8 and must be addressed as BSCA line 2. ICA, local display adapter, and BSCA-2 are mutually exclusive on the Model 8. Local display adapter is available on Models 8 and 12. The locally attached work station is available on Model 4.

Not all CCP devices and features available on Model 10 are available on Models 4, 8, and 12. Users should be familiar with the information in the *Introduction* manual for the model they are using.

In this manual a Model 12 reference to the 5444 indicates the 3340 Direct Access Storage Facility simulation area. A Model 12 reference to the 5445 indicates the 3340 Direct Access Storage Facility main data area.

Prerequisite Publications

You should be familiar with the following publications that describe:

- Communications Control Program:

  IBM System/3 Communications Control Program Messages Manual, GC21-5170

  IBM System/3 Communications Control Program Programmer's Reference Manual, GC21-7579

  IBM System/3 Communications Control Program System Design Guide, GC21-5165

  IBM System/3 Models 4, 8, 10, and 12 Communications Control Program Data Areas and Diagnostic Aids, SY21-0048

  IBM System/3 Models 8, 10, and 12 Communications Control Program System Reference Manual, GC21-7588

  IBM System/3 Communications Control Program Terminal Operator's Guide, GC21-7580

  IBM System/3 Models 10 and 12 Communications Control Program System Operator's Guide, GC21-7581

- Multiline/Multipoint and Multiple Line Terminal Adapter:

  IBM System/3 Disk Systems Binary Synchronous Communications Programming Support Input/Output Control System Logic Manual, SY21-0526

  IBM System/3 Multiple Line Terminal Adapter RPQ Program Logic Manual Supporting RPQs S40028 through S40033, SY21-0527

  IBM System/3 Multiline/Multipoint Binary Synchronous Communications Reference Manual, GC21-7573

  IBM System/3 Disk Sort Reference Manual, SC21-7522

  IBM System/3 Models 8 and 10 System Control Programming Reference Manual, GC21-7512

  IBM System/3 Model 12 System Control Programming Reference Manual, GC21-5130

  IBM System/3 Models 4, 6, 8, 10, and 12 System Generation Reference Manual, GC21-5126

- Model 4

  IBM System/3 Model 4 Introduction, GC21-5146

  IBM System/3 Model 4 CCP Concepts and System Design Guide, GC21-5148

  IBM System/3 Model 4 CCP Programmer's Reference Manual, GC21-5150

- Models 10 and 12 System Management:

  IBM System/3 Models 4, 6, 8, and 10 Disk Systems System Control Program Logic Manual, SY21-0502

  IBM System/3 Disk Systems Data Management and Input/Output Supervisor Logic Manual, SY21-0512

  IBM System/3 Overlay Linkage Editor and Checkpoint/Restart Program Logic Manual, SY21-0530

  IBM System/3 Models 8 and 10 System Control Programming Reference Manual, GC21-7512

  IBM System/3 Model 12 System Control Programming Reference Manual, GC21-5130

Related Publications

Because the Communications Control Program supports communications programs written in RPG II, COBOL, FORTRAN IV, and Basic Assembler, the following publications may be useful:

IBM System/3 Disk Systems RPG II Logic Manual, LY21-0501

IBM System/3 Subset American National Standard COBOL Compiler and Library Program Logic Manual, LY28-6421

IBM System/3 Disk FORTRAN IV Program Logic Manual, LY28-6848

IBM System/3 Basic Assembler Program Logic Manual, LY21-0504

IBM System/3 Models 4, 6, 8, 10, and 12 System Data Areas and Diagnostic Aids, SY21-0045

Contents

Addr	A main storage address.
AID	Attention ID.
AM	Allocation manager.
Arith	Arithmetic variable.
ARR	Address recall register.
ASCII	American national standard code for information interchange.
Bin	Binary number.
Bool	Binary (Boolean) variable.
BSCA	Binary synchronous communication adapter.
calcs	Calculations.
CAM	Compiler access method.
CCP	Communications control program.
CCPIVP	CCP installation verification program.
CDE	Contents directory entry.
CEFE	Customer Engineer Field Engineering.
CH	Disk address (cylinder/head).
Char	Alphameric characters.
Charn	Character variable.
CHRD	Disk address (cylinder/head/record/displacement).
CM	Communications manager.
CMD	Command.
COMARA	Communications area.
COMCOM	Common communication area.
COMMON	Common area.
CP	Command processor.
CPU	Processing unit.
CS or C/S	Disk address (cylinder/sector).
CSD	Disk address (cylinder/sector/byte displacement within sector).
CSDD	Disk address (cylinder/sector/byte displacement from the beginning of that sector).
CSN	Disk address and length (cylinder/sector/number of sectors).
DC	Define constant.
Dec	Zoned decimal characters.
DFCR	Display format control routine.
DFGR	Display format generator routine.
DFF	Display format facility.
DFT	Display format table.
Disp	Displacement.
distrib	Distribution.
DME	Data mode escape.
DPF	Dual program feature.
DSM	Disk system management.
DTF	Define the file.
DTT	Define the table.
EAU	Erase all unprotected.
EBCDIC	Extended binary coded decimal interchange code.

Chapter 1. Introduction to the Components of CCP

CCP (Communications Control Program) is a feature of System/3 Disk System Management (DSM). CCP operates in conjunction with DSM and with MLMP (Multiline/Multipoint) and/or MLTA (Multiple Line Terminal Adapter) IOCS to provide the control required for a communications-based operating system.

CCP includes not only those modules used during the time that a communications network is being controlled, but also modules that are used to prepare for the operation of an online communications network and several auxiliary programs that enhance the serviceability of CCP. Apart from auxiliary service programs and subroutines incorporated into user programs that execute under CCP, the modules that form CCP function in one of four stages:

1. Generation Stage. The stage in which a generalized set of modules distributed as the CCP feature are tailored to form a system appropriate to the user's needs.

2. Installation Stage (Model 4 only). The stage in which the desired version of CCP is copied to the production pack and to the RPG II program preparation pack.

3. Assignment Stage. The stage in which preparatory programs operating directly under Disk System Management are used to establish control information that is used by CCP when controlling an online communications network.

4. Operational Stage. The stage in which CCP controls a communications network. The Operational Stage consists of:

   - Operational Startup. The control information created during the Assignment Stage is gathered to initiate the control of the communications network.

   - CCP Execution. CCP controls communication with terminals of the network and runs user programs requested by operators.

   - Operational Shutdown. CCP is terminated and control is returned to Disk System Management.

GENERATION STAGE (Models 8, 10, and 12 Only)

The user receives a Model 8 or 10 CCP on a 5444 disk pack or a Model 12 CCP on a 3348 disk module. As distributed, CCP is not suitable for immediate production use because the modules must be tailored to the user's hardware configuration and to the functions required in the communication system. Tailoring the distributed modules and recording that tailored system on a disk pack are the functions of the CCP Generation Stage. Figure 1-1 is an overview of CCP Generation.

CCP Generation is not required on Model 4.

Results of Generation

The distribution pack received by the user contains both the modules to be tailored and the programs to perform that tailoring. The output of CCP Generation is a set of modules which may be split among several packs if the user so specifies:

- Production pack. Contains those modules which are used in the Assignment and Operational Stages of CCP, as well as certain service aid programs. This is the pack which must be online when executing any component of CCP beyond generation.

- Program preparation pack. Pack to be used when compiling and link editing user programs which are to run under CCP control. The communications service subroutines (macros, in the case of Basic Assembler programs) and those subroutines which must be incorporated in a user program which accesses unit record devices are copied to the program preparation pack. There may be as many program preparation packs as there are languages to be supported by CCP; or the support for several languages may be directed to the same program preparation pack. A program preparation pack could be the same pack as the production pack.

- Assignment file pack. The Assignment File ($CCPFILE) is allocated and initialized on this pack. This pack may be the same as the production pack. The assignment file pack must be online during the Operational Stage and during the running of several of the preparatory programs, as indicated later in this chapter.

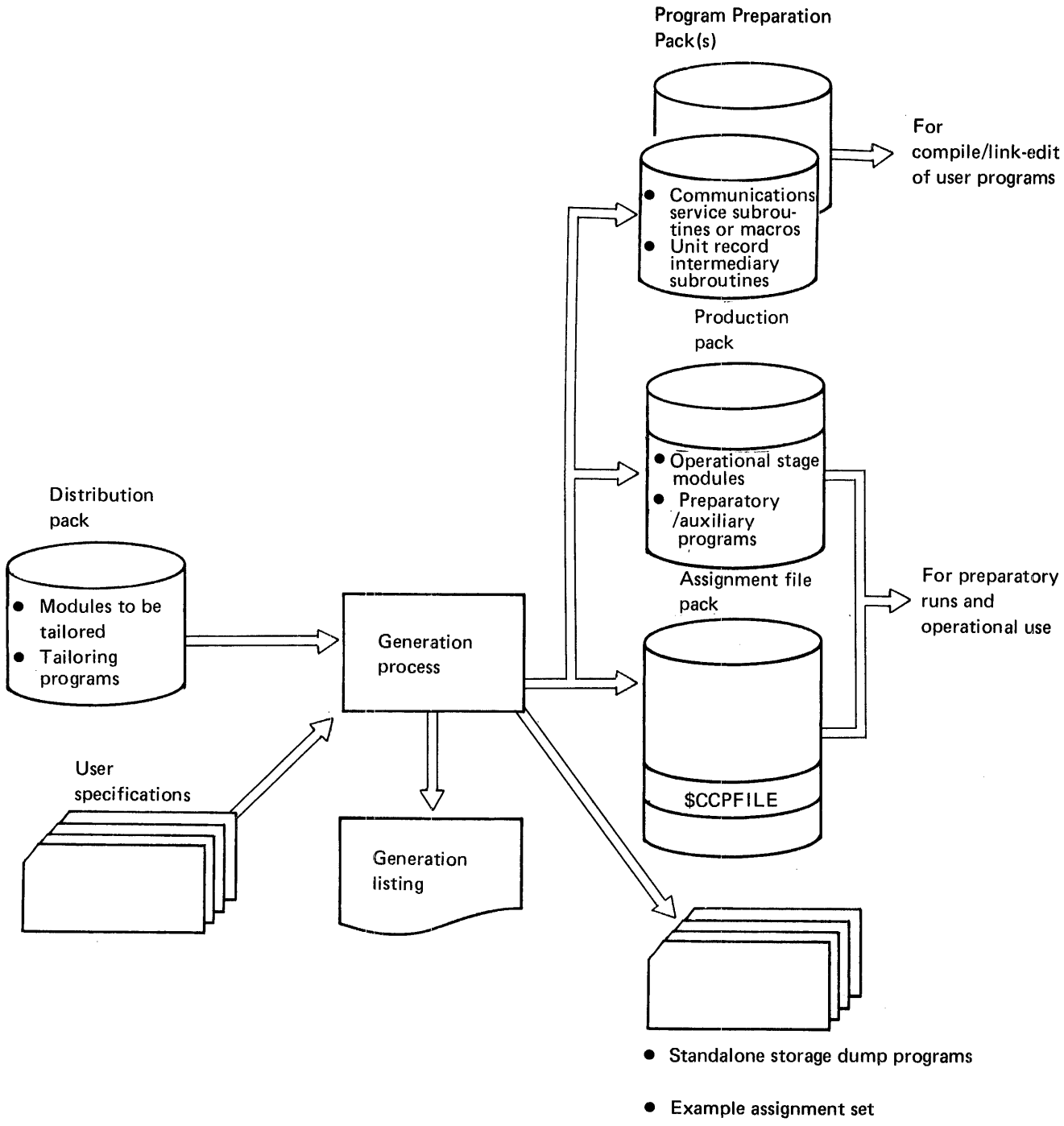


Figure 1-1. Generation (Models 8, 10, and 12 Only)

Tailoring Process

Tailoring during CCP generation takes place at several levels. According to user specifications:

- Load modules are selectively copied to the user's packs.
- Relocatable modules are selectively chosen to be included in a link edit.
- Individual instructions are selectively chosen to be included in the execution-time resident control module.

The last level of tailoring requires the use of the Macro Processor to selectively form the individual instructions in the resident control module, and the use of a language translator — called the SCP Generator — to convert those instructions to machine language.

The user's CCP, once generated, is limited to the set of functions and device types that were specified during generation. Any addition to the hardware types or the functions expected of CCP requires a new generation.

On the other hand, within that set of functions and hardware types, any number of specifics in the user's operating environment may be changed without requiring a new generation to support them. For example, while a new terminal type cannot be supported without a regeneration, the user can vary the configuration of those terminals among lines, the terminal addresses, and the number of terminals attached. Further, while the user has possibly limited the types of disk file support, the generation has not bound him to any specific set of disk files or to any specific set of programs.

The specifics of terminals, disk files, and programs are established by a brief preparatory run known as Assignment Build.

INSTALLATION STAGE (MODEL 4 ONLY)

Installation stage on Model 4 consists of a set of procedures that copy the desired version of CCP to the production pack and RPG II program preparation pack.

Functions Supported

The CCP installation stage is eliminated for Model 4. Instead, two versions of CCP are pregenerated and shipped to the users; one version that supports only 3270 on control station line and a second version that supports all BSCA devices and all BSCA line configurations.

The following CCP generation control statements are used to pregenerate the two versions:

		Minimum Version	Maximum Version
$EIOD	CARD-	NO	NO
	PRINTER-	5203 (note 1)	5203 (note 1)
	N3741-	NO	NO
	DISKS-	'R2, F2'	'R2, F2'
	D5445-	NO	NO
$EFAC	MAXEUP-	4	4
	DPF-	NO	NO
	ESCAPE-	'//////'	'//////'
	PGMCNT-	YES	YES
	FSHARE-	YES	YES
	SYMFIL-	YES	YES
	FORMAT-	YES	YES
	PRUF-	YES	YES
$EPLG	LANG-	RPG II	RPG II
$ESEC	SECURE-	NO (note 2)	NO (note 2)
	LUSI-	0	0
$EFIL	SETS-	(note 3)	(note 3)
	PROGS-	(note 3)	(note 3)
	DFILES-	(note 3)	(note 3)
	TERMS-	(note 3)	(note 3)
	DUMPS-	(note 3)	(note 3)
	CORE-	(note 3)	(note 3)
	TRACE-	1	1

		Minimum Version	Maximum Version
$EBSC	BSCA-	2	2
	DIAL-	NO	YES
	PP-	NO	YES
	MP-	NO	YES
	CS-	YES	YES
	GETMSG-	YES	YES
	ITB-	NO	YES
	RECSEP-	--	1E
	ASCII-	NO	YES
	EBCDIC-	YES	YES
	XPRNCY-	NO	YES
	RESPOL-	YES	YES
	AUTORS-	NO	YES
	TYPE-	3277M1/M2	3277M1/M2
	TYPE-	3275M1/M2	3275M1/M2
	TYPE-	3284M1/M2	3284M1/M2
	TYPE-	3286M1/M2	3286M1/M2
	TYPE-	--	3735
	TYPE-	--	3741
	TYPE-	--	CPU
$EGEN	MINRES-	NO	NO

Notes:

1. The 5203 indicator will be used to indicate 5213.
2. Even though no security is specified on the pregenerated versions, a user can make use of CCP security by specifying the PASSWORD parameter on the SYSTEM statement during the assignment stage.
3. These parameters are used to determine the size of $CCPFILE. Therefore, they are not applicable here.

ASSIGNMENT STAGE AND OTHER PREPARATORY PROGRAMS

Because many of the specifics of the user's operational environment are deliberately not bound at generation, additional programs are provided to permit the user to establish the necessary specifics preparatory to the operational use of CCP. These programs are:

- The Assignment Build Program: $CCPAS and related modules.
- The Assignment List Program: $CCPAL.
- The User Security Information Build Program: $CCPAU.
- The Display Format Facility (DFF) Programs:
  Display Format Generator Program: $CCPDF.
  Printer Format Generator Program: $CCPPF.
  Display Format Test Routine: $CCPDT.
- LOG Build Program (Model 4 only): $CC1BL is provided on Model 4 to create the required $CCPLOG file.

Assignment Build Program

Assignment Build (Figure 1-2) creates, deletes, and replaces user-defined assignment sets in $CCPFILE. An assignment set is a set of control information which defines the specifics (not defined at generation) of an environment in which the execution of CCP will occur. An assignment set defines:

- A line and terminal configuration.
- A set of names by which user programs will refer to terminals.
- The characteristics of each terminal.
- The disk data files to be accessed by user programs.
- The set of user programs which may run and the system resources required by each.
- Optional password security (Model 4 only).

A number of different assignment sets may be defined in $CCPFILE. Upon each run of the CCP Operational Stage, one assignment set is chosen by the system operator to apply to the entire run.

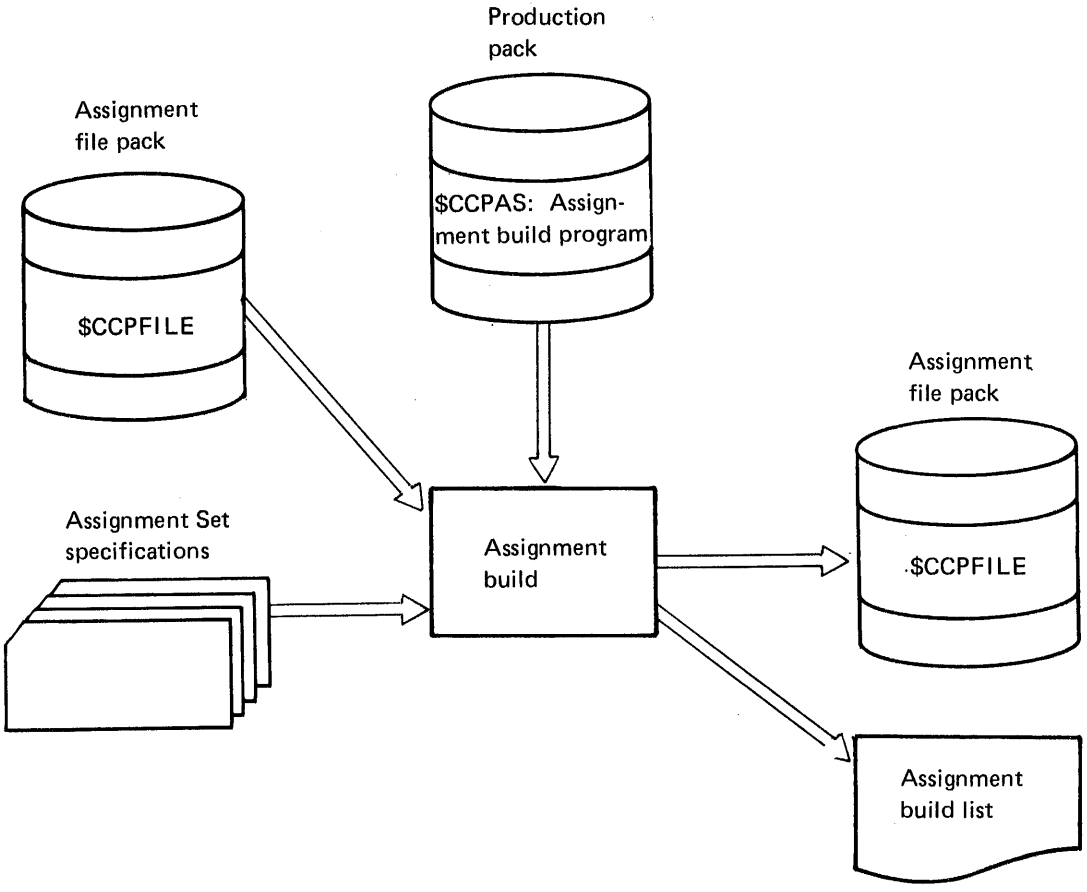


Figure 1-2. Assignment Build

Assignment List Program

The Assignment List Program ($CCPAL) lists on the system print device those portions of $CCPFILE specified by the user. This program can be used to list, in tabular form, the contents of any assignment set currently in $CCPFILE. It can also be used to list the configuration record of $CCPFILE, in which are recorded the system functions and hardware configuration specified by the user during CCP generation. Figure 1-3 is an overview of Assignment List.

One of the system functions which can be requested at CCP generation is the maintenance of a count of requests made for each user program during operational runs of CCP. If this function was specified at CCP Generation, Assignment List can be used to list the number of times each program of an assignment set has been requested since the last time the cumulative counts were reset. Assignment List will, upon the user's request, reset those counts after listing them.

User Security Information Build Program (Models 8, 10, and 12 Only)

At CCP generation, a user-written routine to check the validity of every sign-on of a terminal during a CCP execution can be provided. To perform this check, the user's routine must have access to a set of information (prepared by the installation) that defines what information must accompany each sign-on in order for the sign-on to be considered valid.

The User Security Information Build Program ($CCPAU) accepts installation-defined security information and records that information in a module ($CC4Z9) in the object library of the production pack (Figure 1-4). The contents of this module will be accessible to the user's security routine during the execution of CCP.

The User Security Information Build Program is not supported on Model 4.

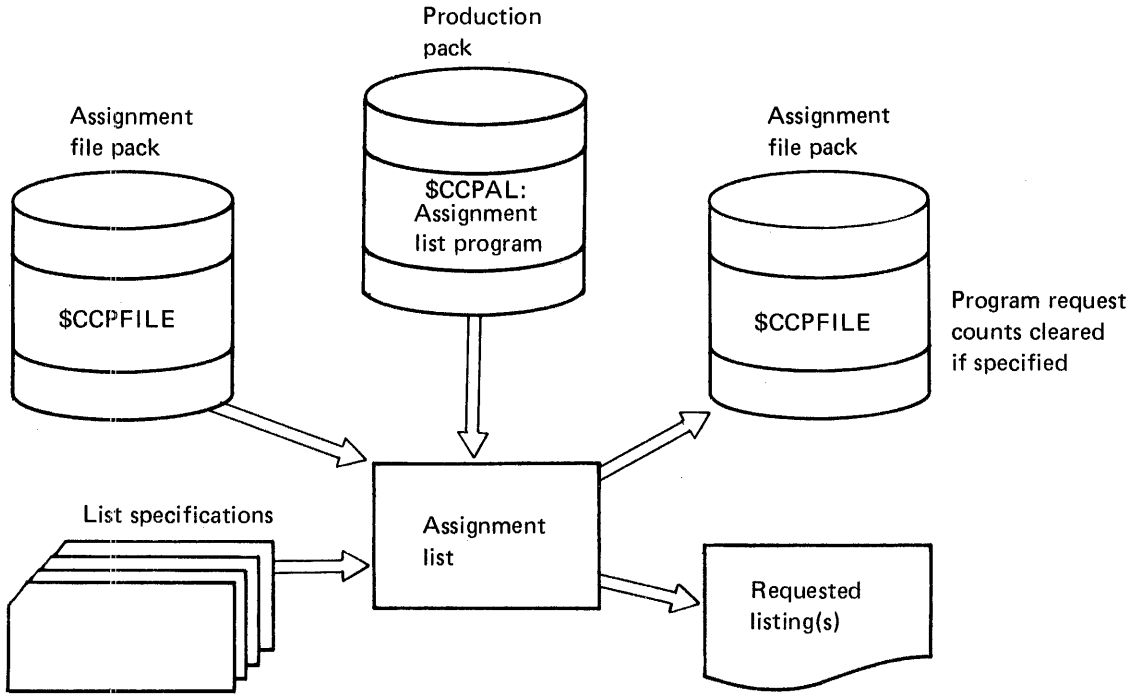


Figure 1-3. Assignment List

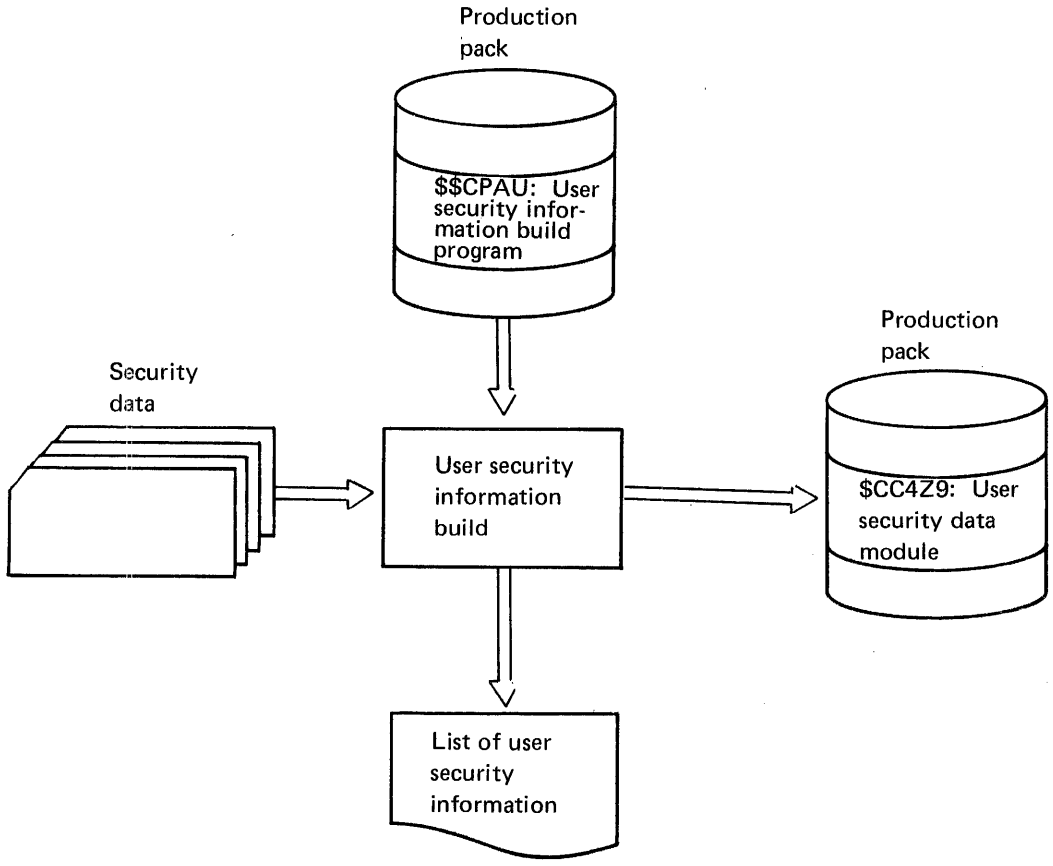


Figure 1-4. User Security Information Build (Models 8, 10, and 12 Only)

Display Format Generator Program

A significant feature in the ease of use of CCP is the Display Format Facility (DFF). DFF permits user programs, during the execution of CCP, to access and manipulate pre-defined screen formats to be displayed or printed on a 3270 Information Display System.

The Display Format Generator Program ($CCPDF) constructs pre-defined formats and initial data contents from specifications provided by the user and records them in an object library as load modules named $Zxxxx. The modules are accessible to user programs which make use of DFF during the execution of CCP. Figure 1-5 is an overview of the Display Format Generator Program.

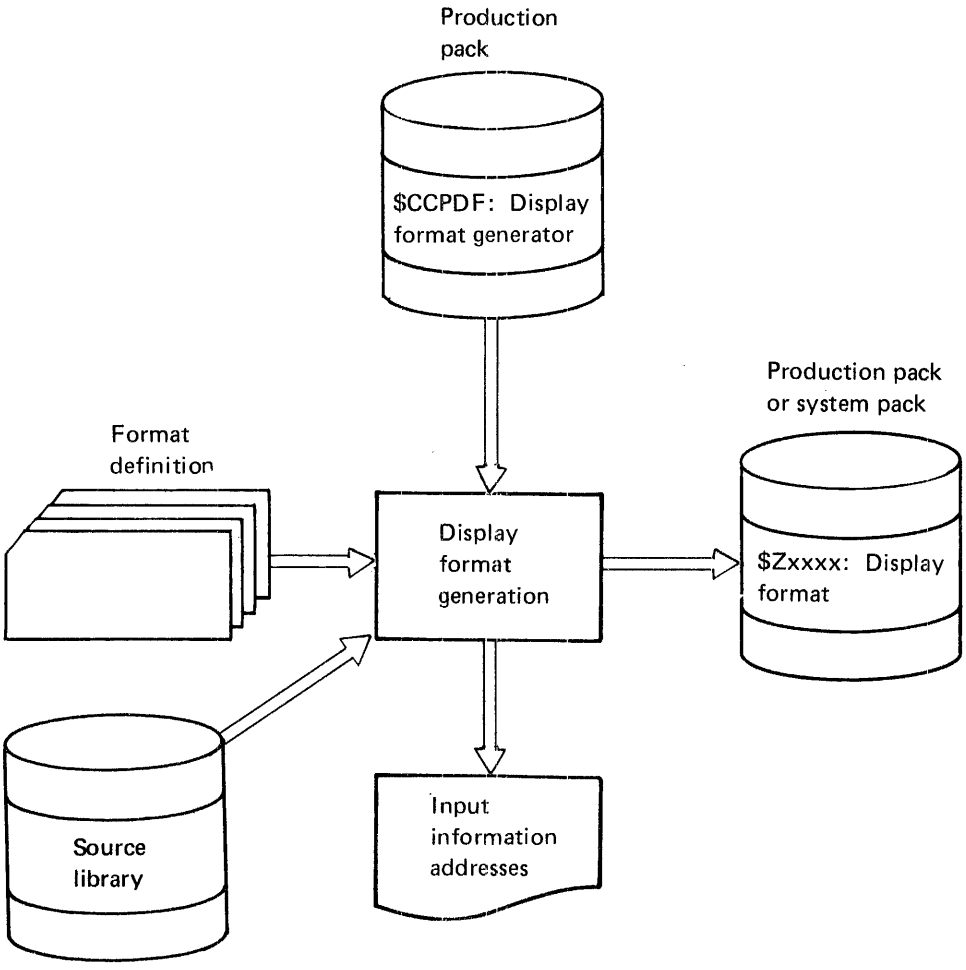


Figure 1-5. Display Format Generator

Printer Format Generator Program

DFF permits user programs, during the execution of CCP, to access and manipulate pre-defined printer formats to be printed on a 3270 Information Display System Printer.

The Printer Format Generator Program ($CCPPF) constructs pre-defined formats and initial data contents from specifications provided by the user and records them in an object library as load modules named $Zxxxx. The modules are accessible to user programs that use DFF during the execution of CCP. Figure 1-5.1 is an overview of the Printer Format Generator Program.

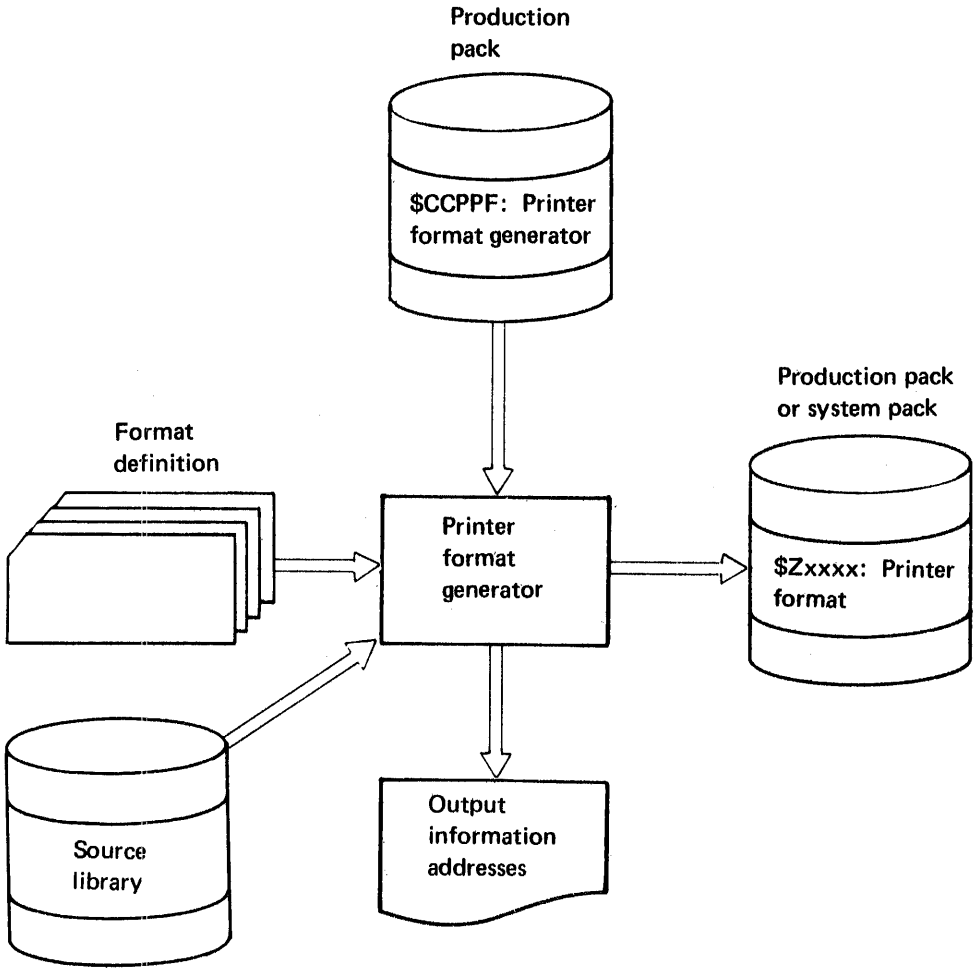


Figure 1.5.1. Printer Format Generator

THE OPERATIONAL STAGE

Startup

The Operational Stage begins with the startup of CCP, initiated by the system operator via OCL. Under the direction of Startup (Figure 1-6), the system operator selects from $CCPFILE the assignment set defining the environment which applies to the current execution of CCP. The system operator then may make minor modifications to the specifications in the assignment set.

CCP Startup then prepares for the execution of CCP by:

- Locating and initializing all CCP transient routines.
- Locating all user programs and writing required information about those programs from the object library directory to $CCPFILE.
- Allocating and opening all disk files to be used in this execution of CCP.
- Loading the execution-time resident control routine $CC4.
- Building control blocks in main storage which reflect the specifics of the defined environment.
- Opening communication lines.
- Turning control over to the resident control routine.

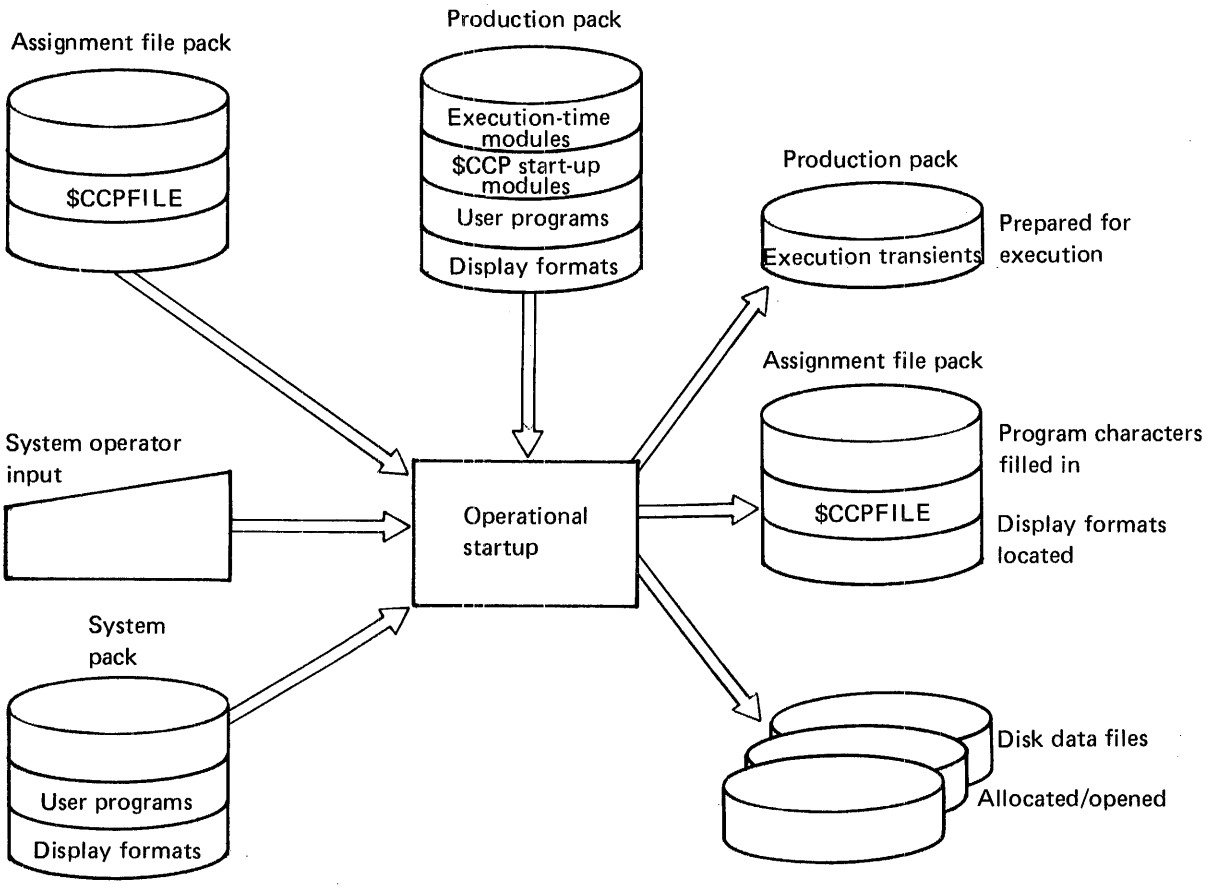


Figure 1-6. Operational Startup

Execution

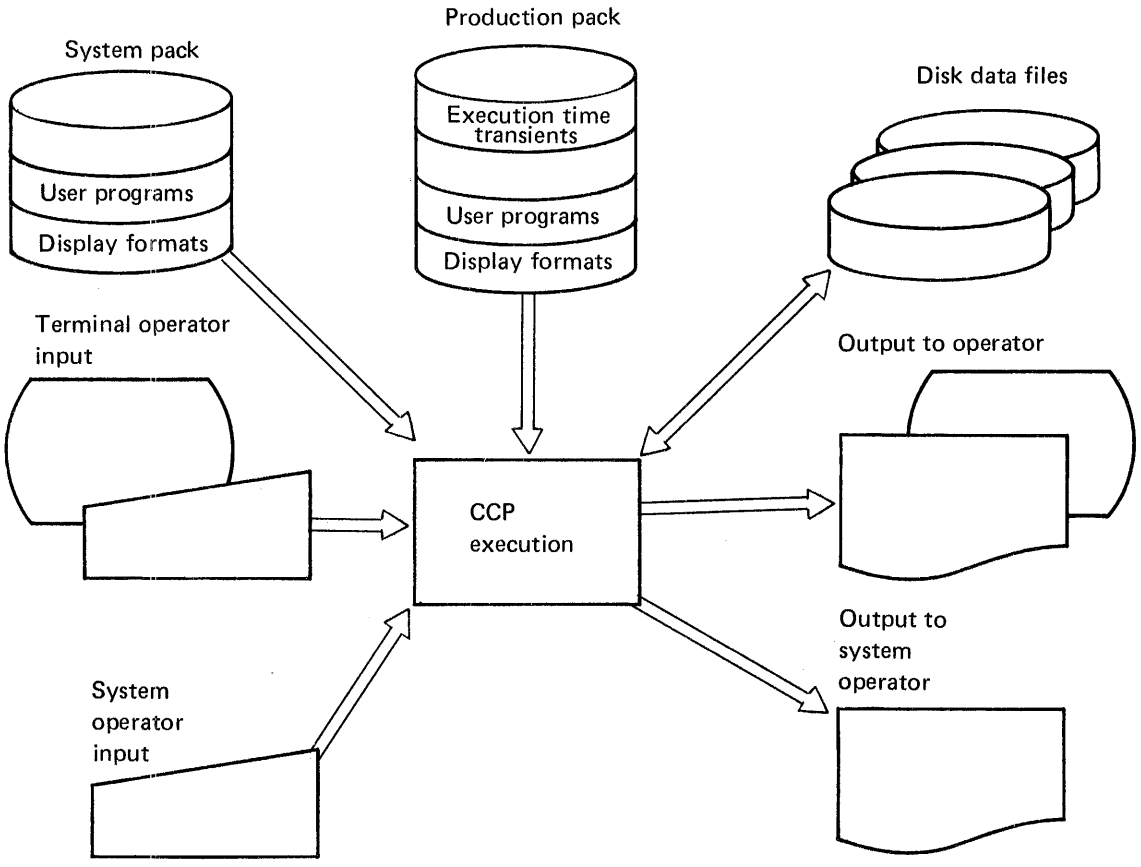
CCP enters Execution (Figure 1-7) at the completion of Startup. During execution CCP monitors the user's terminal network to accept commands and program requests from those terminals authorized to issue them. CCP loads user programs as required and provides an interface for those programs to the terminals with which they communicate. During execution CCP schedules all communications operations and manages contention for the use of communications lines.

CCP Execution is primarily controlled by the resident control routine, which was tailored at the source program level to the user's specifications during CCP generation. In order to respond to commands, to schedule, initiate, terminate programs, and to react to exceptional conditions in the communications network, however, transient routines are used in addition to the resident control routine.

During CCP execution, any user program which is running under CCP co-resides in the program-level partition with the resident control routine and its transients. Main storage, both for user programs and communications buffer space, is managed by CCP. If the user chose the capability at generation, multiple user programs may be concurrently loaded and executing; that is, CCP provides for multitasking.

When a user program is requested, all resources required by that program — main storage, files, terminals, and unit record devices — are allocated by CCP to the user task before that task is initiated. Similarly, upon termination of a user task, CCP de-allocates the resources of that task.

Access to disk data files is controlled by CCP during CCP execution. To provide for satisfactory program initiation and termination performance, CCP opens and closes the DTFs in user programs from information previously obtained from the open of the disk file at Startup. In a multitasking system, the user may have elected at generation to support the sharing, across several tasks, of files being updated; if so, CCP controls the concurrent access to such files.



ure 1-7. CCP Execution

During CCP execution the system operator has ultimate control over the actions of the system and has the ability to display on the system console typewriter the status of a number of elements of the system. In addition, the system operator has control over the triggering of several service aid traces.

CCP execution is terminated by request from the system operator that the system shutdown. After all user programs currently running and scheduled go to completion, CCP Shutdown begins.

Operational Shutdown

The function of Shutdown (Figure 1-8) is to systematically terminate the execution of CCP and return control to Disk System Management. To accomplish this, Shutdown:

- Terminates all communications operations.
- Closes communications lines.
- Updates program request counts in $CCPFILE if that option was in effect.
- Closes disk data files opened at Startup.
- Goes to end-of-job.

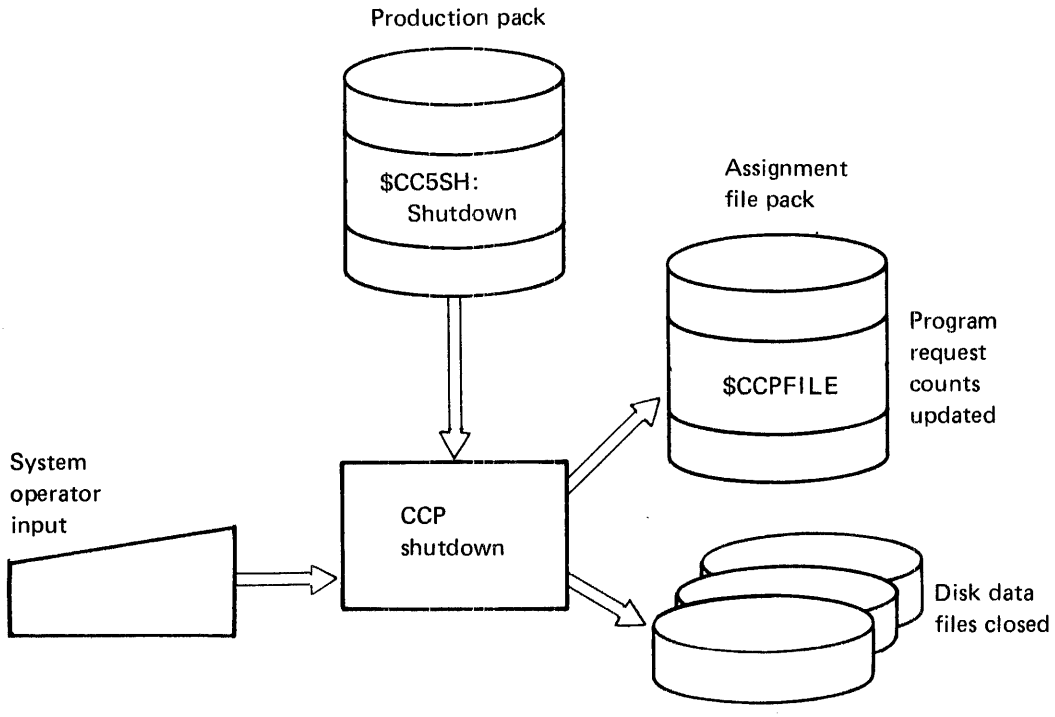


Figure 1-8. Operational Shutdown

AUXILIARY PROGRAMS

Several auxiliary programs are provided with the CCP feature as service aids.

Disk-to-Printer Dump of Main Storage Trace

Space in $CCPFILE is set aside during the execution of CCP to accommodate dumps of main storage and the recording on disk of the entries made by a CCP trace.

If a user task terminates abnormally, CCP writes all of main storage to disk if space is available. If the CCP trace facility was included in the current CCP run and the system operator requested the recording on disk of all trace entries, the trace entries are also recorded in $CCPFILE.

The Disk-to-Printer Dump Program ($CCPDD) can be used to dump to the system print device any or all of the main storage dumps taken during the preceding CCP execution as well as any trace entries from that execution. The Disk-to-Printer Dump Program (Figure 1-9) runs directly under control of Disk System Management.

Standalone Main Storage Dump Programs (Models 10 and 12)

If CCP itself terminates abnormally during execution, it is possible that the DSM CEFE main storage dump might not be operable. Therefore, four MFCU-loadable main storage dump programs are provided – one for each possible print chain image:

1. AN or LC (standard 48 character)
2. HN (COBOL/FORTRAN 48 character)
3. PN (60 character)
4. TN (120 character)

Installation Verification Program

As an aid in determining if a generated CCP is truly operable with the user's configuration, an installation verification program (CCPIVP) is included as part of the CCP feature. Immediately after generation and an Assignment Build run, CCP may be started up and this program run as a verification of the generation, even if the user has no user programs prepared to test the generation of the system.

Listing of $CCPLOG

The Log List Program ($CCPLL) can be used to list the contents of $CCPLOG on the printer.

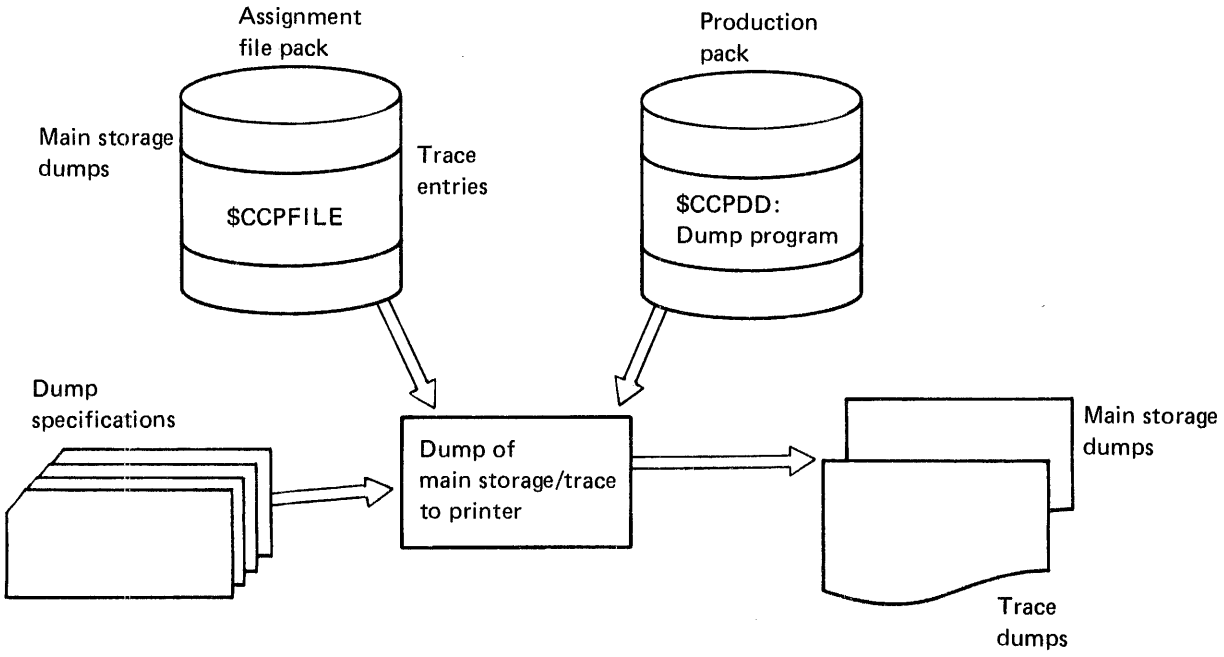


Figure 1-9. Disk-to-Printer Dump of Storage/Trace

EOJ	End of job.
EOT	End of transmission.
ERP	Error recovery procedure.
ESL	External symbol list.
FCT	File control table.
FDT	Field descriptor table.
FSB	File specification block.
FT	Format table.
HDB	High density buffer.
HPL	Halt program level.
IAR	Instruction address register.
ICA	Integrated communications adapter.
ID	Identification.
IFT	Input format table.
I/O	Input/output.
IOB	Input/output block.
IOCS	Input/output control system.
IOS	Input/output supervisor.
IPL	Initial program load.
ITB	Intermediate text block.
LCA	Local communications adapter.
LCB	Line control block.
LCT	Line control table.
Len	Length.
MDT	Modified data tag.
MFCU	Multi-function card unit.
MLMP	Multiline/multipoint
MLTA	Multiple line terminal adapter.
MPX	Macro processor.
MRT	Multiple requesting terminal.
NEP	Never-ending program.
OCL	Operation control language.
OHA	Output hold area.
OLE	Overlay linkage editor.
OLT	Online test.
parm	Parameter.
PAS	Program appended storage.
PCT	Program control table *or* Program characteristics table.
PFGR	Printer format generator routine.
PID	Program information department.
PLCA	Program level communication area.
PRUF	Program request under format.
PSR	Program status register.
PTAM	Pseudo tape access method.
PTTC/EBCD	Perforated tape and transmission code/extended binary coded decimal.
RAT	Relocation adcon table.
reorg	Reorganization.
REQ	Request.
RIB	Request indicator byte.
RLD	Relocation list dictionary.
R4	R4 hold queue; associated with $CC4R4.
SBA	Set buffer address.
SCA	System communication area.
SCB	Storage control block.
SCP	System control program.
SDF	Short disk.

SDR	Statistical data recording.
SIO	Start input/output.
SIT	System information table.
SQB	Sector queue block.
stmt	Statement.
STT	Switched terminal block.
SWA	Scheduler work area.
SYSBFR	Control statement input buffer.
SYSCOM	System communication region.
TAS	Terminal attribute set.
TAT	Terminal attributes table.
TCB	Task control block.
TM	Termination manager.
TNT	Terminal name table.
TP	Teleprocessing.
TT	Terminal table.
TUB	Terminal unit block.
TUT	Terminal used table.
UPA	User program area.
UPSI	User program switch indicator.
VTOC	Volume table of contents.
WCC	Writer control character.
XCTL	Transfer control.
XDT	Symbolic file table.
XREF	Cross-reference.

Chapter 2. CCP Generation (Models 8, 10, and 12 Only) And Installation (Model 4 Only)

Introduction to CCP Generation (Models 8, 10, and 12 Only)

Function

The function of the Generation Stage of CCP is to:

- Generate (using the SCP Generator – see Chapter 3) modules of the execution-time resident control routine that require tailoring at the source level.
- Link edit the generated modules and certain other relocatable modules to form the execution-time resident control routine.
- Copy this module and other load modules to the designated CCP production pack.
- Allocate and initialize the CCP Assignment File ($CCPFILE) on the designated assignment file pack.
- Copy selected relocatable modules (subroutines for user programs) to one or more designated program preparation packs.

CCP Generation is not supported on Model 4.

Procedure

There are two methods for doing the CCP generation. The first method uses cards as the primary input medium. The second method, a cardless-oriented procedure, uses the source and procedure libraries as the primary input medium.

The basic procedure for card-oriented CCP Generation is:

1. A sample control statement deck is punched from the source library of the distribution pack.
2. The sample deck is modified by the user to fit specifications and reentered as input to the next step of Generation.
3. A full job stream to accomplish the necessary functions is punched out for the user.
4. The job stream is entered. At conclusion of the job, CCP is generated and the CCP Assignment File ($CCPFILE) is ready for the user's initial Assignment run. CCP generation may be accomplished on a Model 10 Disk System whether the CCP generated is to be used on that particular system or not.

The basic procedure for the cardless-oriented CCP generation is:

1. Source and procedure members are printed from the distribution pack.
2. The user modifies the sample procedures to the user's system configuration. This is input to the next step of generation.
3. The user enters the CCP specifications to create source and procedure members which are used to generate the user's version of the CCP. The CCP assignment file ($CCPFILE) is now ready for the user's initial assignment run.

For more information, refer to *Method of Operation* later in this chapter.

Note: The printer paper resulting from generation should be saved in case of required maintenance by IBM Field Engineering. This paper is the only documentation of the user's unique system and the precise sequence of events during this particular CCP generation. The user should also consider back-up procedures in case the CCP distribution pack or the CCP production packs are inadvertently destroyed.

Restrictions

The following restrictions apply to CCP Generation:

- System/3 Disk System Management, including the appropriate BSCA and/or MLTA IOCS, must have already been generated.
- An appropriately sized object library and source library must have been allocated on the designated CCP production pack (5444 for Model 10, or the simulation area on the 3348 for Model 12). The libraries may already contain non-CCP modules. The production pack may be any pack other than the distribution pack. It may or may not be a DSM system pack, and might even be the system pack used during CCP generation.
- The object library on the production pack must have been reorganized since the last deletion or replacement of a module in that library. In order to ensure optimum execution-time performance, it is important that all CCP load modules copied to the object library of the production pack be contiguous.
- The assignment file pack must contain sufficient space for the allocation of $CCPFILE.
- A program preparation pack to be used for the compilation of RPG II programs must already contain, in its object library, the DSM data management routines (relocatable modules) for any unit record devices to be supported by CCP.
- Sufficient work file space must be available, on 5444 packs online during generation, for SCP Generator and Overlay Linkage Editor runs.

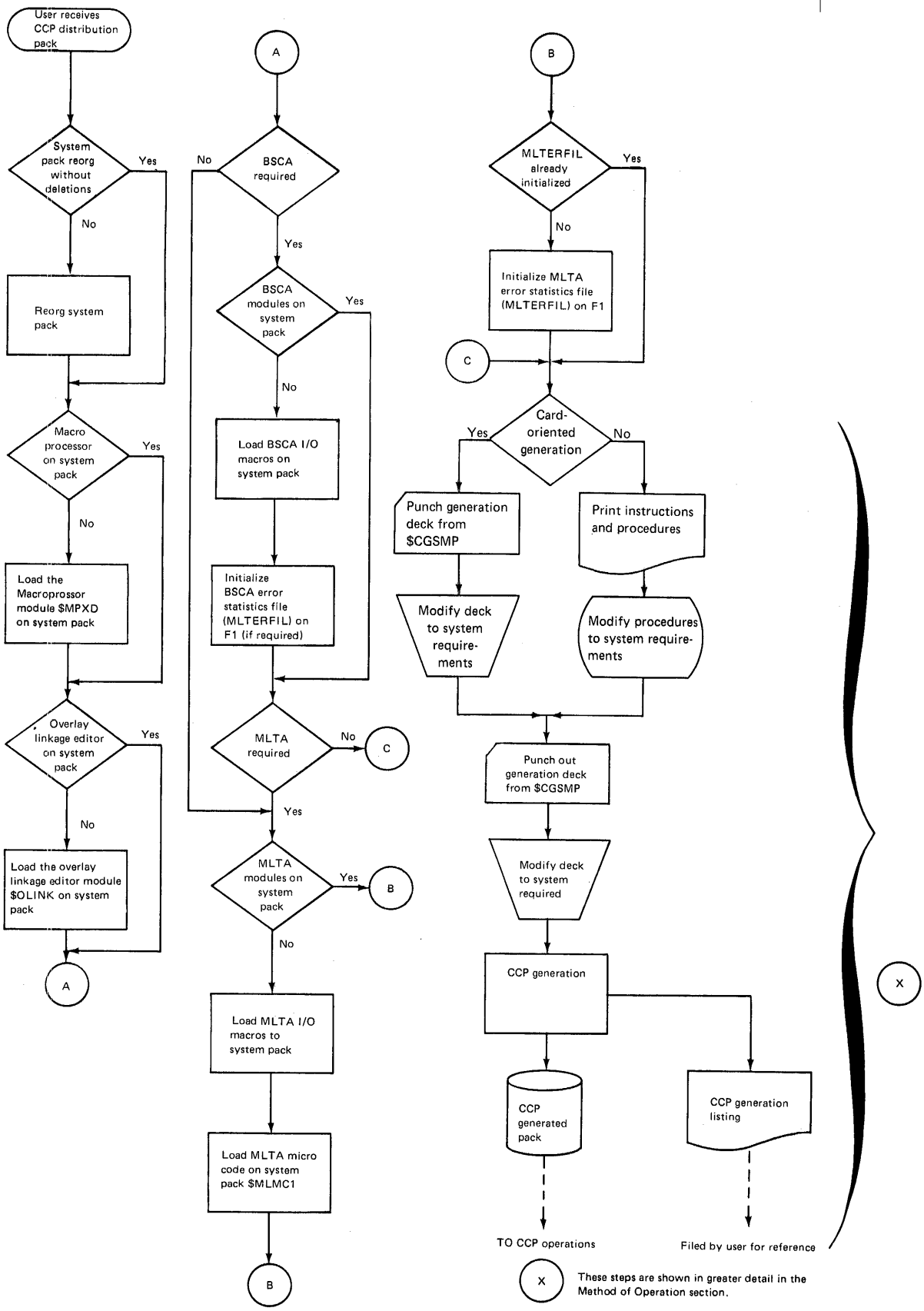


Figure 2-1. General Procedures for a CCP Generation (Models 8, 10, and 12 Only)

Method of Operation (Card-Oriented Generation)

Card-oriented CCP Generation requires two passes through the system. The first pass consists of preparatory operations:

- Copying to the distribution pack certain DSM modules which must be located there during the generation process (Models 4, 8, and 10 only)
- Evaluation of the user's Generation control statements
- Creation of input to the second pass of Generation

As an aid to the user in preparing his input to the first pass of Generation, a sample input deck is provided in the source library of the distribution pack. During the second pass of Generation, the output from the first pass is processed to form a usable CCP tailored to the user's specifications.

STEPS IN CARD-ORIENTED CCP GENERATION

There are six required steps in generating CCP. (Figure 2-2 shows several of the six steps and Figure 2-3 is an overview of all the steps.) A seventh step is recommended: testing the generated CCP using the Installation Verification Program CCPIVP. CCPIVP is a distributed program that uses the generated CCP system to make sure it has been correctly generated.

Step 1 (Card Oriented)

The user enters the following statements from the system input device:

```
// LOAD $MAINT,dsunit

// RUN

// COPY FROM-diunit,TO-PRTPCH,LIBRARY-S,
NAME-$CGSMP

// END
```

dsunit – The unit on which the DSM system pack is mounted.

diunit – The unit on which the CCP distribution pack is mounted.

Step 2 (Card Oriented)

The system prints and punches from the source library the sample first pass input deck ($CGSMP). (Part 1 of Figure 2-2.) This deck consists of three parts (for Model 12) or four parts (for Models 4, 8, and 10):

1. Instructions to the user for modifying the punched deck.
2. OCL and COPY control statements to copy the Macro Processor ($MPX.ALL) and parts of the Overlay Linkage Editor ($OLYNX and $OLBO) from the DSM system pack to the distribution pack (Models 4, 8, and 10 only).
3. OCL to call the Macro Processor followed by sample $E Generation control statements to generate a CCP system.
4. OCL to call $CC1PP (CCP Generation Print/Punch Utility).

Step 3 (Card Oriented)

The user modifies the sample deck to reflect the requirements of his CCP system. This step can be performed either by modifying the punched deck or by keying the required OCL and Generation control statements on the console typewriter.

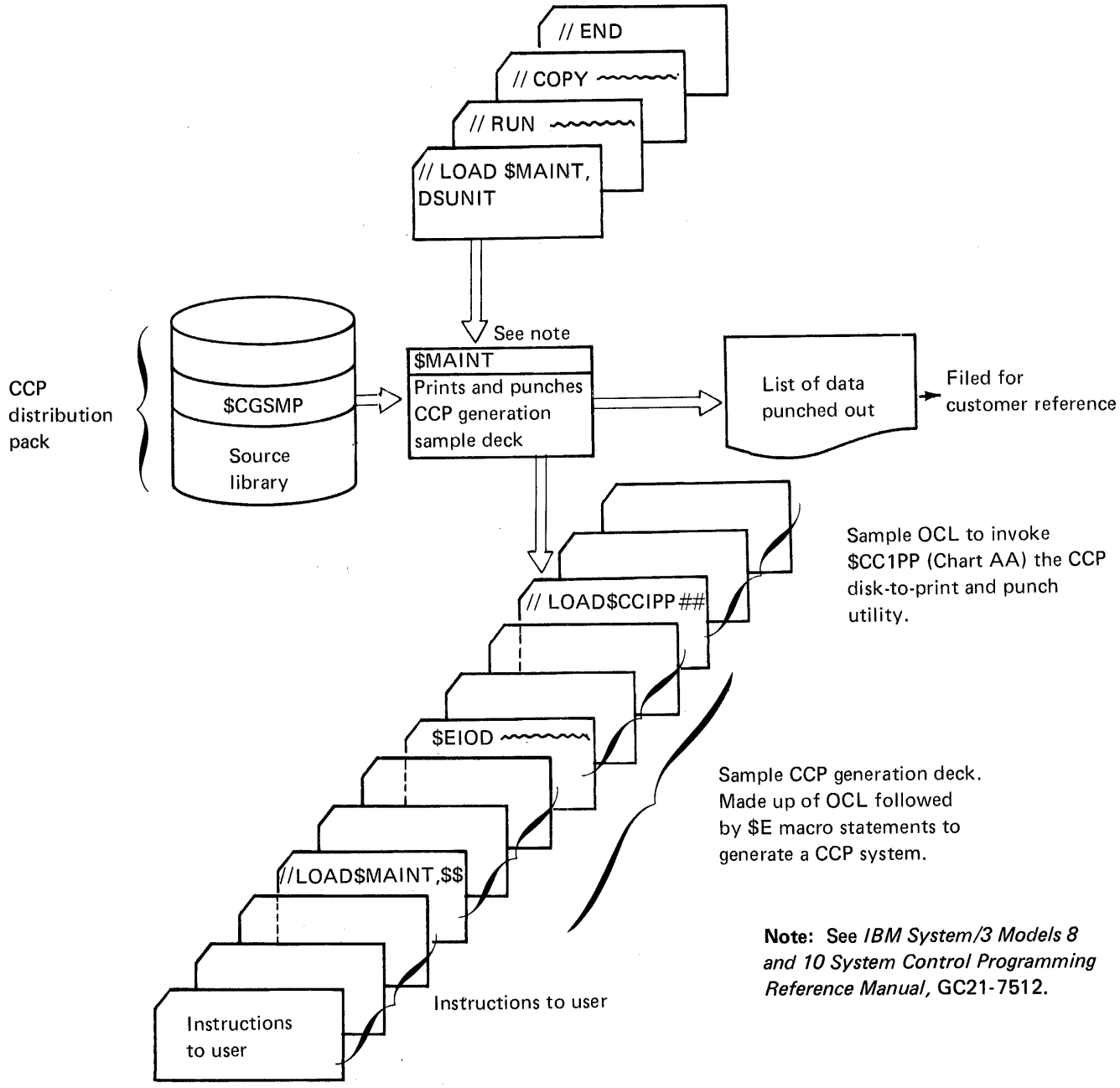
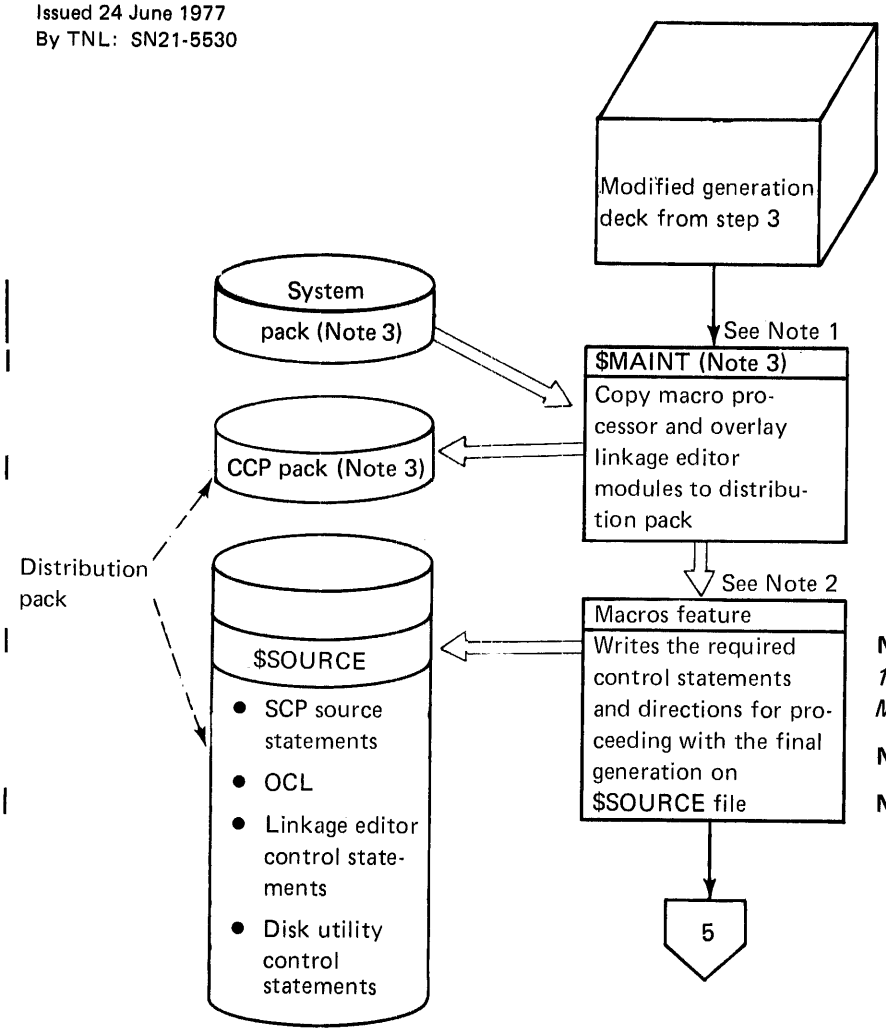


Figure 2-2 (Part 1 of 4). Card-Oriented CCP Generation Step 2

Step 4 (Card Oriented)

This step (Figure 2-2, Part 2) consists of the following operations:

1. The Library Maintenance Program ($MAINT) copies from the system pack the Macro Processor ($MPX.ALL) and parts of the Overlay Linkage Editor ($OLYNX and $OLBO) to the CCP distribution pack (Models 4, 8, and 10 only).
2. The system invokes the Macro Processor to expand the CCP Generation control statements ($E macro statements). All statements are diagnosed for errors and global variables are set to record options specified by the user. During processing of the last statement ($EGEN), records which represent input to the second pass of Generation are written to the work file $SOURCE if no errors were encountered in user specifications. The content of these records will vary according to the values of the global variables set by user specifications.



Note 1: See *IBM System/3 Models 8 and 10 System Control Programming Reference Manual*, GC21-7512.

Note 2: See *Chapter 3. SCP Generator.*

Note 3: Used for Models 4, 8, and 10 only.

Figure 2-2 (Part 2 of 4). Card-Oriented CCP Generation Step 4

Three types of errors can occur during step 4:

1. A syntax error such as a misspelled keyword, invalid operand format, invalid continuation card, etc.

2. An error in statement sequence or the omission of a required statement.

3. Specification of a valid keyword with an invalid value or the omission of a required keyword.

The first type of error is noted by the Macro Processor. For each error detected a Macro Processor error message is written into $SOURCE after the macro statement containing the error. The Macro Processor immediately terminates processing of that generation statement without any further checking.

The last two error types are noted by diagnostics internal to the macro definitions themselves. An error switch is set on, and an error record is written into $SOURCE for each macro statement containing an error.

If an error of any type is encountered, no input to the second pass of Generation is created and the diagnostic messages are printed during step 5 by the program $CC1PP.

If no errors have been detected, the following records are generated into $SOURCE during the processing of the $EGEN statement:

1. The records required to generate the module which initializes $CCPFILE:

   - OCL to invoke the SCP Generator to read card input and generate relocatable module $CC1FC.
   - Statements for generating the initialization data for $CCPFILE as a relocatable module.
   - OCL to invoke the Overlay Linkage Editor.
   - Overlay Linkage Editor statements for link editing the provided relocatable module $CC1BP. and the new relocatable data module $CC1FC together to form the load module $CC1BF.

2. The records required to generate the CCP execution-time resident module:

   - OCL to invoke the Macro Processor to read card input and write output to $SOURCE.
   - Source statements and macro statements creating the first source-generatable portion of the execution-time resident module.
   - OCL to invoke the SCP Generator to read input statements from $SOURCE and generate a relocatable module ($CC4#1) which will be the first part of the execution-time resident load module $CC4.
   - OCL to invoke the Macro Processor to read card input and write output to $SOURCE.
   - Source statements and macro statements creating the second source-generatable portion of the execution-time resident module.
   - OCL to invoke the SCP Generator to read input statements from $SOURCE and generate a relocatable module ($CC4#2) which will be the second part of the execution-time resident load module $CC4.
   - OCL to invoke the Overlay Linkage Editor.
   - Overlay Linkage Editor statements to link edit $CC4#1, $CC4#2, and other distributed relocatable modules to form the CCP execution-time resident module $CC4.

   If remap is requested (FORMAT-PL1), the following records are required to create a second execution-time resident module:

   - OCL to invoke the Macro Processor to read card input and write output to $SOURCE.
   - Source statements and macro statements creating source-generatable execution-time resident module ($CC4#3).
   - OCL to invoke the SCP Generator to read input statements from $SOURCE and generate a relocatable module ($CC4#3).
   - OCL to invoke the Overlay Linkage Editor.
   - Overlay Linkage Editor statements to link edit $CC4#3 to form the CCP execution-time resident module ($CC4#3).

3. The records required to generate the necessary IOCS loadable trace module(s). One trace module is created for BSCA, if supported, and one for MLTA, if supported. The following are generated for each module:

   - OCL to invoke the Overlay Linkage Editor.
   - Overlay Linkage Editor statements to link edit the loadable trace module.

4. The records required to copy Assignment Stage modules, Startup modules, CCP execution-time transient modules, the Shutdown module, and service aid programs to the source and object libraries of the production pack:

   - OCL to run $MAINT.
   - COPY statements to copy the above load modules from the CCP distribution pack to the production pack.
   - A // END statement.

5. Optionally, the records required to build (but not enter information into) the initial data load module for the user terminal sign on security information:

   - OCL to invoke the SCP Generator to read card input and generate relocatable module $CC4Z9.
   - Statements for generating a module of the required size, containing all hex zeros.
   - OCL to invoke the Overlay Linkage Editor.
   - Link edit statements to link edit the user terminal sign on security data into a load module $CC4Z9.

6. OCL to invoke load module $CC1BF to initialize $CCPFILE.

7. The records required to copy to each program preparation pack the Communications Service Subroutines (macros for Basic Assembler Language), and to copy (and rename if necessary) the unit record intermediary data management subroutines:

   - OCL to run $MAINT.
   - COPY statements to copy the above modules to the user's program preparation pack(s) and for RPG to rename the DSM unit record data management modules.
   - A // END statement.

8. The records required to punch out modules used to service and verify the generated system:

   - The card loadable storage dump programs.
   - The OCL linkage edit statements and a sample assignment set for running the Installation Verification Program (CCPIVP).

This page intentionally left blank.

Step 5 (Card Oriented)

The OCL for running $CC1PP (the CCP Generation Utility) modified during step 2, is now run (Figure 2-2, Part 3).

$CC1PP prints the user's input to the previous step and any diagnostic messages generated during that step.

If any errors were detected, an additional message is issued directing the user to correct the errors and repeat step 4.

If no errors were detected, $CC1PP performs its second function. The user determines this function by responding to the keyword CARD on the $EGEN macro control statement. In a card-oriented generation, the response is CARD-YES. When CARD-YES is specified, the input to the second pass of generation (step 6) is printed and punched. Also, a message is issued directing the user to proceed to the second pass of generation.

If CARD-NO is specified, refer to the cardless generation step 5 for a description of the second function of $CC1PP.

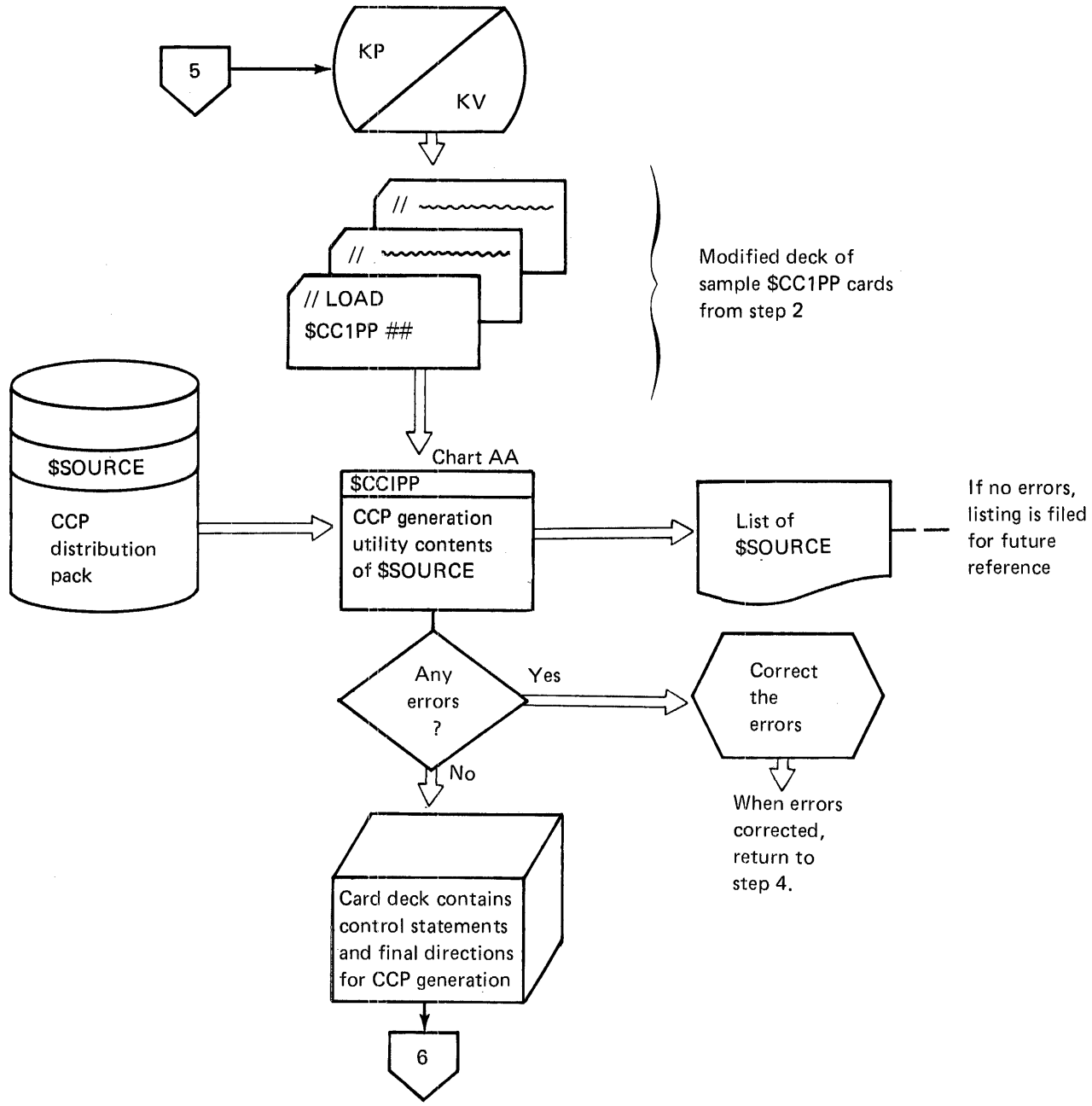


Figure 2-2 (Part 3 of 4). Card-Oriented CCP Generation Step 5

Step 6 (Card Oriented)

The user places the punched output from the previous step without modification into the system input device (Figure 2-2, Part 4). This produces the following:

1. Creation of a load module which contains the initial contents of $CCPFILE and the instruction code to initialize that file ($CC1BF).
2. Source generation and linkage edit of the CCP execution-time resident module ($CC4).
3. Linkage edit of the execution-time loadable IOCS trace modules.
4. A copy of the load modules for the Assignment and Operational Stages and of service aid modules.
5. Initialization of module $CC4Z9 if SECURE-USER was specified in the $ESEC statement.
6. Initialization of $CCPFILE.
7. A copy of subroutines (and macros) to be used in compiling and link editing application programs which are to run under CCP.
8. The following punched output cards:
   a. Stand alone storage dumps.
   b. OCL and Overlay Linkage Editor control statements that will link edit the Installation Verification Program (CCPIVP).
   c. OCL and control statements for an Assignment Build run.

On successful completion of step 6 the user has a CCP pack that is ready for an Assignment Build run.

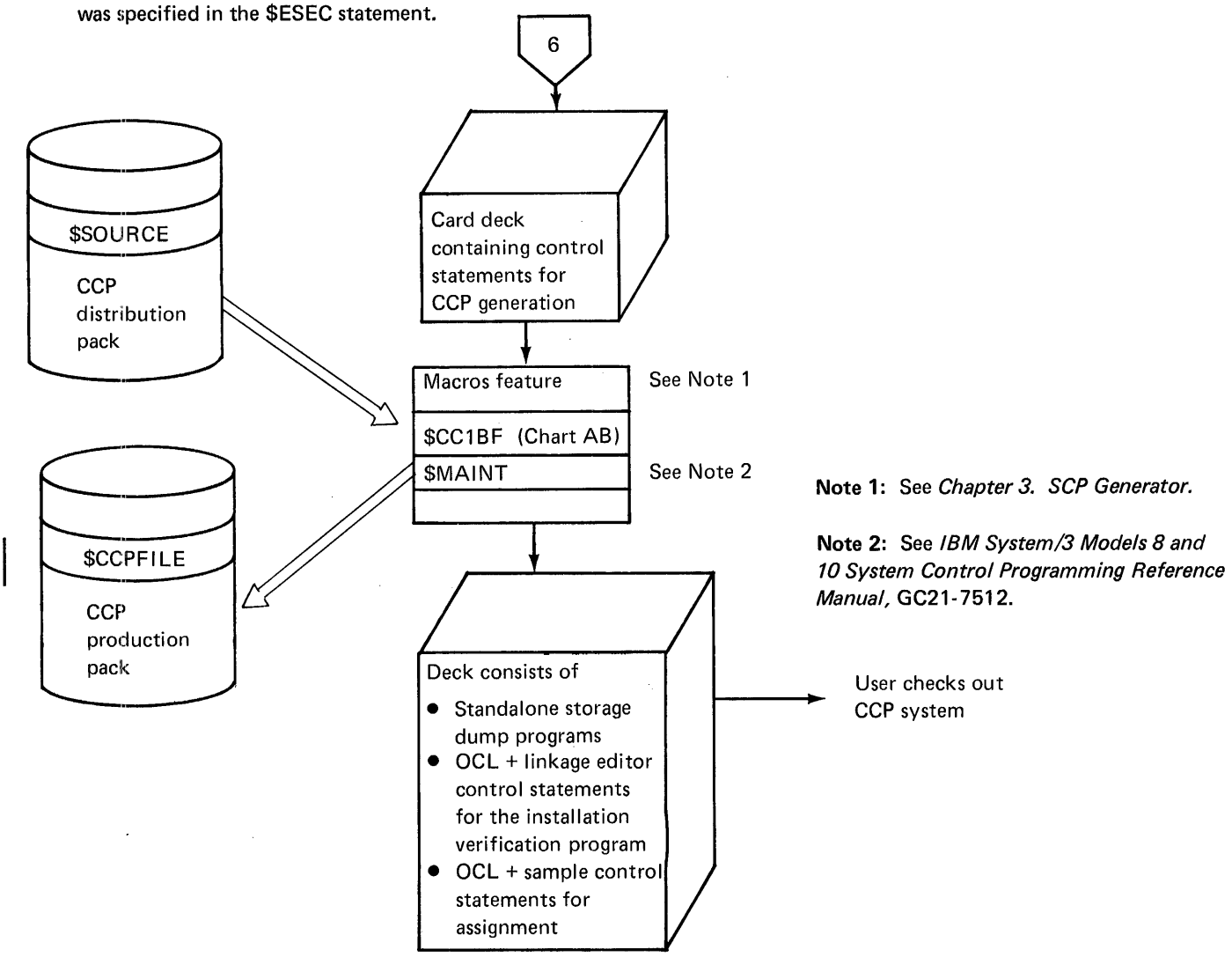


Figure 2-2 (Part 4 of 4). Card-Oriented CCP Generation Step 6

Method of Operation (Cardless-Oriented Generation)

Cardless CCP Generation requires only one pass through the system. To aid the user in preparing input for this generation pass, two source members and five procedures (for Models 4, 8, and 10) or four procedures (for Model 12) are provided in the source library of the distribution pack. Source members provided are:

$CG1GI	Cardless CCP generation instructions.
$CG1GM	Sample $E macro control statements.

Procedures provided are:

$CG1G1	Controlling procedure for CCP cardless generation.
$CG1G2	Prepares the CCP distribution pack (Models 4, 8, and 10 only).
$CG1G3	Processes the user's CCP generation specifications.
$CG1G4	Prints results of user's specifications and prepares the $SOURCE file for procedure $CG1G5.
$CG1G5	Using the contents of the $SOURCE file, creates the source and procedure members that will complete the CCP generation.

STEPS IN CARDLESS-ORIENTED CCP GENERATION

Six steps are required in cardless-oriented CCP Generation. (Figures 2-2.1 - 2-2.3).

Step 1 (Cardless Oriented)

The user enters the following statements from the system input device:

```
// LOAD $MAINT,dsunit
// RUN
// COPY FROM-diunit,TO-PRINT,LIBRARY-S,
   NAME-$CG1.ALL
// COPY FROM-diunit,TO-PRINT,LIBRARY-P,
   NAME-$CG1.ALL
// END
```

dsunit — The unit on which the DSM system pack is mounted.

diunit — The unit on which the CCP distribution pack is mounted.

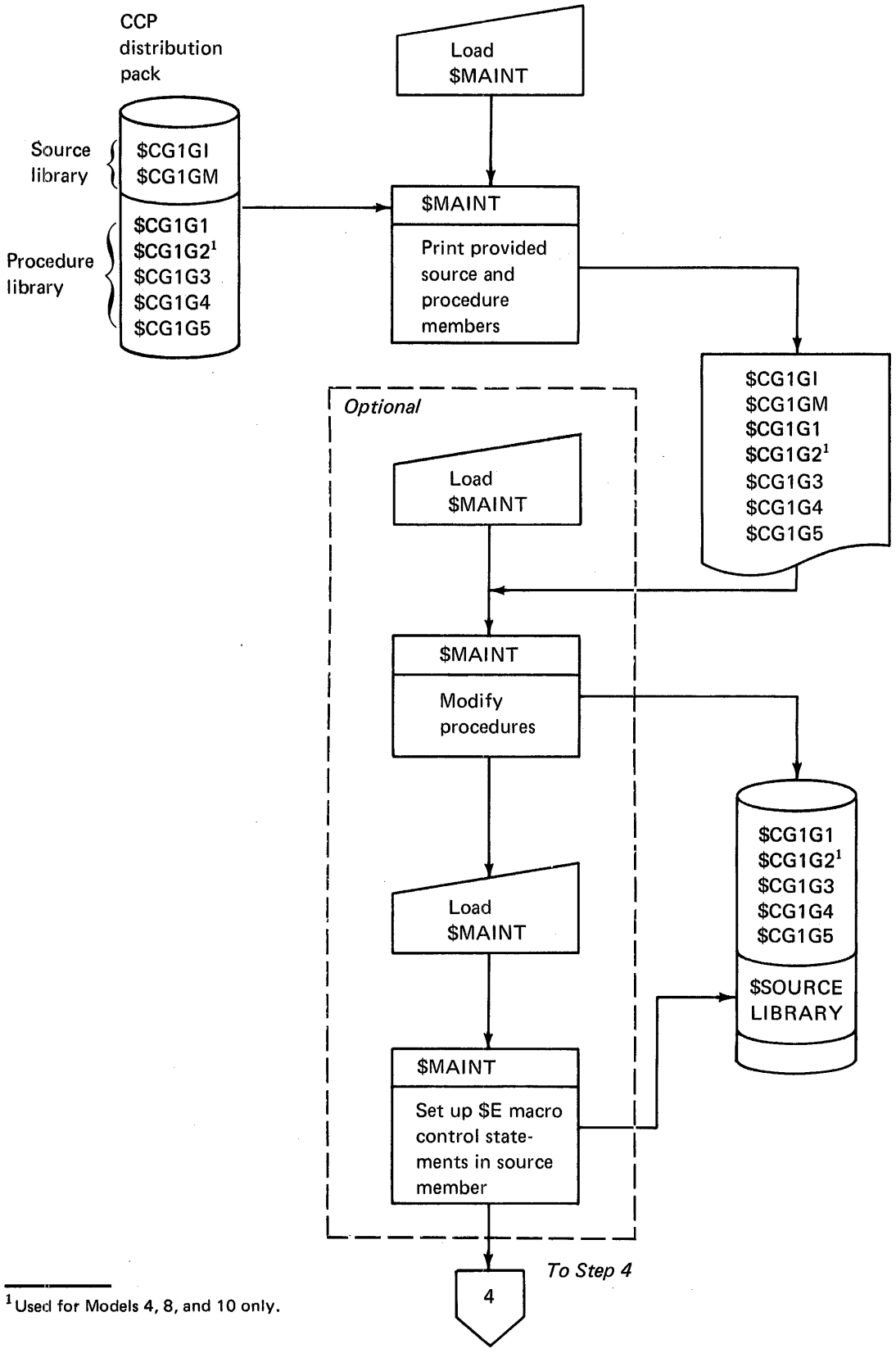
Step 2 (Cardless Oriented)

The system prints, from the source library, the source and procedure members provided. The listing has three parts:

1. Instructions to the user.
2. Sample $E macro control statements.
3. The procedures that control the CCP generation.

Step 3 (Cardless Oriented)

Optionally, following the printed instructions, the user modifies the five provided procedures to reflect the requirements of the system. The user modifies the sample source member, $CG1GM, or creates a new source member to reflect the CCP system requirements.



[1] Used for Models 4, 8, and 10 only.

Figure 2-2.1. Cardless-Oriented CCP Generation Steps 1-3

Step 4 (Cardless Oriented)

This step consists of the following operations:

1. The user calls the procedure $CG1G1 from the CCP distribution pack. This procedure controls all of step 4 and step 5 of the cardless CCP generation.

2. For Models 4, 8, and 10 only, the procedure $CG1G1 calls procedure $CC1G2. This procedure copies, from the system pack, the macro processor and parts of the overlay linkage editor.

3. The procedure $CG1G1 calls procedure $CC1G3. This procedure invokes the macro processor which expands the CCP generation control statements ($E macro statements). The generation control statements are read from the described source member, or the macro processor requests the generation control statements from the system input device.

   All statements are checked for errors, and global variables are set to record options specified by the user. If no errors are found in the user specifications and the keyword CARD-NO has been specified in the $EGEN statement, records which represent input to step 6 are written to the workfile, $SOURCE, during processing of the last statement ($EGEN). The contents of these records vary according to the values of the global variables set by user specifications.

Three types of errors can occur during step 4:

1. A syntax error such as a misspelled keyword, invalid operand format, invalid continuation card, etc.

2. An error in statement sequence or the omission of a required statement.

3. Specification of a valid keyword with an invalid value or the omission of a required keyword.

The first type of error is noted by the Macro Processor. For each error detected a Macro Processor error message is written into $SOURCE after the macro statement containing the error. The Macro Processor immediately terminates processing of that generation statement without any further checking.

The last two error types are noted by diagnostics internal to the macro definitions themselves. An error switch is set on, and an error record is written into $SOURCE for each macro statement containing an error.

If an error of any type is encountered, no input to step 6 is created and the diagnostic messages are printed during step 5 by the program $CC1PP.

If no errors have been detected, the following records are generated into $SOURCE during the processing of the $EGEN statement:

1. The records required to generate the module which initializes $CCPFILE:

   - OCL to invoke the SCP Generator to generate relocatable module to $CC1FC.
   - Statements for generating the initialization data for $CCPFILE as a relocatable module.
   - OCL to invoke the Overlay Linkage Editor.
   - Overlay Linkage Editor statements for link editing the provided relocatable module $CC1BP and the new relocatable data module $CC1FC together to form the load module $CC1BF.

2. The records required to generate the CCP execution-time resident module:

   - OCL to invoke the Macro Processor and write output to $SOURCE.
   - Source statements and macro statements creating the first source-generated portion of the execution-time resident module.
   - OCL to invoke the SCP Generator to generate a relocatable module ($CC4#1), which will be the first part of the execution-time resident load module $CC4.
   - OCL to invoke the Macro Processor to write output to $SOURCE.
   - Source statements and macro statements creating the second source-generatable portion of the execution-time resident module.
   - OCL to invoke the SCP Generator to read input statements from $SOURCE and generate a relocatable module ($CC4#2) which will be the second part of the execution-time resident load module $CC4.
   - OCL to invoke the Overlay Linkage Editor.
   - Overlay Linkage Editor statements to link edit $CC4#1, $CC4#2, and other distributed relocatable modules to form the CCP execution-time resident module $CC4.

If remap is requested (FORMAT-PL1), the following records are required to create a second execution-time resident module:

- OCL to invoke the Macro Processor to read card input and write output to $SOURCE.
- Source statements and macro statements creating source-generatable execution-time resident module ($CC4#3).
- OCL to invoke the SCP Generator to read input statements from $SOURCE and generate a relocatable module ($CC4#3).
- OCL to invoke the Overlay Linkage Editor.
- Overlay Linkage Editor statements to link edit $CC4#3 to form the CCP execution-time module ($CC4#3).

3. The records required to generate the necessary IOCS loadable trace module(s). One trace module is created for BSCA, if supported, and one for MLTA, if supported. The following are generated for each module:
   - OCL to invoke the Overlay Linkage Editor.
   - Overlay Linkage Editor statements to link edit the loadable trace module.
4. The records required to copy Assignment Stage modules, Startup modules, CCP execution-time transient modules, the Shutdown module, and service aid programs to the source and object libraries of the production pack:
   - OCL to run $MAINT.
   - COPY statements to copy the above load modules from the CCP distribution pack to the production pack.
   - A // END statement.
5. Optionally, the records required to build (but not enter information into) the initial data load module for the user terminal sign on security information:
   - OCL to invoke the SCP Generator to generate relocatable module $CC4Z9.
   - Statements for generating a module of the required size, containing all hex zeros.
   - OCL to invoke the Overlay Linkage Editor.
   - Link edit statements to link edit the user terminal sign on security data into a load module $CC4Z9.
6. OCL to invoke load module $CC1BF to initialize $CCPFILE.
7. The records required to copy to each program preparation pack the Communications Service Subroutines (macros for Basic Assembler Language), and to copy (and rename if necessary) the unit record intermediary data management subroutines:
   - OCL to run $MAINT.
   - COPY statements to copy the above modules to the user's program preparation pack(s) and for RPG to rename the DSM unit record data management modules.
   - A // END statement.
8. The records required to print the sample assignment set control statements, and copy them to the CCP production pack.

Step 5 (Cardless Oriented)

This step consists of the following three operations:

1. Procedure $CG1G1 calls procedure $CG1G4 which executes the generation utility $CC1PP. $CC1PP prints the user's input to the previous step and any diagnostic messages generated during that step. If any errors are detected, an additional message is issued directing the user to correct the errors and restart the generation.

2. If no errors are detected, $CC1PP performs its second function. The user determines this function by responding to the keyword CARD on the $EGEN macro control statement. In a cardless generation that response is CARD-NO. If CARD-NO is specified, the input to step 6 is printed and the $SOURCE workfile is reorganized to eliminate the user's specification statements.

   CARD-NO specified on the $EGEN statement causes the $EGEN macro to insert the necessary // COPY and // CEND statements required for the following $MAINT file-to-library run. The first // COPY statement inserted by $EGEN macro is recognized by $CC1PP and causes $CC1PP to reorganize the file. If this // COPY statement is not recognized, $CC1PP assumes a card-oriented generation and punches the step 6 records.

   If the user specified CARD-YES, refer to card-oriented generation step 5 for a description of the second function of $CC1PP.

3. Procedure $CG1G1 calls procedure $CG1G5 which executes $MAINT file-to-library function. $MAINT copies the $SOURCE workfile to the source library on the CCP distribution pack. This, in turn, creates all necessary source and procedure members to complete the CCP generation.

Step 6 (Cardless Oriented)

Procedure $CG1G1 calls the generated procedure, $CCPSA, now in the source library on the CCP distribution pack. $CCPSA controls the remaining portions of the CCP generation which produces the following:

1. Creation of a load module which contains the initial contents of $CCPFILE and the instruction code to initialize that file ($CC1BF).

2. Source generation and linkage edit of the CCP execution-time resident module ($CC4).

3. Linkage edit of the execution-time loadable IOCS trace modules.

4. A copy of the load modules for the Assignment and Operational Stages and of service aid modules.

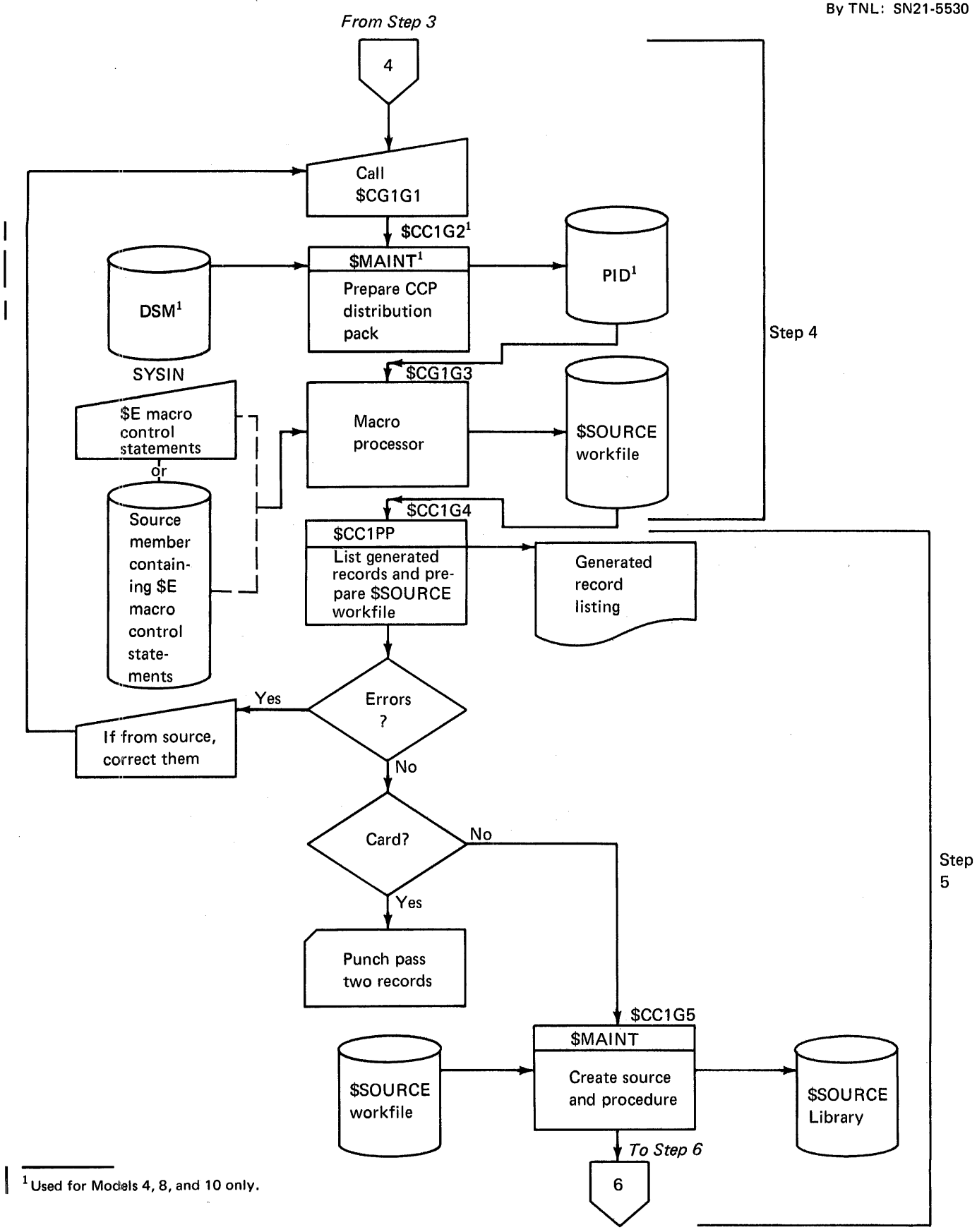
5. Initialization of module $CC4Z9 if SECURE-USER was specified in the $ESEC statement.

6. Initialization of $CCPFILE.

7. A copy of subroutines (and macros) to be used in compiling and link editing application programs which are to run under CCP.

8. Printed output of control statements for an Assignment Build run.

On successful completion of step 6 the user has a CCP pack that is ready for an Assignment Build run.



[1] Used for Models 4, 8, and 10 only.

Figure 2-2.2. Cardless-Oriented CCP Generation Steps 4 and 5

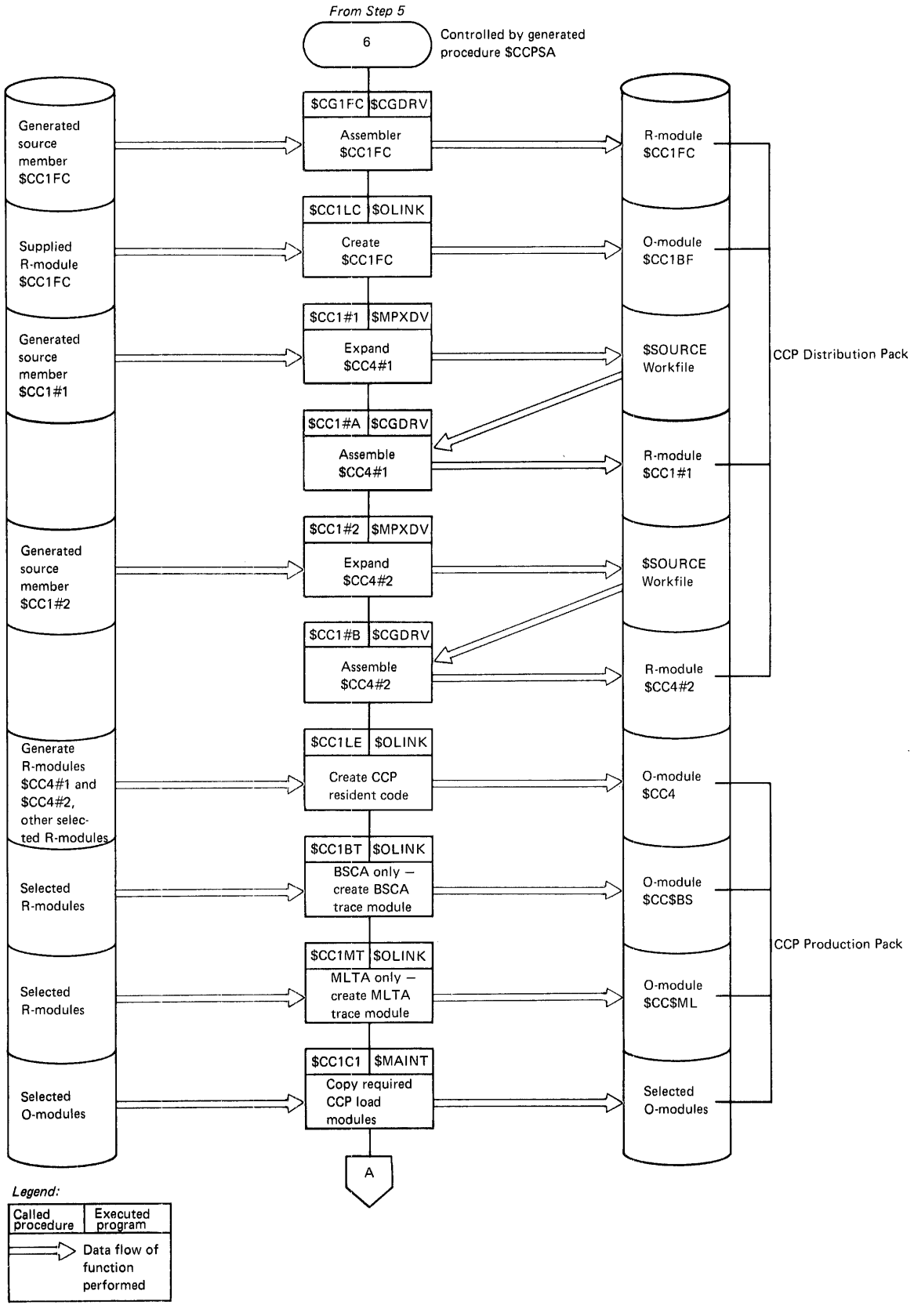


Figure 2-2.3 (Part 1 of 2). Cardless-Oriented CCP Generation Step 6

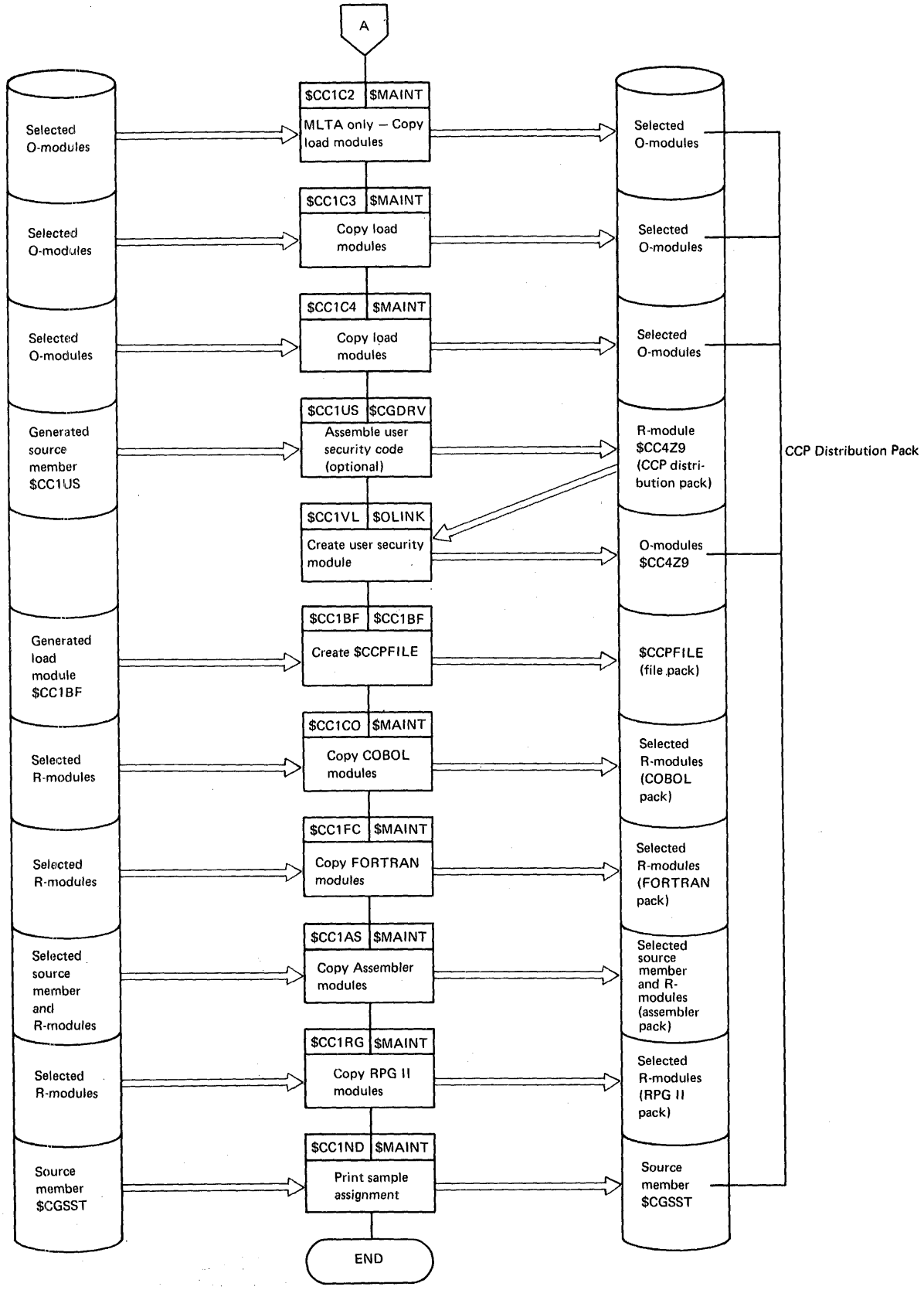


Figure 2-2.3 (Part 2 of 2). Cardless-Oriented CCP Generation Step 6

CCP GENERATION GLOBAL VARIABLES

The following tables define the global variables used during Macro Processor runs in the first and second passes (Figure 2-3) of CCP Generation. The tables contain the following information about each global variable:

- Name. Name of the global variable.
- Type. Data type of the variable, as follows:

  Arith – arithmetic variable.

  Bool – binary (Boolean) variable.

  Charn – character variable. The digit n after char (for example char4) specifies the number of characters in the character variable.

- Stmt. Indicates the Generation control statement which contains the parameter from which the global variable is set.
- Keywd. Indicates the keyword in the Generation control statement whose parameter is used to set the global variable's value.
- Value and Meaning. Indicates the values that the global variable may have and the meaning of each value.

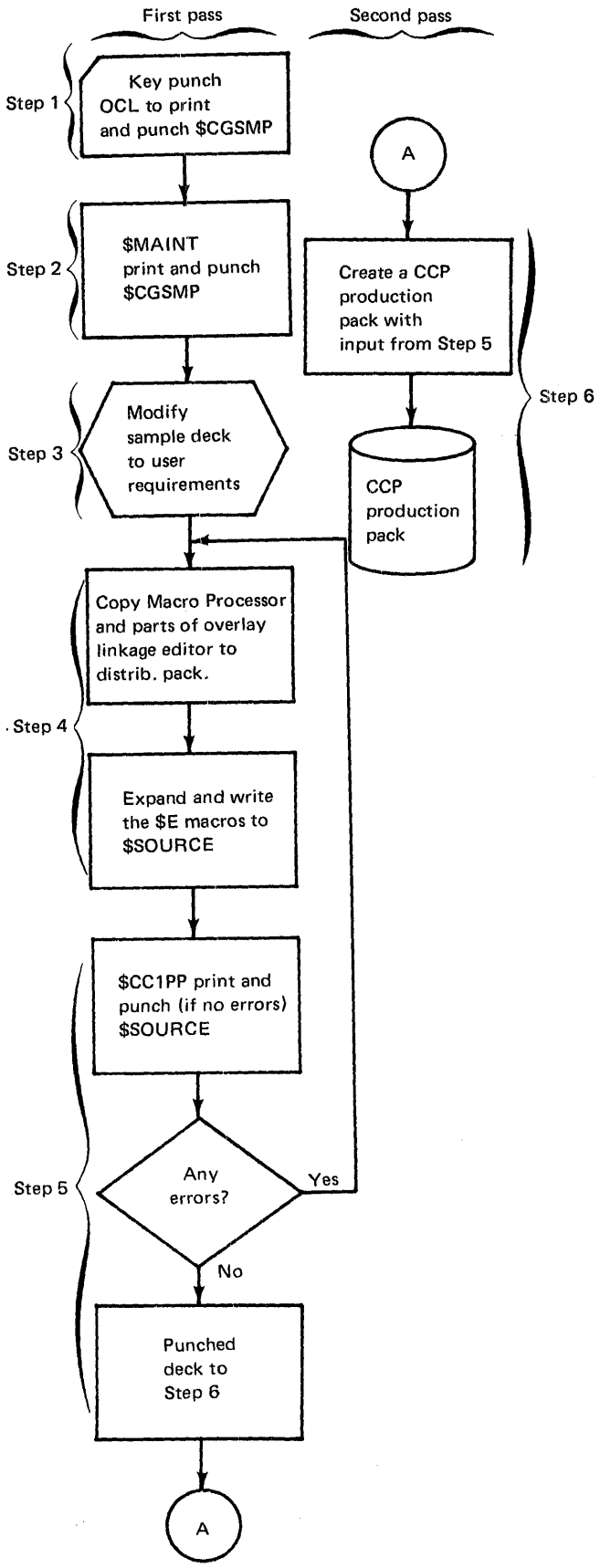


Figure 2-3. Card-Oriented Generation Passes

First Pass

Name	Type	Stmt	Keywd	Value and Meaning
&SEQ	Arith	ALL		Sequence Control. Advances in value from 0 to 10 as each type of Generation control statement is processed.
&TERR	Bool	ALL		Termination error switch.
&URDEV	Char4	$EIOD	CARD PRINTR	Unit record support. *Position* *Value* *Meaning* 1-2 00 CARD–NO (card not supported) 10 CARD–MFCU 01 CARD–1442 11 CARD–'MFCU,1442' 3-4 00 PRINTR–NO (printer not supported) 10 PRINTR–5203 01 PRINTR–1403
&DISK	Char4	$EIOD	DISKS	Disk support. *Position* *Value* *Meaning* 1-2 00 DISKS–NO (only F1 and R1 supported) 10 DISKS–R2 11 DISKS–'R2,F2' 3-4 00 D5445–NO (no 5445 disk support) 10 D5445–D1 11 D5445–'D1, D2'
&N3741	Bool	$EIOD	N3741	1 = N3741–YES (3741 as unit record device). 0 = N3741–NO (3741 not supported).
&NUTSK	Arith	$EFAC	MAXEUP	MAXEUP–#. Number of concurrently executing user programs that can be run under CCP as specified by the user. Minimum = 1, maximum = 8.
&FDPF	Bool	$EFAC	DPF	1 = DPF–YES, DPF support. 0 = DPF–NO, no DPF support.
&FDME	Bool	$EFAC	ESCAPE	1 = String specified in the Data Mode Escape keyword ESCAPE–value. 0 = ESCAPE–NO, no terminal can interrupt a program and talk directly to CCP.
&X1DME &X2DME	Char	$EFAC	ESCAPE	The Data Mode Escape string (except for the closing apostrophe) as specified (character or hex): CL6' cccccc XL6' xxxxxxxxxxxx
&FPGC	Bool	$EFAC	PGMCNT	1 = PGMCNT–YES, a count to be kept of the number of times a user program was requested. 0 = PGMCNT–NO, no count to be kept.
&FSHR	Bool	$EFAC	FSHARE	1 = FSHARE–YES, shared file update supported. 0 = FSHARE–NO, shared file update not supported.

Name	Type	Stmt	Keywd	Value and Meaning
&FSYM	Bool	$EFAC	SYMFIL	1 = SYMFIL—YES, symbolic file supported. 0 = SYMFIL—NO, symbolic files not supported.
$FDFF	Bool	$EFAC	FORMAT	2 = FORMAT—PL1, Display Format Facility moved to PL1. 1 = FORMAT—YES, 3270 Display Format Facility supported. 0 = FORMAT—NO, Display Format Facility not supported.
&FSRT	Bool	$EFAC	SORT	1 = SORT—YES. 0 = SORT—NO.
&FPL1	Arith	$EFAC	FORMAT	2 = FORMAT—PL1, Display Format Facility moved to PL1.
&FRUF	Bool	$EFAC	PRUF	1 = PRUF—YES, Program Request Under Format supported. 0 = PRUF—NO, Program Request Under Format not supported.
&PLG	Char4	$EPLG	LANG	Programming language support, LANG—xxxxx. *Position* *Value* *Meaning* 1-4 1000 COBOL supported. 1-4 0100 FORTRAN supported. 1-4 0010 Assembler supported. 1-4 0001 RPG II supported. More than one language can be supported, therefore there can be many character combinations for &PLG. Check the SRL for the restrictions.
&UPCBL	Char2	$EPLG	PPUNIT	PPUNIT—R1/F1/R2/F2. Unit to which COBOL support routines are to be copied.
&UPFOR	Char2	$EPLG	PPUNIT	PPUNIT—R1/F1/R2/F2. Unit to which FORTRAN support routines are to be copied.
&UPASM	Char2	$EPLG	PPUNIT	PPUNIT—R1/F1/R2/F2. Unit to which Assembler support routines are to be copied.
&UPRPG	Char2	$EPLG	PPUNIT	PPUNIT—R1/F1/R2/F2. Unit to which RPG II support routines are to be copied.
&CPW	Bool	$ESEC	SECURE	1 = SECURE—CCP, CCP password checking included. 0 = CCP password checking not included.
&UPW	Bool	$ESEC	SECURE	1 = SECURE—USER, user security routine supported. 0 = User security routine not supported.
&LUS	Arith	$ESEC	LUSI	LUSI—value. Length of the user security information if &UPW = 1.
&NS	Arith	$EFIL	SETS	SETS—value. Maximum number of assignment sets in $CCPFILE.
&NPM	Arith	$EFIL	PROGS	PROGS—value. Maximum number of user programs in any one assignment set.
&NDF	Arith	$EFIL	DFILES	DFILES—value. Maximum number of disk files in any one assignment set.

Name	Type	Stmt	Keywd	Value and Meaning
&NT	Arith	$EFIL	TERMS	TERMS—value. Maximum number of terminals in any one assignment set.
&DMP	Arith	$EFIL	DUMPS	DUMPS—value. Space reserved in CCPFILE for the specified number of dynamic main storage dumps.
&COR	Arith	$EFIL	CORE	CORE—nnK. Object CPU size. *Value* *Meaning* 24576 Storage—24K (Model 10 only) 32768 Storage—32K (Model 10 only) 49152 Storage—48K 0 Storage—64K 1 Storage—80K 2 Storage—96K
&TRC	Arith	$EFIL	TRACE	TRACE—value. Number of tracks reserved in $CCPFILE for CCP trace entries.
&UFIL	Char2	$EFIL	FLUNIT	FLUNIT—R1/F1/R2/F2. Disk drive on which $CCPFILE is to be allocated.
&PFIL	Char6	$EFIL	FLPACK	FLPACK—packname. Name of the $CCPFILE pack.
&TFIL	Arith	$EFIL	TRKLOC	TRKLOC—value. Beginning track location for $CCPFILE. TRKLOC—0. Operand not specified.
&MLA	Arith	$EMLA	LINES	LINES—value. Number of teleprocessing lines in the user's MLTA support. LINES—0. No MLTA support.
&MNOX	Bool	$EMLA	XLATE	1 = XLATE—YES, translation between EBCDIC and line code always to occur. 0 = XLATE—NO, additional code to be included to inhibit translate when so requested in user programs.
&MT40	Bool	$EMLD	TYPE	1 = TYPE—2740, 2740 device supported. 0 = 2740 device not supported.
&MT41	Bool	$EMLD	TYPE	1 = TYPE—2741, 2741 device supported. 0 = 2741 device not supported.
&MT50	Bool	$EMLD	TYPE	1 = TYPE—1050, 1050 device supported. 0 = 1050 device not supported.
&MFSC	Bool	$EMLD	TYPE	1 = Station control supported. 0 = Station control not supported.
&MFSW	Bool	$EMLD	TYPE	1 = Switched lines supported. 0 = Switched lines not supported.
&MFBR	Bool	$EMLD	TYPE	1 = Buffer receive supported. 0 = Buffer receive not supported.

Name	Type	Stmt	Keywd	Value and Meaning
&MFTC	Bool	$EMLD	TYPE	1 = Transmit control supported. 0 = Transmit control not supported.
&MFCK	Bool	$EMLD	TYPE	1 = Checking terminal supported. 0 = Checking terminal not supported.
& MFNK	Bool	$EMLD	TYPE	1 = Non-checking terminal supported. 0 = Non-checking terminal not supported.
&XM4E	Bool	$EMLD	TYPE	1 = 2740 PTTCEBCD code supported. 0 = 2740 PTTCEBCD code not supported.
&MD1	Char8	$EMLD	TYPE	MLTA device support. *Position* *Value* *Meaning* 1 1 TYPE–1050 2 1 TYPE–1050D 3 1 TYPE–2740 4 1 TYPE–2740S 5 1 TYPE–2740C 6 1 TYPE–2740SC 7 1 TYPE–2740D 8 1 TYPE–2740DT Zero value indicates that the device is not supported.
&MD2	Char8	$EMLD	TYPE	MLTA device support. *Position* *Value* *Meaning* 1 1 TYPE–2740DC 2 1 TYPE–2740DTC 3 1 TYPE–2740M2S 4 1 TYPE–2740M2SB 5 1 TYPE–2740M2SC 6 1 TYPE–2740M2SCB 7 1 TYPE–2741 8 1 TYPE–2741D Zero value indicates that the device is not supported.
&MD3	Char4	$EMLD	TYPE	MLTA device support. *Position* *Value* *Meaning* 1 1 TYPE–SYS7C 2 1 TYPE–SYS7SC 3 1 TYPE–SYS7DC 4 1 TYPE–CMCSTD Zero value indicates that the device is not supported.

Name	Type	Stmt	Keywd	Value and Meaning
&MXC	Char4	$EMLD	XMCODE	MLTA line transmission code support. *Position* *Value* *Meaning* 1 1 XMCODE–CORR 2 1 XMCODE–PTTCEBCD (2740/1) 3 1 XMCODE–PTTCBCD 4 1 XMCODE–PTTCEBCD (1050) Zero value indicates that the code is not supported.
&BSC	Arith	$EBSC	BSCA	BSCA–value. Number of BSCA lines to be supported. BSCA–0. BSCA is not supported.
&BLT	Char4	$EBSC	PP MP CS DIAL	BSCA line type support. *Position* *Value* *Meaning* 1 1 PP–YES (point to point) 2 1 MP–YES (multipoint) 3 1 CS–YES (control station) 4 1 DIAL–YES (switched line) Zero value indicates that the control logic for the line type is not included.
&BFA	Char8	$EBSC	GETMSG ITB RESPOL AUTORS EBCDIC ASCII XPRNCY	BSCA features supported. *Position* *Value* *Meaning* 1 1 GETMSG–YES (get message) 2 1 ITB–YES (intermediate text block) 3 1 Always 1 4 1 RESPOL–YES (storage resident polling) 5 1 AUTORS–YES (automatic response to polling) 6 1 EBCDIC–YES (EBCDIC transmission code) 7 1 ASCII–YES (ASCII transmission code) 8 1 XPRNCY–YES (text transparency feature) Zero indicates that the feature is not supported.
&RSB	Char2	$EBSC	RECSEP	RECSEP–value. Value of the record separator character (1E if not specified).
&BD1	Char8	$EBSD	TYPE	BSCA device support. *Position* *Value* *Meaning* 1 1 TYPE–3275M1 2 1 TYPE–3277M1 3 1 TYPE–3284M1 4 1 TYPE–3286M1 5 1 TYPE–3275M2 6 1 TYPE–3277M2 7 1 TYPE–3284M2 8 1 TYPE–3286M2 Zero value indicates that a device is not supported.

Name	Type	Stmt	Keywd	Value and Meaning
&BD2	Char8	$EBSD	TYPE	BSCA device support.

Position	*Value*	*Meaning*
1	1	TYPE—3735
2	1	TYPE—CPU
3	1	TYPE—3741
4-8	—	Unused

Second Pass

Name	Type	Stmt	Keywd	Value and Meaning
&DPF	Bool	$EFAC	DPF	1 = Dual programming feature supported. 0 = Dual programming feature not supported.
&MTK	Bool	$EFAC	MAXEUP	1 = Multiple user tasks supported. 0 = Single user task supported.
&MLTA	Bool	$EMLA	LINES	1 = MLTA supported. 0 = MLTA not supported.
&NOM	Bool	$EMLA	LINES	1 = MLTA not supported. 0 = MLTA supported.
&BSCA	Bool	$EBSC	BSCA	1 = BSCA supported. 0 = BSCA not supported.
&NOB	Bool	$EBSC	BSCA	1 = BSCA not supported. 0 = BSCA supported.
&ONE	Bool	$EMLA $EBSC	LINES BSCA	1 = Single adapter support for MLTA or BSCA. 0 = Both MLTA and BSCA supported.
&DME	Bool	$EFAC	ESCAPE	1 = Data Mode Escape support. 0 = Data Mode Escape not supported.
&NDME	Bool	$EFAC	ESCAPE	1 = No Data Mode Escape support. 0 = Data Mode Escape support.
&DFF	Bool	$EFAC	FORMAT	1 = Display Format Facility supported. 0 = Display Format Facility not supported.
&FPL1	Arith	$EFAC	FORMAT	2 = Display Format Facility moved to PL1.
&SRT	Bool	$EFAC	SORT	1 = Sort Facility supported. 0 = Sort Facility not supported.
&NDFF	Bool	$EFAC	FORMAT	1 = Display Format Facility not supported. 0 = Display Format Facility supported.

Name	Type	Stmt	Keywd	Value and Meaning
&MIN	Bool	$EGEN	MINRES	1 = Minimum core system supported. 0 = Minimum core system not supported.
&CL	Bool	$EGEN	CARD	1 = Card-oriented generation. 0 = Cardless-oriented generation.
&OR	Arith	$EFIL	CORE	*Value* *Meaning* 49152 Storage—48K 0 Storage—64K 1 Storage—80K 2 Storage—96K
&SHR	Bool	$EFAC	FSHARE	1 = Update file sharing supported. 0 = Update file sharing not supported.
$USEON	Bool	$ESEC	SECURE	1 = User sign on routine used. 0 = User sign on routine not used.
&SYSON	Bool	$ESEC	SECURE	1 = CCP password security required. 0 = CCP password checking not required.
&PUCNT	Bool	$EFAC	PGMCNT	1 = Program request count supported. 0 = Program request count not supported.
&URMFU	Bool	$EIOD	CARD	1 = MFCU supported. 0 = MFCU not supported.
&UR142	Bool	$EIOD	CARD	1 = 1442 supported. 0 = 1442 not supported.
&URPRT	Bool	$EIOD	PRINTR	1 = 5203/1403 supported. 0 = 5203/1403 not supported.
&UR41	Bool	$EIOD	3741	1 = 3741 supported. 0 = 3741 not supported.
&NSCTL	Bool	$EMLA	TYPE	1 = Station control not supported. 0 = Station control supported.
&NSW	Bool	$EMLA	TYPE	1 = Switched line not supported. 0 = Switched line supported.
&N1050	Bool	$EMLA	TYPE	1 = 1050 not supported. 0 = 1050 supported.
&N2741	Bool	$EMLA	TYPE	1 = 2741 not supported. 0 = 2741 supported.
&NBFR	Bool	$EMLA	TYPE	1 = Buffered receive not supported. 0 = Buffered receive supported.

Name	Type	Stmt	Keywd	Value and Meaning
&NMOVE	Bool	$EMLA	XLATE	1 = Move without translate not supported. 0 = Move without translate supported.
&NCPU	Bool	$EBSD	TYPE	1 = CPU-to-CPU not supported. 0 = CPU-to-CPU supported.
&NITB	Bool	$EBSC	ITB	1 = Intermediate text block not supported. 0 = Intermediate text block supported.
&NTSP	Bool	$EBSC	XPRNCY	1 = Text transparency not supported. 0 = Text transparency supported.
&N32	Bool	$EBSD	TYPE	1 = 3270 not supported. 0 = 3270 supported.
&N37	Bool	$EBSD	TYPE	1 = 3735 not supported. 0 = 3735 supported.
&N41	Bool	$EBSD	TYPE	1 = 3741 not supported (terminal). 0 = 3741 supported (terminal).
&NAS	Bool	$EBSC	ASCII	1 = ASCII not supported. 0 = ASCII supported.
&NPP	Bool	$EBSC	PP	1 = Point to point not supported for BSCA. 0 = Point to point supported for BSCA.
&NMP	Bool	$EBSC	MP	1 = Multipoint tributary not supported for BSCA. 0 = Multipoint tributary supported for BSCA.
&NSWL	Bool	$EBSC	DIAL	1 = BSCA switched line not supported. 0 = BSCA switched line supported.
&NCS	Bool	$EBSC	SC	1 = Control station not supported for BSCA. 0 = Control station supported for BSCA.
&EBIDA	Bool	$EBSC	DA	1 = DA—YES, display adapter supported. 0 = DA—NO, display adapter not supported.

CCP Generation Utility ($CC1PP)

CHART: AA
FUNCTIONS:
- Print the user's input to the first pass and any messages from the CCP Generation macros.
- Determine if, on the first pass of CCP Generation, any errors were detected.
- If any errors, summarize the extent of errors and exit.
- If no errors, print that output from the first pass of Generation which serves as input to the second pass of Generation.
- If card-oriented generation, punch the past two records.
- If cardless-oriented generation, reorganize the $SOURCE file to eliminate the user input records.

ENTRY POINT: PPEXEC
INPUT: 96-byte records in the $SOURCE file which were:
- Input from the user
- Error message output directly from Macro Processor
- Messages (error and warning) from the Generation macros
- Second pass input produced by the Generation macros
- Trailer record from the Generation macro $EGEN
- Slash-asterisk record representing end of $SOURCE

OUTPUT:
- Printed listing of all input from the user (whether errors or not).
- Printed listing of all messages from the Generation macros (whether errors or not).
- Printed listing of all error messages directly from Macro Processor. The converted Macro Processor error messages, the number and abbreviated text are as follows:

MPX Error	*CCP Error*	*CCP Abbreviated Text*
NF	CC901E	Invalid stmt id or misplaced source lib
OC	CC902E	Invalid stmt format or previous err
IK	CC903E	A keyword used is not valid
IR	CC904E	Parameter missing or has invalid form
ID	CC905E	Invalid delimiter or delimiter placement
IC	CC906E	Comma after last OPND, but no continuation
CE	CC907E	On continuation of statement, cols. 1-13 are not blank
OTHER	CC909E	Error xx from Macro Processor — possible CCP error

- Halt issued if user specification error.
- Punched cards or reorganized $SOURCE file (if no errors were detected) to serve as input to the second pass of generation:
  - OCL, source, and link-edit control statements to establish the initial contents of $CCPFILE. (A load module, $CC1BF, is created and stored on the distribution pack to be used to initialize $CCPFILE according to the user's input.)
  - OCL, source, and link-edit control statements to generate the tailored CCP execution-time resident supervisor ($CC4, written as a load module to the production pack).
  - OCL and utility control statements to copy the required load modules of Assignment, Startup, Shutdown, and transients of the CCP Execution Stage to the production pack.
  - If the user chose the user-security option, then OCL, source, and link-edit control statements to create the (empty) user-information module $CC4Z9 as a load module on the production pack.
  - OCL to cause the initialization of $CCPFILE (by execution of the load module $CC1BF).
  - OCL and utility control statements to copy the modules for the support of each language (macros for Assembler, relocatable modules for the other languages) to the pack(s) specified by the user.
  - OCL and utility control statements to punch out:
    - Overlay Linkage Editor control statements for the link edit of the Installation Verification Program
    - A sample assignment deck
    - $OLINK traces
    - The stand-alone storage dumps (card oriented only)

EXTERNAL REFERENCES:
- Program level communication area for date (NPDATE).
- CAM for input/output from the $SOURCE file.
- Printer data management routine ($$LPRT) for printed output.
- System punch routine for punched output.

EXIT, NORMAL: DSM end-of-job routine.
EXITS, ERROR:
- DSM end-of-job routine after halt.
- Halt/syslog to a U– halt with following subhalts:
  - PU – punch unallocatable
  - HE – disk file permanent error
  - CC – CCP internal error
  - F2 – User specification error

Build Initial Contents of $CCPFILE ($CC1BF)

CHART: AB
FUNCTIONS: Write the configuration and directory sectors of $CCPFILE as specified by the user. (The encoded version of those specifications (the configuration and directory sectors of $CCPFILE) is contained within this load module, having been link-edited from relocatable module $CC1FC.)
ENTRY POINT: EXECBF
INPUT: NONE
OUTPUT:
- Configuration and directory sectors of $CCPFILE.
- Completion message on the system log device.

EXTERNAL REFERENCES:
- Disk data management module $$DAUB for disk output
- Halt/syslog routine for printing of the completion message.
- Module $CC1FC created previously and link-edited with this module.

EXIT, NORMAL: DSM EOJ routine.
EXIT, ERROR: Halt/syslog to a halt with subhalt AF on permanent disk error.

Create the $CCPLOG File ($CC1BL)

FUNCTION: Create the file needed by CCP on Model 4 for a console log file.
ENTRY POINT: $CC1BL
INPUT: OCL file statement for the $CCPLOG file.
OUTPUT: The $CCPLOG file exists as specified on the file statement.
EXIT, NORMAL: End of job.
EXTERNAL REFERENCE: Allocate, open, and close.

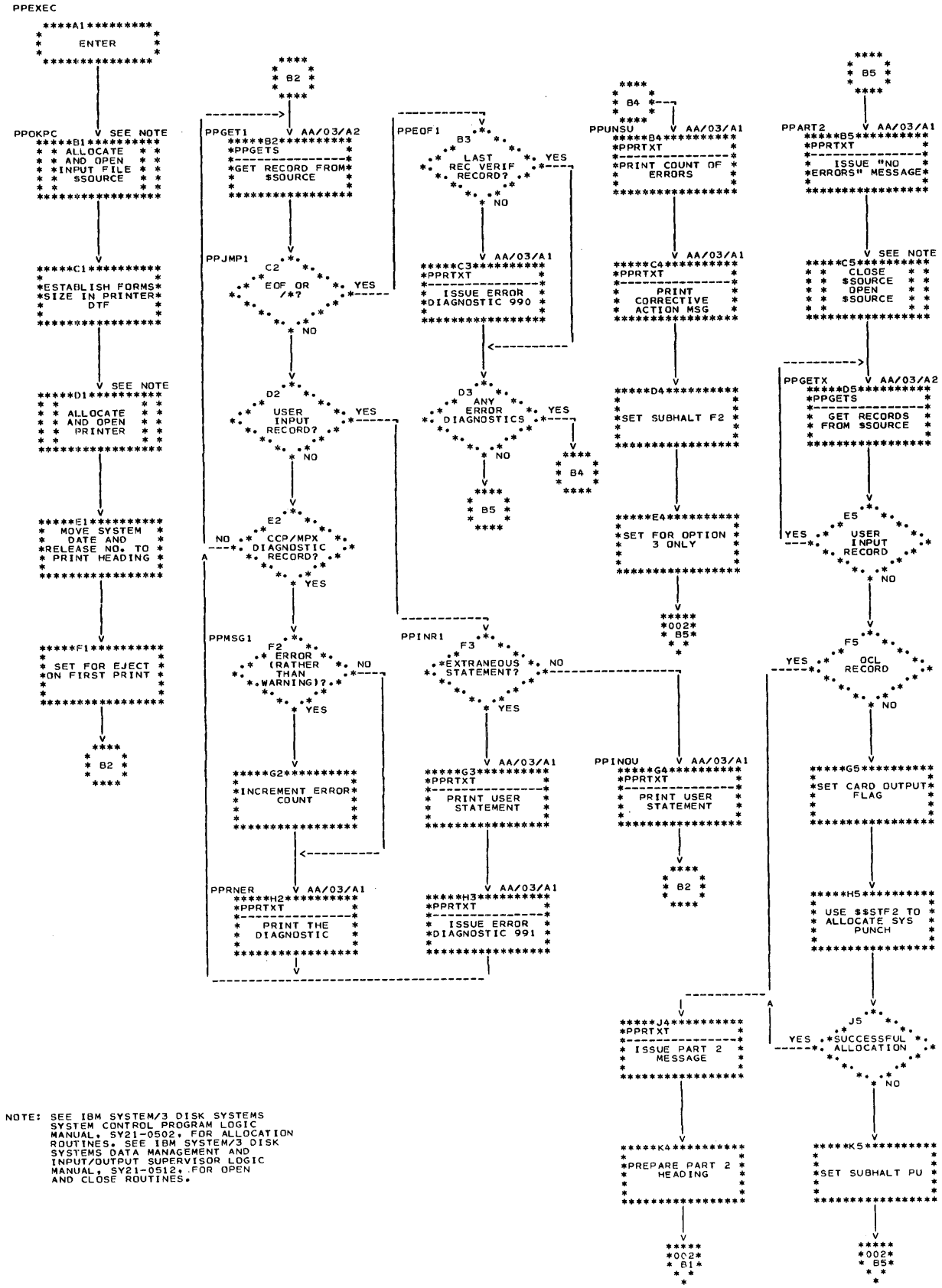


Chart AA (Part 1 of 3). CCP Generation Utility ($CC1PP) (Models 8, 10, and 12 Only)

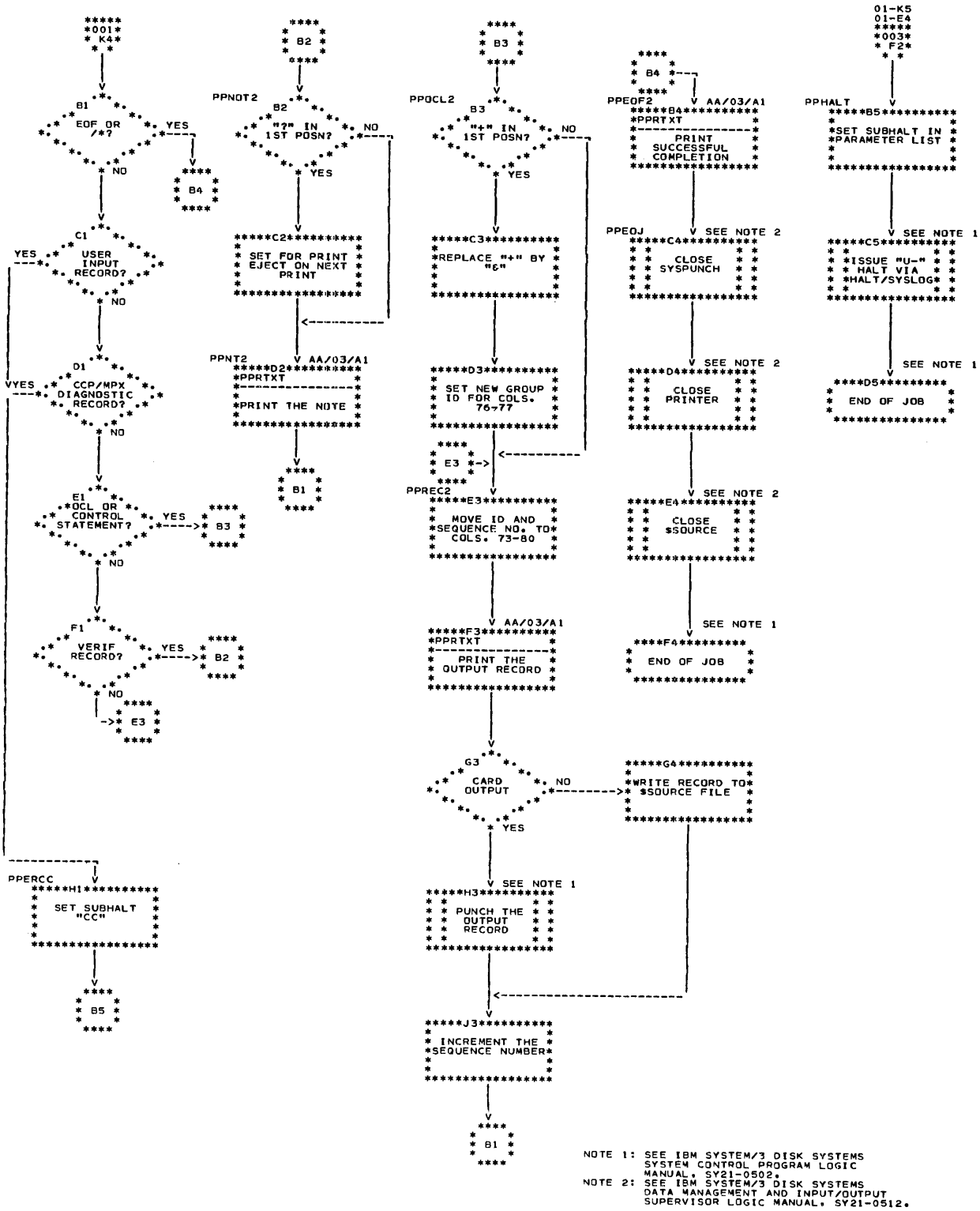


Chart AA (Part 2 of 3). CCP Generation Utility ($CC1PP) (Models 8, 10, and 12 Only)

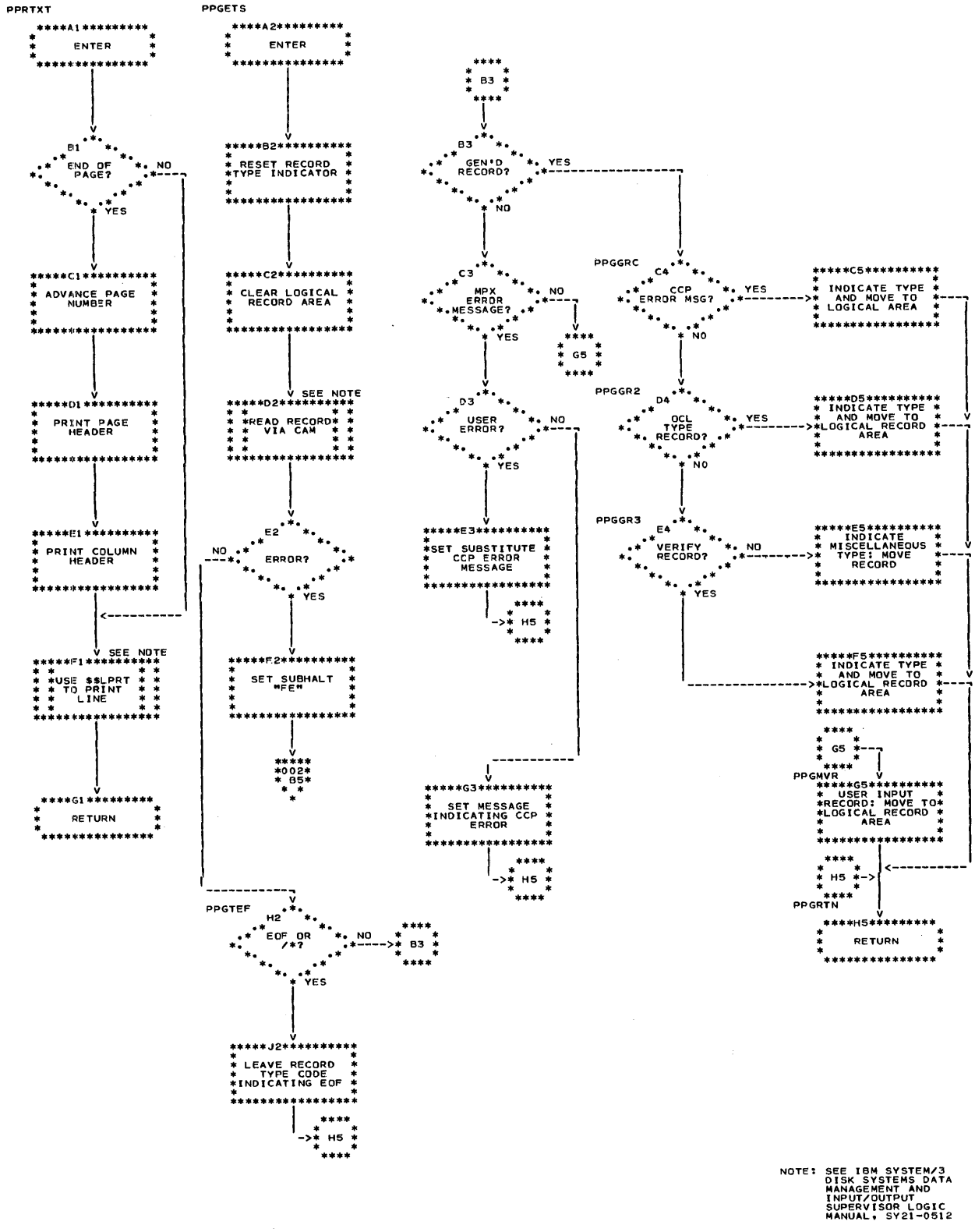


Chart AA (Part 3 of 3). CCP Generation Utility ($CC1PP) (Models 8, 10, and 12 Only)

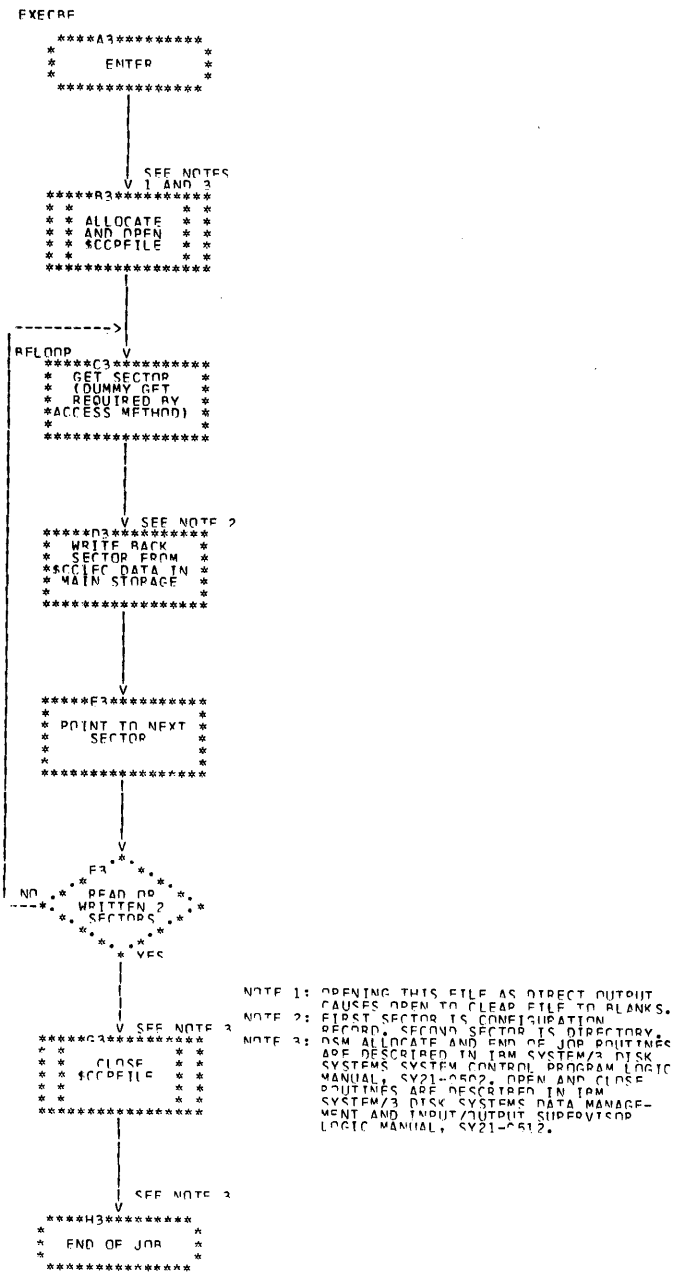


Chart AB. Build Initial Contents of $CCPFILE ($CC1BF)

Introduction to CCP Installation (Model 4 Only)

Function

The Installation Stage of CCP on Model 4:

- Copies the necessary load modules for the designated version of CCP to the CCP production pack.
- Copies the necessary relocatable modules (subroutines for user programs) to the designated program preparation pack.
- Allocates and initializes the CCP assignment file ($CCPFILE) and CCP log file ($CCPLOG).
- Sets up the correct printer intermediary module, which is used by the RPG II compiler when compiling and link-editing user programs that require the printer.

Procedure

To perform CCP installation:

- Call procedures that copy the necessary modules.
- Load the programs that initialize the $CCPFILE and $CCPLOG.

METHOD OF OPERATION

To get the desired version of CCP from the distribution pack to the CCP production pack, the system pack must be on F1 and include the overlay linkage editor, the CCP distribution pack must be on R1, and you must run the following procedure:

Keywords		Response
READY–		
CALL	NAME–	CALL
	UNIT–	verVxx
		R1
MODIFY		
RUN		

ver specifies the desired version which is either:

MIN: 3270 only on the control station line
MAX: all supported BSCA devices and all BSCA line configuration

xx is F1, R2, or F2 that specifies the unit on which the desired CCP production pack is placed. For example:

Keywords		Response
READY–		
CALL	NAME–	CALL
	UNIT–	MAXVF2
		R1
MODIFY		
RUN		

This copies the maximum version of CCP from the distribution pack on R1 to the production pack on F2.

To transfer the CCP subroutines used with RPG II to the desired RPG II program preparation pack, the RPG II program preparation pack must be in a non-CCP mode and you must run the following procedure:

Keywords		Response
READY–		
CALL	NAME–	CALL
	UNIT–	RPGVxx
		R1
MODIFY		
RUN		

xx is F1, R2, or F2 that specifies the unit on which the desired RPG II program preparation pack is located.

To create the file for $CCPFILE, you must run the following job:

Keywords		Response
READY–		
LOAD	NAME–	LOAD
	UNIT–	$CC1BF
		nn
FILE	NAME–	$CCPFILE
	UNIT–	code
	PACK–	name
	TRACKS–	number
	LOCATION–	track number
	RETAIN–	P
MODIFY		
RUN		

To create the file for $CCPLOG, you must run the following job:

Keywords		Response
READY–		LOAD
LOAD	NAME–	$CC1BL
	UNIT–	nn
FILE	NAME–	$CCPLOG
	UNIT–	code
	PACK–	name
	TRACKS–	number
	LOCATION–	track number
	RETAIN–	P
MODIFY		
RUN		

nn is the unit that the CCP production pack is on: F1, F2, or R2.

Chapter 3. SCP Generator (Models 8, 10, and 12 Only)

Introduction

The IBM System/3 Models 8, 10, and 12 System Control Program Generator (SCP Generator) is a language processor that generates relocatable object modules from input to the CCP Generation Stage (see Chapter 2 for a description of the Generation Stage). This language processor is always used to generate at least three relocatable object modules, and may be used to generate an optional fourth module:

- $CC1FC, initialization data for the assignment file ($CCPFILE). See Chapters 4 and 5 for a description of the CCP Assignment Programs.
- $CC4#1, $CC4#2, and $CC4#3 containing resident CCP code.
- $CC4Z9, a null module required only for user-written security routines.

All input for the language processor resides in a disk source file ($SOURCE). The source file is processed by the following phases:

$CGNIN Processor Initialization Phase. Initializes the processor.

$CGNCM Source Compression Phase. Reads source file ($SOURCE) and generates intermediate text in the work file ($WORK2).

$CGNSB Symbol Table Build Phase. Reads the intermediate text and builds symbol table in main storage.

$CGNSF Symbol Table Overflow Processing Phase. Called only if symbol table overflows. Tests intermediate text following the overflow for previously defined symbols.

$CGNSS Symbol Substitution Phase. Places values from last (or only) symbol table into the intermediate text term records (see index entry *Data Area Formats, Term Records)*. Builds an ESL (External Symbol List) table.

$CGNPE ESL Output Phase. Writes ESL records in the object file ($WORK) and prints the ESL.

$CGNPS Source/Object Output Phase. Generates object code and source/object listing, and writes the object code in the object file ($WORK).

$CGNBX Build XREF File Phase. Builds a cross-reference file in the work file ($WORK2).

$CGNSX Merge and List Cross Reference. Sorts the file built by $CGNBX, generates the cross-reference listing, and fetches the Overlay Linkage Editor to put the object module in the object library.

A detailed account of each phase is contained in *Program Organization* in this chapter.

SYSTEM REQUIREMENTS

Listed below is the minimum system configuration for the SCP Generator:

- A processing unit with a minimum of 24K bytes of main storage (to contain DSM and the SCP generator).
- IBM 1403 or 5203 Printer with Universal Character Set Feature PN (60 character set) Interchangeable Chain Cartridge.

  Note: A 48 character chain can be used with the SCP Generator. However, the user must be willing to accept substitute characters.
- IBM 5444 Disk Drive.

STORAGE REQUIREMENTS

The SCP Generator requires 16K bytes of main storage for execution.

PREREQUISITE PUBLICATIONS

Effective use of SCP Generator requires an understanding of the following manuals:

- *IBM System/3 Basic Assembler Reference Manual*, SC21-7509.
- *IBM System/3 Models 4, 6, 8, and 10 Disk Systems System Control Program Logic Manual*, SY21-0502.
- *IBM System/3 Disk System Data Management and Input/Output Supervisor Logic Manual*, SY21-0512.

Method of Operation

This section describes the functional flow of logic and data through the various phases of the language processor.

PHASE-TO-PHASE COMMUNICATION AND FILE USAGE

Phase-to-Phase Communication Tables

Phase-to-phase communication during execution of the processor is accomplished by using communication tables:

1. $CGNIN Communication Tables. The following communication tables are loaded with $CGNIN and remain in main storage until the processor completes execution:
   - Communications Area (COMARA). Contains a transfer vector for all data management routines, address tables for any working storage that must be addressed but exists outside an execution phase, and the name of the current execution phase. Disk DTFs and IOBs for the source, work, and object files are also in COMARA.
   - Common Area (COMMON). A working storage/constant block used by all phases of the processor for intraphase and interphase communication.

   Note: The Compiler Access Method (CAM) is also loaded with $CGNIN. CAM is described in detail in the *Program Organization* section of this chapter.

2. $CGNPE Communication Tables. The following communication tables are loaded with $CGNPE and remain in storage until the processor completes execution. For more information about the following tables, see index entry *Data Area Formats*.
   - Header Area. Used for passing the header of the listing from phase to phase. Accessed through COMARA.
   - Printer DTF. Used by all output phases. Accessed through COMARA.
   - Printer Buffer. Used as a properly aligned print buffer (X'7C') boundary.

Register Conventions

1. The processor uses Index Register 1 as the communications register. XR1 points either to the Communications Area (COMARA) or to the Common Area (COMMON) at all times. When it points to COMARA, XR1 has the symbolic name CAP; when it points to COMMON, XR1 has the symbolic name CMP. Each of these two communications tables contains the displacement of the other, so XR1 can be switched from one to the other with one instruction. At the entry to each phase except $CGNIN, XR1 points to COMARA. The first instruction of each phase moves the phase name to COMARA so that COMARA always contains the name of the current execution phase.

2. Index Register 2 is used as a DTF pointer, base register, and work register. The symbolic name of XR2 for each case is DTF, BR2, and WK2, respectively. The content of XR2 at the entry to a phase is unpredictable.

Work File ($WORK2)

The work file is a scratch file used by the processor for intermediate storage (on disk storage drive). Data is passed back and forth between the work file and processor phases throughout execution of the processor.

Source File ($SOURCE)

The source file is used by the processor for storing the source program. It provides source records for $CGNCM and $CGNPS. The source file is loaded before execution of the processor.

Object File ($WORK)

The object program produced by the language processor is written to the object file by $CGNPE and $CGNPS. $CGNSX passes control to the Overlay Linkage Editor to put the object file (module) in the object library.

ILLUSTRATED OVERVIEW

Diagram 3M.0010 shows the main storage load structure for each phase in the processor. It also indicates which areas remain the same from one phase to the next and which areas are overlaid with different code.

Diagram 3M.0020 illustrates the overall flow of logic and data through the processor. The logic flow is traced from phase to phase, with the major functions of each phase listed under the name of the phase. The data flow is traced between phases and the work, source, and object files. Any output produced by a phase is indicated as data flow.

Note: Two dark lines are shown in each phase. The area between the lines is newly loaded by the phase. The area above the first line and below the second remains the same as it was for the preceding phase.

Diagram 3M.0010. Main Storage Load Structure for the SCP Generator (Models 8, 10, and 12 Only)

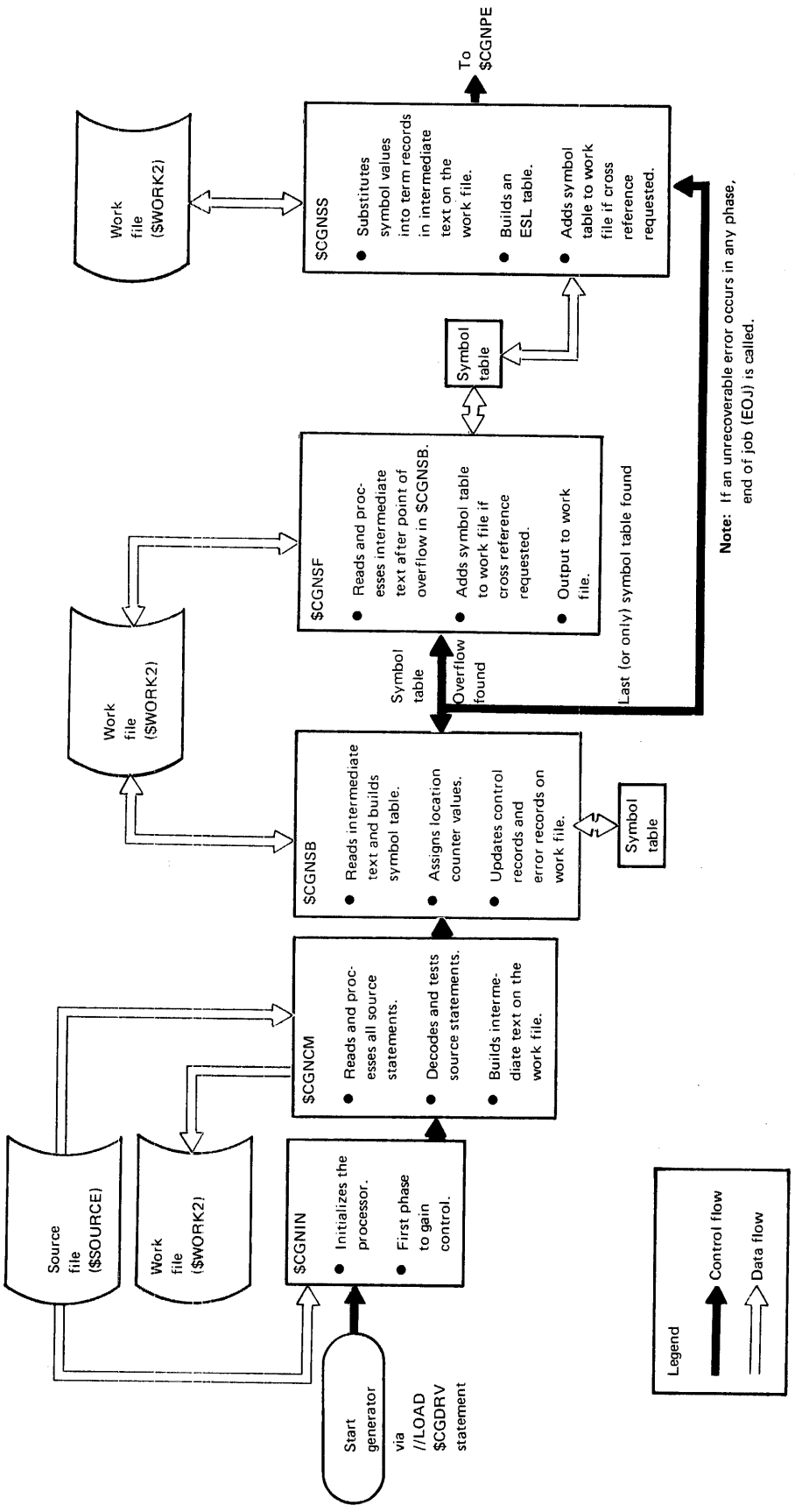


Diagram 3M.0020 (Part 1 of 2). Control and Data Flow in the SCP Generator (Models 8, 10, and 12 Only)

Note: If an unrecoverable error occurs in any phase, end of job (EOJ) is called.

Diagram 3M.0020 (Part 2 of 2). Control and Data Flow in the SCP Generator (Models 8, 10, and 12 Only)

Program Organization

This section describes in detail each of the phases and routines that perform the functions of the SCP Generator. The description of each phase consists of a main storage map, appropriate flowcharts, and supporting text derived from prologues in the phase listings. This text is intended only to supplement the flowcharts (some of the simpler routines within each phase do not have flowcharts).

Processor Initialization Phase ($CGNIN – PID Name is $CGDRV)

ENTRY POINT: INI000 – entered from DSM scheduler. (See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.)
MAIN STORAGE MAP: Figure 3-1.
CHART: BA
FUNCTION: Initializes the processor (INI000 – Chart BA).
INPUT: Opened files – $SOURCE, $WORK, and $WORK2.
OUTPUT: Communications Area (COMARA) and Compiler Access Method (CAM).
EXITS:

- Normal: Control is passed to $CGNCM.
- Error: Control is passed to the DSM scheduler via Halt/Syslog or EOJ transients. (See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.)

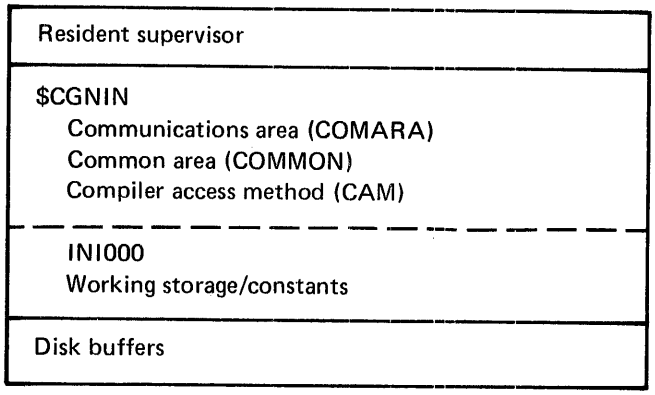


Figure 3-1. Main Storage Map of $CGNIN

Source Compression Phase ($CGNCM)

ENTRY POINT: CMI000 – entered from $CGNIN via Fetch. (See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.)
MAIN STORAGE MAP: Figure 3-2.
CHARTS: BB – BH
FUNCTIONS:

- Initializes $CGNCM for processing (CMI000 – Chart BB).
- Controls $CGNCM processing of source records, including sequence checking if requested (CMP000 – Chart BB).
- Tests name field (CMN000 – no chart).
- Writes a control record and a name record if a valid symbol is present (CMU000 – no chart).
- Performs actions required when $CGNCM processing is completed (CML000 – Chart BG).
- Fetches $CGNSB (CML000 – no chart).
- Determines the operation code specified on the current source statement (CMO000 – no chart).
- Determines if a referenced byte contains either a valid alphabetic or numeric character (CMC000 – no chart).
- Searches a source record; records following information in COMMON.

  1. Length of name (if present).
  2. Column number of rightmost byte of operation code.
  3. Length of operation byte.
  4. Column number of leftmost byte of operand.
  5. Length of operand field.
  6. Column number of rightmost byte of operand (CMS000 – no chart).

- Converts zoned decimal strings to binary equivalents (CMV000 – no chart).
- Decodes and tests the syntax of the operand field of the current source statement for proper syntactical coding (CMY000 – Chart BC: see also *Figure 3-3. Syntax Checking*).

- Determines if the operand format set by CMY000 is valid for the current type of source statement (CMX000 – no chart).
- If operand format is valid, sets up operand byte in the intermediate text control record build area (CMX000 – no chart).
- Determines object length (in bytes) of current machine source statement (CMR000 – no chart).
- Prepares each term in the operand field for CMD000 by searching for the end of the current term and temporarily placing a blank in that byte (CMK000 – Chart BE).
- Creates intermediate text term record for each valid term in the operand field of the current source record (CMD000 – Chart BF).
- Processes all assembler operations (CMA000 – Chart BD).
- Creates intermediate text term records for the operand field of the TITLE and DC statements (CMA000 – Chart BD).
- Provides interfaces with data management for disk operations (CMW000, CMW010, CMW020, CMW030, CMW100 – Chart BH).

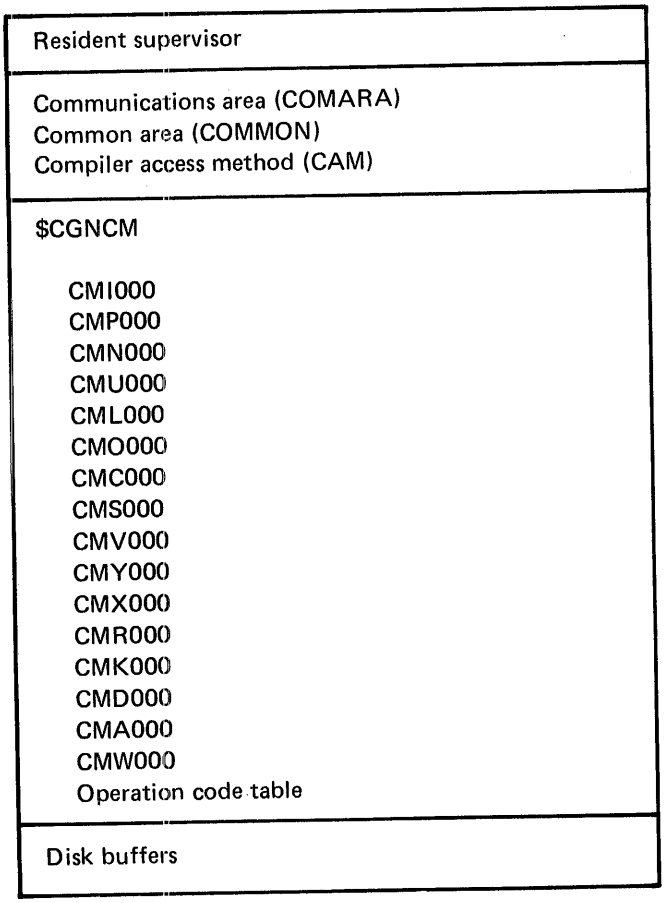


Figure 3-2. Main Storage Map of $CGNCM

INPUT:
- Source file ($SOURCE). Source records are read from $SOURCE with disk data management.
- Common Area (COMMON). This area remains in main storage from $CGNIN.
- Operation Code Table. Contains information about all instructions supported by the processor. This table is divided into five sections, each of which can be accessed by means of pointers containing the address of the rightmost byte of the first mnemonic in that section. (Each pointer also contains a one-byte count of the number of mnemonics in the section.)

  1. Section 1, all one-character mnemonics.
  2. Section 2, all two-character mnemonics.
  3. Section 3, all three-character mnemonics.
  4. Section 4, all four-character mnemonics.
  5. Section 5, all five-character mnemonics.

  The operation code table is loaded with $CGNCM. It is included as input here because it is used by the phase as a model to which source records are compared.

OUTPUT:
- Intermediate text in the work file ($WORK2). Intermediate text records are written in $WORK2 with disk data management.
- Parameters stored in COMMON.

  1. TITLEN, name from the first title statement.
  2. MODULE, module name from the start statement.
  3. SEQCNT, count of sequence errors.
  4. ESLCNT, count of ESL table entries.

EXITS:
- Normal: Control is passed to $CGNSB.
- Error: Control is passed to the scheduler via Halt/Syslog or EOJ transients. (See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.)

Branch table[2]	Acceptable syntactical possibilities for currently loaded branch tables[1] / Possible Expression[3]	Any non-syntactical element[4]	End of field	Left parentheses	comma	Right parentheses
1	START	Record element start. Load branch table 2	Error	Error	Error	Error
2	X	Continue	Compute and store element length return	Compute and store element length load branch table 3	Compute and store element length load branch table 1	Error
3	X(	Record element start. Load branch table 7	Error	Error	Load branch table 4	Error
4	X(, or X(X,	Record element start. Load branch table 5	Error	Error	Error	Error
5	X(X,X or X(,X	Continue	Error	Error	Error	Compute and store element length. Load branch table 6
6	X(X,X) or X(,X) or X(X)	Error unless DC/DS	End of operand, end of field	Error unless DC/DS	End of operand, not end of field	Error
7	X(X	Continue	Error	Error	Compute and store element length. Load branch table 4	Compute and store element length. Load branch table 6

Notes:

(1) Acceptable syntactical possibilities are listed across the top of the table.

(2) Reading across from each Branch table number the action taken is indicated in the case of each syntactical possibility. (Example: In Branch table 1, only a non-syntactical element is acceptable. All other possibilities will cause an error bit to be turned on in the error record build areas.

(3) For each Branch table number, the Possible expression column shows the format that the operand being tested will be in up to that point.

(4) A non-syntactical element in an operand is a character or series of characters other than the parentheses and commas which form the syntactical elements. (Example: X(X,X) ... The Xs are non-syntactical elements.)

Figure 3-3. Syntax Checking

Symbol Table Build Phase ($CGNSB)

ENTRY POINT: SBI000 – entered from $CGNCM or $CGNSF via Fetch. (See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.)
MAIN STORAGE MAP: Figure 3-4.
CHARTS: CA – CK
FUNCTIONS:
- Initializes phase for processing (SBI000 – Chart CA).
- Controls processing of the intermediate text file until end of data is found (SBP000 – Chart CB).
- Performs functions associated with phase termination (SBL000 – no chart).
- Performs $CGNSB processing of machine instructions. Symbols are placed in symbol table (SBM000 – Chart CC).
- Test DC/DS specification for valid length and for duplication; stores symbol (if present) in table (SBD000 – Chart CD).
- Performs $CGNSB processing of ORG and EQU statements (SBO000 – Chart CE).
- Processes START statements (SBS000 – Chart CF).
- Processes EXTRN statements (SBE000 – Chart CJ).
- Updates location counter by object length of current statement (SBC000 – no chart).
- Checks for overflow of storage (SBC000 – no chart).
- Checks maximum location counter value (SBC000 – no chart).
- Evaluates all expressions in the operand field of the current source record (SBV000 – Chart CG).
- Performs multiplication calculations (SBX000 – no chart).
- Moves symbol and its attributes to symbol table (SBY000 – Chart CH).
- Counts symbols in table; controls calling of $CGNSF when table overflows (SBY000 – Chart CH).
- Searches symbol table for a given symbol (SBR000 – Chart CI).
- Provides interfaces with data management for disk operations (SBW000, SBW010, SBW100 – Chart CK).

INPUT: Intermediate text in $WORK2.

1. Name Records. Used to build symbol table.

2. Control Records. Used to obtain lengths for location counter assignments and to assign symbol lengths, attributes, and values.

3. Term Records. Used to obtain length allocation for DC/DS statements, location counter changes for ORG statements, and symbol length, attributes, and values for EQU statements.

OUTPUT:
- Symbol Table. Built in lower main storage and designed to fill all space between phase code and disk buffers.
- Symbol Table Parameters in COMMON. Used by $CGNSF and $CGNSS to access symbol table.

EXITS:
- Normal:

  1. Control is passed to $CGNSF on symbol table overflow.

  2. Control is passed to $CGNSS when last (or only) symbol table is processed.

- Error: Control is returned to scheduler via Halt/Syslog or EOJ transients. (See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.)

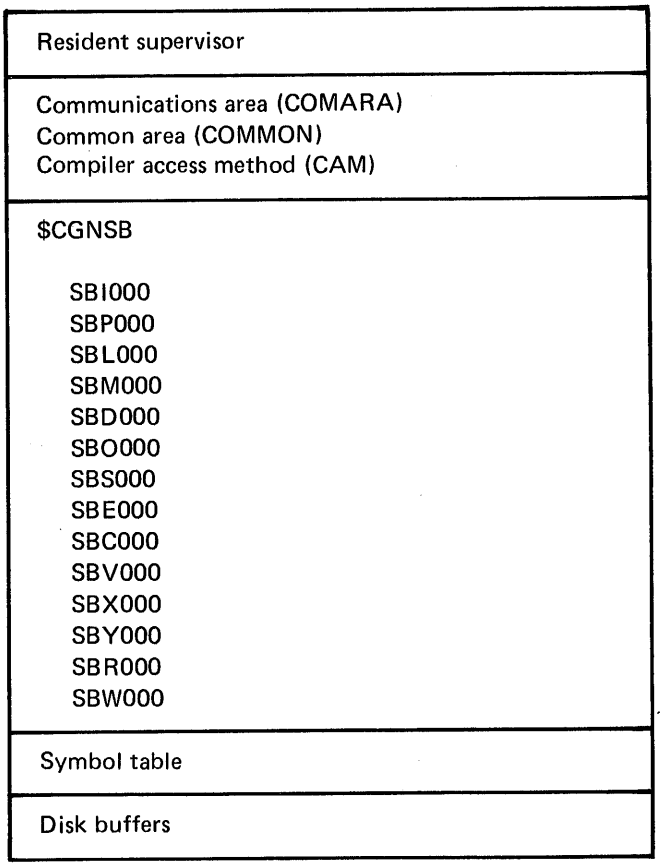


Figure 3-4. Main Storage Map of $CGNSB

Symbol Table Overflow Phase ($CGNSF)

ENTRY POINT: SFI000 – entered from $CGNSB via Fetch. (See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.)
MAIN STORAGE MAP: Figure 3-5.
CHARTS: DA – DD
FUNCTIONS:
- Controls processing of the intermediate text in $WORK2 (SFP000 – Chart DA).

- Processes intermediate text term records, resolving all symbols defined in the current symbol table (SFT000 – Chart DB).
- Tests name records for previously defined symbols (SFN000 – Chart DC).
- Searches the symbol table for a given symbol (SFS000 – no chart, but same as SBR000 in $CGNSB – Chart CI).
- Provides interface for disk data management (SFW000 – Chart DD).

INPUT:
- Intermediate text in $WORK2.
- Symbol table.
- COMMON symbol table parameters.

OUTPUT: All output is to $WORK2.

1. Intermediate text.

   a. Symbols in the term records that are defined in the present symbol table are updated with the symbol's value and attributes.
   b. Name records containing previously defined symbols are updated with a previously defined symbol indicator.

2. Symbol Table Entries. If a cross reference has been requested, all symbol table entries are added to the end of $WORK2.

EXITS:
- Normal: Control is passed to $CGNSB for continued symbol processing.
- Error: Control is returned to the scheduler via Halt/Syslog or EOJ transients. (See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.)

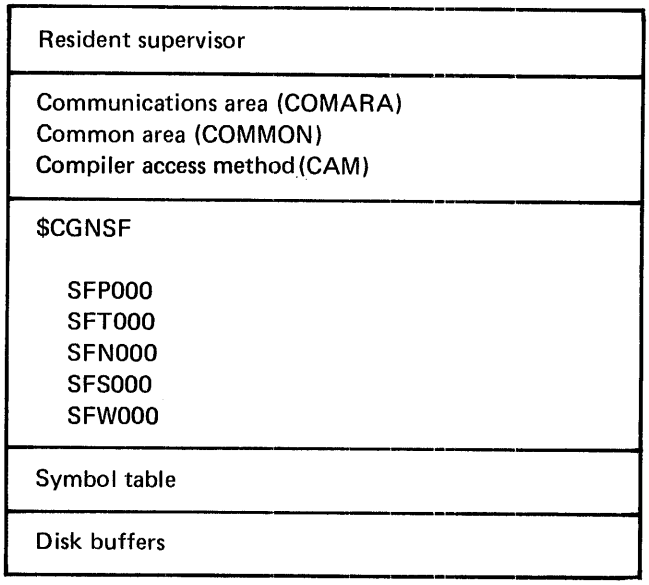


Figure 3-5. Main Storage Map of $CGNSF

Symbol Substitution Phase ($CGNSS)

ENTRY POINT: SSI000 – entered from $CGNSB via Fetch. (See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.)

MAIN STORAGE MAP: Figure 3-6.

CHARTS: EA – EE

FUNCTIONS:
- Initializes phase for processing (SSI000 – Chart EA).
- Controls phase processing (SSP000 – Chart EB).
- Performs functions associated with termination of $CGNSS (SSL000 – Chart EC).
- Tests term records for unresolved symbols and resolves those symbols in the current symbol table (SST000 – no chart, but same as SFT000 in $CGNSF – Chart DB).
- Builds ESL table entries for valid EXTRN and ENTRY statements (SSE000 – Chart ED).
- Adds EXTRN and ENTRY entries to ESL (SSU000 – no chart).
- Searches symbol table for a given symbol (SSS000 – no chart, but same as SBR000 in $CGNSB – Chart CI).
- Provides interface for disk data management (SSW000 – Chart EE).

INPUT:
- Symbol table in main storage.
- Associated symbol table parameters in COMMON.
- Intermediate text in $WORK2.

OUTPUT:
- All term symbols that are defined in the current symbol table are updated with their values and attributes in the intermediate text.
- Symbol Table Entries. If a cross-reference has been requested, all symbol table entries are added to the end of $WORK2.
- ESL table passed to $CGNPE.

EXITS:
- Normal: Control is passed to $CGNPE.
- Error: Control is returned to the scheduler via Halt/Syslog or EOJ transients. (See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.)

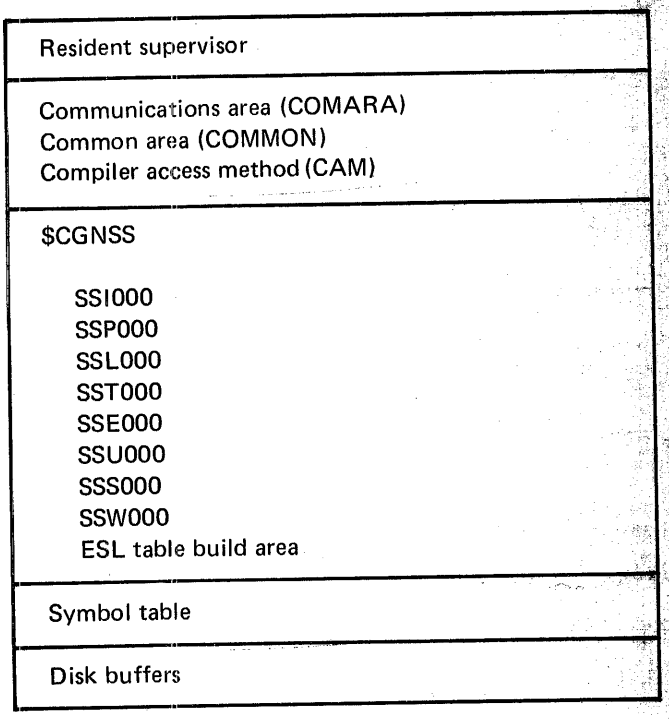


Figure 3-6. Main Storage Map of $CGNSS

ESL Output Phase ($CGNPE)

ENTRY POINT: PEI000 – entered from $CGNSS via Fetch. (See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.)

MAIN STORAGE MAP: Figure 3-7.

CHARTS: FA – FF

FUNCTIONS:

- Initializes $CGNPE (PEI000 – Chart FA).
- Sorts, builds, and writes out ESL object records (PEP000 – Chart FB).
- Prints the ESL table (PES000 – Chart FC).
- Performs functions associated with termination of the phase (PEL000 – Chart FD).
- Prints the printer listing header and counts pages (PEG000 – no chart).
- Provides interface for disk data management (PEW000 – Chart FE).
- Provides interface for printer data management (PER000 – Chart FF).

INPUT:

- ESL table moved below phase end by $CGNSS.
- Parameters in COMMON.

  1. ESLCNT, contains a count of ESL table entries.
  2. MODULE, module name.
  3. TITLEN, TITLE name.
  4. ESLTBA, ESL table address.

OUTPUT:

- ESL object records are placed in the $WORK.
- ESL table is sorted, then printed.
- The following information is passed to $CGNPS via COMMON:

  1. PAGCNT, current printer page size.
  2. LPSIZE, current printer line size.
  3. Other miscellaneous areas of COMMON are initialized.

- Listing header in the printer communications area is initialized for the source/object listing done in $CGNPS.

EXITS:

- Normal: Control is passed to $CGNPS.
- Error: Control is returned to the scheduler via Halt/Syslog or EOJ transients. (See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.)

WORKING STORAGE BLOCK: Contains all working storage and data constants that are required by $CGNPE and are not in the communications area.

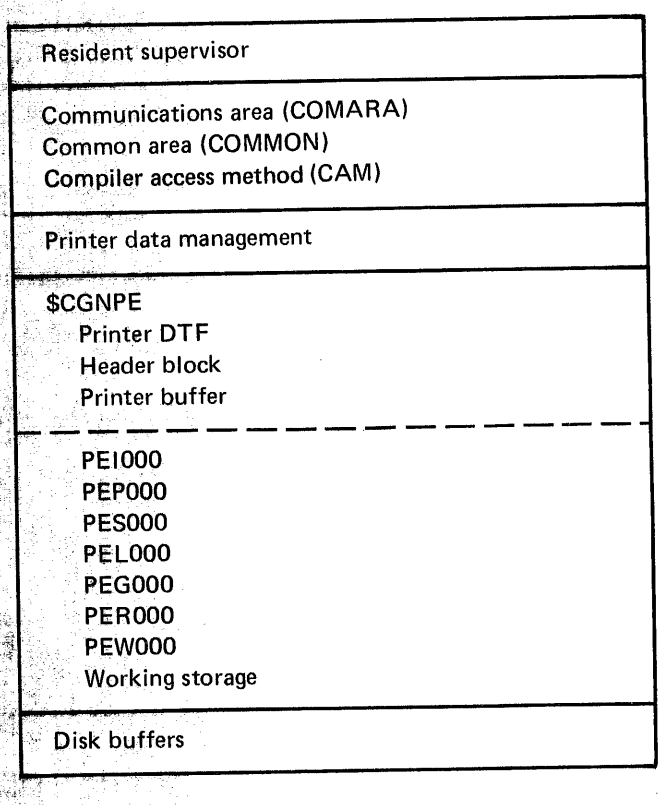


Figure 3-7. Main Storage Map of $CGNPE

Source/Object Output Phase ($CGNPS)

ENTRY POINT: PSI000 – entered from $CGNPE via Fetch. (See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.)

MAIN STORAGE MAP: Figure 3-8.

CHARTS: GA – GI

FUNCTIONS:

- Initializes $CGNPS for processing (PSI000 – Chart GA).
- Controls processing of intermediate text file until end of data or end of file is reached (PSC000 – Chart GB).
- Searches for control records; puts error code in listing if any errors are found (PSK000 – no chart).
- Performs functions associated with termination of phase (PST000 – Chart GC).
- Puts object code in $WORK (PSH000 – Chart GD).
- Converts binary data to hexadecimal representation (PSX000 – no chart).
- Evaluates the expressions in the operand field of the current source record (PSE000 – no chart, but the same as SBV000 in $CGNSB – Chart CG).
- Initializes PSZ000 at the start of the phase and for every ISEQ start request (PSY000 – Chart GE).
- Performs all editing of the print image required by the line printers: 132 column printer, 120 column printer, and 96 column printer (PSZ000 – Chart GE).
- Controls page size of printed output (PSP000 – no chart).
- Counts pages (increments page count parameter) and prints current header (PSG000 – no chart).
- Performs binary multiplication operations (PSM000 – no chart).
- Processes machine instructions, translating them into object code (PSS000 – Chart GF).
- Processes Group 1 instructions: DROP, ENTRY, EQU, ORG, START, USING (PSB000 – Chart GG).
- Processes Group 2 instructions: DC, DS, EJECT, END, EXTRN, ISEQ, PRINT, SPACE, TITLE (PSA000 – Chart GH).
- Provides interface for printer data management (PSR000 – no chart, but same as PER000 in $CGNPE – Chart FF).
- Provides interface for disk data management (PSW000 – Chart GI).

INPUT:

- $SOURCE
- Intermediate text in $WORK2.

OUTPUT:

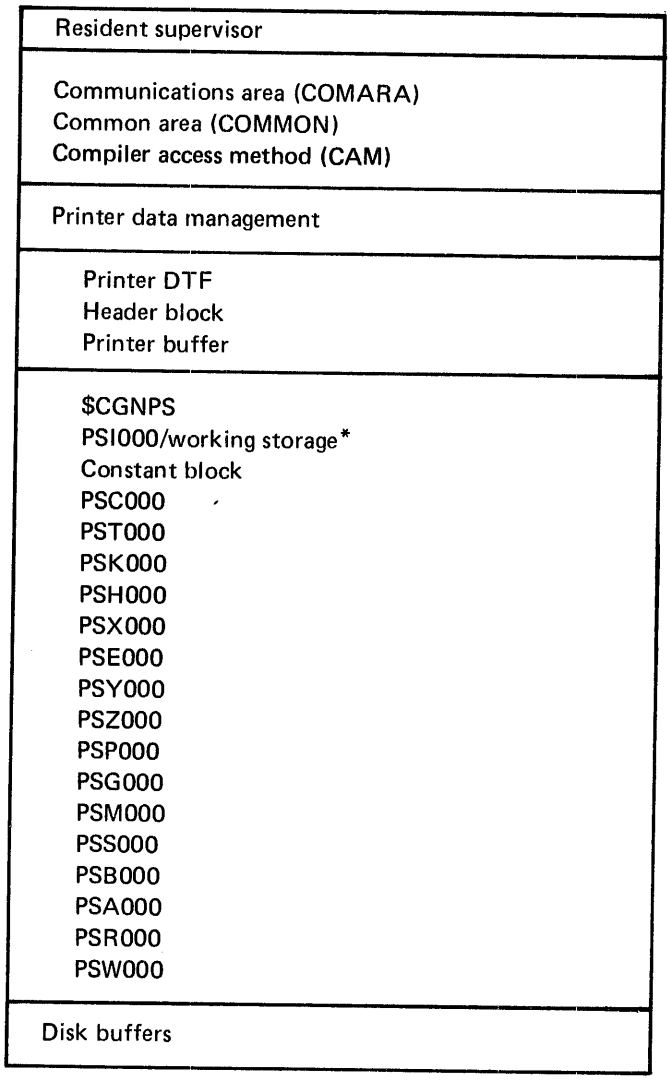
- Printer listing of the source statements and any object code generated by them if LIST is specified.
- Object deck is placed in $WORK.

EXITS:

- Normal: Control is passed to $CGNBX if XREF, to $CGNSX if NOXREF.
- Error: Control is passed to the scheduler via Halt/Syslog or EOJ transients. (See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.)

WORKING STORAGE BLOCK: All required working storage that is not in the communications area resides in this block.

CONSTANT BLOCK: The constant block is loaded into the area immediately following the initialization routine (PSI000); it is moved into COMMON when the phase is initialized.



*PSI000 and its associated constant block occupy the same storage as $CGNPS working storage.

Figure 3-8. Main Storage Map of $CGNPS

Build Cross Reference (XREF) File Phase ($CGNBX)

ENTRY POINT: BXI000 – entered from $CGNPS via Fetch. (See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.)
MAIN STORAGE MAP: Figure 3-9.
CHARTS: HA – HF
FUNCTIONS:
- Initializes phase for processing (BXI000 – Chart HA).
- Controls the building of the XREF sort file (BXP000 – Chart HB).
- Performs functions associated with termination of the phase (BXL000 – Chart HC).
- Moves records from the work area to the XREF file build area (BXM000 – Chart HD).
- When XREF file build area is full, branches to BXS000 for sorting, then branches to BXW100 to write blocks to the XREF sort file (BXM000 – Chart HD).
- Sorts the contents of current block in XREF sort file build area (BXS000 – Chart HE).
- Provides interface for disk data management (BXW000 – Chart HF).

INPUT:
- Intermediate text contained in $WORK2.
- Symbol tables contained in $WORK2.

OUTPUT:
- XREF sort file.
- COMMON fields:

1. XRPIMG, print image area blanked.
2. XRBLKC, count of blocks in sort file.
3. XRPASC, count of passes required to merge the file.
4. XRFRST, relative sector address of first block in file.
5. XRLAST, relative sector address of last block in file.
6. XRAVL1, sort file availability table is cleared.

- Listing header area is initialized for cross reference listing.

EXITS:
- Normal: Control is passed to $CGNSX.
- Error: Control is returned to the scheduler via EOJ. (See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.)

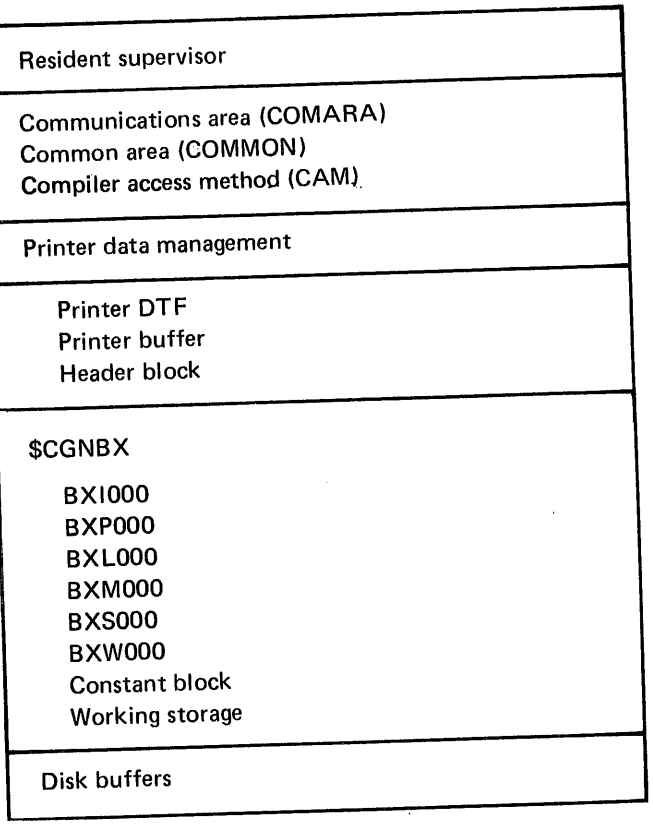


Figure 3-9. Main Storage Map of $CGNBX

Merge and List Cross Reference Phase ($CGNSX)

ENTRY POINT: SXI000 – entered from $CGNBX via Fetch. (See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.)
MAIN STORAGE MAP: Figure 3-10.
CHARTS: IA – IE
FUNCTIONS:
- Initializes phase for processing (SXI000 – Chart IA).
- Controls the merging of the XREF sort file (SXS000 – Chart IB).
- Controls the moving of records from the input areas to the output area (SXO000 – Chart IC).
- Controls the writing and chaining together of output strings and the reading of chained input strings (SXO000 – Chart IC).
- Creates the XREF listing at the last pass of the sort file merge (SXL000 – Chart ID).
- Performs functions associated with termination of the phase (SXT000 – Chart IC).
- Provides the interface for reading to and writing from the sort file (SXW000 – Chart IE).

- Prints the header for the printer listing and counts pages (SXG000 – no chart).
- Controls page size of printed output (SXP000 – no chart).
- Converts binary numbers to decimal (SXV000 – no chart).
- Provides interface for printer data management (SXR000 – no chart).

INPUT:
- XREF file.
- COMMON parameters:

1. XRPIMG, print image area blanked.
2. XRBLKC, count of blocks in file.
3. XRPASR, number of passes required to merge file.
4. XRFRST, C/S address of first block in file.
5. XRLAST, C/S address of last block in file.

- Listing header in block.

OUTPUT:
- Cross reference listing.
- Error summary statements of error counts.

EXITS:
- Normal:

1. If object output exists, control is passed to $OLYNX of the Overlay Linkage Editor. (See *IBM System/3 Overlay Linkage Editor and Checkpoint/Restart Programs Logic Manual*, SY21-0530.)
2. If no object output exists, control is returned to the scheduler via EOJ. (See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.)

WORKING STORAGE BLOCK: All required working storage not in COMMON resides in this block.

CONSTANT BLOCK: All required constants that are not in COMMON reside in this block.

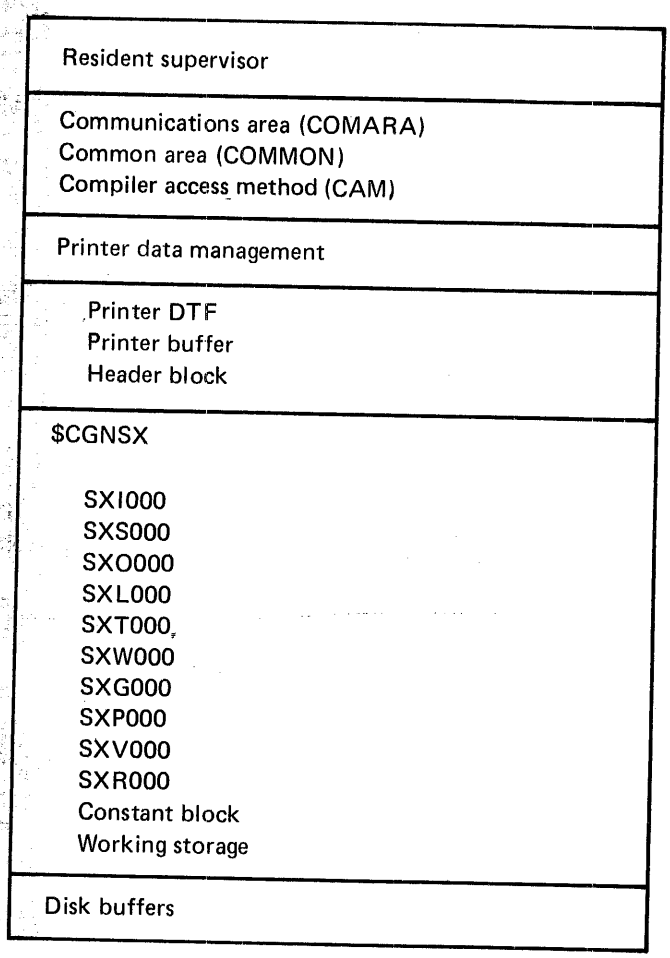


Figure 3-10. Main Storage Map of $CGNSX

Compiler Access Method (CAM)

ENTRY POINT: CAM001 – can be accessed through COMARA by any phase (CAM resides in $CGNIN).

CHART: JA

FUNCTIONS:
- Retrieves up to 255 sectors at a time according to a library relative sector number.
- Loads up to 255 sectors at a time according to a binary relative sector number.

INPUT: $SOURCE and $WORK2.

OUTPUT: $WORK and $WORK2.

EXIT: Returns to caller.

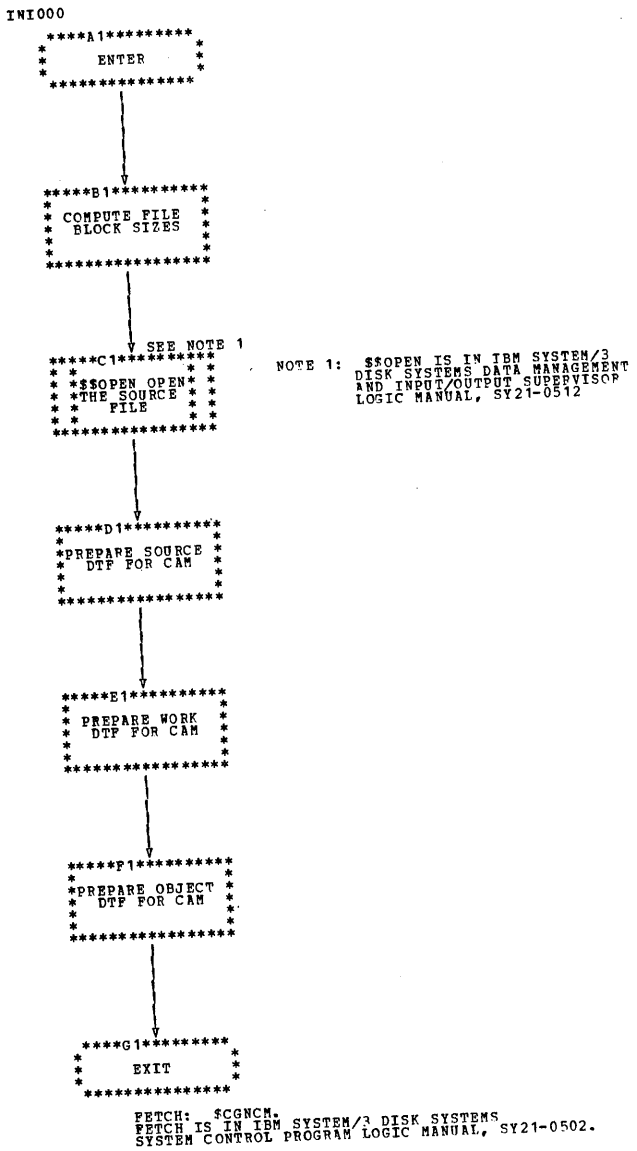


Chart BA. $CGNIN Initialization Routine (INI000)

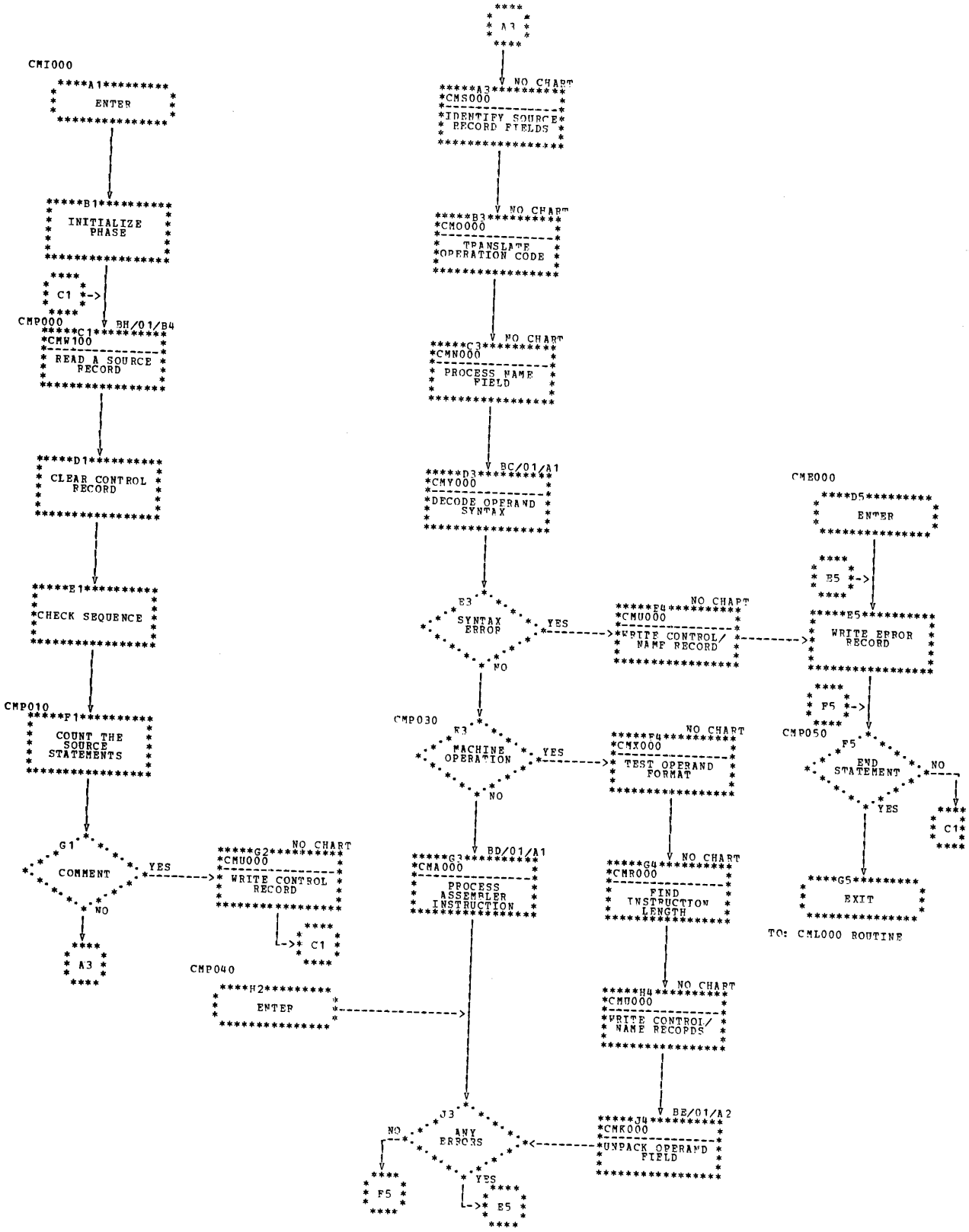


Chart BB. $CGNCM Initialization Routine (CMI000) and Main Control Routine (CMP000)

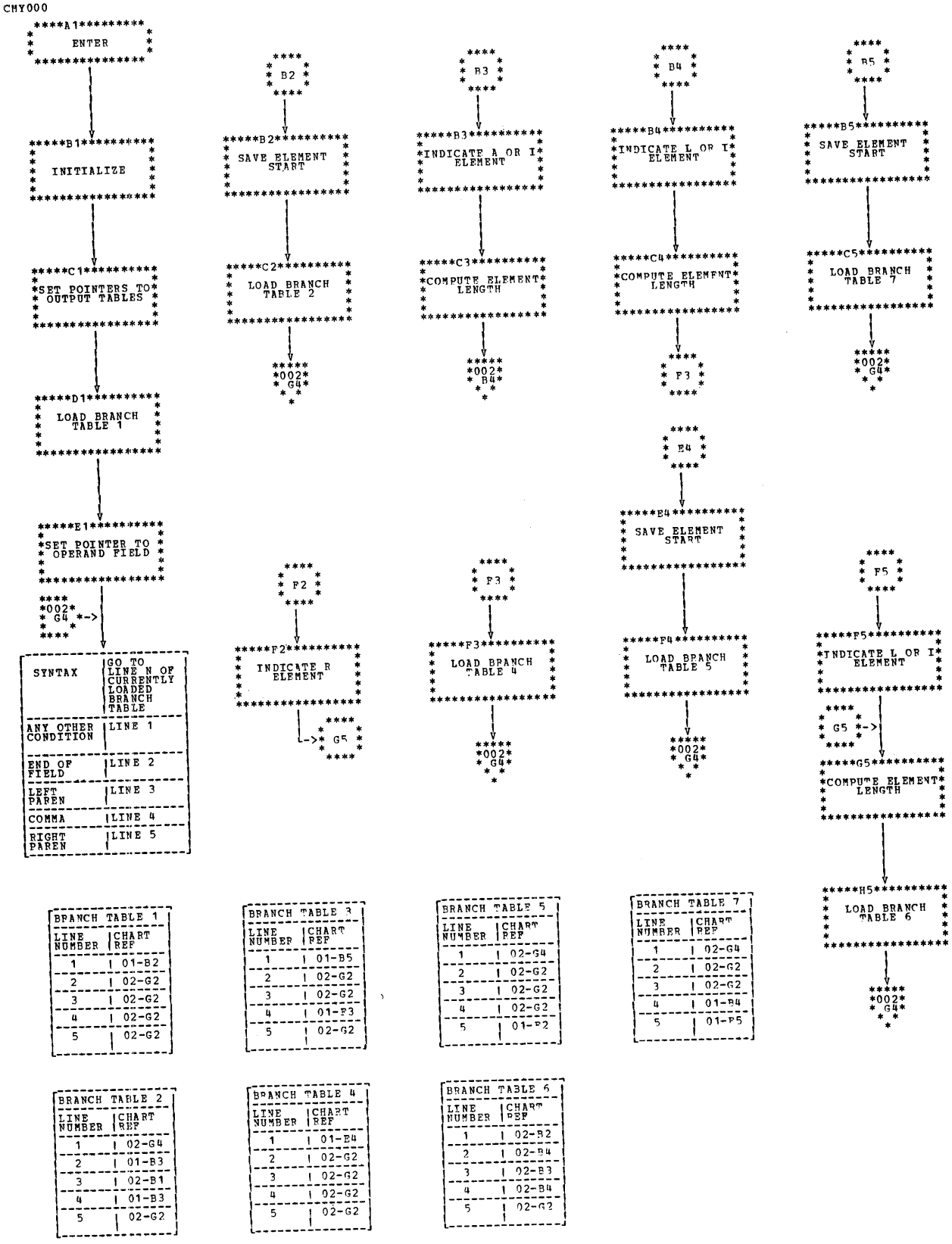


Chart BC (Part 1 of 2). $CGNCM Operand Syntax Checking Routine (CMY000)

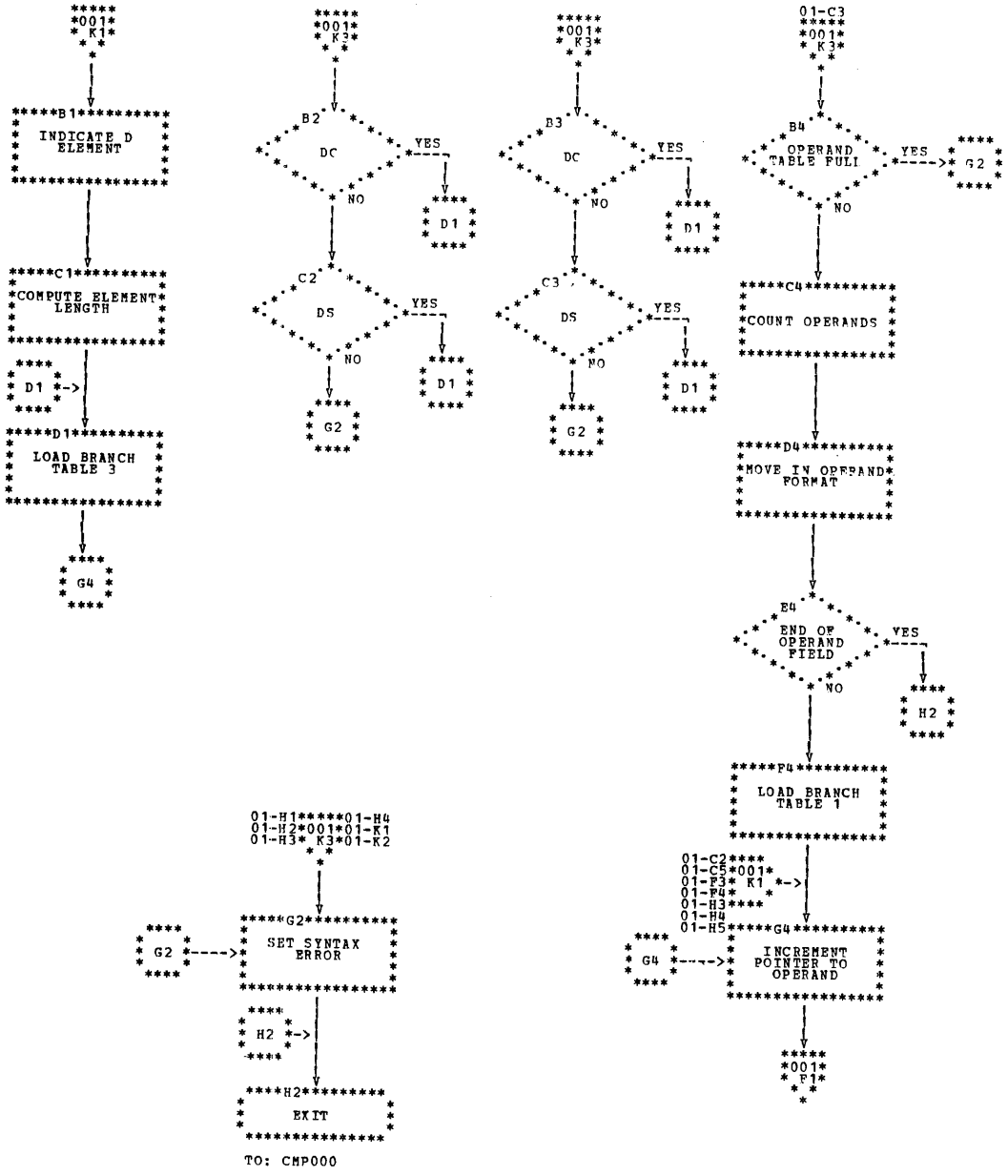


Chart BC (Part 2 of 2). $CGNCM Operand Syntax Checking Routine (CMY000)

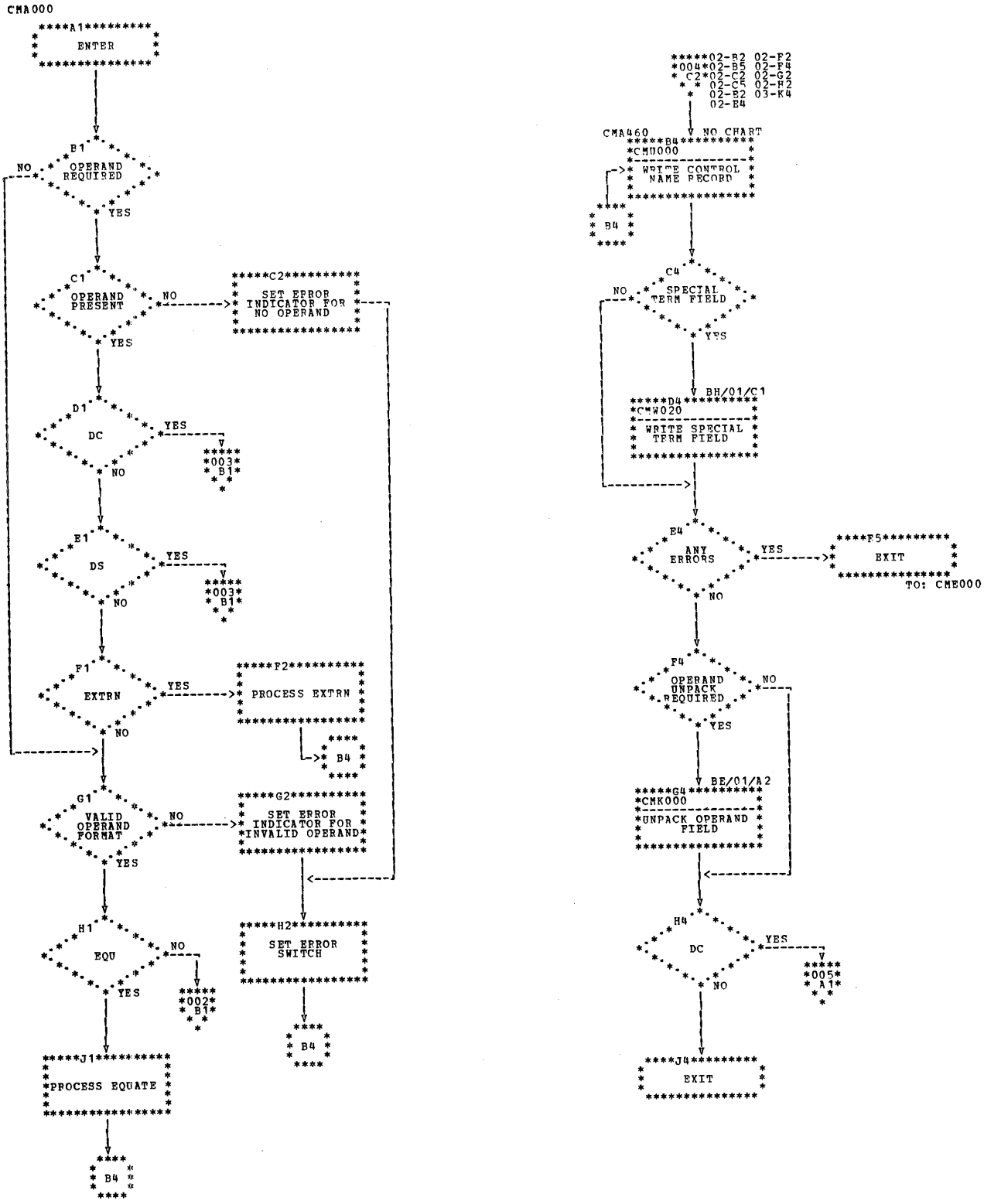


Chart BD (Part 1 of 5). $CGNCM Assembler Instruction Processing Routine (CMA000)

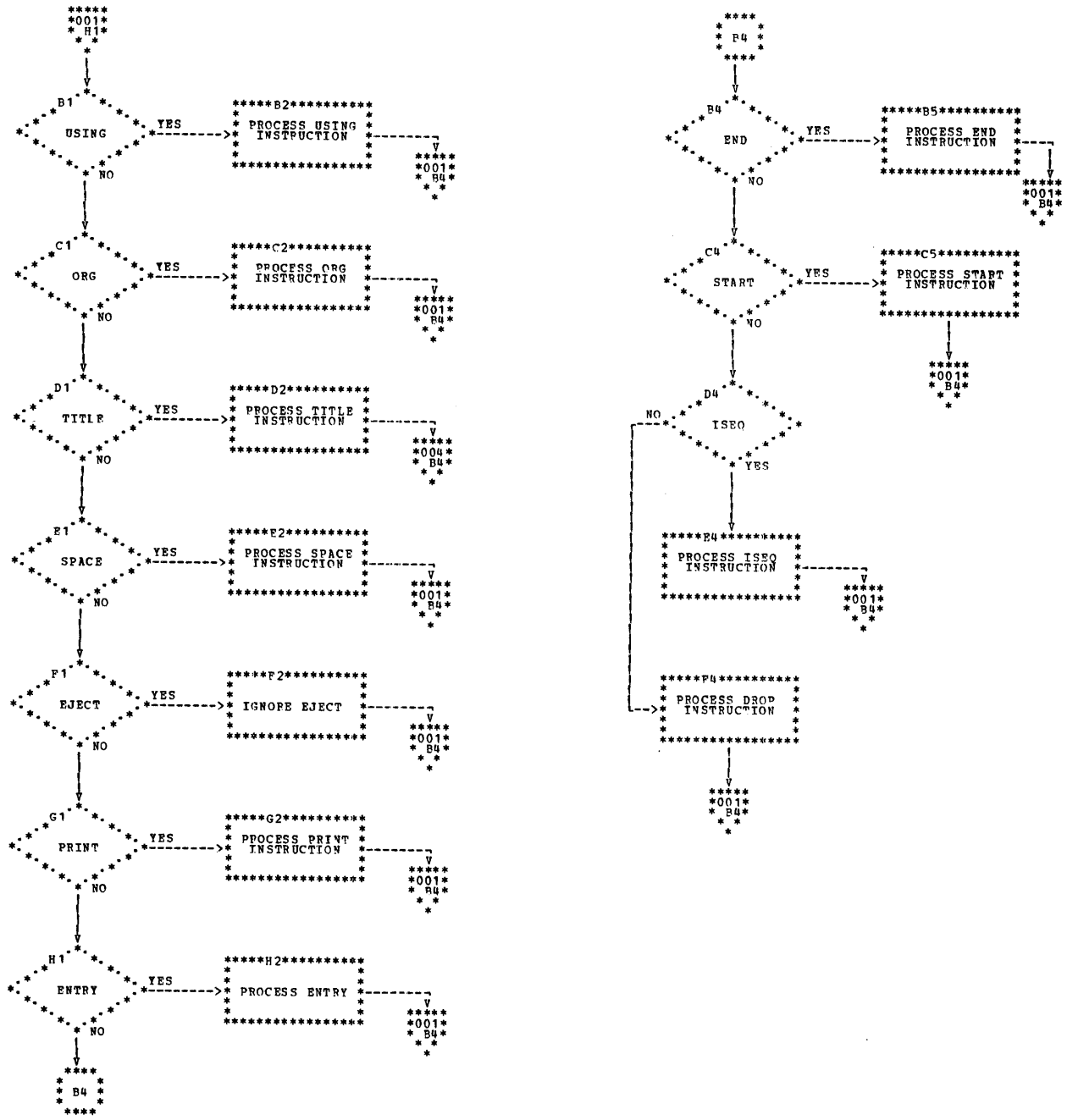


Chart BD (Part 2 of 5). $CGNCM Assembler Instruction Processing Routine (CMA000)

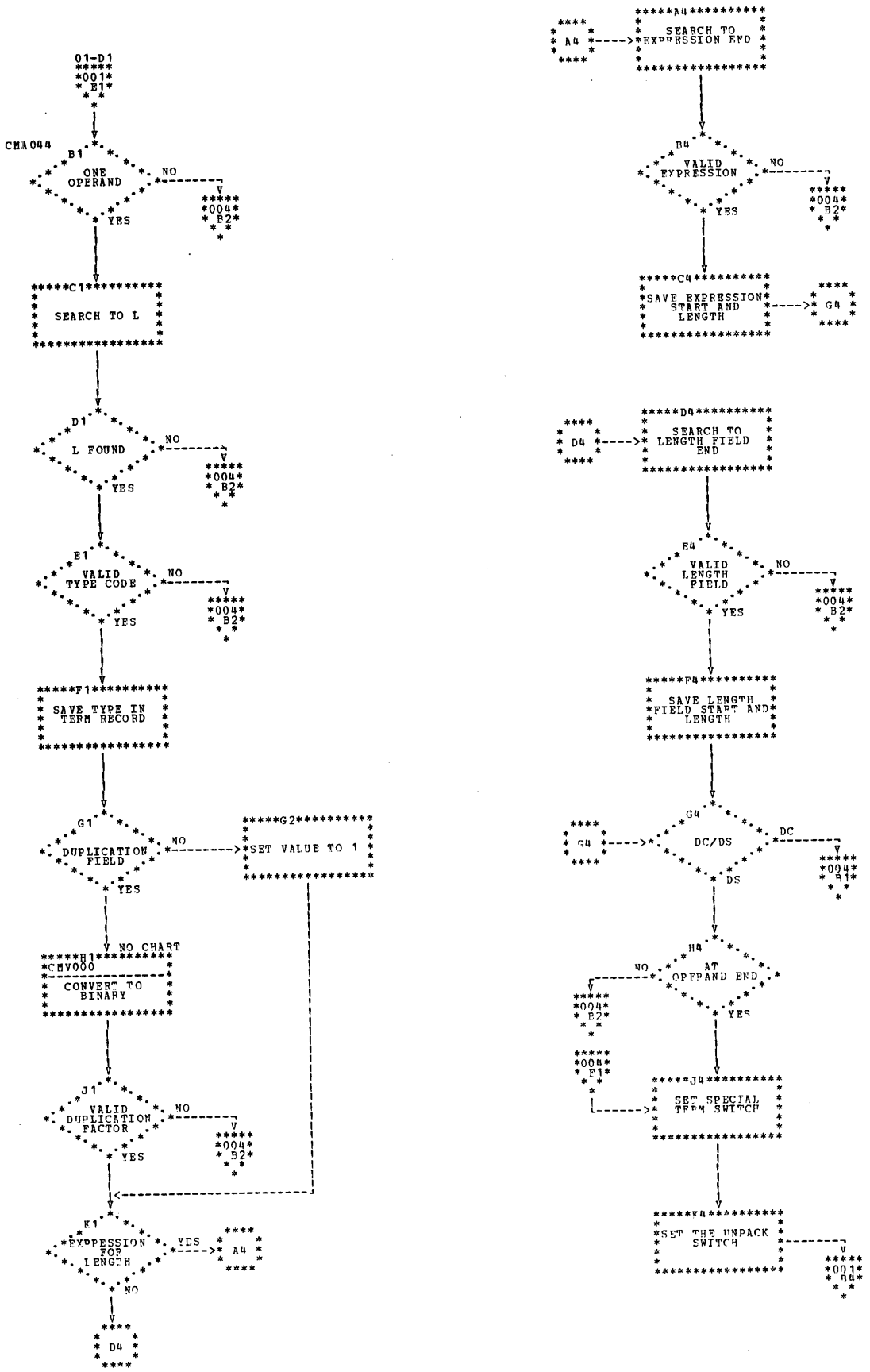


Chart BD (Part 3 of 5). $CGNCM Assembler Instruction Processing Routine (CMA000)

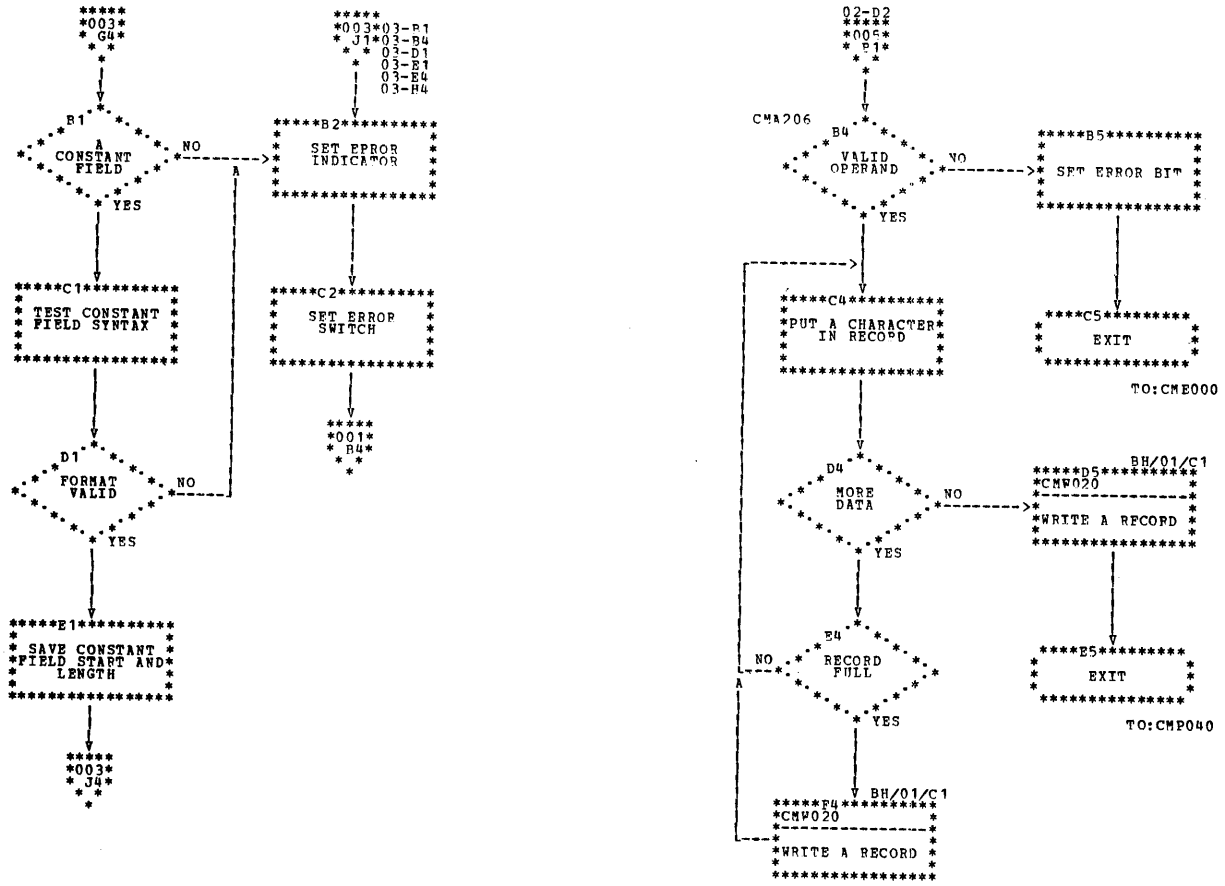


Chart BD (Part 4 of 5). $CGNCM Assembler Instruction Processing Routine (CMA000)

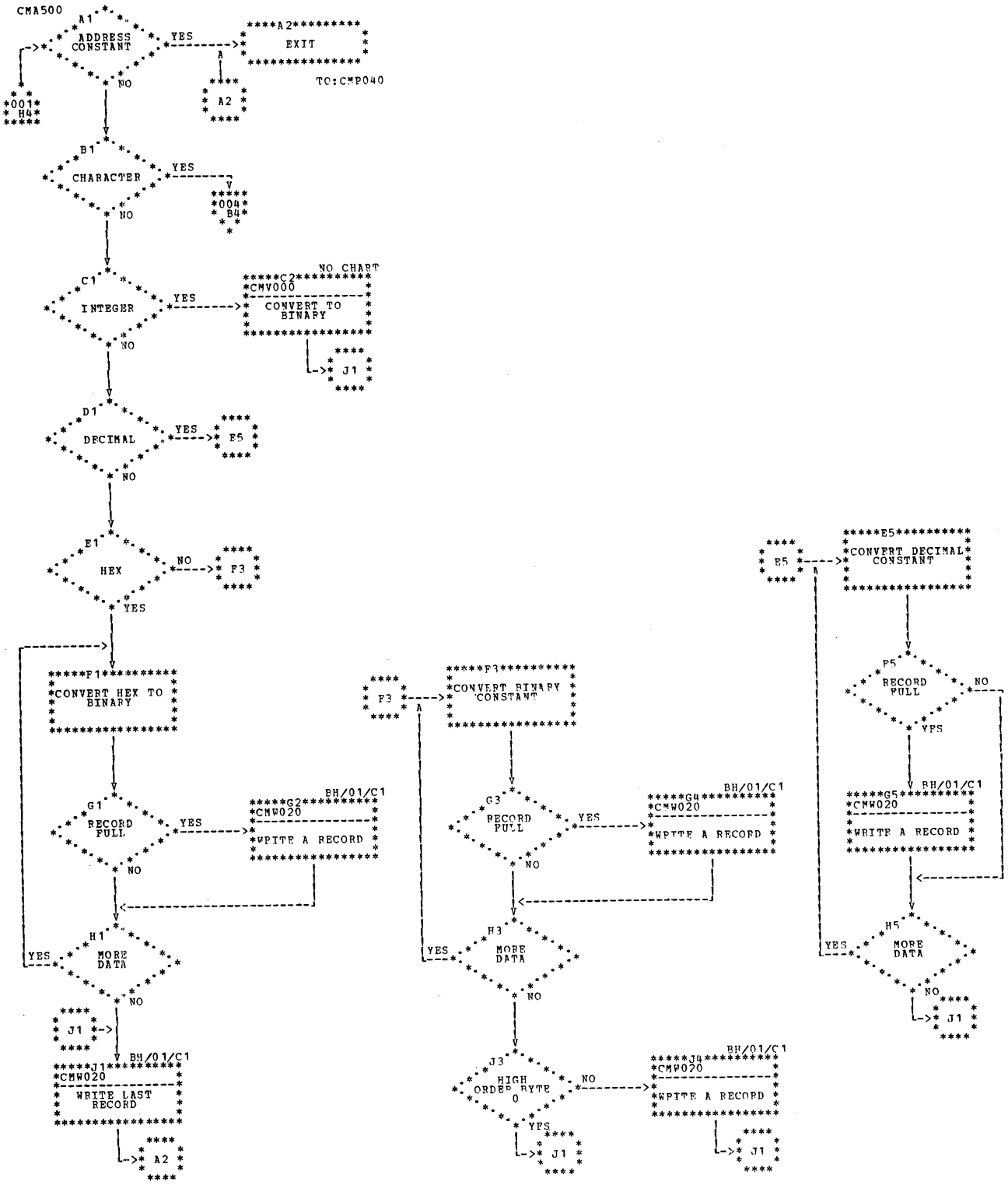


Chart BD (Part 5 of 5). $CGNCM Assembler Instruction Processing Routine (CMA000)

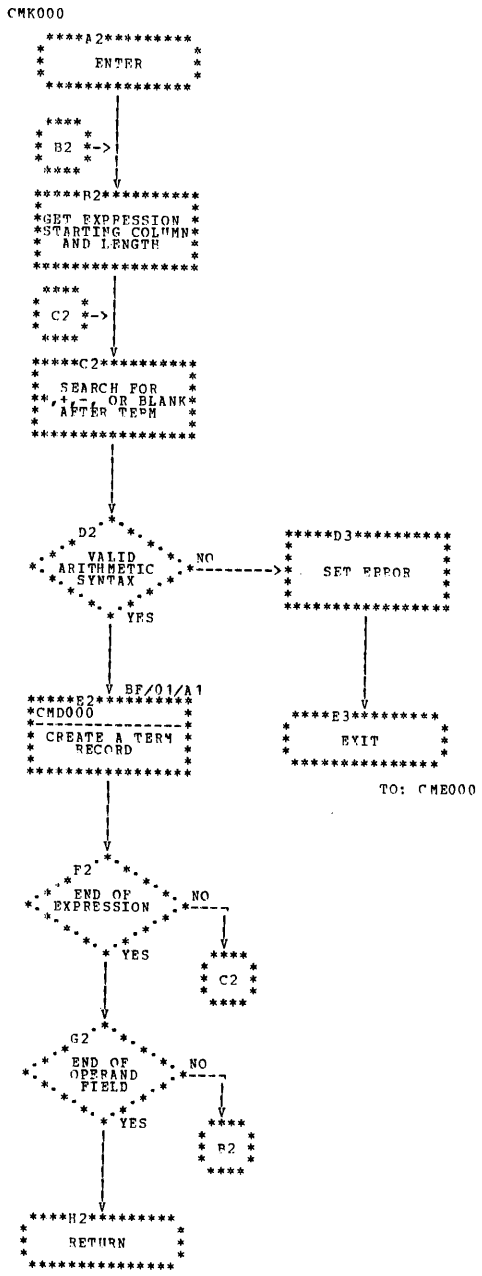


Chart BE. $CGNCM Prepare Operand for Unpack Routine (CMK000)

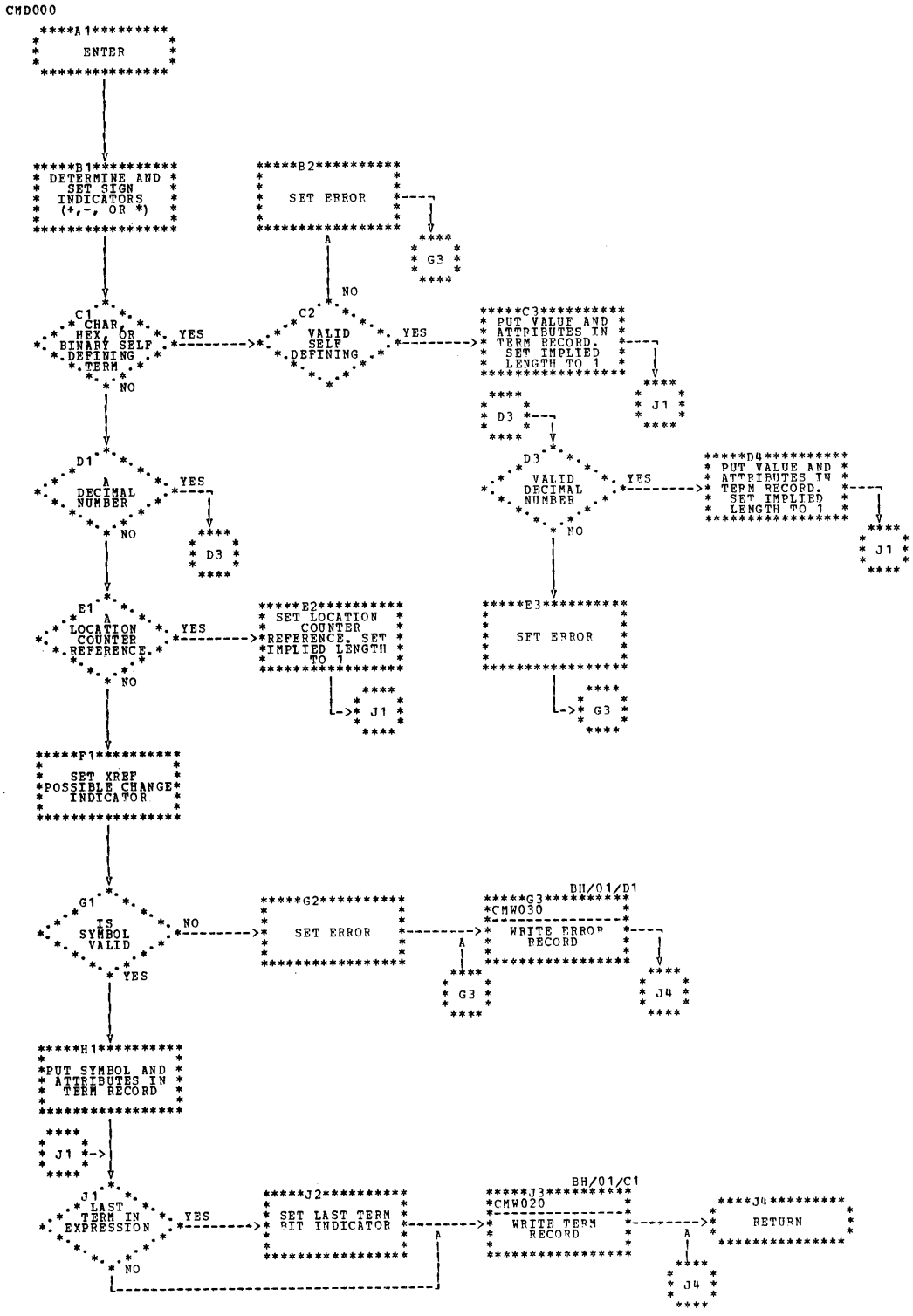
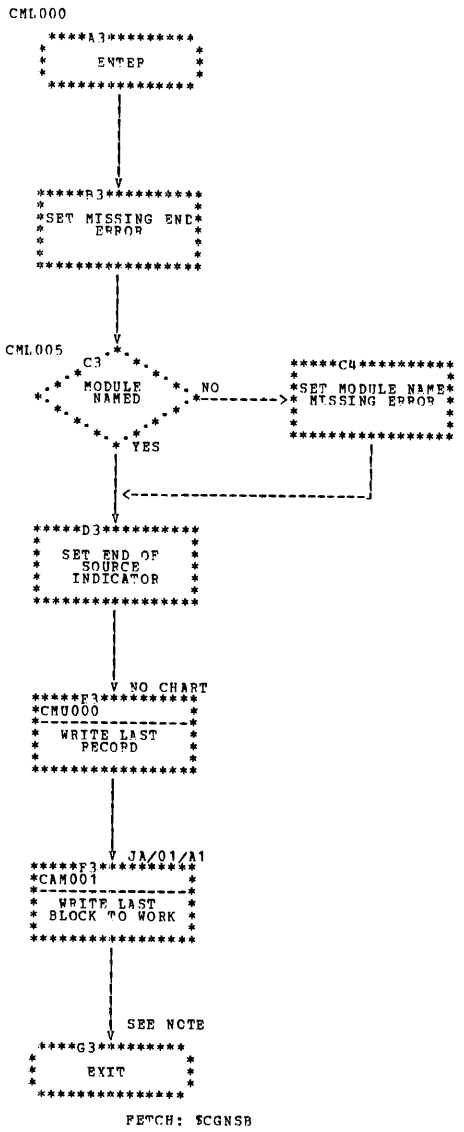


Chart BF. $CGNCM Operand Unpack Routine (CMD000)



NOTE : FETCH IS IN IBM SYSTEM/3 DISK SYSTEMS SYSTEM CONTROL PROGRAM LOGIC MANUAL, SY21-0502

Chart BG. $CGNCM Termination Routine (CML000)

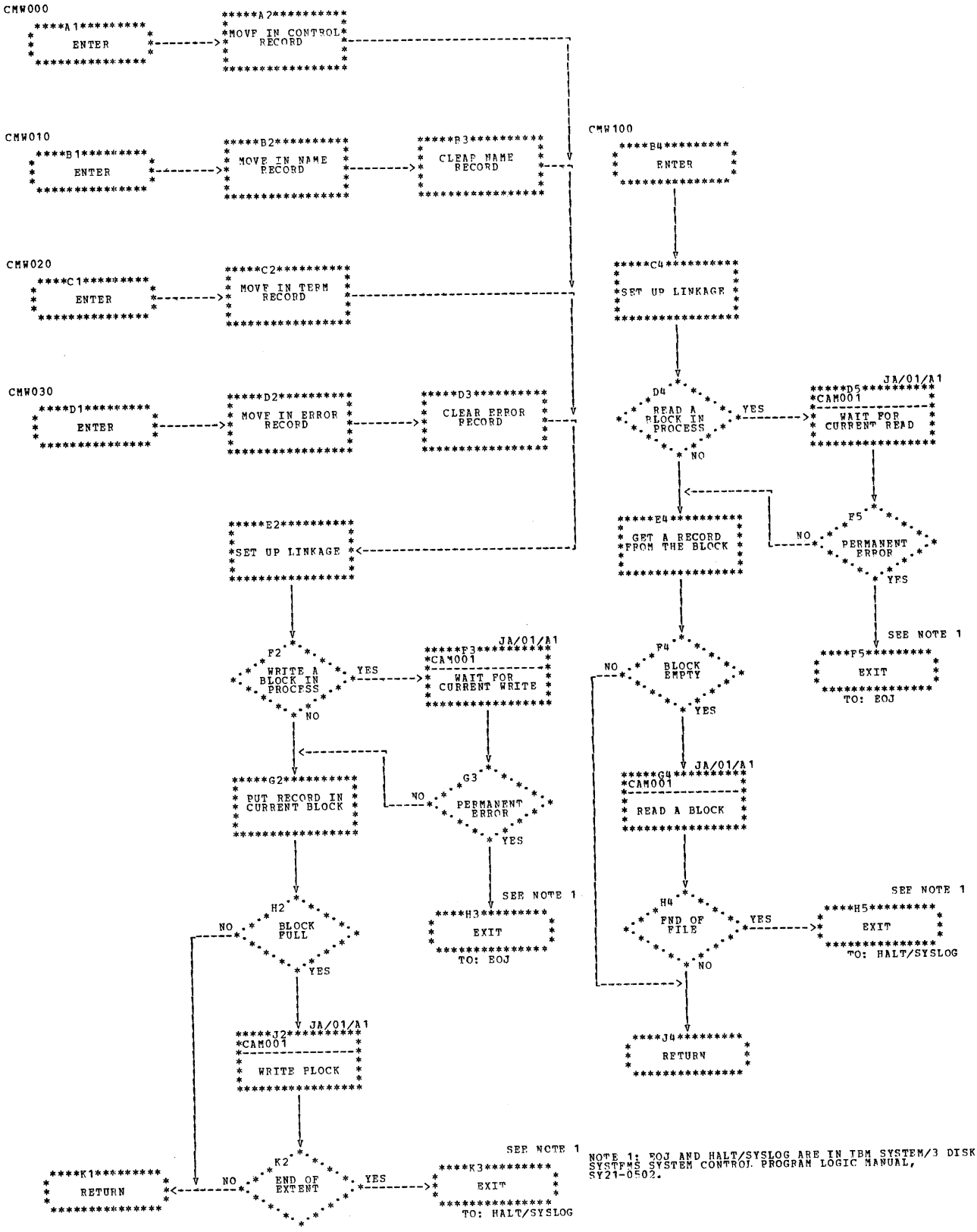


Chart BH. $CGNCM Disk Data Management Interfaces (CMW000)

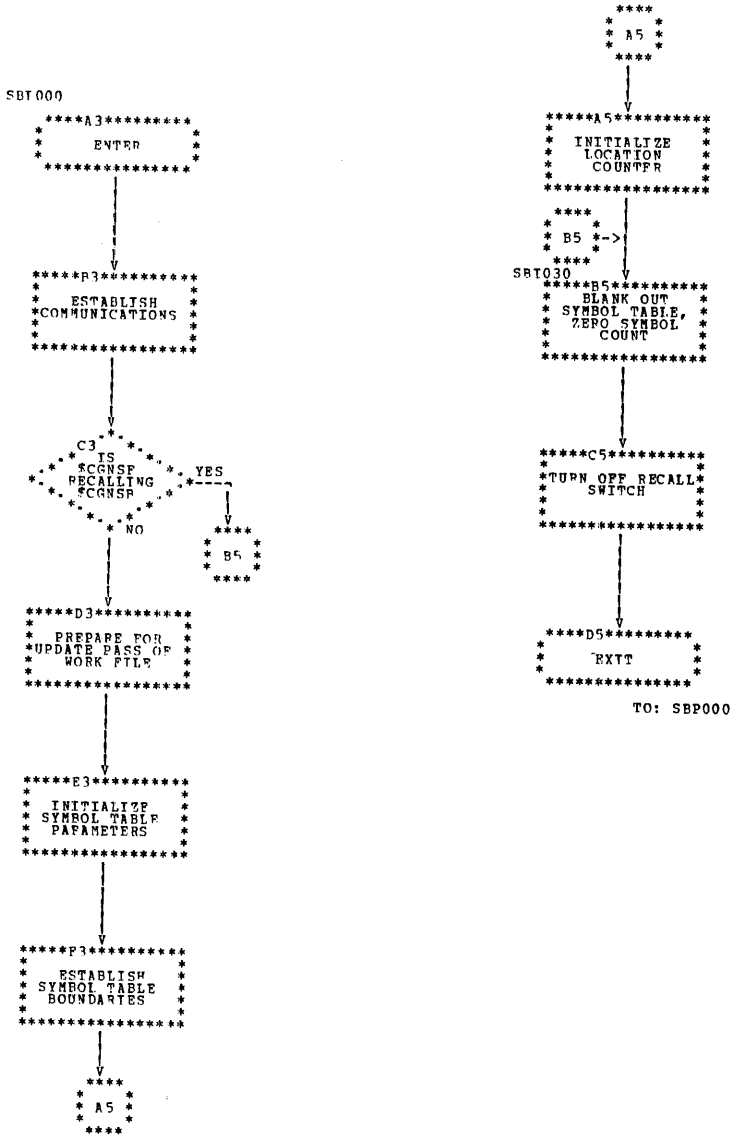
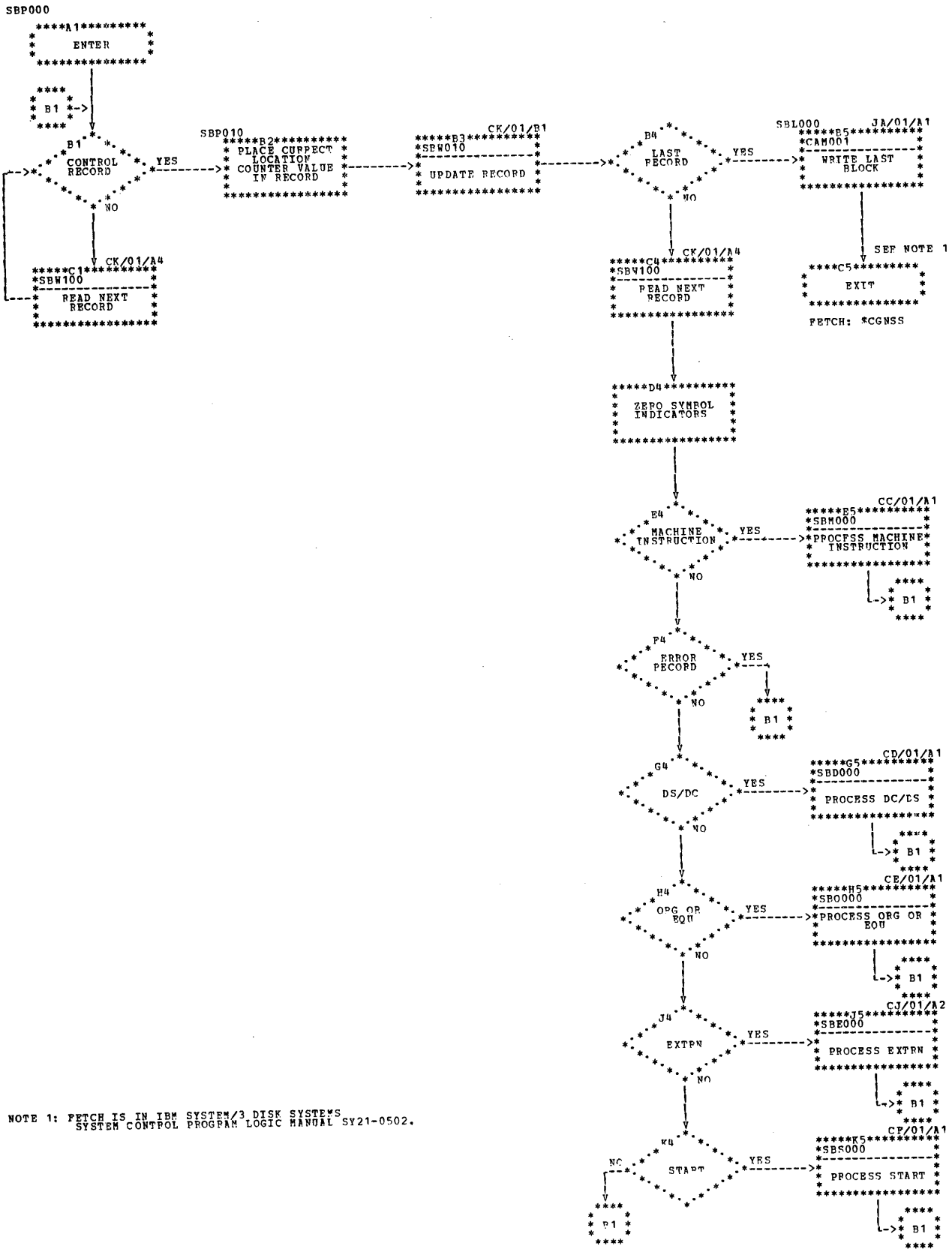


Chart CA. $CGNSB Initialization Routine (SBI000)



NOTE 1: FETCH IS IN IBM SYSTEM/3 DISK SYSTEMS SYSTEM CONTROL PROGRAM LOGIC MANUAL SY21-0502.

Chart CB. $CGNSB Main Control Routine (SBP000)

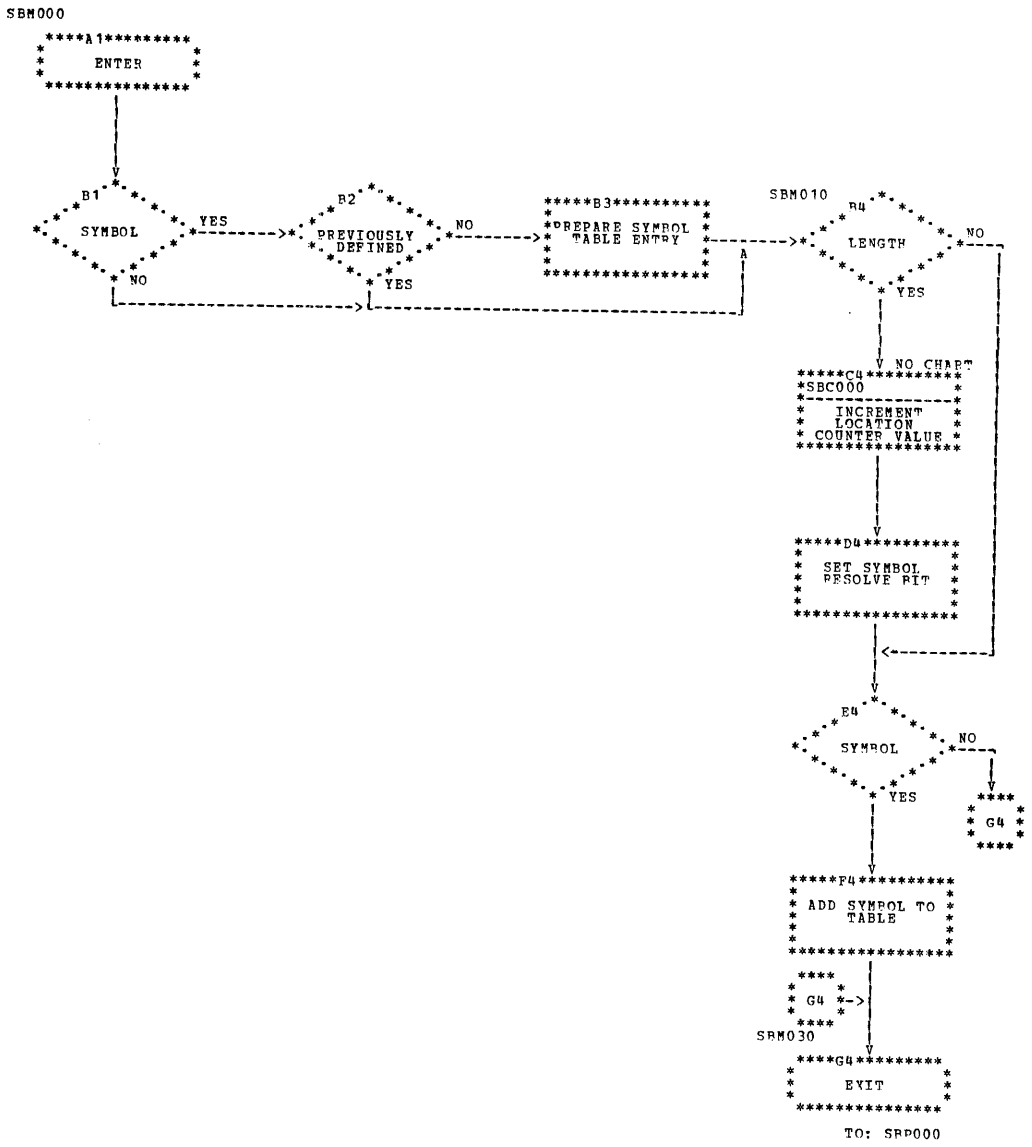


Chart CC. $CGNSB Machine Instruction Processing Routine (SBM000)

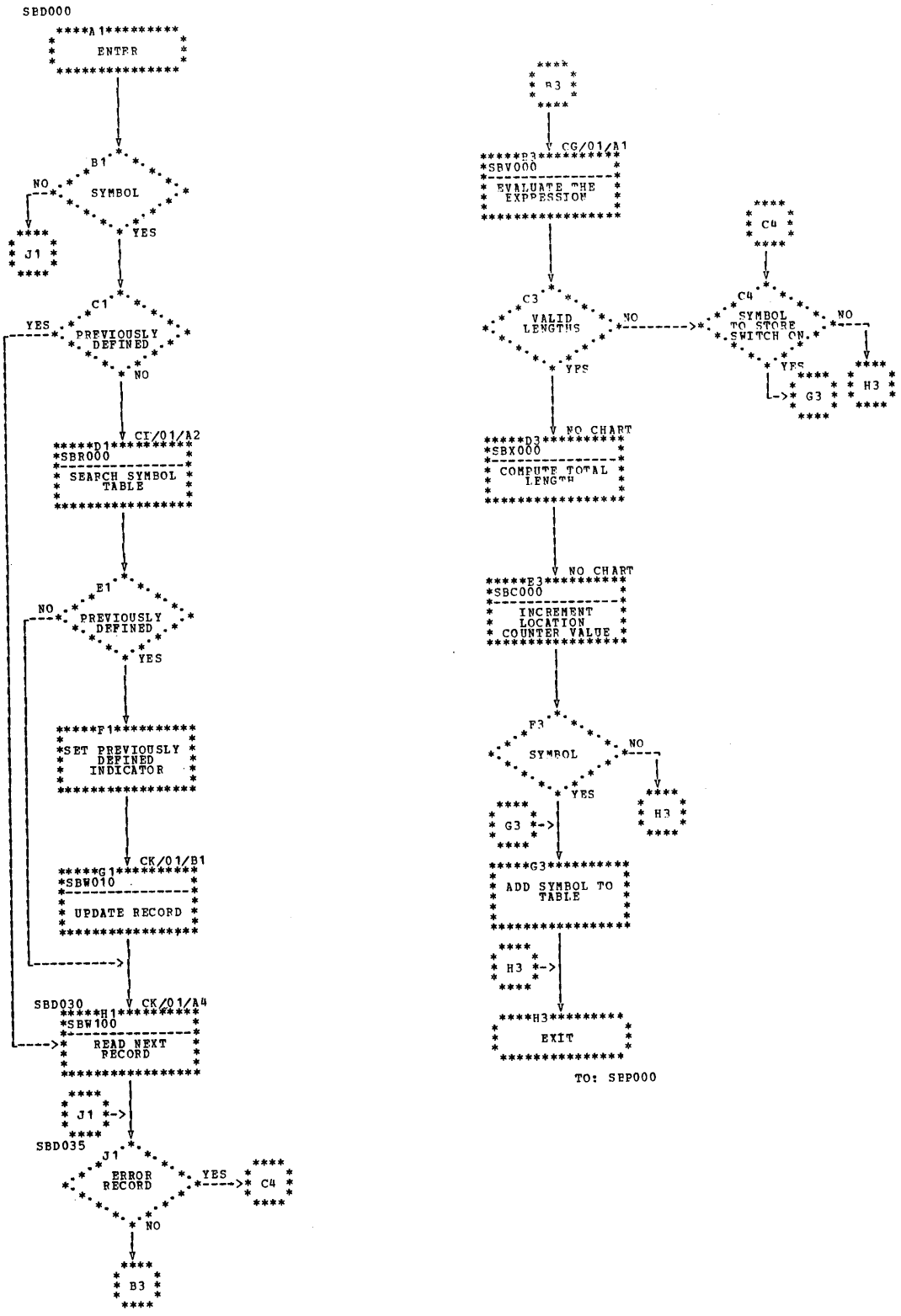


Chart CD. $CGNSB DC/DS Processing Routine (SBD000)

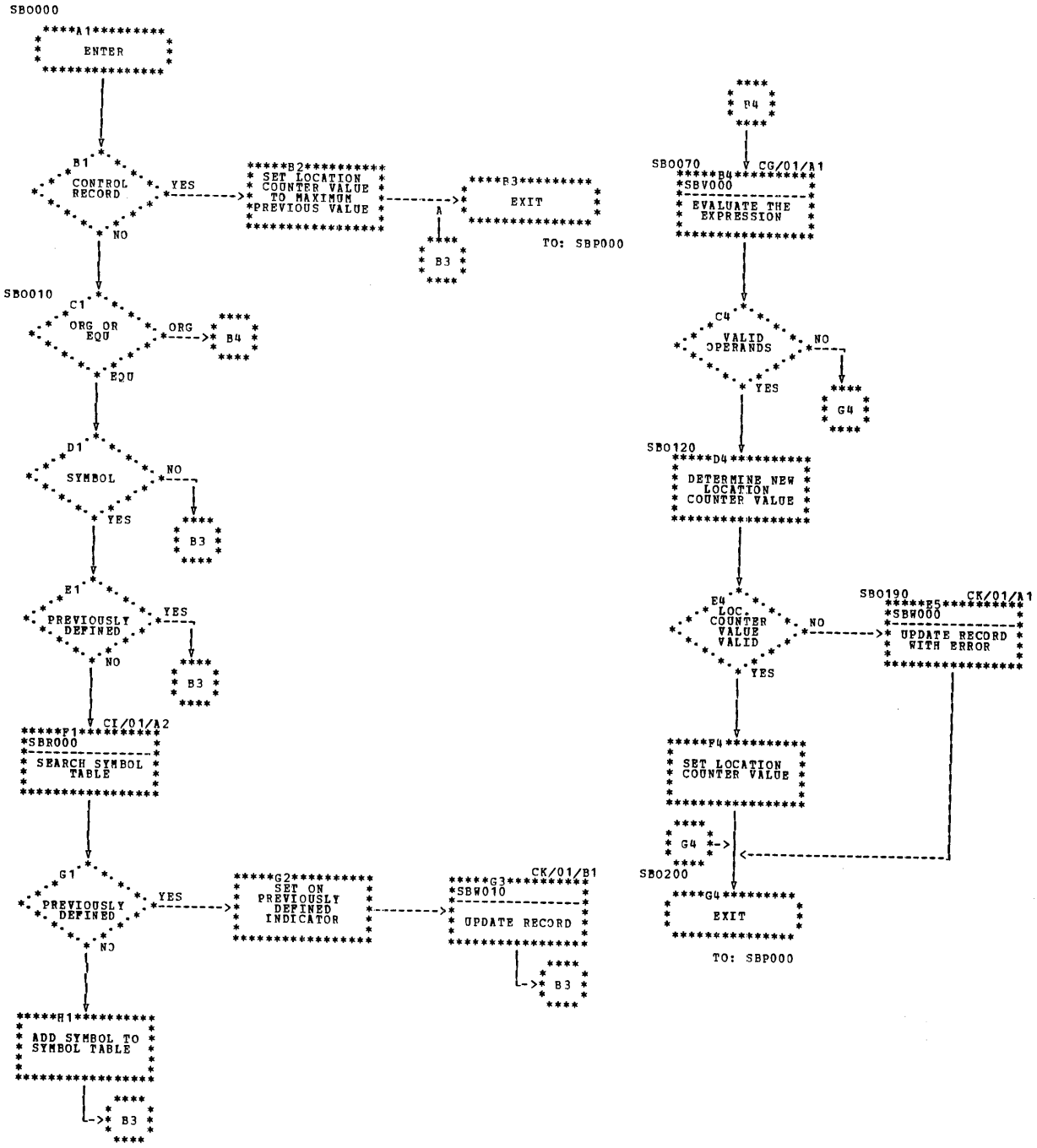


Chart CE. $CGNSB ORG/EQU Processing Routine (SBO000)

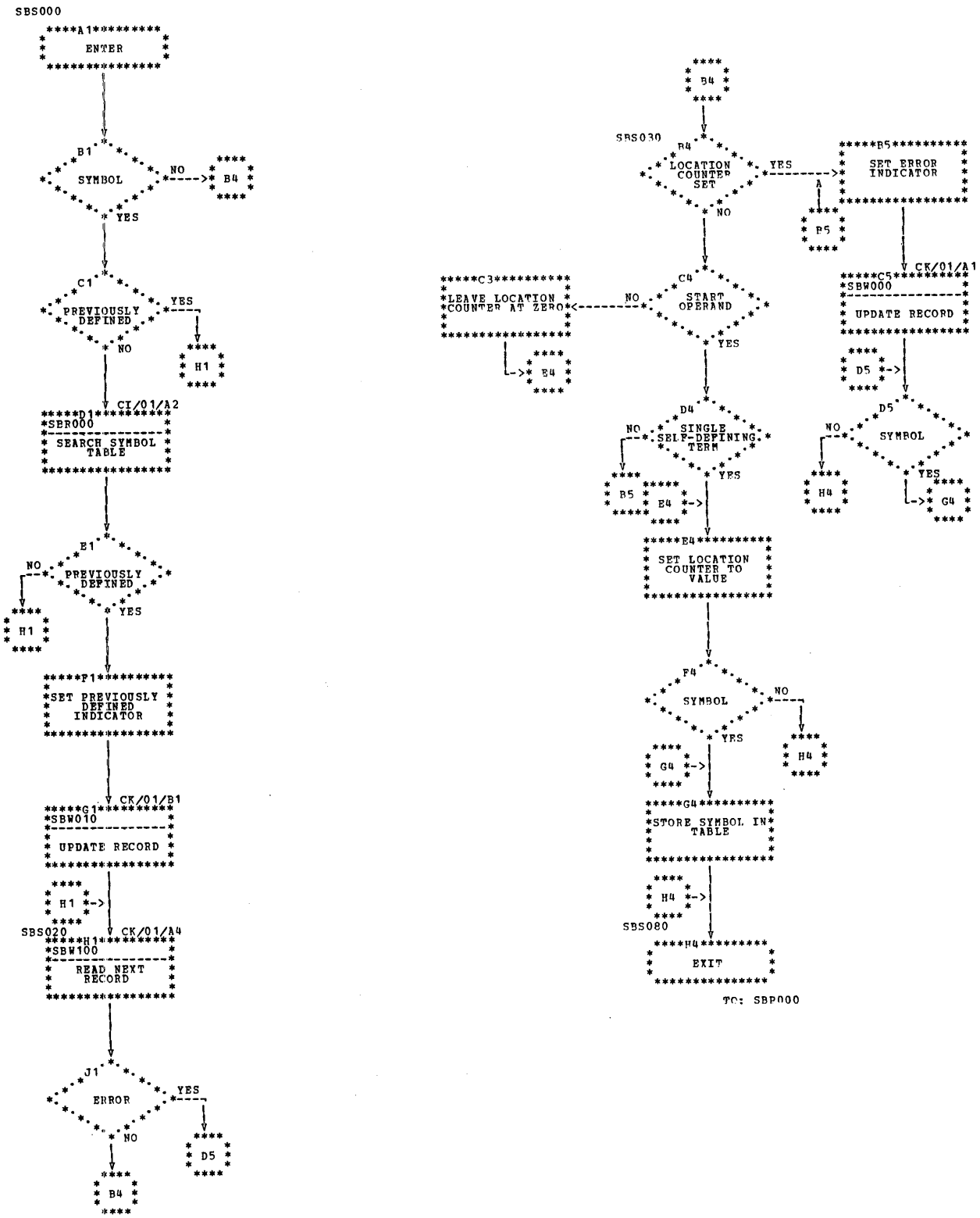


Chart CF. $CGNSB START Processing Routine (SBS000)

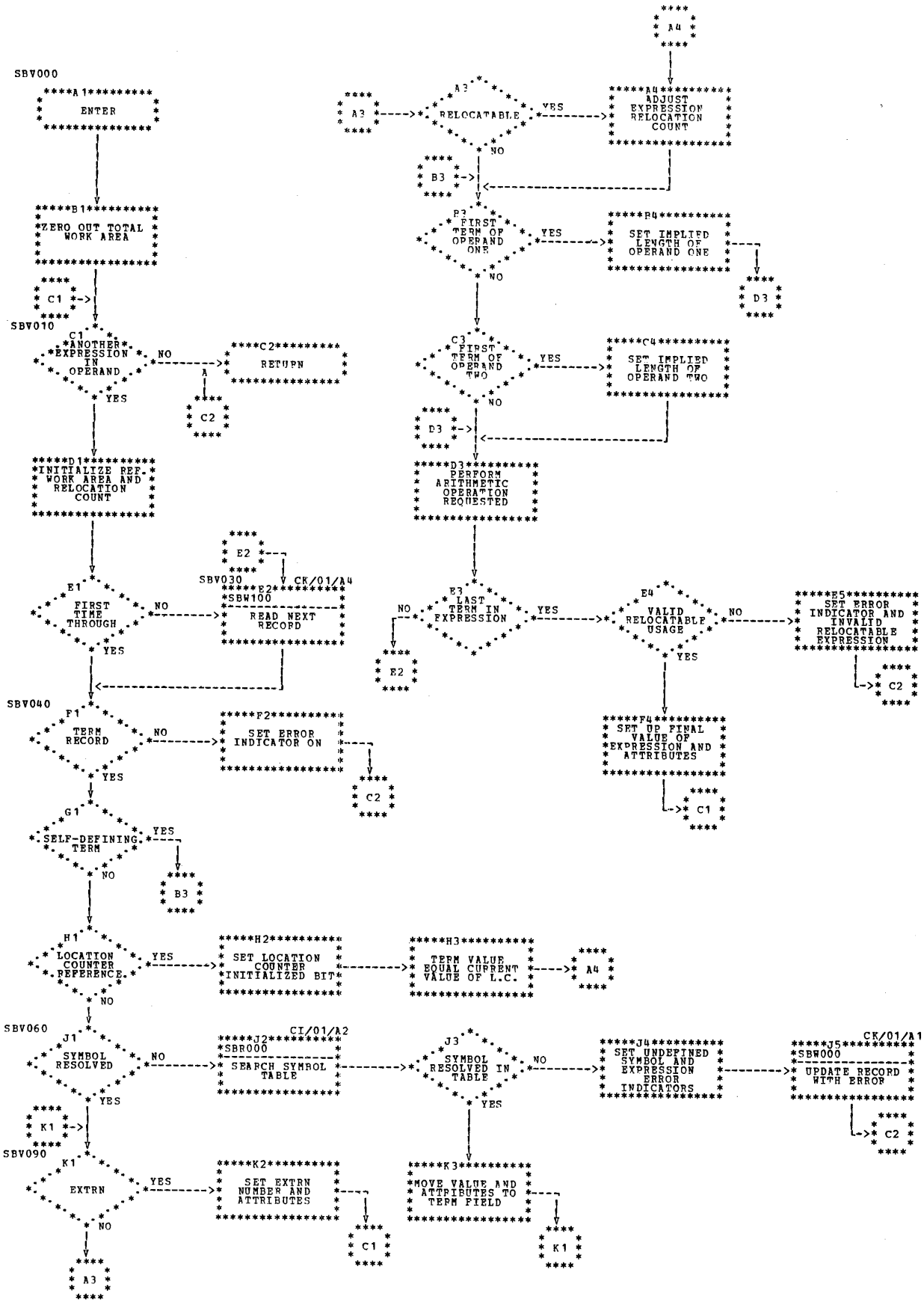


Chart CG. $CGNSB Expression Evaluation Routine (SBV000)

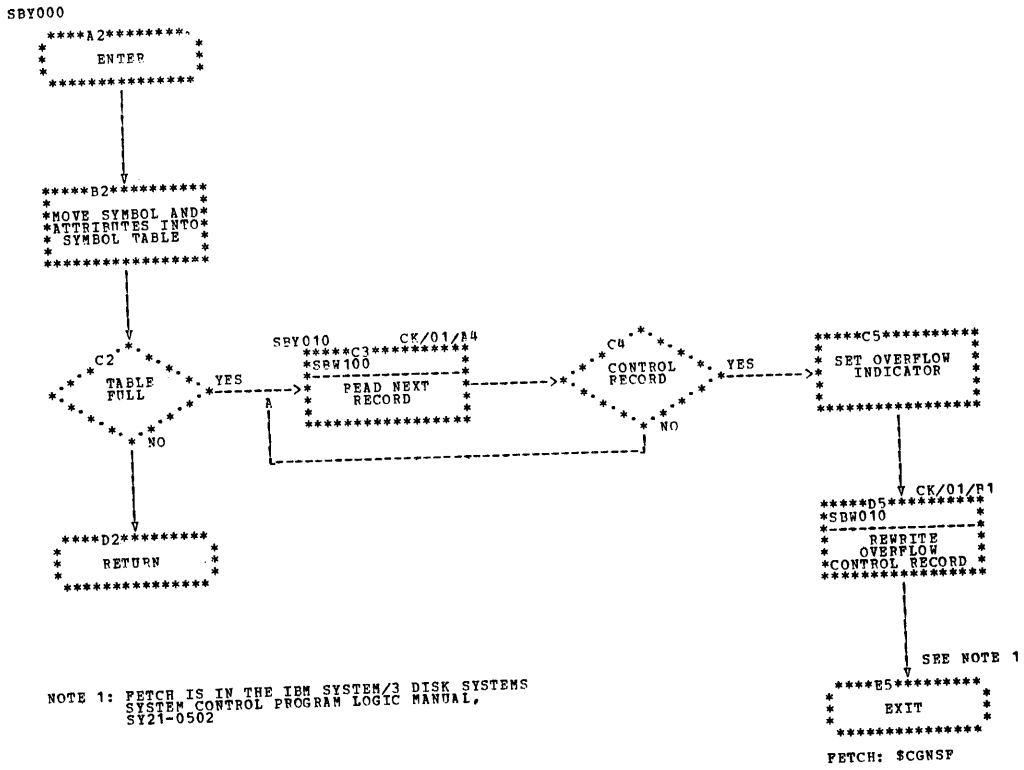


Chart CH. $CGNSB Store and Count Symbol Routine (SBY000)

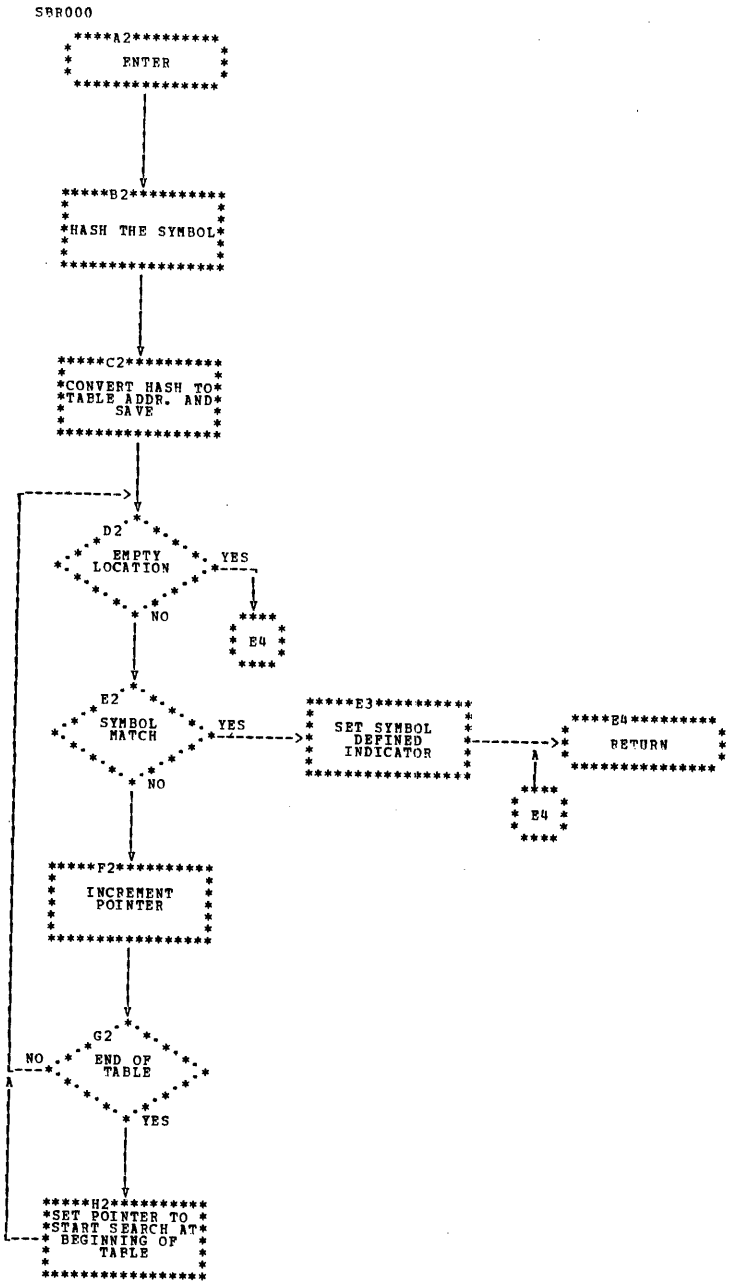


Chart CI. $CGNSB Search Symbol Table Routine (SBR000)

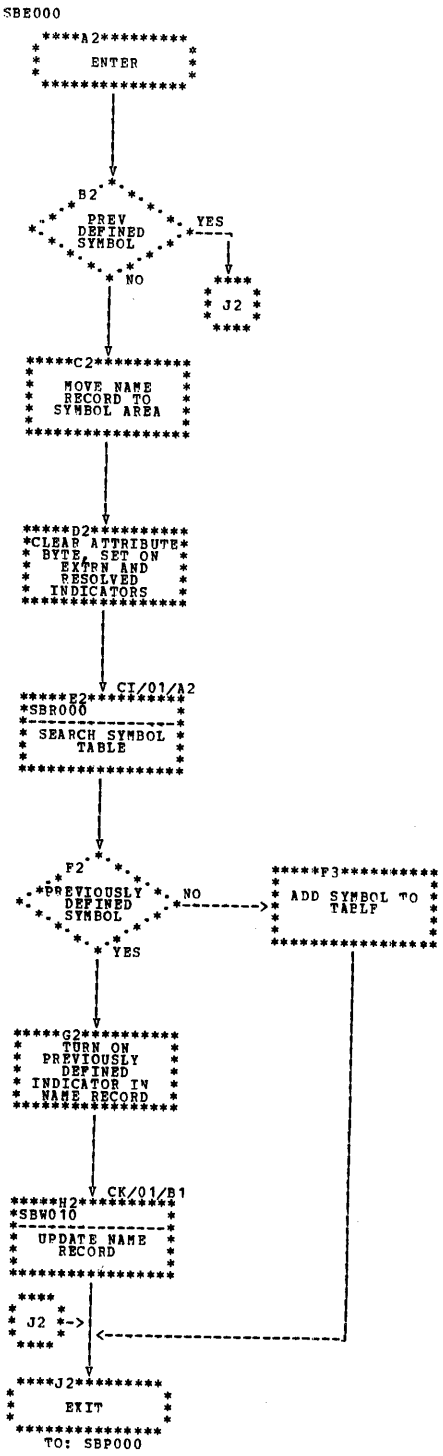


Chart CJ. $CGNSB EXTRN Processing Routine (SBE000)

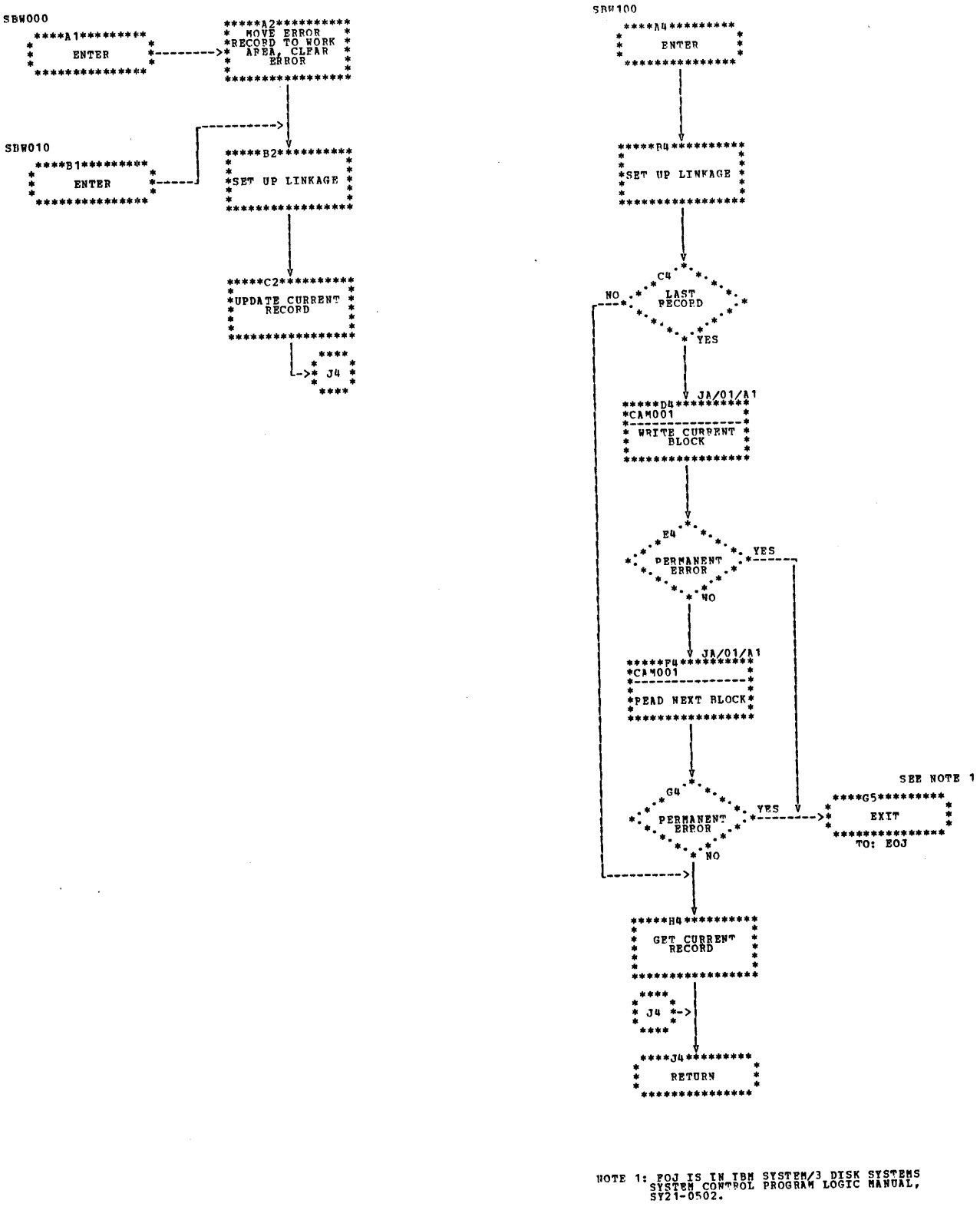
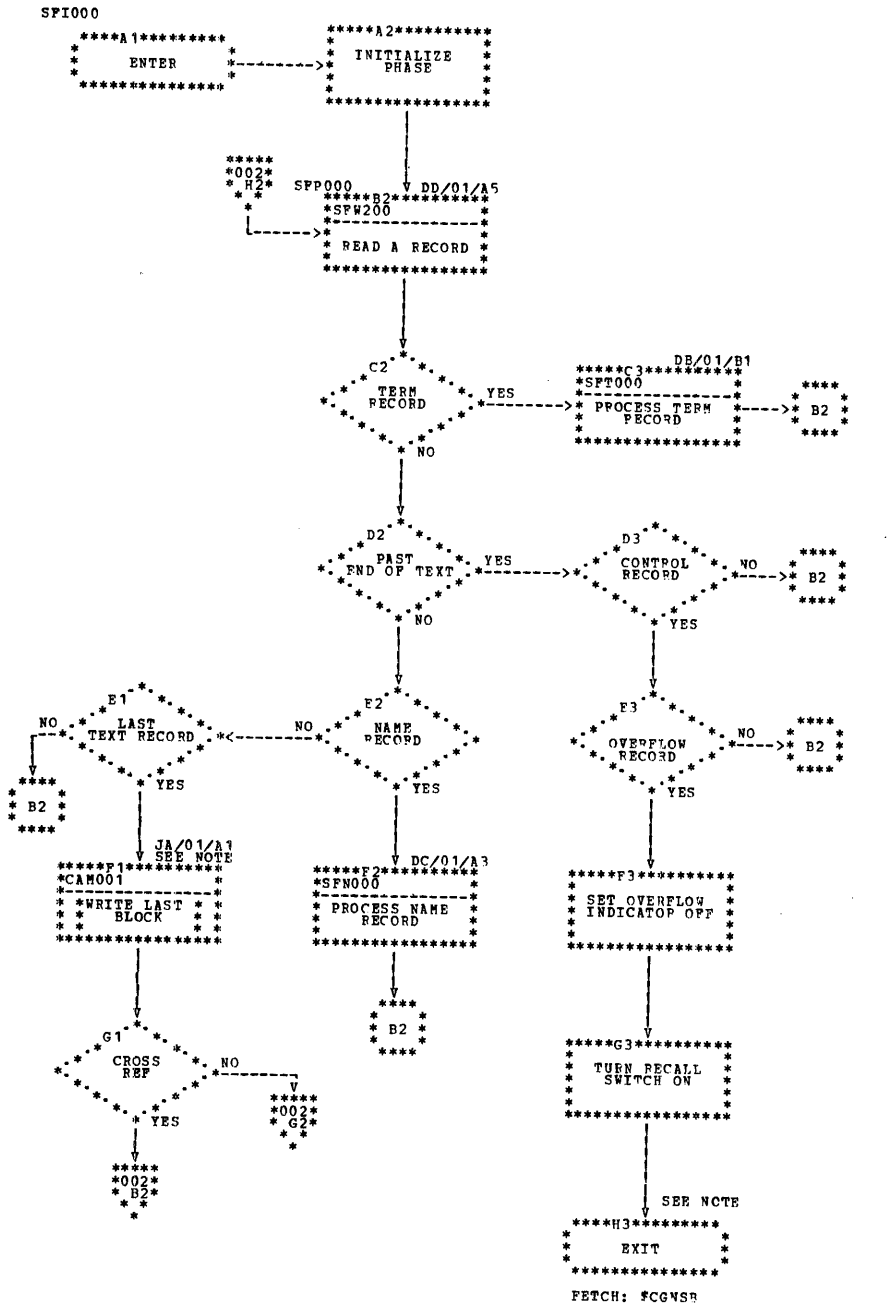


Chart CK. $CGNSB Disk Data Management Interfaces (SBW000)



NOTE : FETCH IS IN IBM SYSTEM/3 DISK SYSTEMS SYSTEM CONTROL PROGRAM LOGIC MANUAL, SY21-0502.

Chart DA (Part 1 of 2). $CGNSF Initialization Routine (SFI000) and Main Control Routine (SFP000)

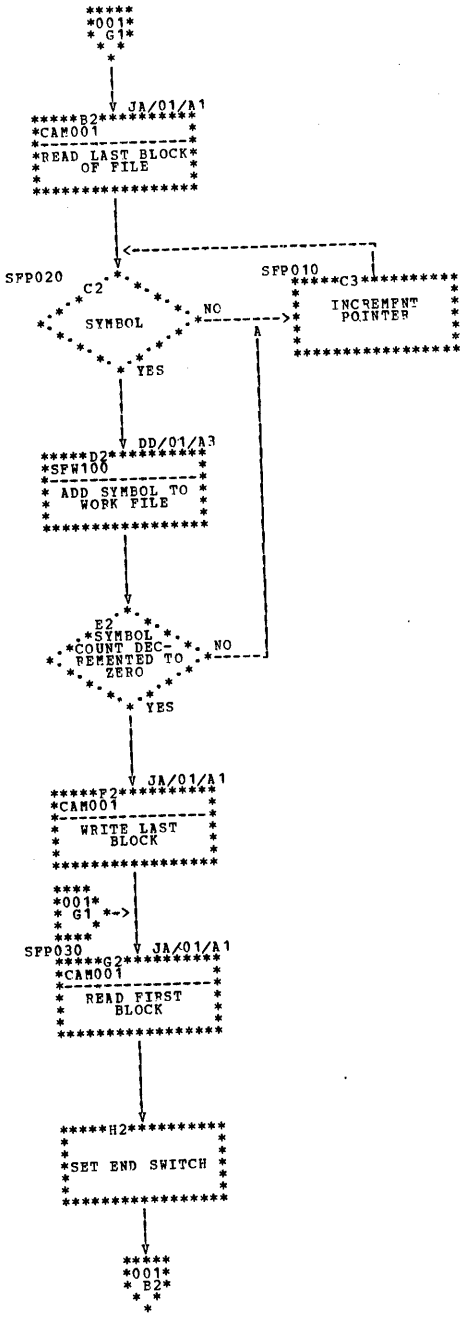


Chart DA (Part 2 of 2). $CGNSF Initialization Routine (SFI000) and Main Control Routine (SFP000)

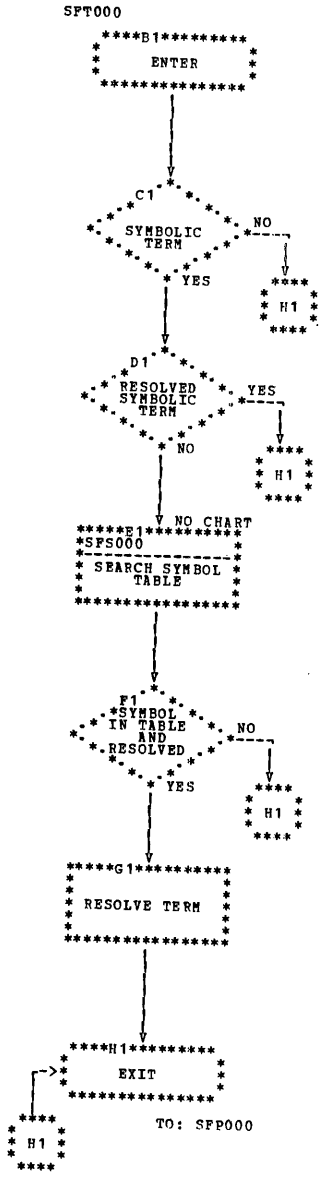


Chart DB. $CGNSF Term Record Processing Routine (SFT000)

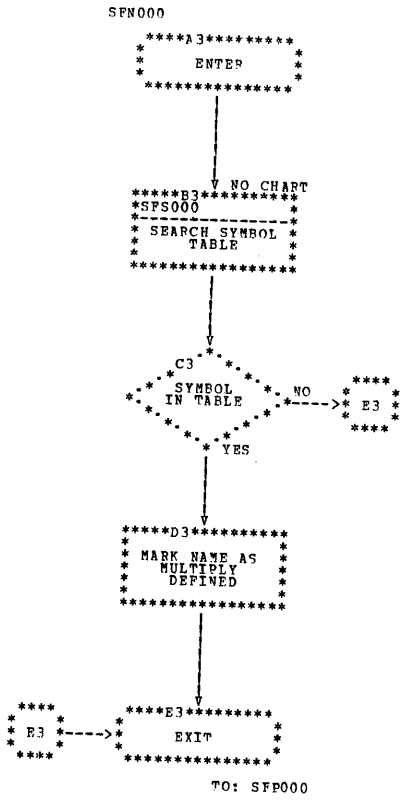
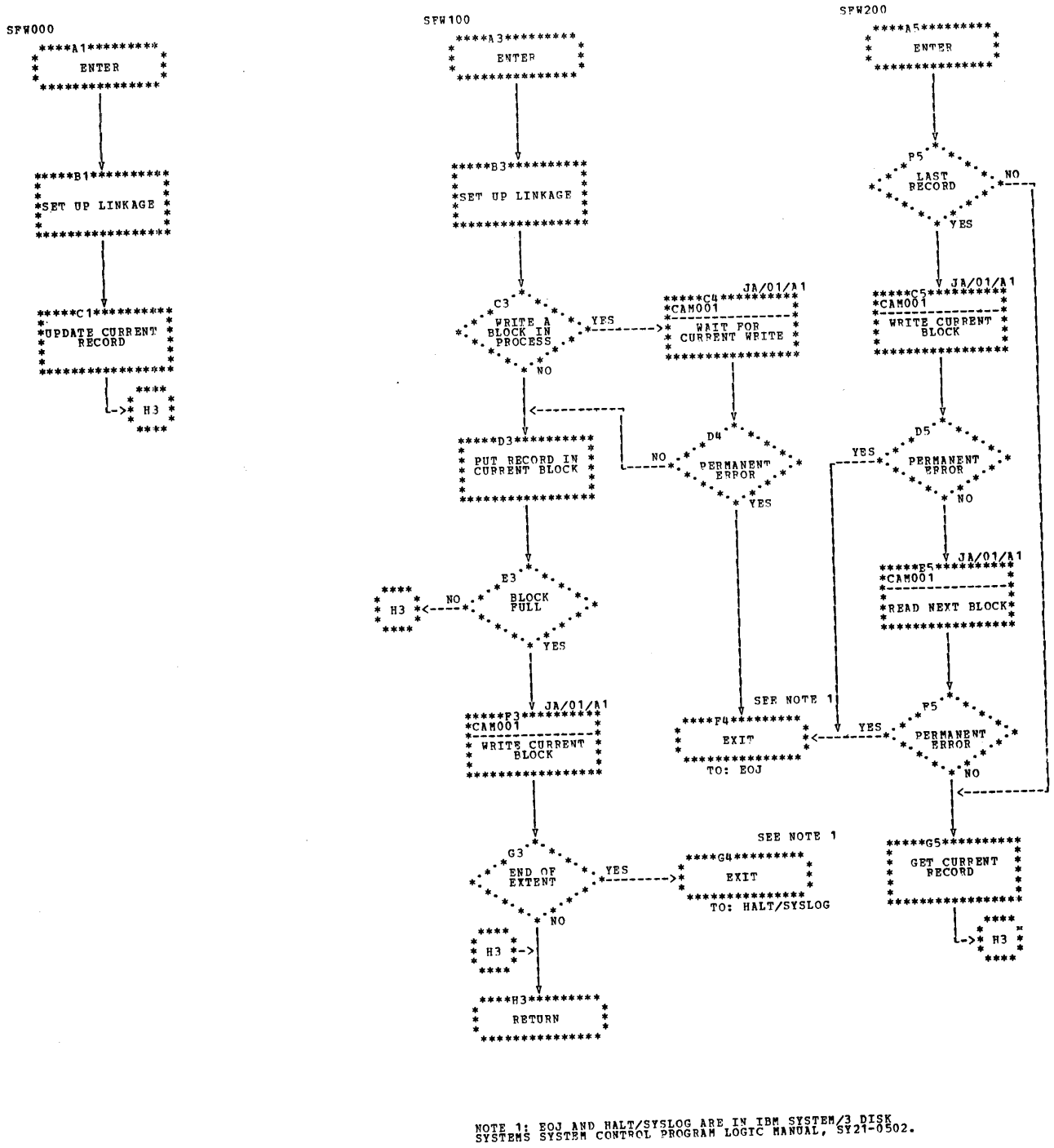


Chart DC. $CGNSF Name Record Processing Routine (SFN000)



NOTE 1: EOJ AND HALT/SYSLOG ARE IN IBM SYSTEM/3 DISK SYSTEMS SYSTEM CONTROL PROGRAM LOGIC MANUAL, SY21-0502.

Chart DD. $CGNSF Disk Data Management Interfaces (SFW000)

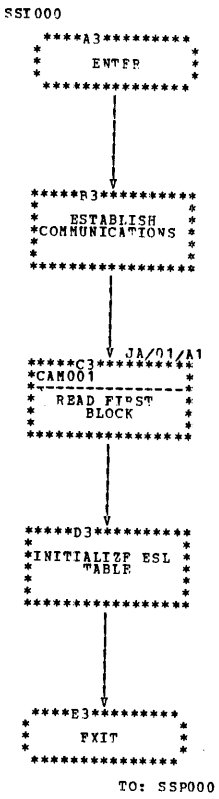


Chart EA. $CGNSS Initialization Routine (SSI000)

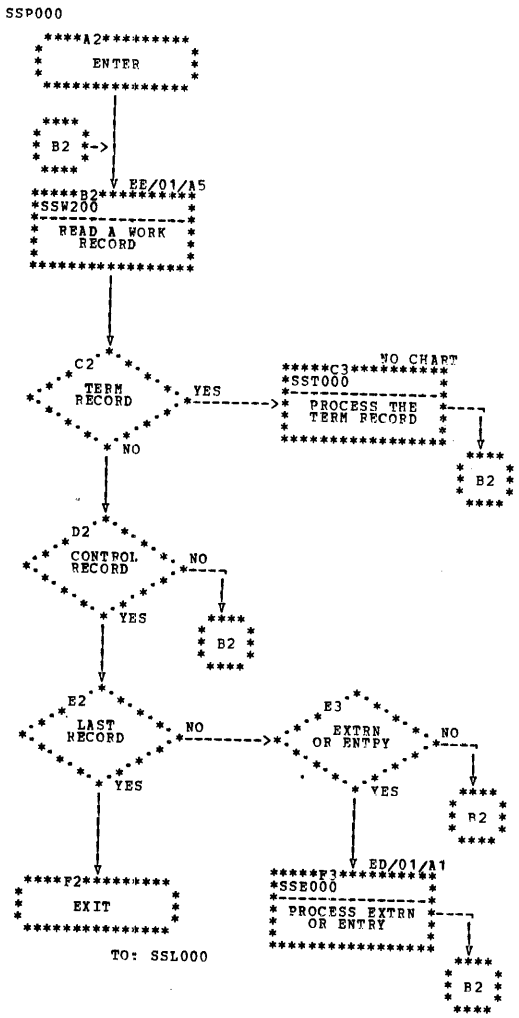


Chart EB. $CGNSS Main Control Routine (SSP000)

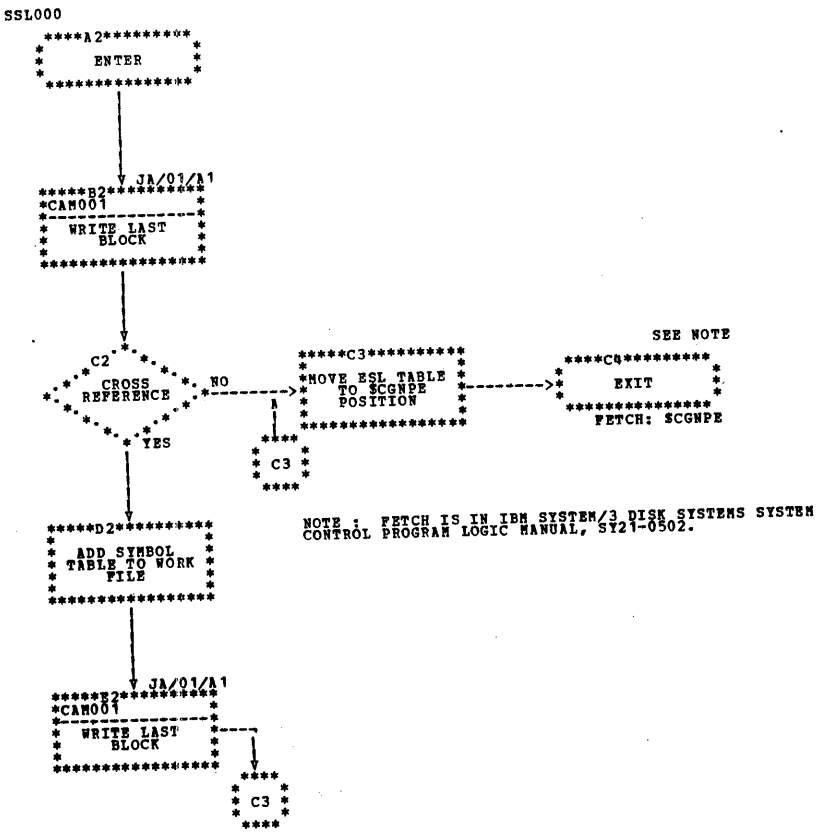


Chart EC. $CGNSS Termination Routine (SSL00)

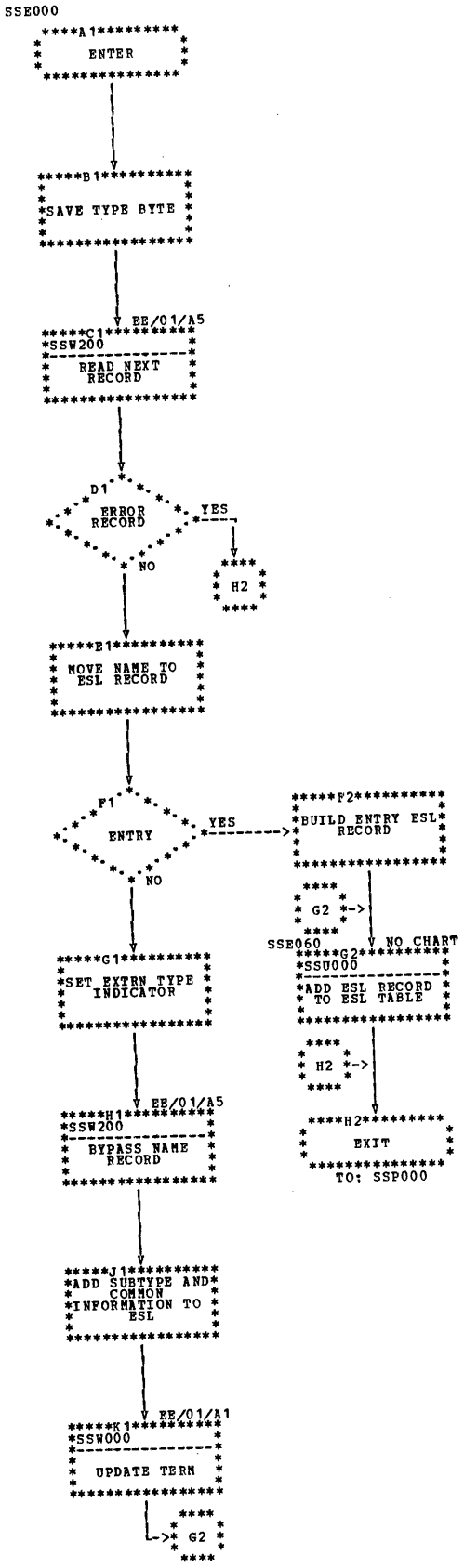
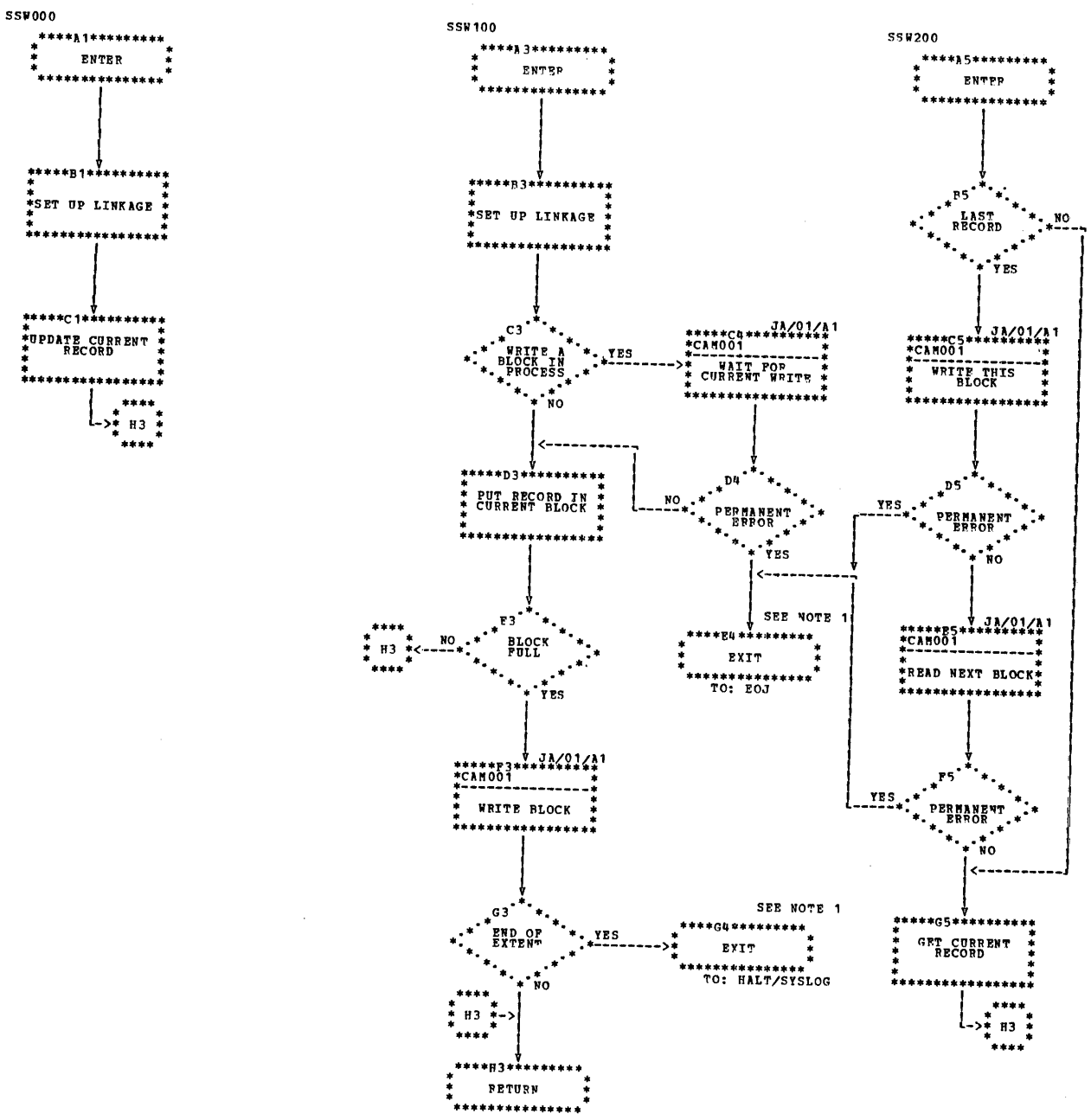


Chart ED. $CGNSS EXTRN/ENTRY Processing Routine (SSE000)



NOTE 1: EOJ AND HALT/SYSLOG ARE IN IBM SYSTEM/3 DISK SYSTEMS SYSTEM CONTROL PROGRAM LOGIC MANUAL, SY21-0502.

Chart EE. $CGNSS Disk Data Management Interfaces (SSW000)

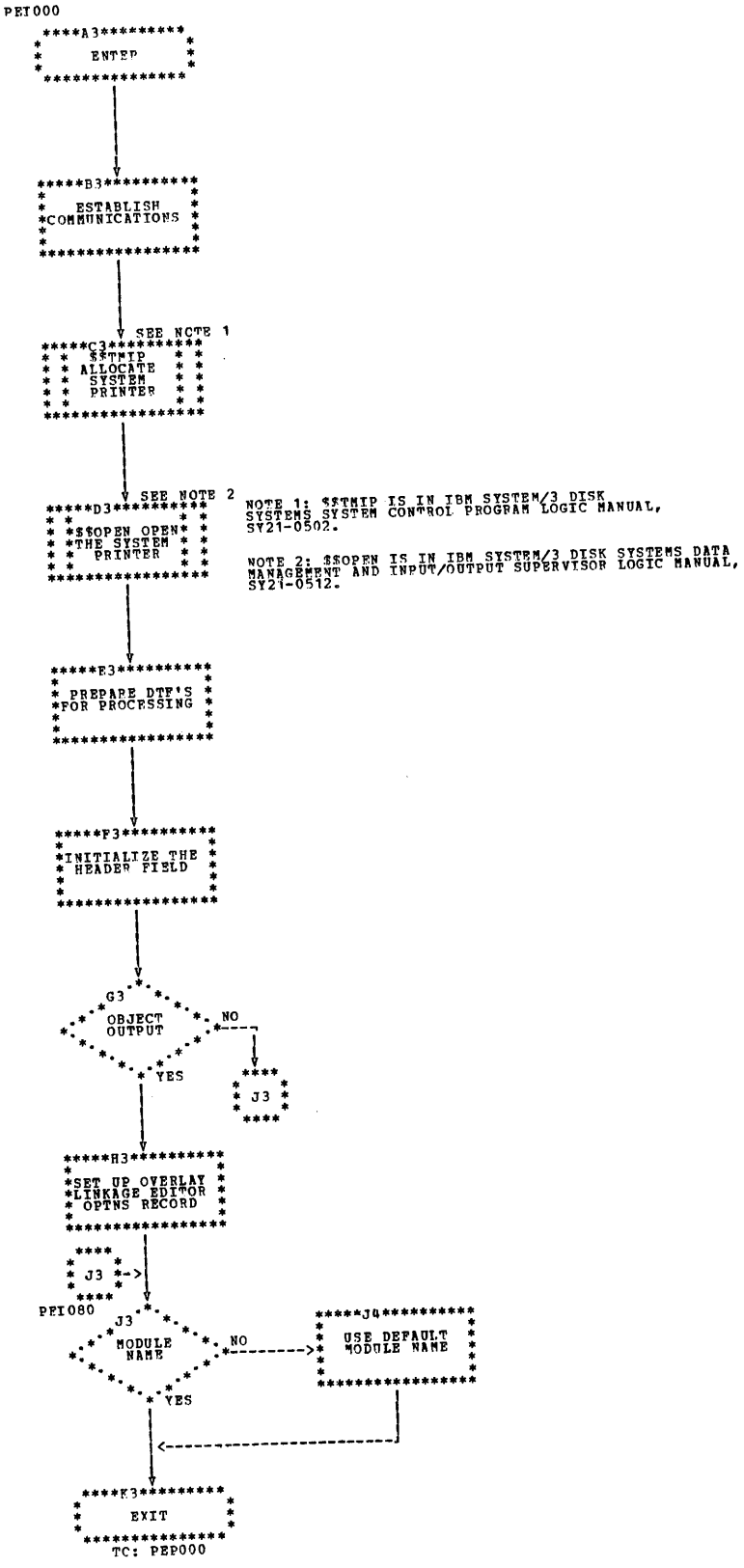


Chart FA. $CGNPE Initialization Routine (PEI000)

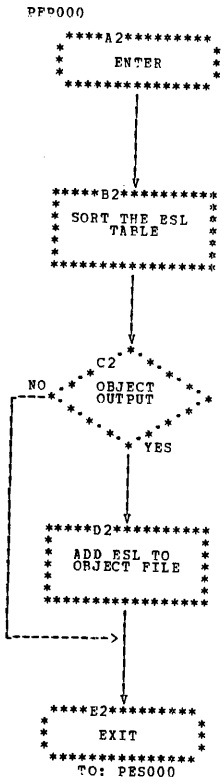


Chart FB. $CGNPE ESL Object Output Routine (PEP000)

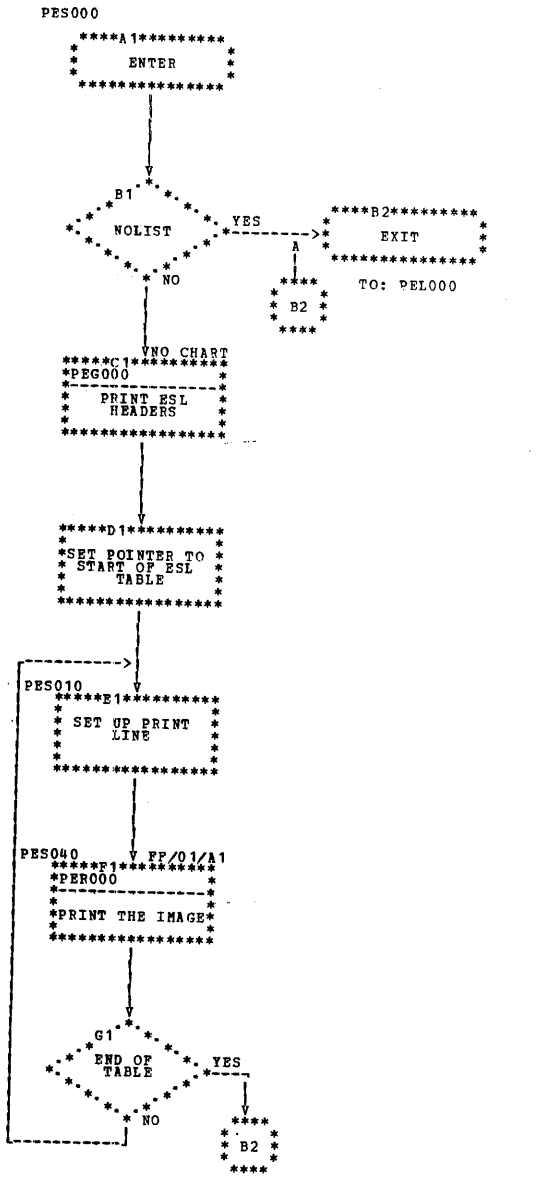
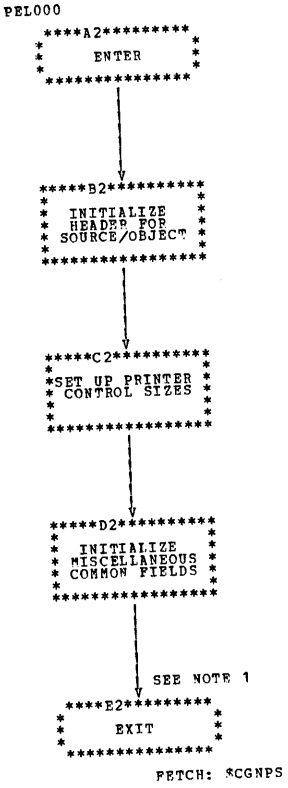
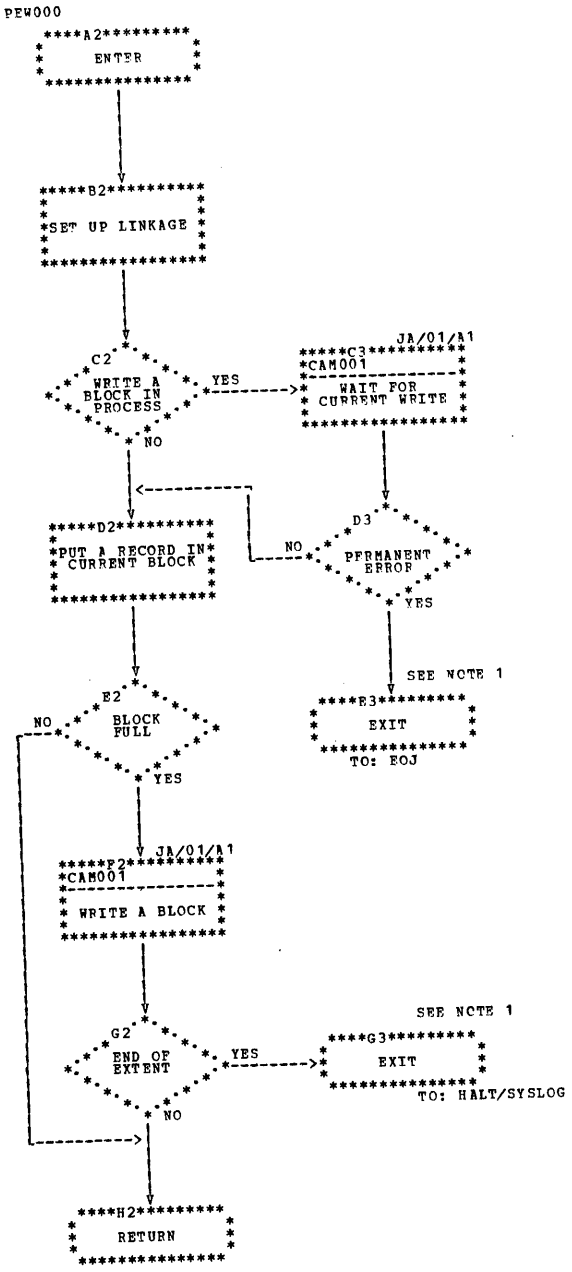


Chart FC. $CGNPE ESL Table Print Routine (PES000)



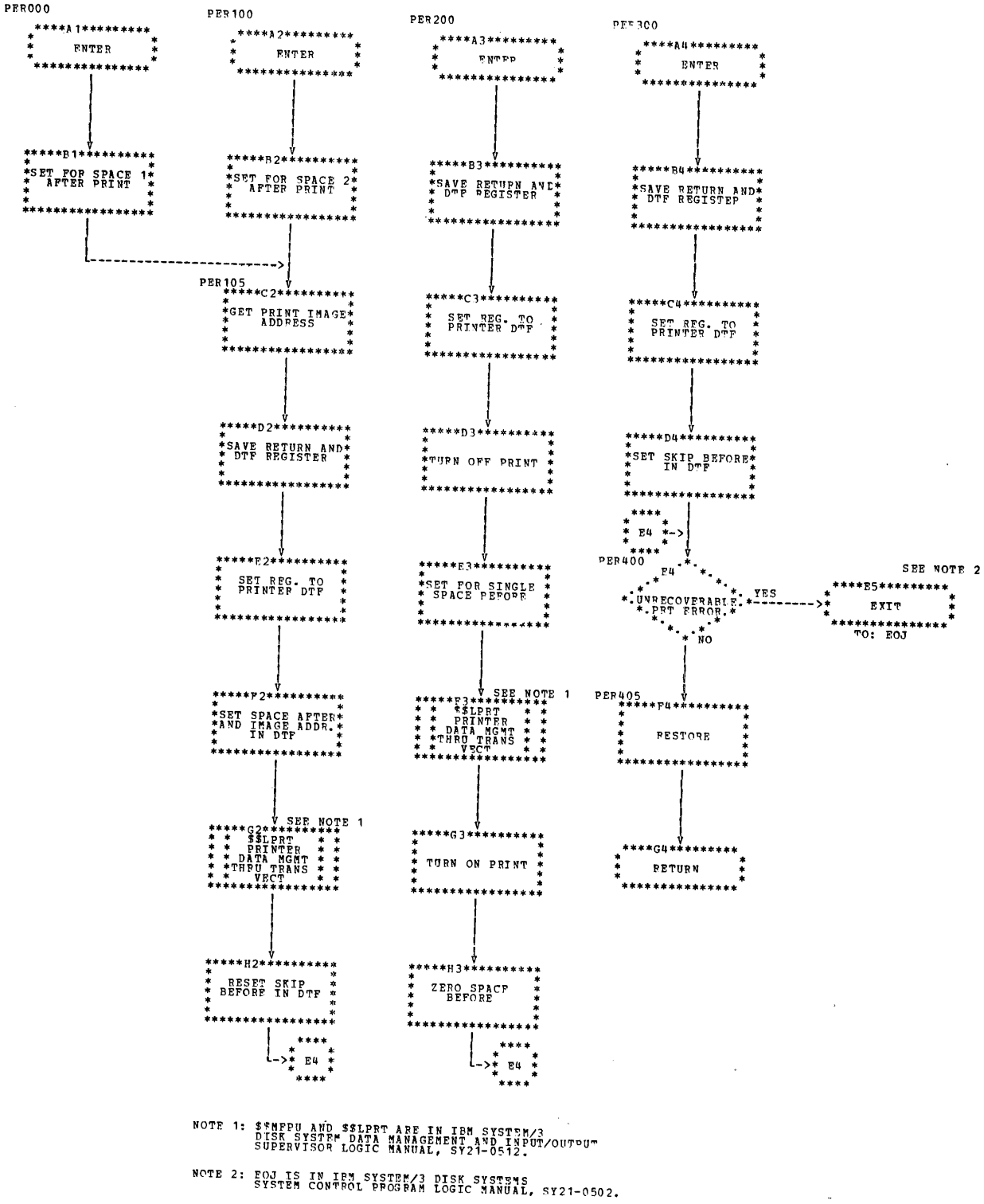
NOTE 1: FETCH IS IN THE IBM SYSTEM/3 DISK SYSTEMS SYSTEM CONTROL PROGRAM LOGIC MANUAL, SY21-0502.

Chart FD. $CGNPE Termination Routine (PEL000)



NOTE 1: EOJ AND HALT/SYSLOG ARE IN IBM SYSTEM/3 DISK SYSTEMS SYSTEM CONTROL PROGRAM LOGIC MANUAL, SY21-0502.

Chart FE. $CGNPE Disk Data Management Interfaces (PEW000)



NOTE 1: $$MFPU AND $$LPRT ARE IN IBM SYSTEM/3 DISK SYSTEM DATA MANAGEMENT AND INPUT/OUTPUT SUPERVISOR LOGIC MANUAL, SY21-0512.

NOTE 2: EOJ IS IN IBM SYSTEM/3 DISK SYSTEMS SYSTEM CONTROL PROGRAM LOGIC MANUAL, SY21-0502.

Chart FF. $CGNPE Printer Data Management Interface (PER000)

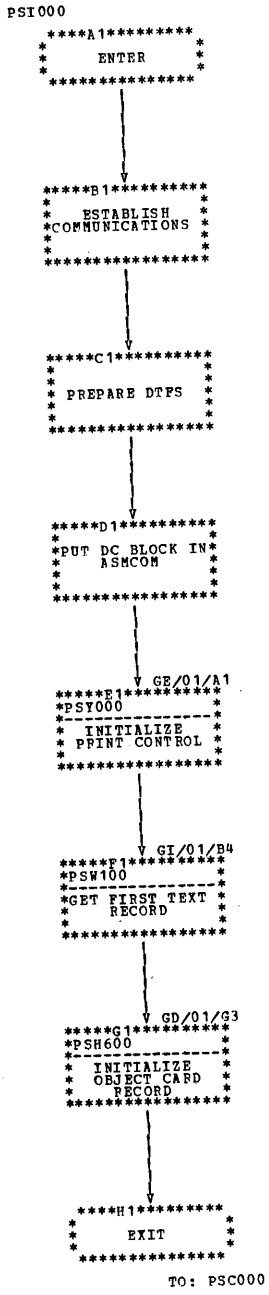


Chart GA. $CGNPS Initialization Routine (PSI000)

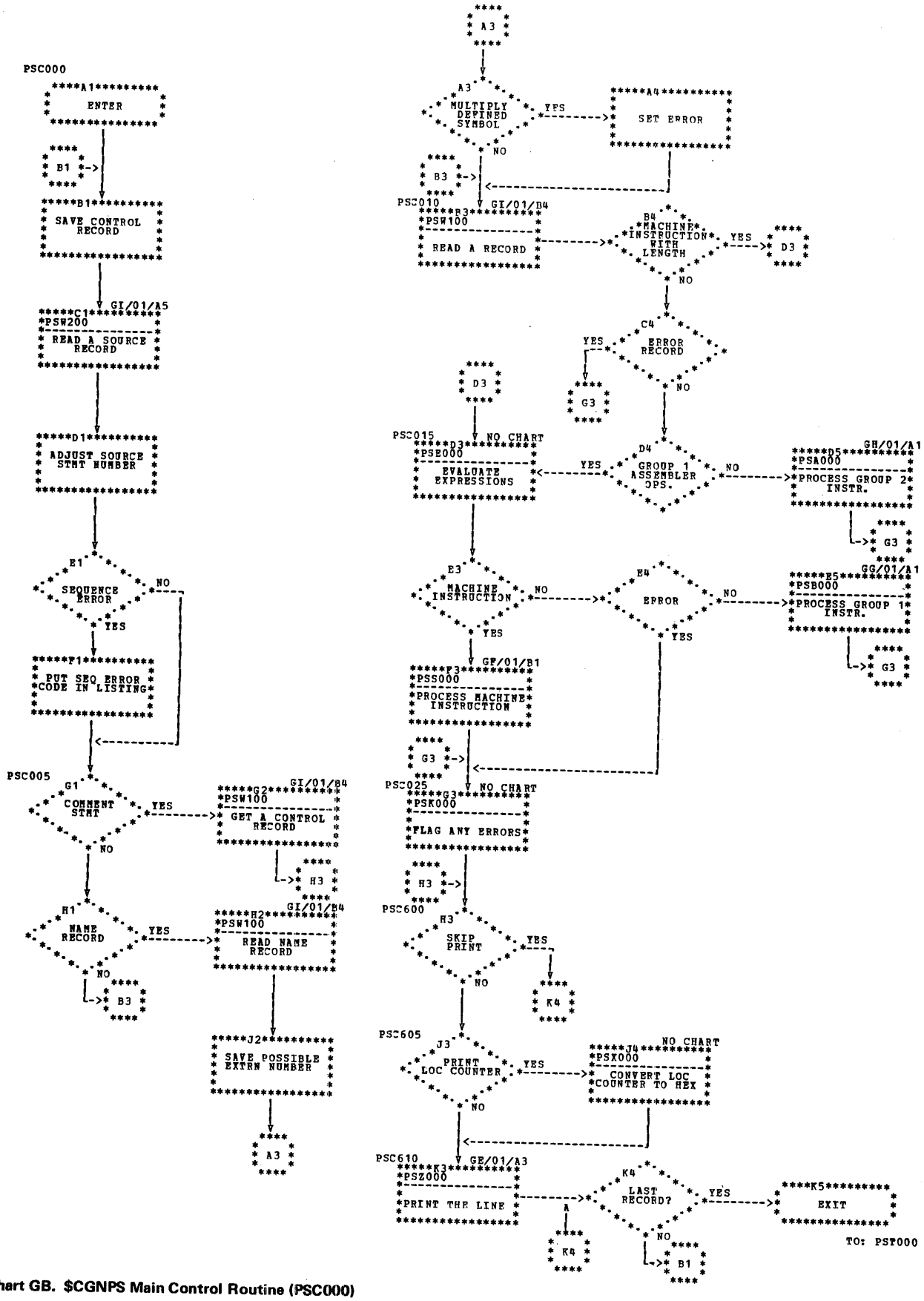
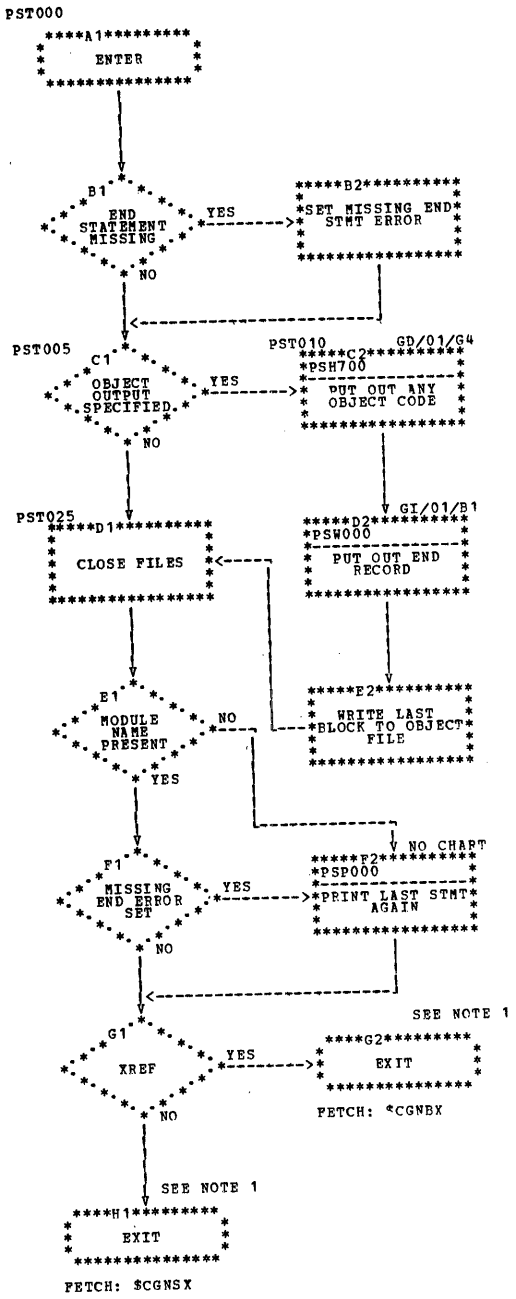


Chart GB. $CGNPS Main Control Routine (PSC000)



NOTE 1: FETCH IS IN IBM SYSTEM/3 DISK SYSTEMS SYSTEM CONTROL PROGRAM LOGIC MANUAL, SY21-0502

Chart GC. $CGNPS Termination Routine (PST000)

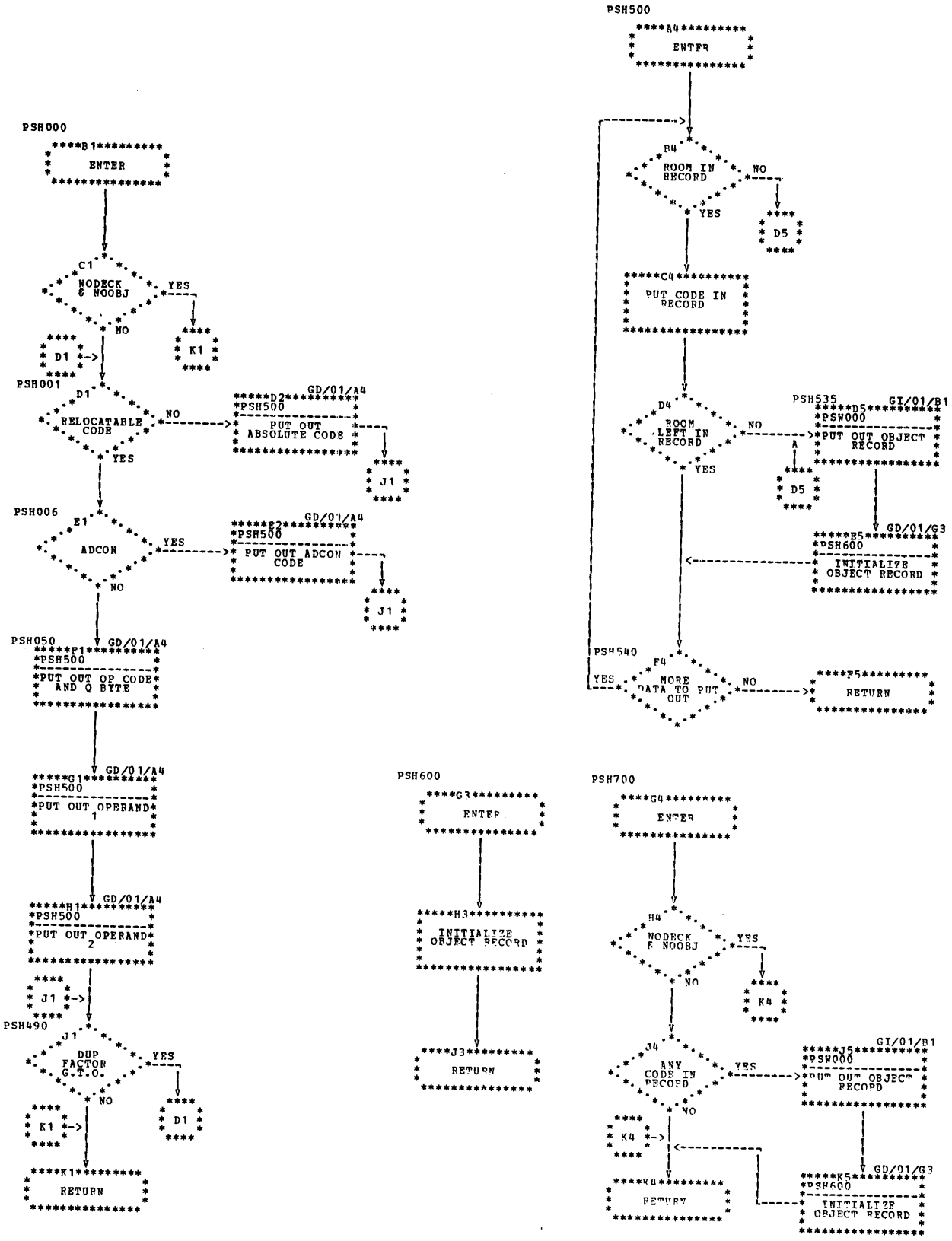


Chart GD. $CGNPS Put Object Code in Punch File Routine (PSH000)

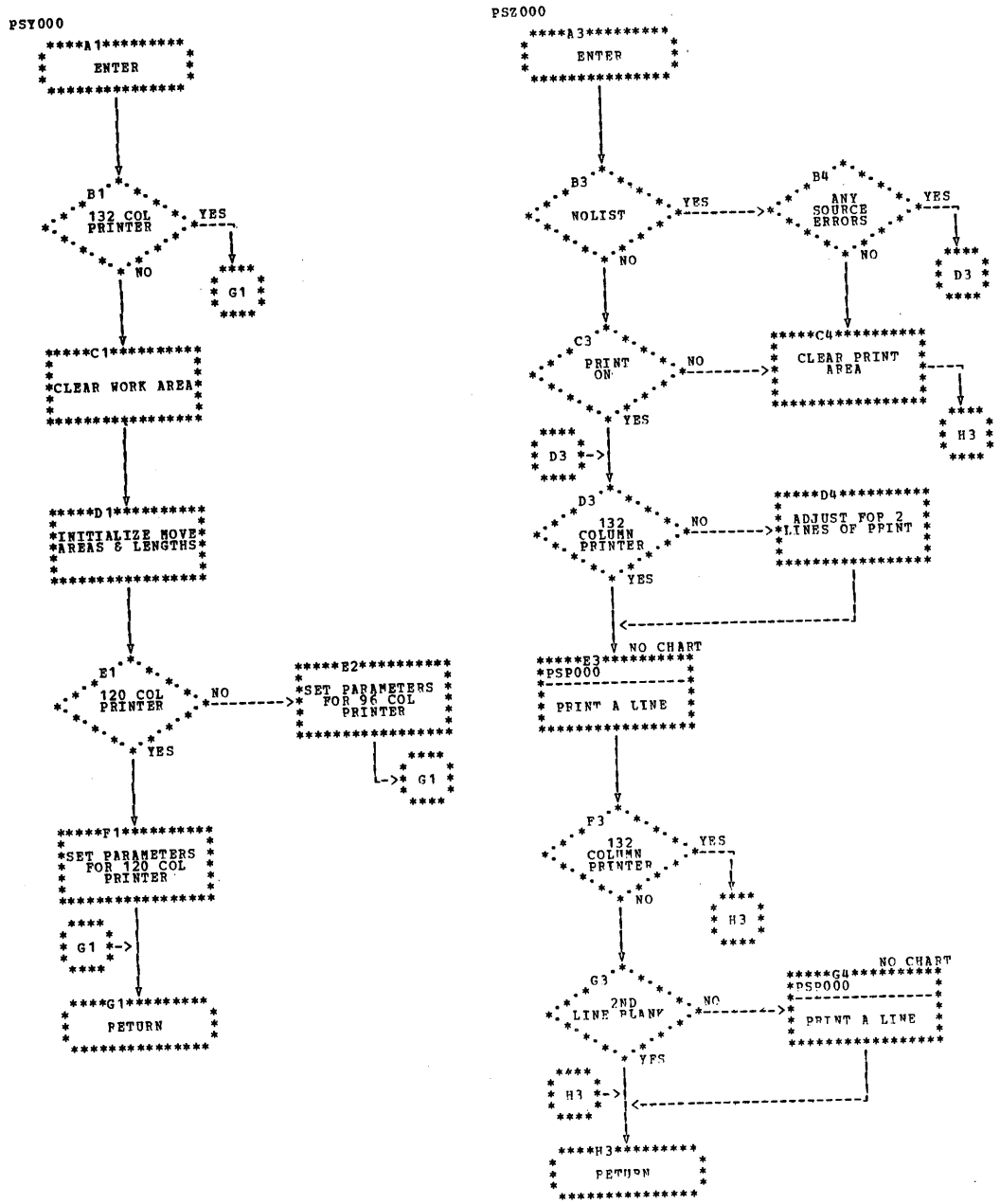


Chart GE. $CGNPS Print Image Edit Initialization Routine (PSY000) and Print Image Edit Routine (PSZ000)

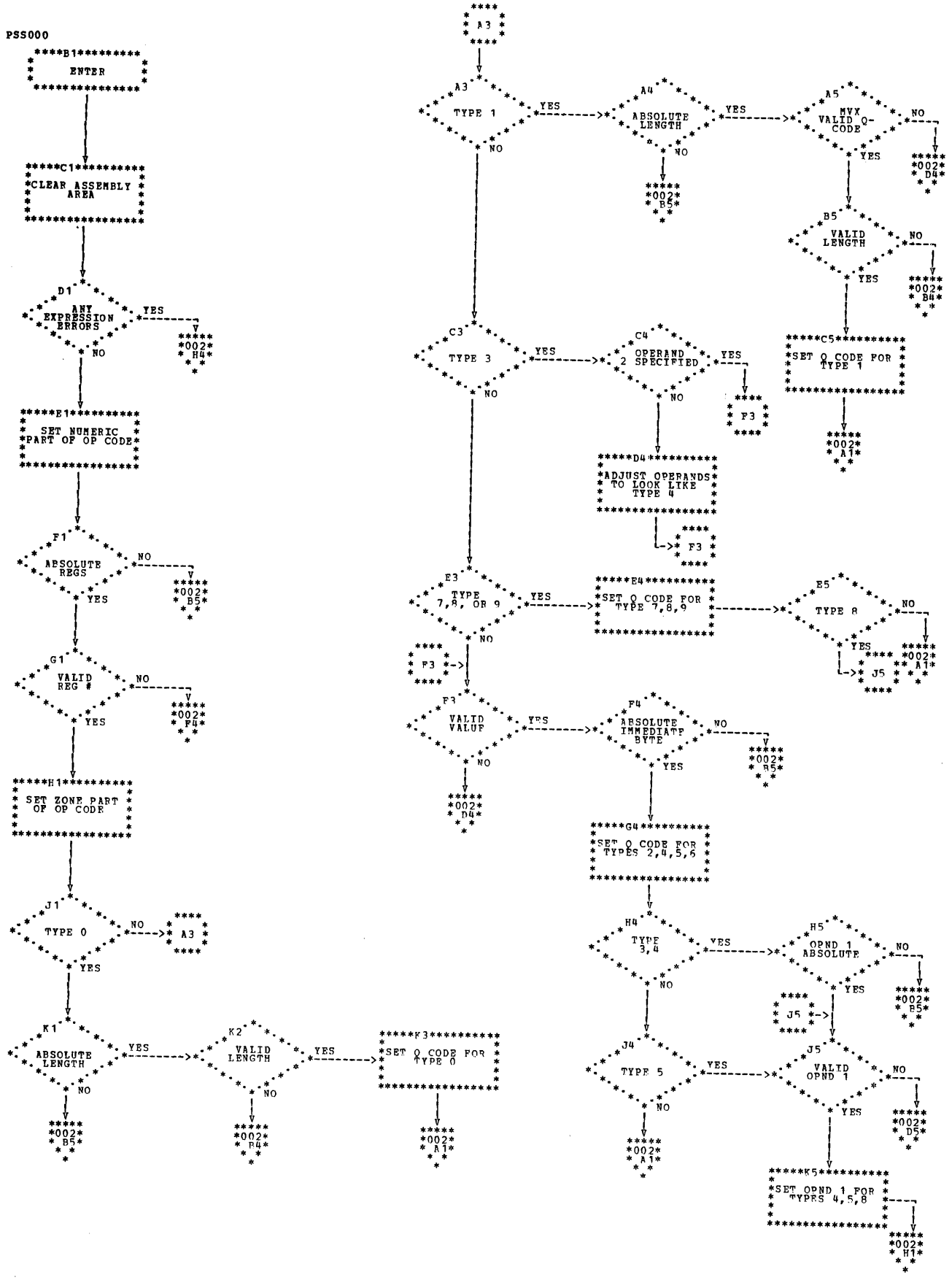


Chart GF (Part 1 of 2). $CGNPS Machine Instruction Processing Routine (PSS000)

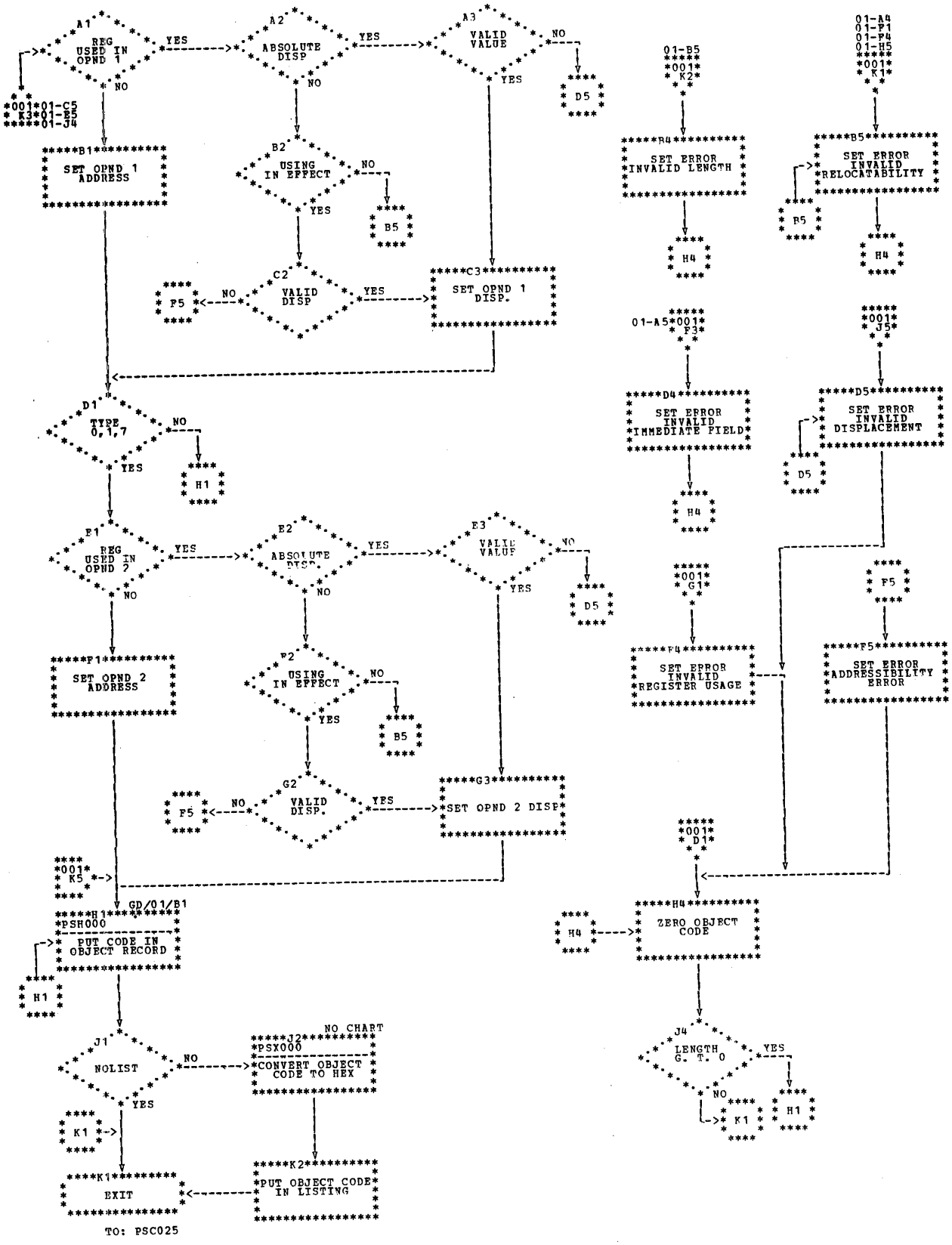


Chart GF (Part 2 of 2). $CGNPS Machine Instruction Processing Routine (PSS000)

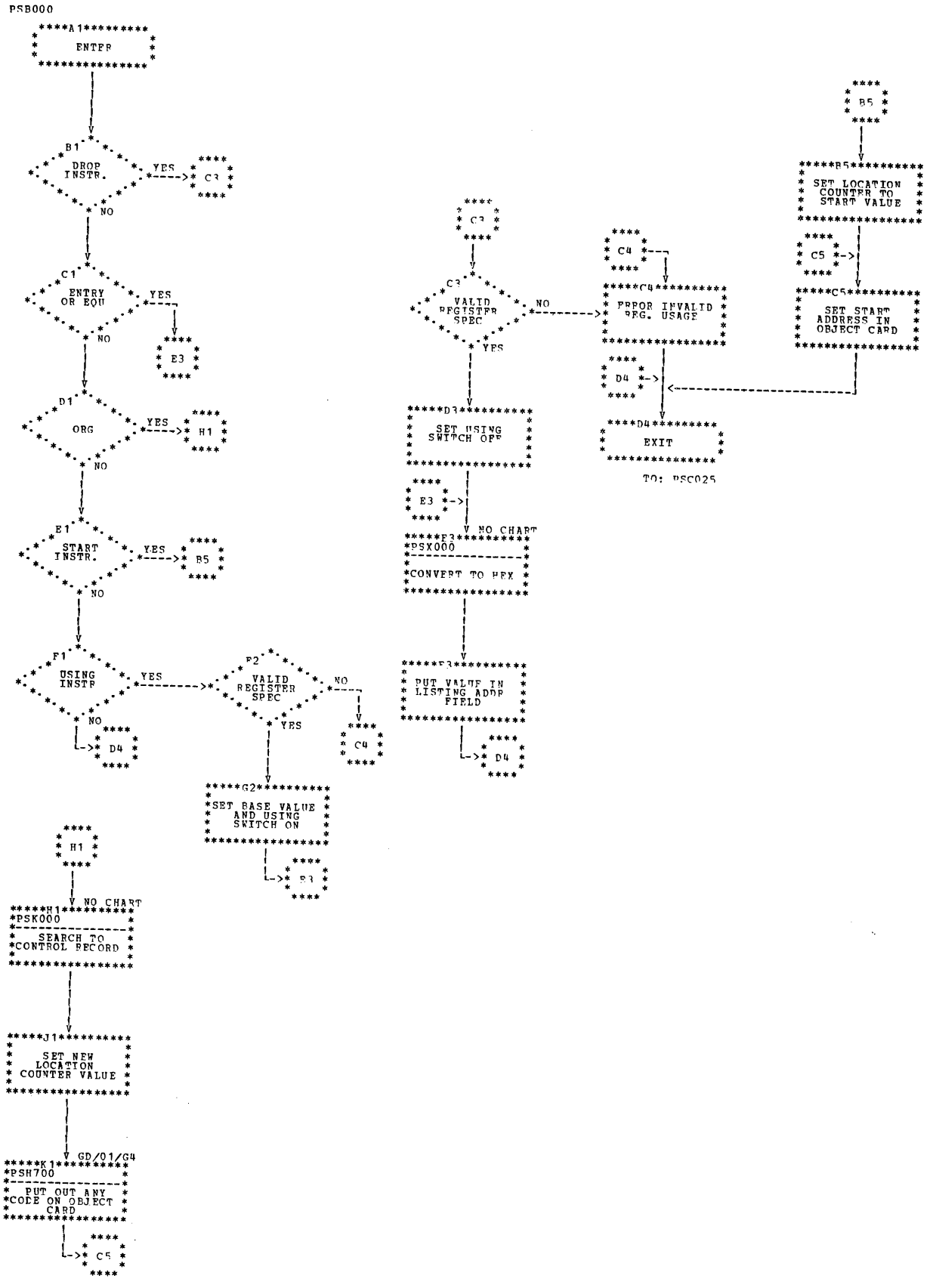


Chart GG. $CGNPS Group 1 Assembler Instruction Processing Routine (PSB000)

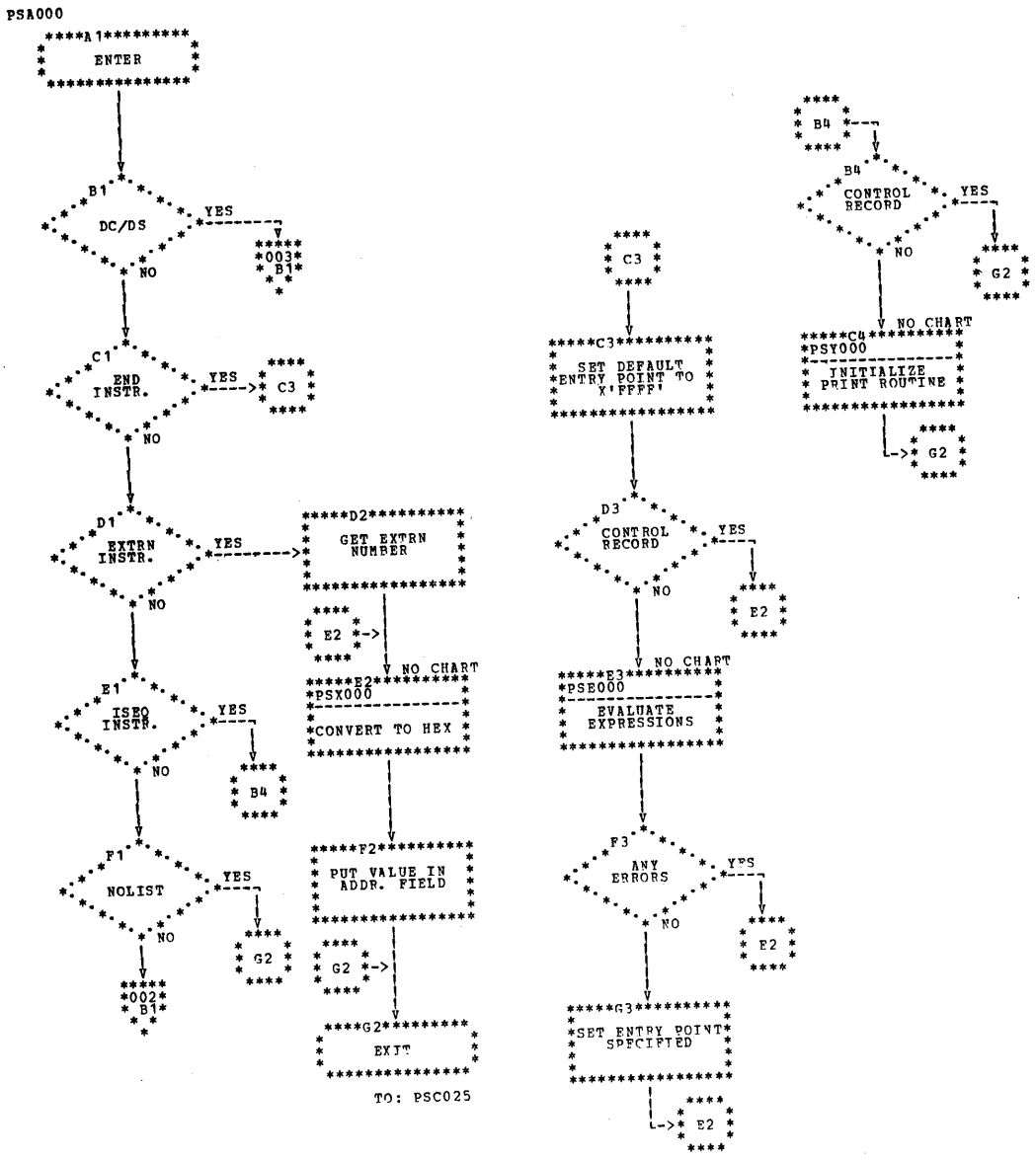


Chart GH (Part 1 of 3). $CGNPS Group 2 Assembler Instruction Processing Routine (PSA000)

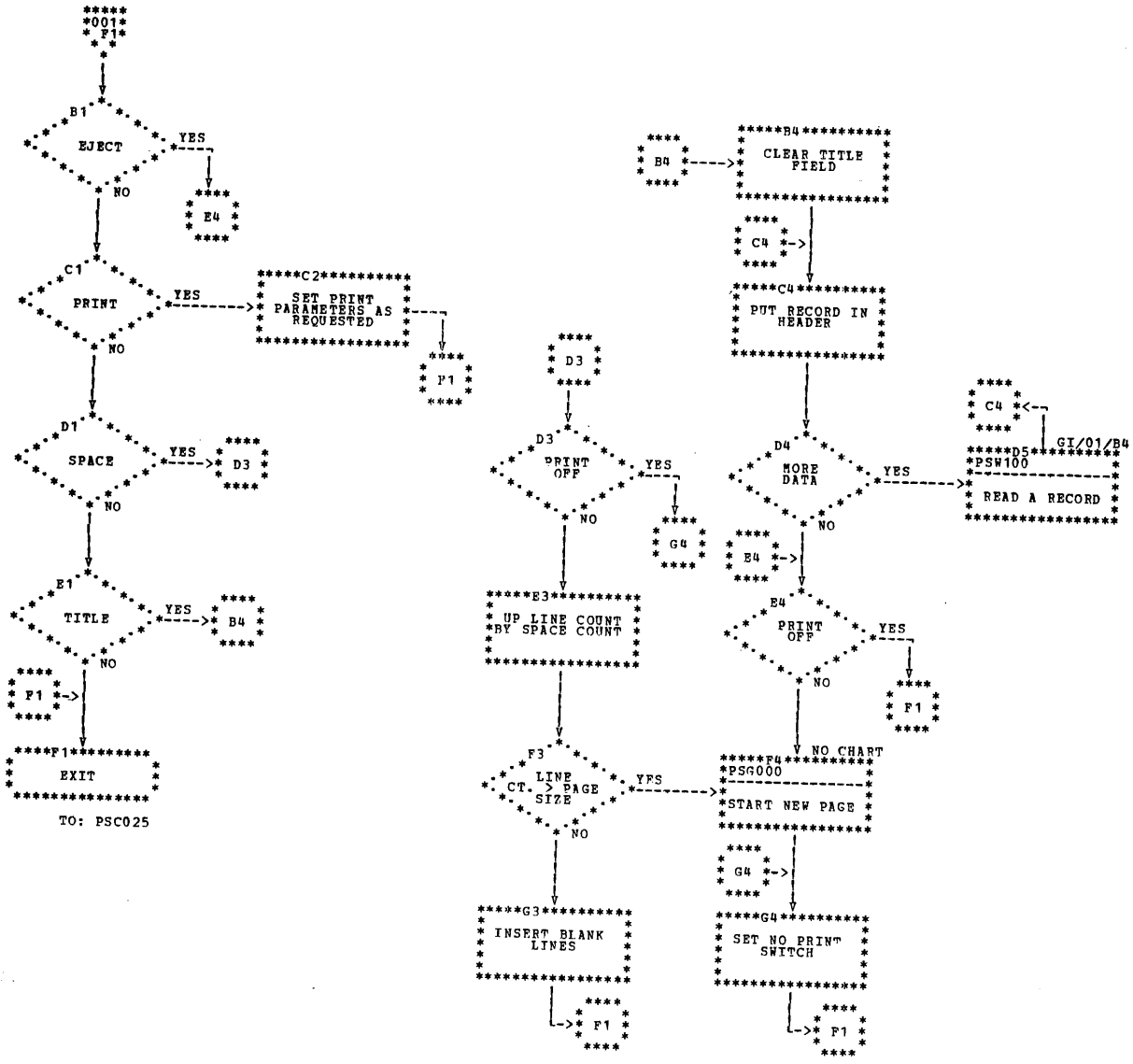


Chart GH (Part 2 of 3). $CGNPS Group 2 Assembler Instruction Processing Routine (PSA000)

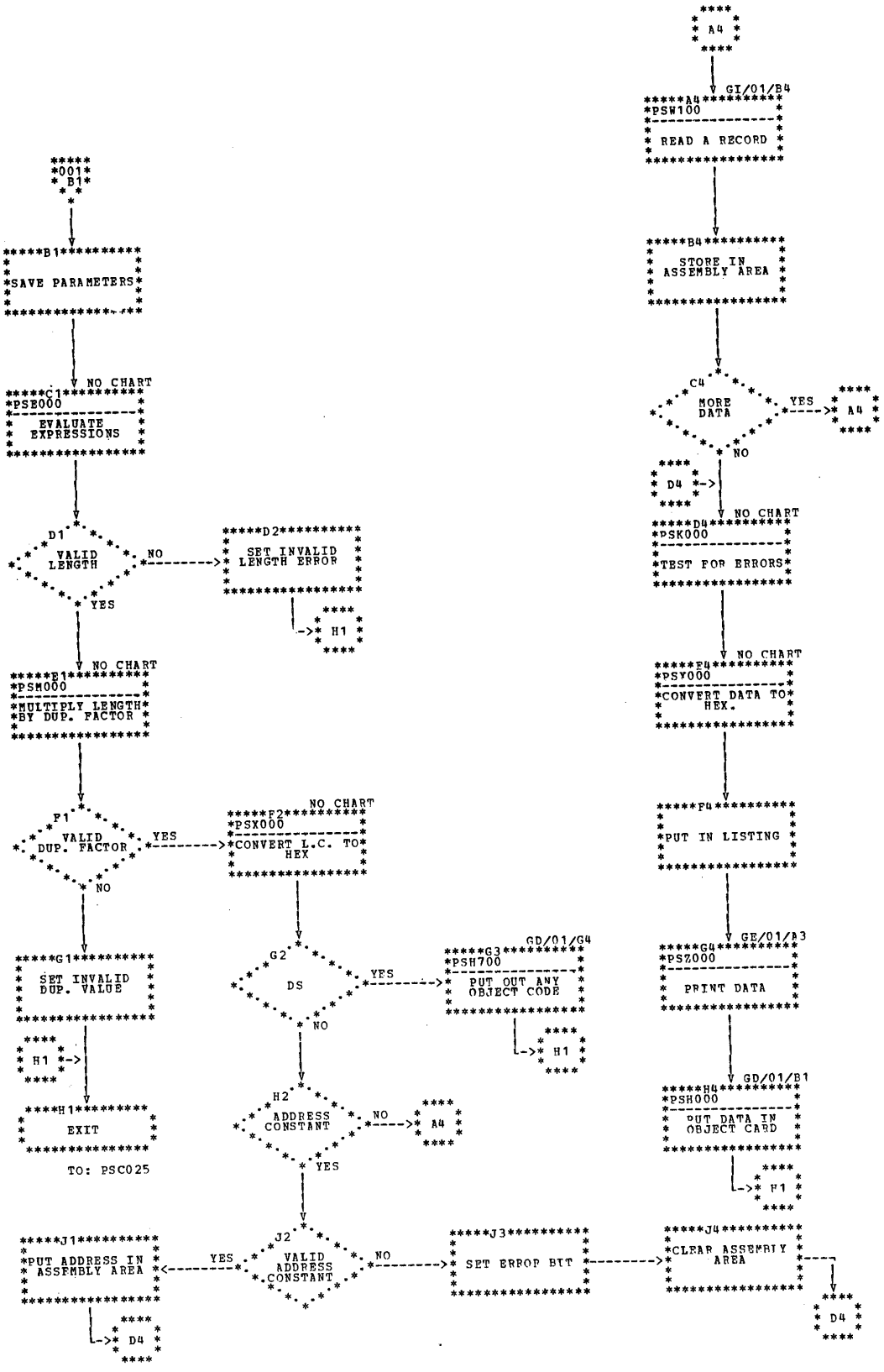


Chart GH (Part 3 of 3). $CGNPS Group 2 Assembler Instruction Processing Routine (PSA000)

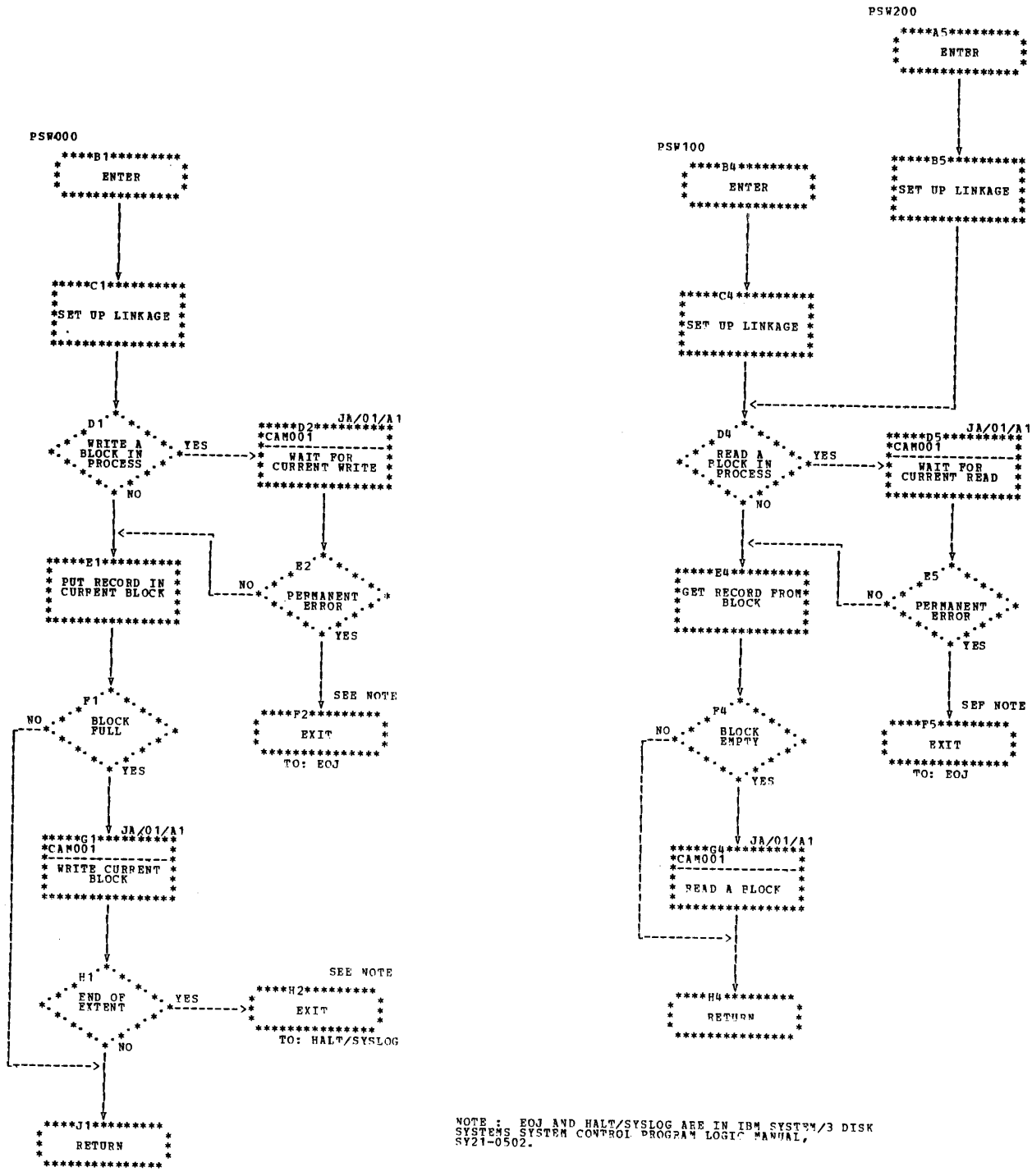


Chart GI. $CGNPS Disk Data Management Interface (PSW000)

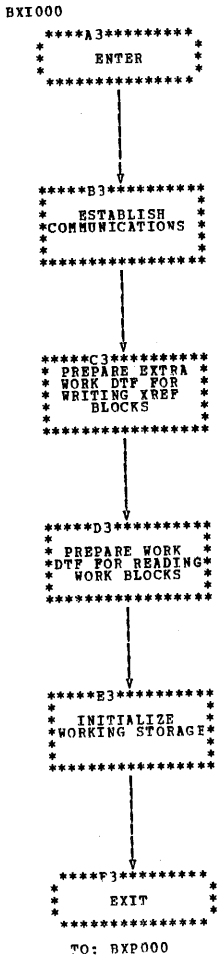


Chart HA. $CGNBX Initialization Routine (BXI000)

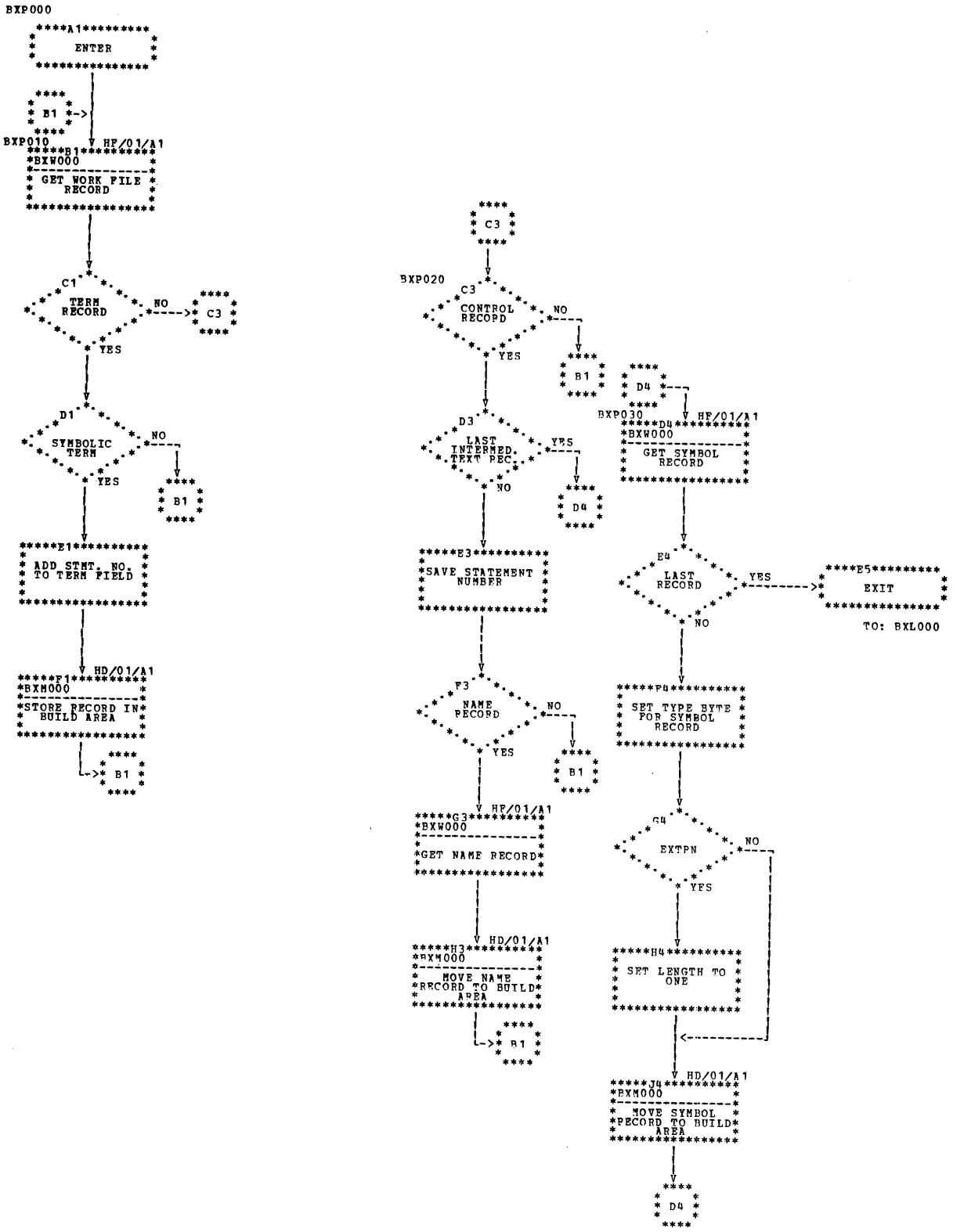


Chart HB. $CGNBX Cross Reference Build Routine (BXP000)

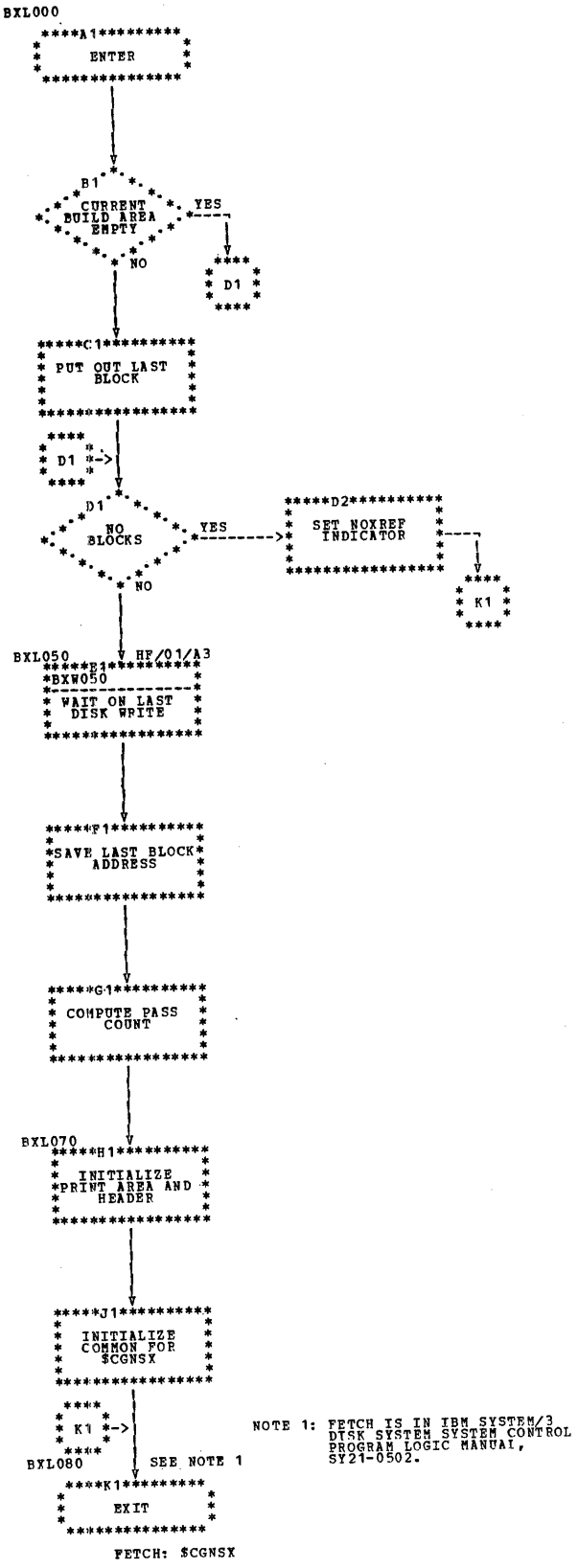


Chart HC. $CGNBX Termination Routine (BXL000)

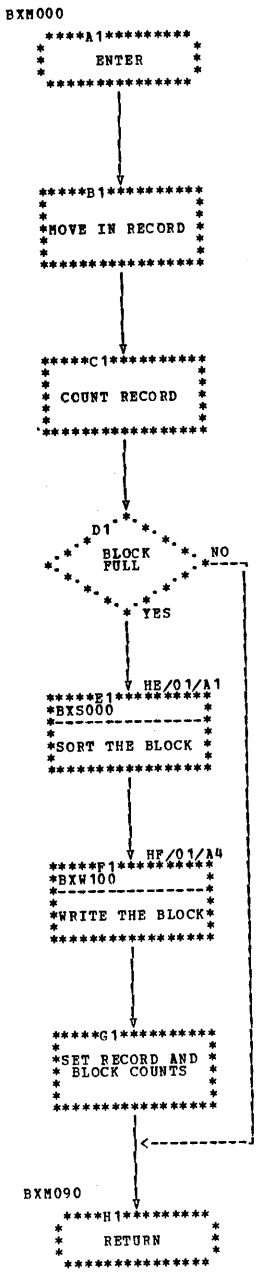


Chart HD. $CGNBX Build Area Move and Output Routine (BXM000)

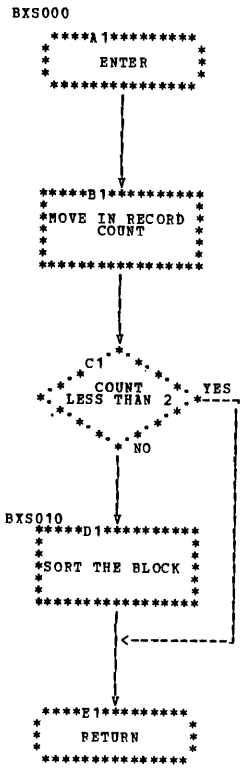
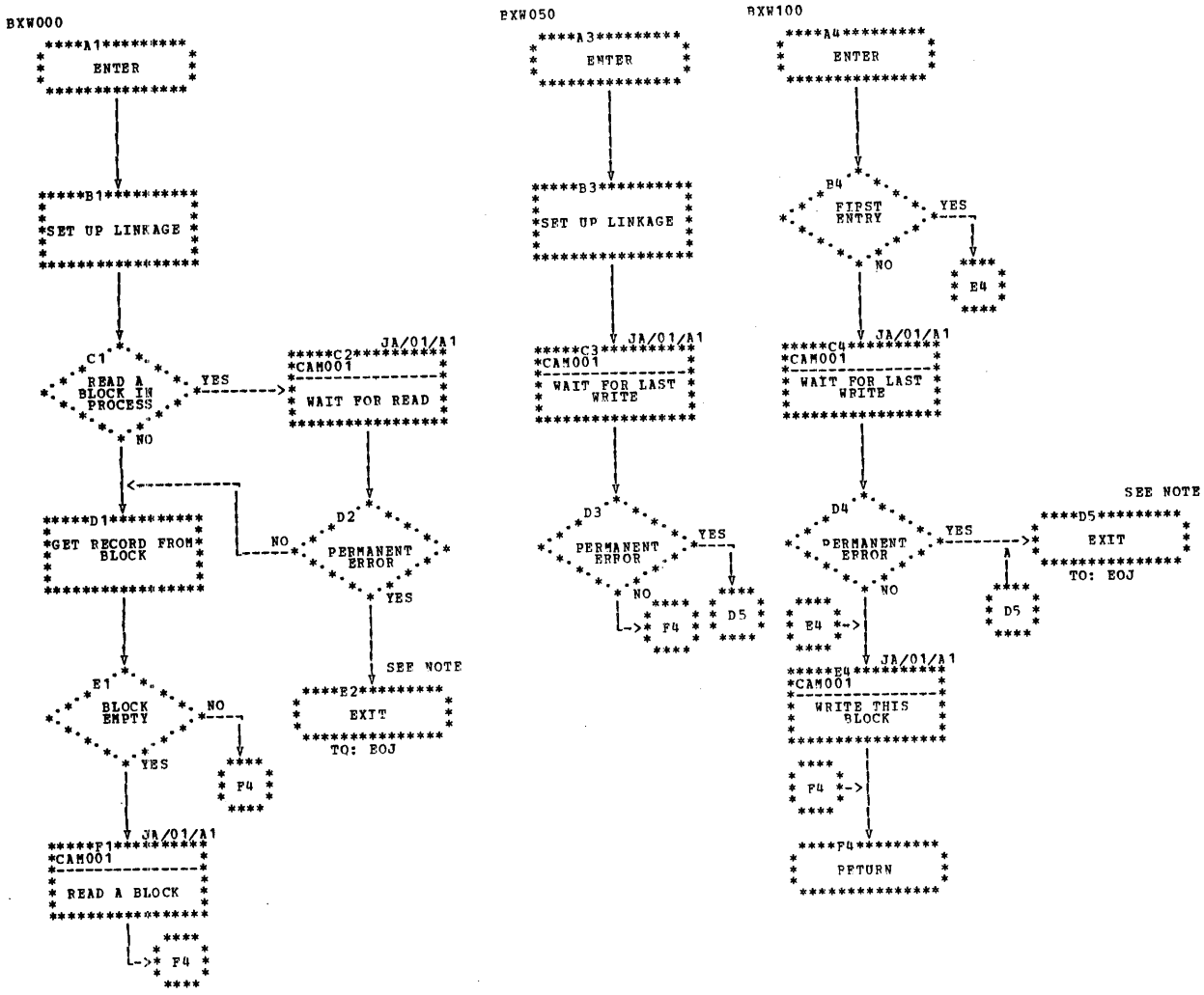


Chart HE. $CGNBX Cross Reference File Block Sort Routine (BXS000)



NOTE : EOJ IS IN IBM SYSTEM/3 DISK SYSTEMS SYSTEM CONTROL PROGRAM LOGIC MANUAL, SY21-0502.

Chart HF. $CGNBX Disk Data Management Interfaces (BXW000)

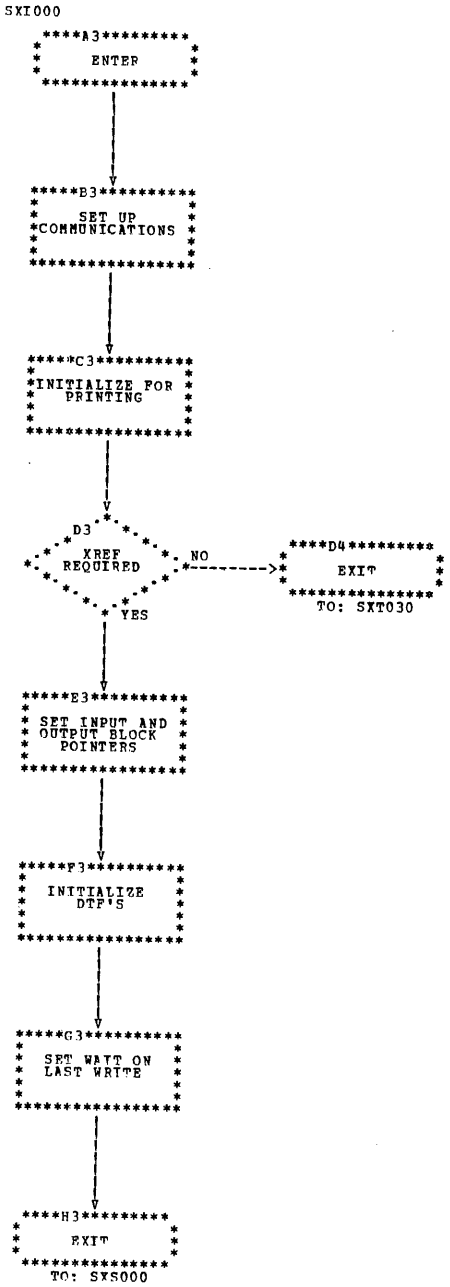


Chart IA. $CGNSX Initialization Routine (SXI000)

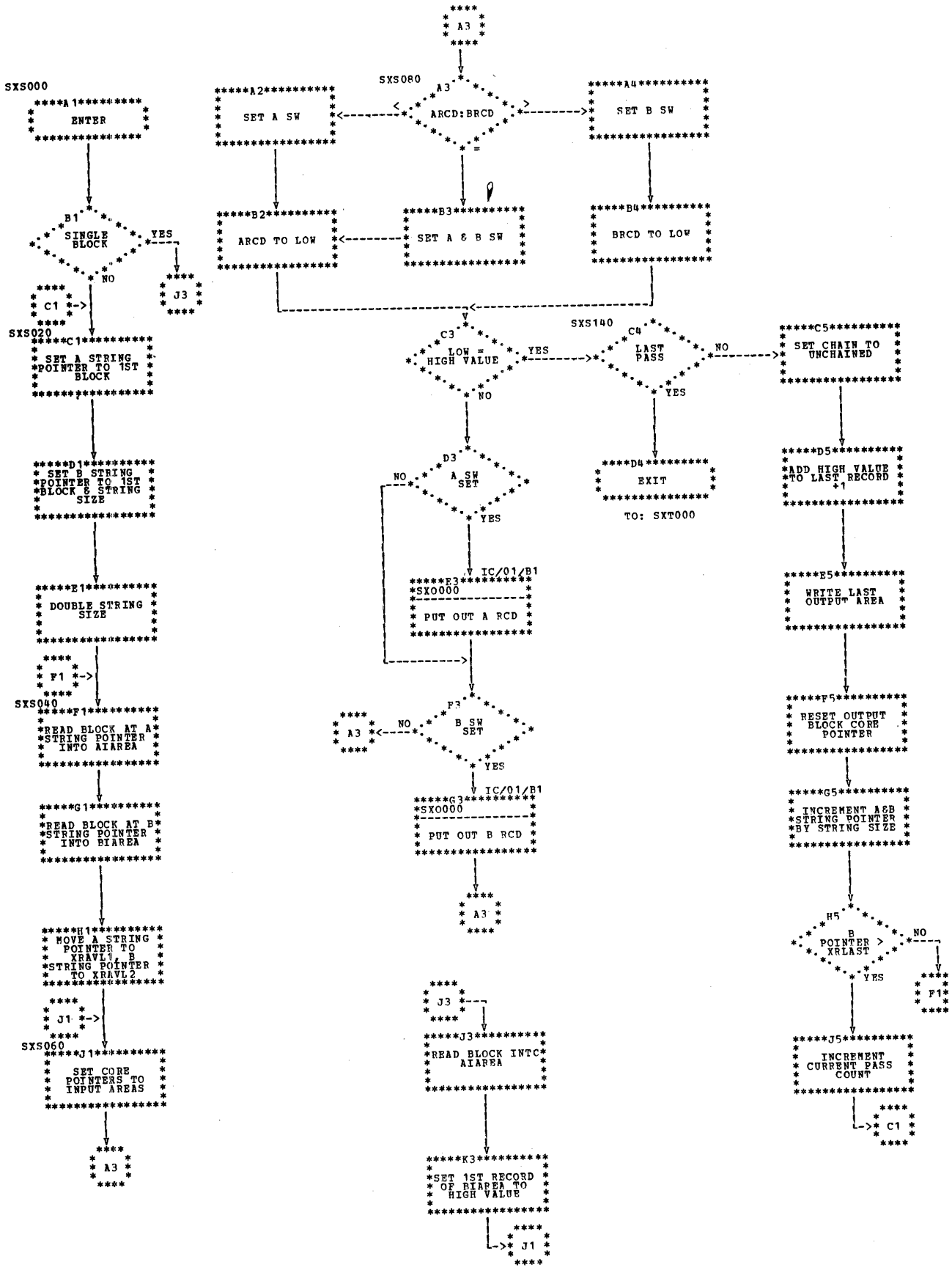


Chart IB. $CGNSX Merge Control Routine (SXS000)

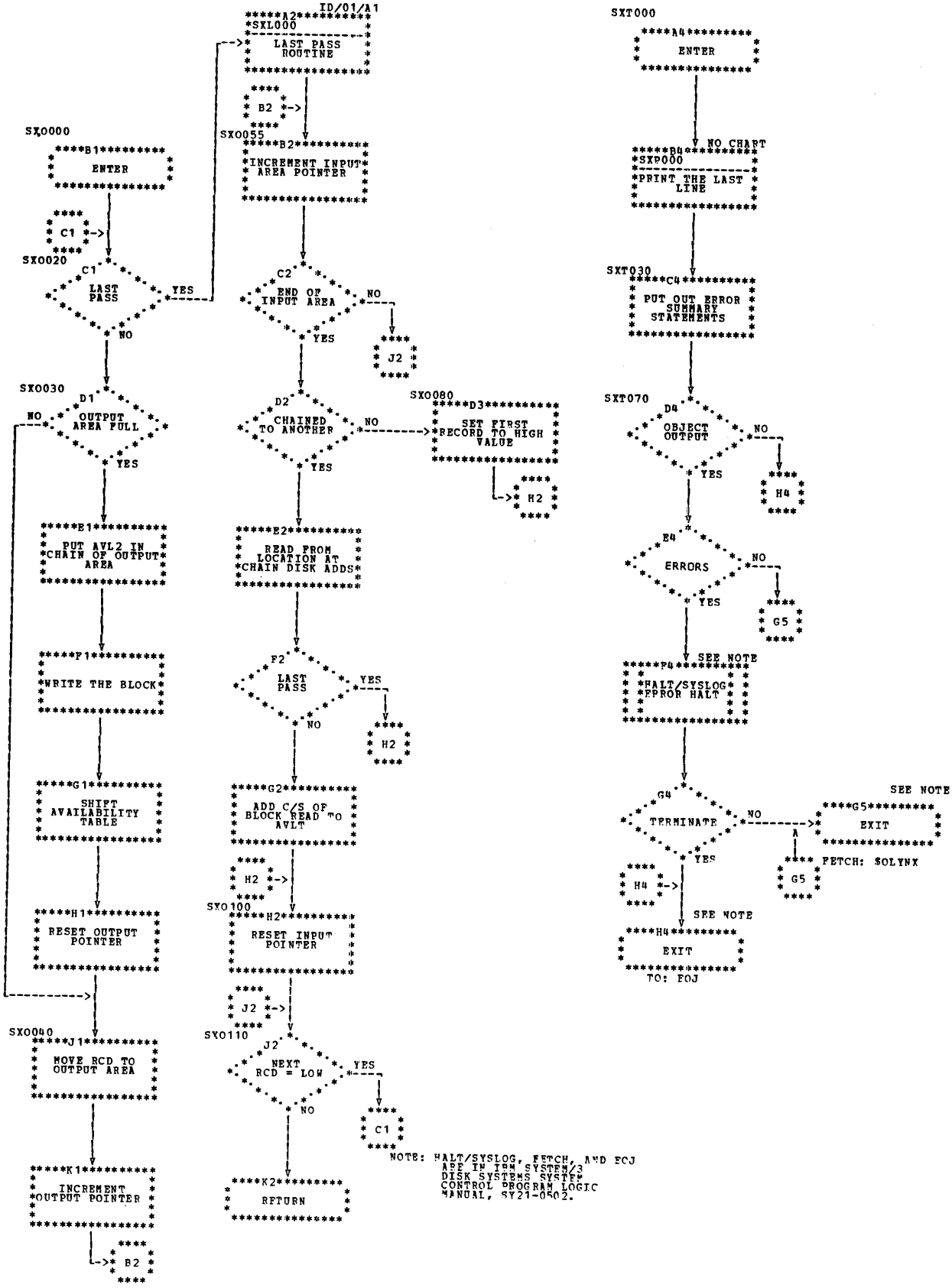


Chart IC. $CGNSX Merge Output Routine (SXO000) and Termination Routine (SXT000)

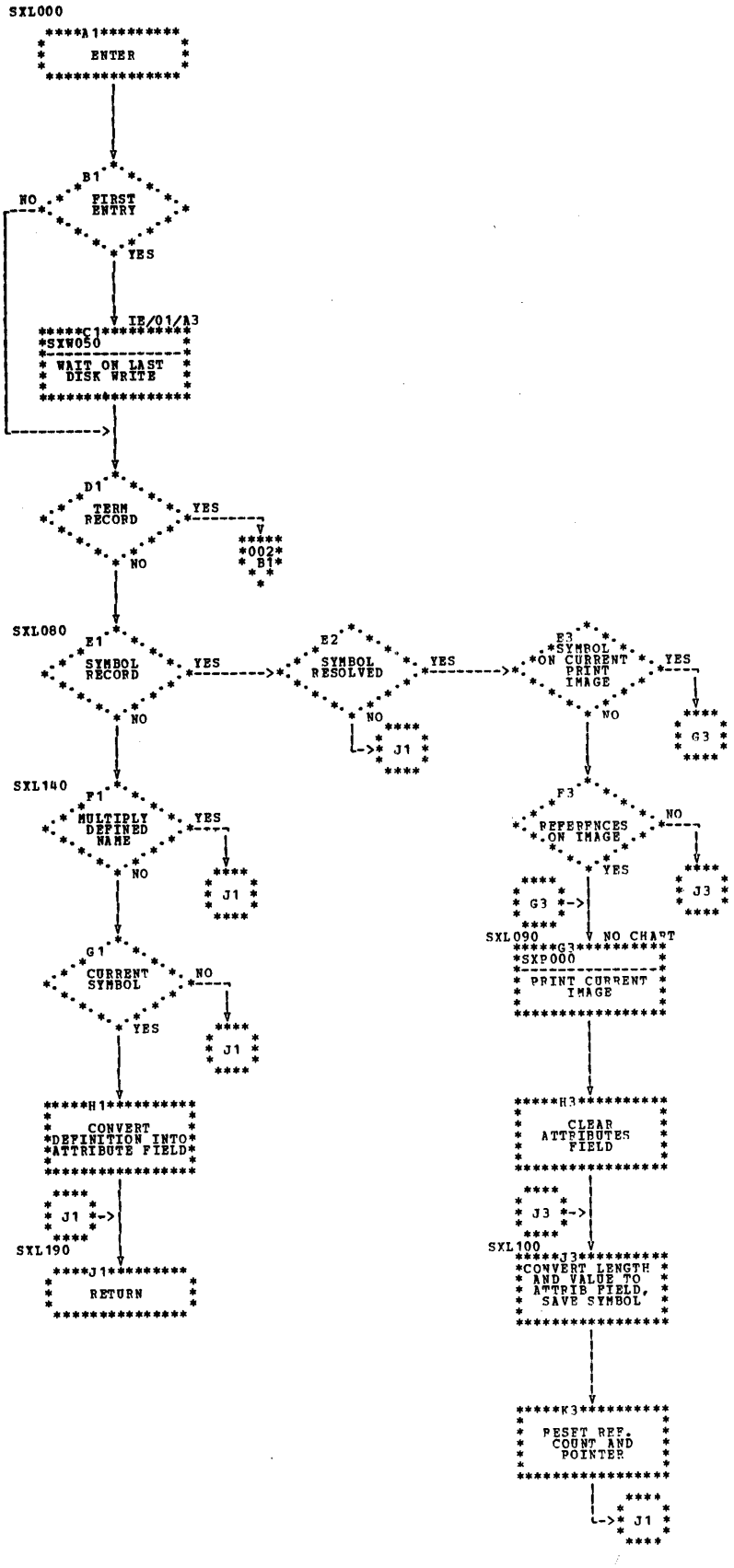


Chart ID (Part 1 of 2). $CGNSX Last Pass Output Routine (SXL000)

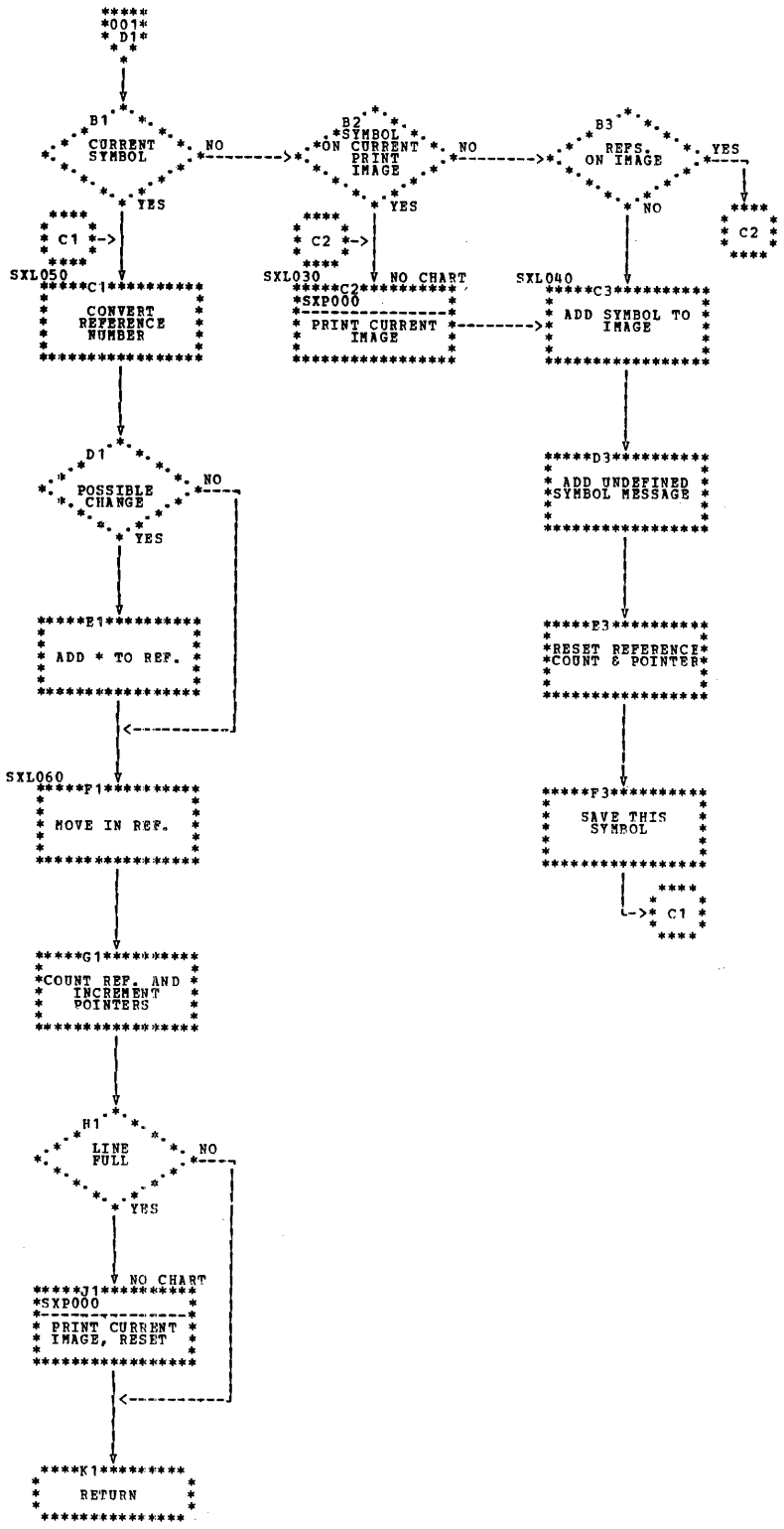


Chart ID (Part 2 of 2). $CGNSX Last Pass Output Routine (SXL000)

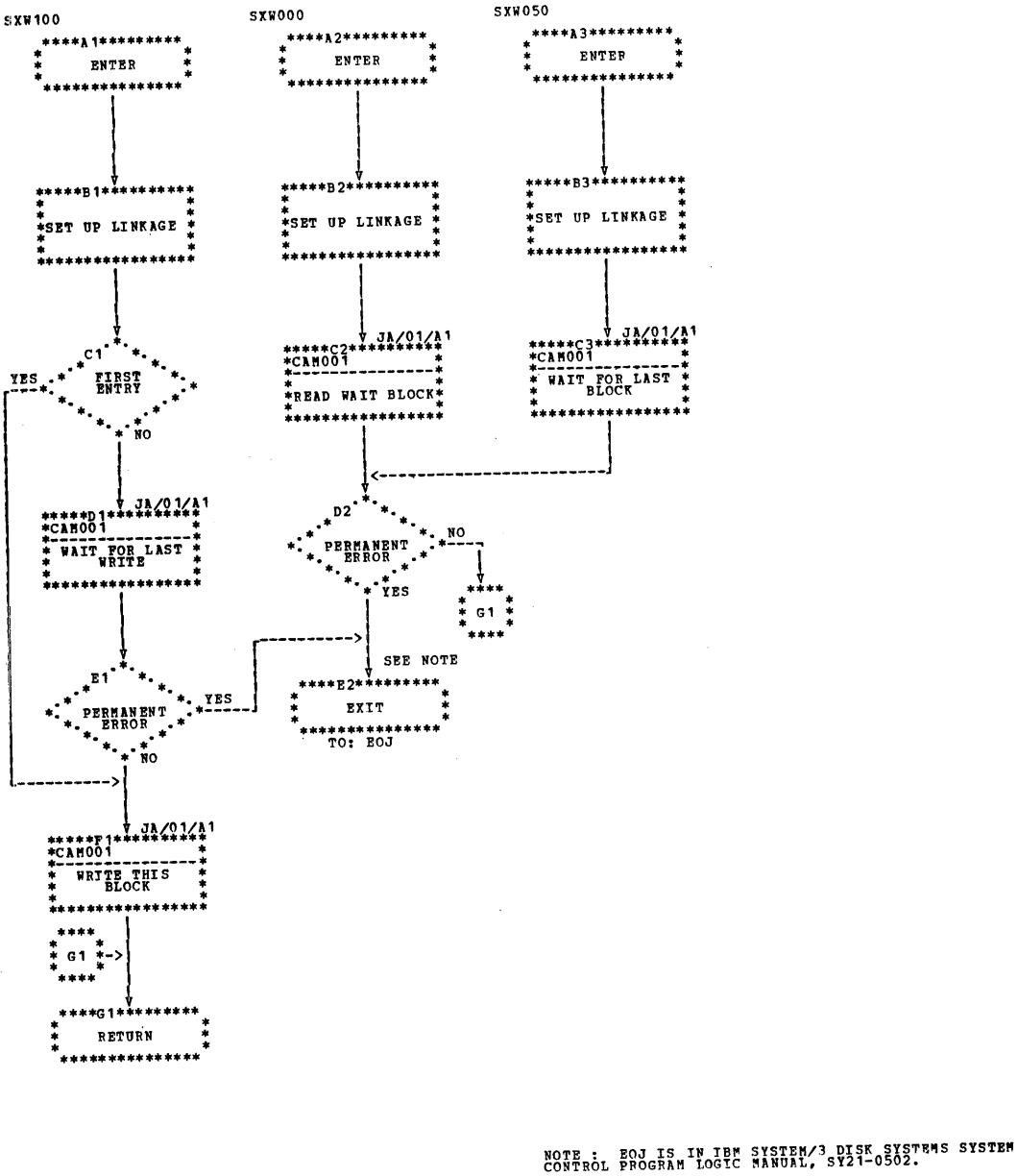


Chart IE. $CGNSX Disk Data Management Interfaces (SXW000)

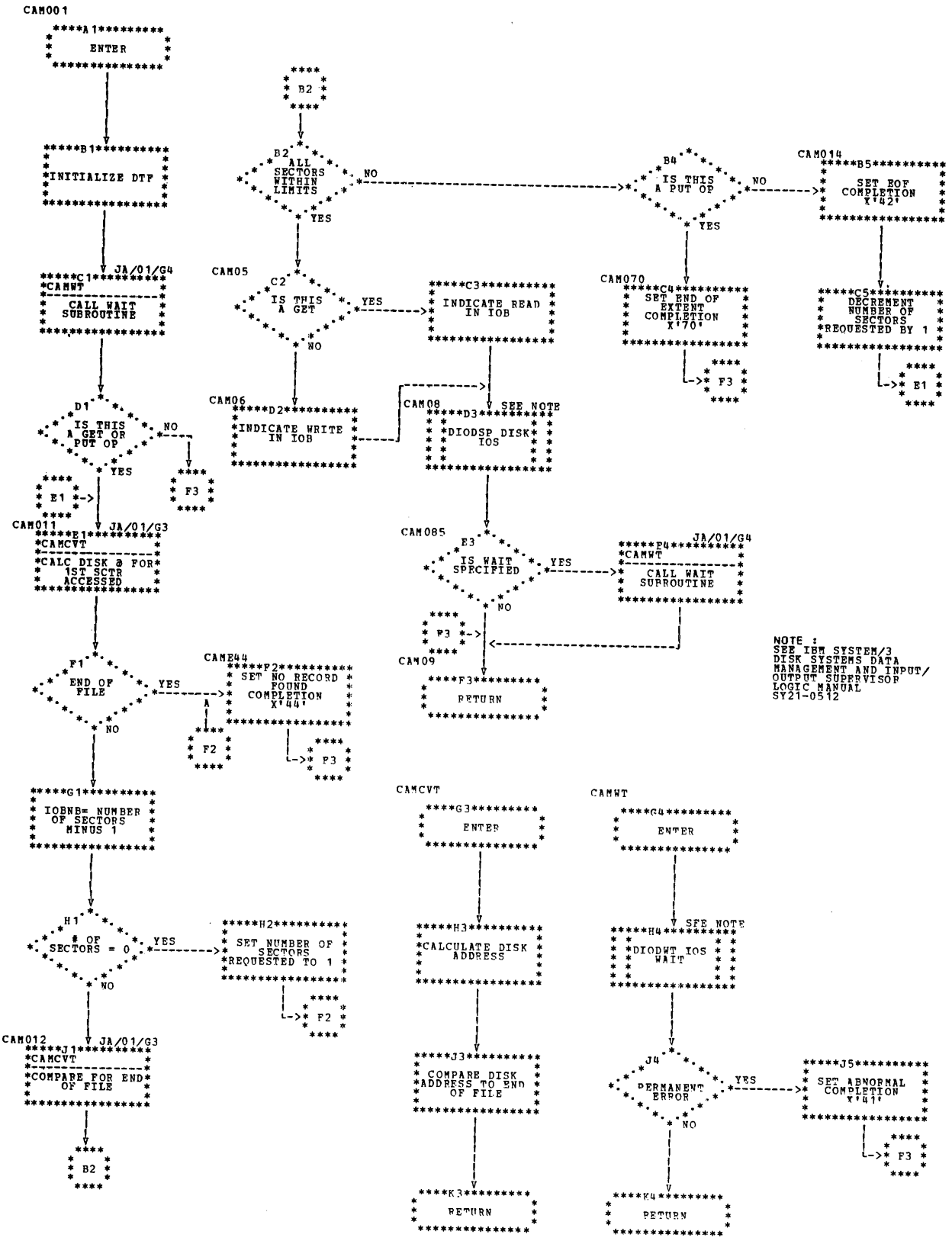


Chart JA. Compiler Access Method (CAM001)

Chapter 4. Assignment Build

Introduction

The Assignment Build Program ($CCPAS) operates under control of System/3 Disk System Management and requires 20K of main storage for execution. It is run to create, replace, modify, or delete CCP assignment sets in the Assignment File $CCPFILE (Figure 4-1). Any combination of these actions may be requested in a single run of the program. The Assignment File, allocated and initialized by CCP Generation, must exist before running Assignment Build. The file must contain a CCP configuration record and file directory.

Method of Operation

When the user loads Assignment Build, the mainline phase ($CCPAS) receives control. The mainline phase verifies that $CCPFILE is usable, loads the Syntax Scan Routine ($CC2SS), processes the SET statement, controls loading and execution of all overlay phases, handles all I/O operations, and handles logging of statements and messages and going to end of job.

The Syntax Scan Routine ($CC2SS) is called by the appropriate statement processor to scan the input control statement, issue messages if required, and modifies the keyword parameter list (contained within the calling statement processor) with the parameter values contained in the control statement.

The SET statement must be the first control statement processed and specifies the ACTION to be performed by Assignment Build. The ACTION parameter also determines the required sequence for the remaining control statements. There are four possible assignment actions:

- ACTION-CREATE. Create a new assignment set as specified by the remaining control statements and place the new set in $CCPFILE following the last existing set.
- ACTION-REPLACE. The REPLACE action is similar to CREATE except that before the end-of-set phase places the new set in $CCPFILE, it first deletes the replaced set and moves up all sets that follow the replaced set to occupy the freed space. Then, as with CREATE, the new set is placed after the last existing set.
- ACTION-DELETE. Delete an assignment set from $CCPFILE and move up all sets following the deleted set to occupy the freed space.
- ACTION-SYSMOD. Modify an existing assignment set System Information Table. (The modified table occupies the same space in $CCPFILE as it did before the action was performed.)

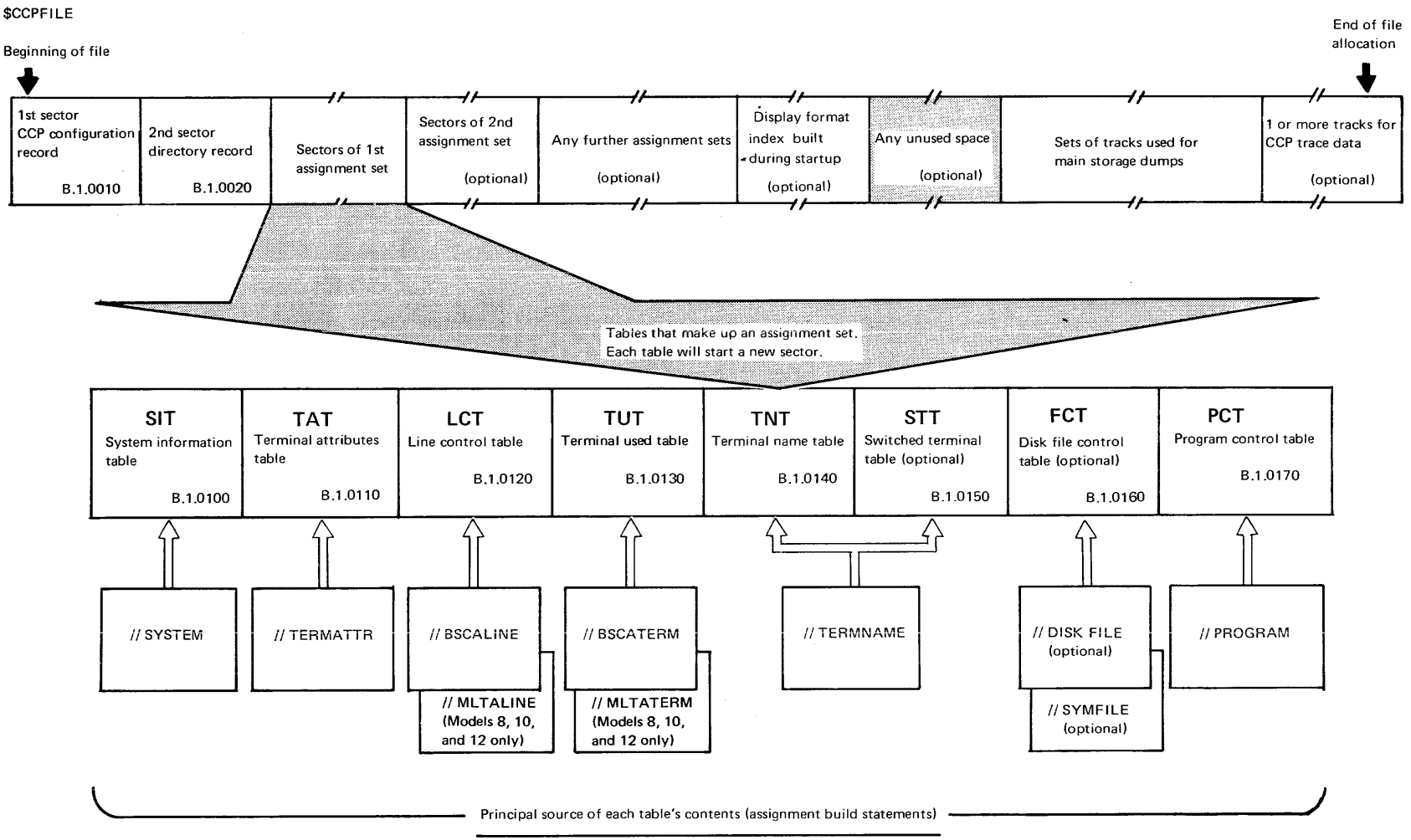


Figure 4-1. Disk Layout of $CCPFILE

Any combination of assignment actions may be performed with a single execution of Assignment Build. Diagram 4M.0100 shows normal module and data flow of Assignment Build. Figure 4-2 shows the main storage lay-out of Assignment Build.

CONTROL STATEMENT PROCESSORS

Each control statement, except the SET and SOURCE statements is processed by its individual overlay statement processor. The order of loading and executing the various statement processors is determined by the sequence of the input control statements. Only one overlay statement processor will exist in main storage at a time. The control statements and their processors are listed below.

Program Organization

Each statement processor processes one or more statements, performs diagnostics on each statement, and builds its associated table in $CCPWORK and a compressed version of the same table in the main storage table save area. Each table built in main storage will be used by one or more succeeding phases for diagnostics and for determining various parameter defaults and values as required.

For a CREATE or REPLACE action, all tables of the assignment set are built in $CCPWORK to ensure the set is error free before $CCPFILE is modified. For a SYSMOD action the SIT table is modified in the disk I/O buffer and then, if the SIT table is error free, written to $CCPFILE.

Control Statement	Processor	Diagram	Associated Tables
// SET	$CCPAS	4P.0140	None
// SYSTEM	$CC2SY	4P.0300	SIT (System Information Table)
// TERMATTR	$CC2TA	4P.0400	TAT (Terminal Attributes Table)
// BSCALINE	$CC2BL	4P.0500	LCT (Line Control Table)
// BSCATERM	$CC2BT	4P.0600	TUT (Terminal Used Table)
// MLTALINE	$CC2ML	4P.0700	LCT (Line Control Table) (Models 8, 10, and 12 only)
// MLTATERM	$CC2MT	4P.0800	TUT (Terminal Used Table) (Models 8, 10, and 12 only)
// TERMNAME	$CC2TN	4P.0900	TNT (Terminal Name Table)
// DISK/SYMFILE	$CC2DF	4P.1000	FCT (File Control Table)
// PROGRAM	$CC2PG	4P.1100	PCT (Program Control Table)
// SOURCE	$CCPAS	4P.0110	None

DIAGNOSTICS

Three stages of diagnostics are performed on each input control statement:

- Preliminary diagnostics. Performed by the Mainline Statement Read routine ($CCPAS).
- Syntax checking. Performed by the Syntax Scan Routine ($CC2SS).
- Logic diagnostic. Performed by each statement processor.

Two types of messages may be issued during diagnostics: warnings, which allow processing of the set to continue, and termination errors, which allow only the read routine diagnostics and syntax checking to continue. After a termination error occurs, control statements continue to be processed but no logic checking is performed and the associated tables are not built in either the main storage table save area or in $CCPWORK. If termination error occurs during the processing of an assignment set, $CCPFILE is not altered by the requested action. Termination errors in one assignment action do not affect succeeding assignment actions.

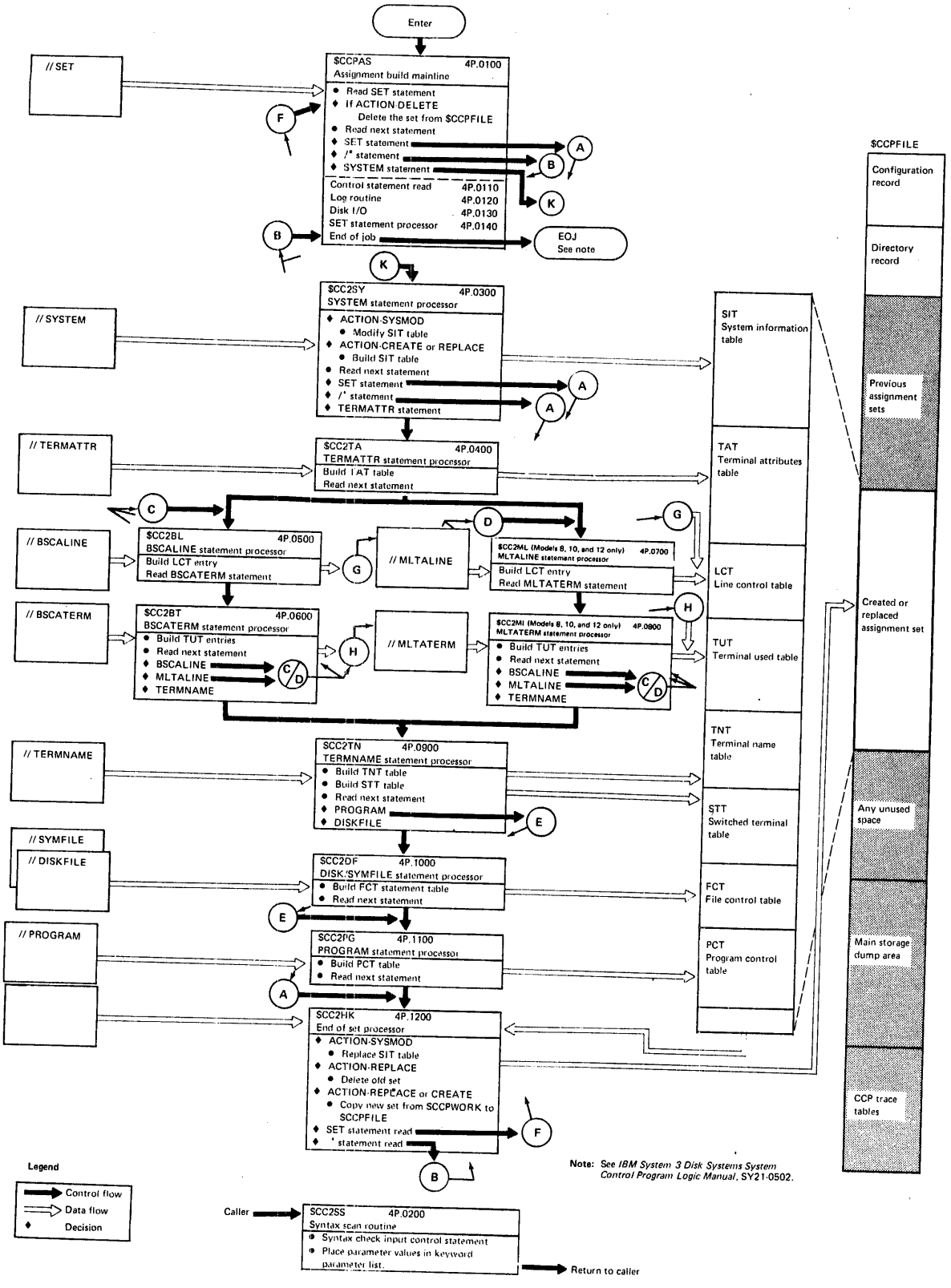
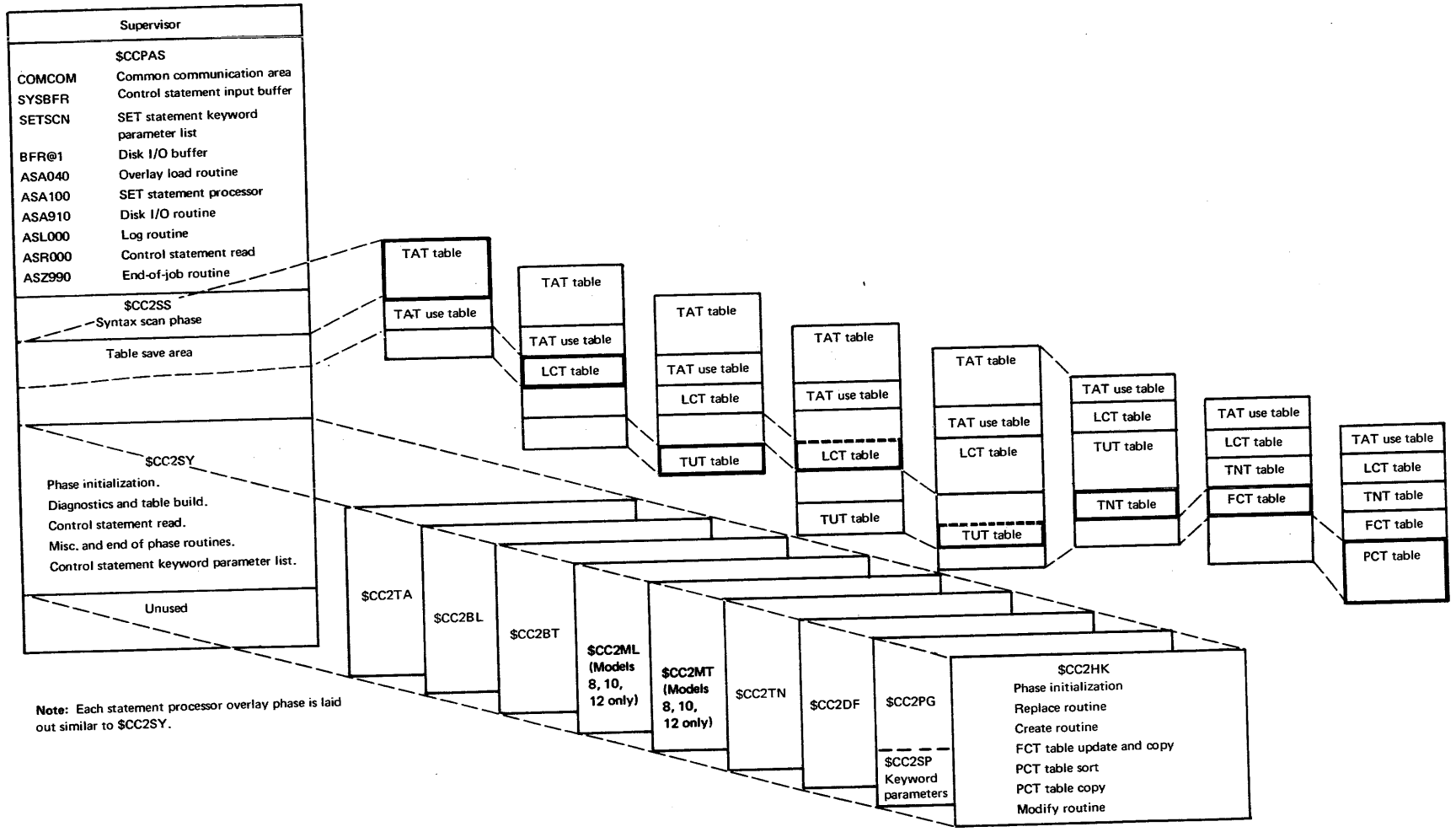


Diagram 4M.0100. Overall Module and Data Flow of Assignment Build



Note: Each statement processor overlay phase is laid out similar to $CC2SY.

Figure 4-2. Main Storage Layout of Assignment Build

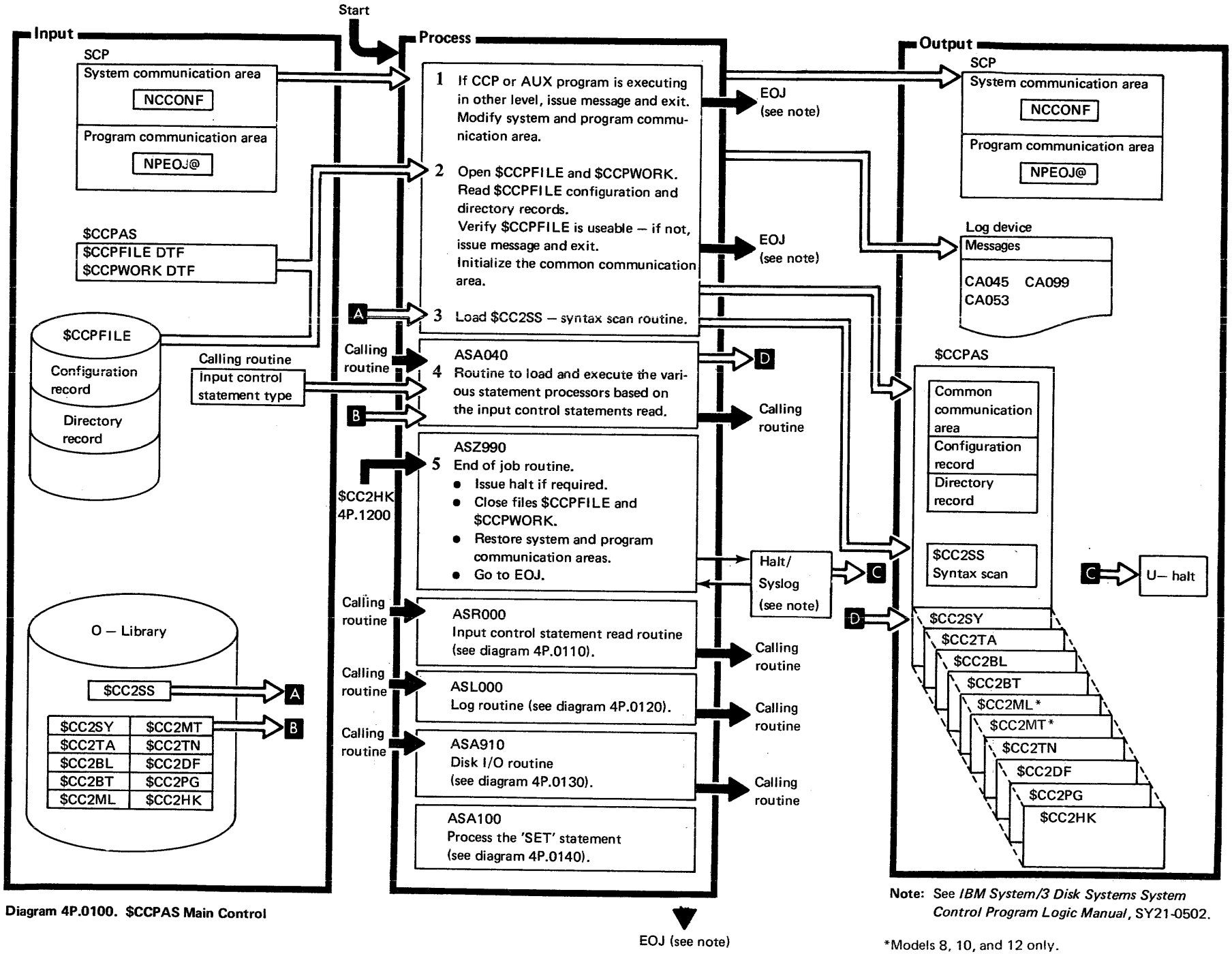
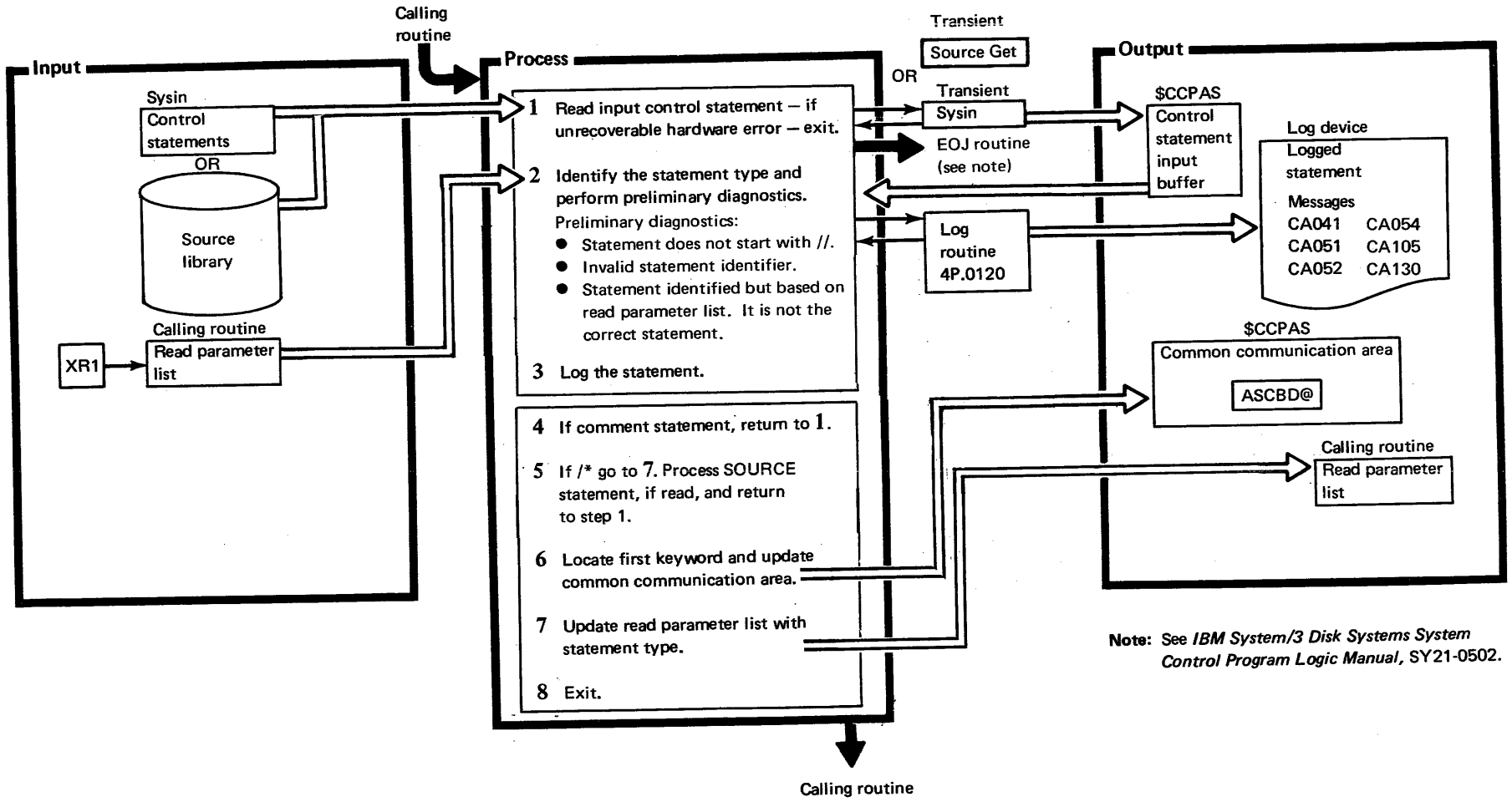


Diagram 4P.0100. $CCPAS Main Control

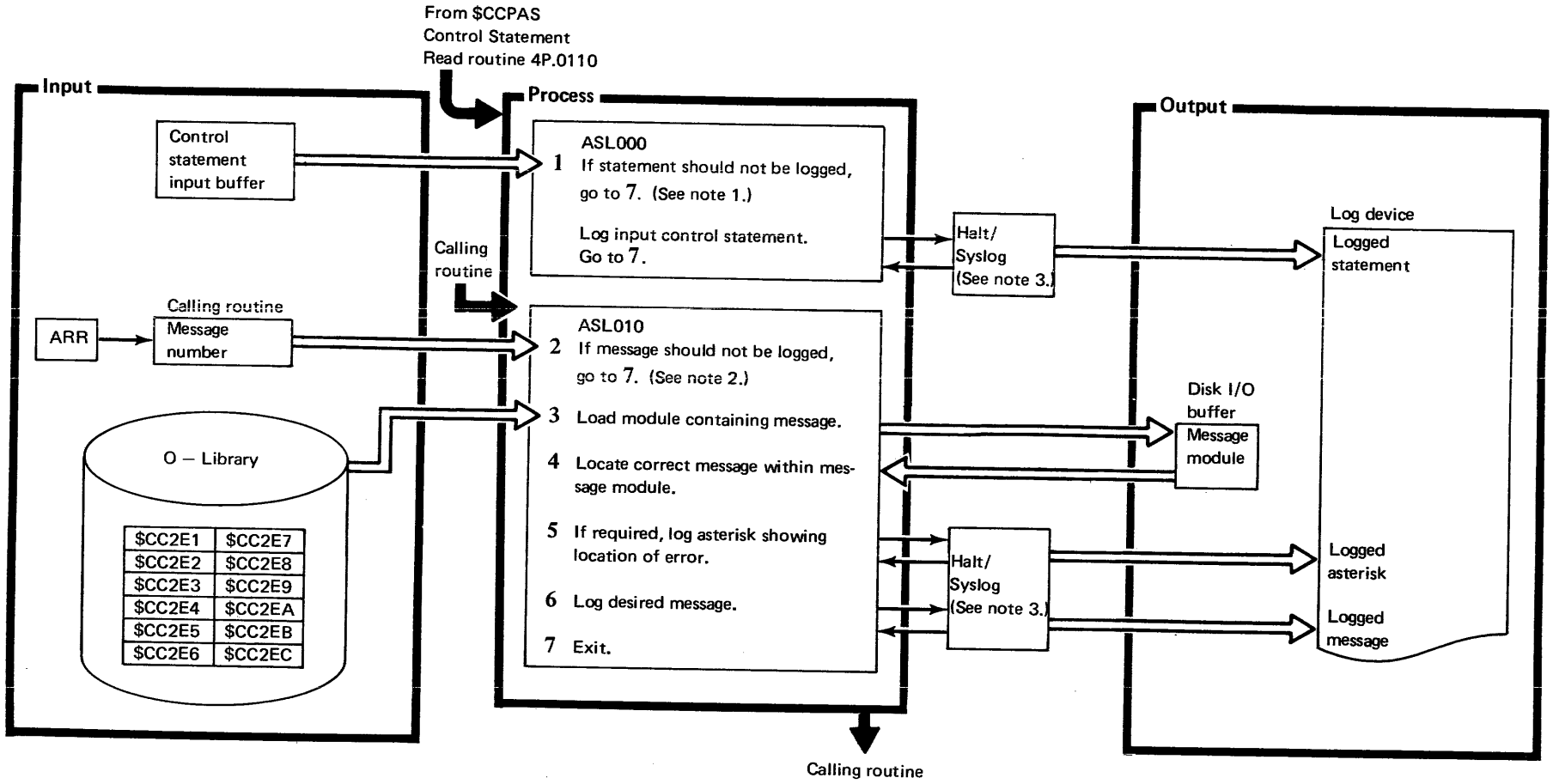
Note: See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.

*Models 8, 10, and 12 only.



Note: See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.

Diagram 4P.0110. $CCPAS Control Statement Read Routine

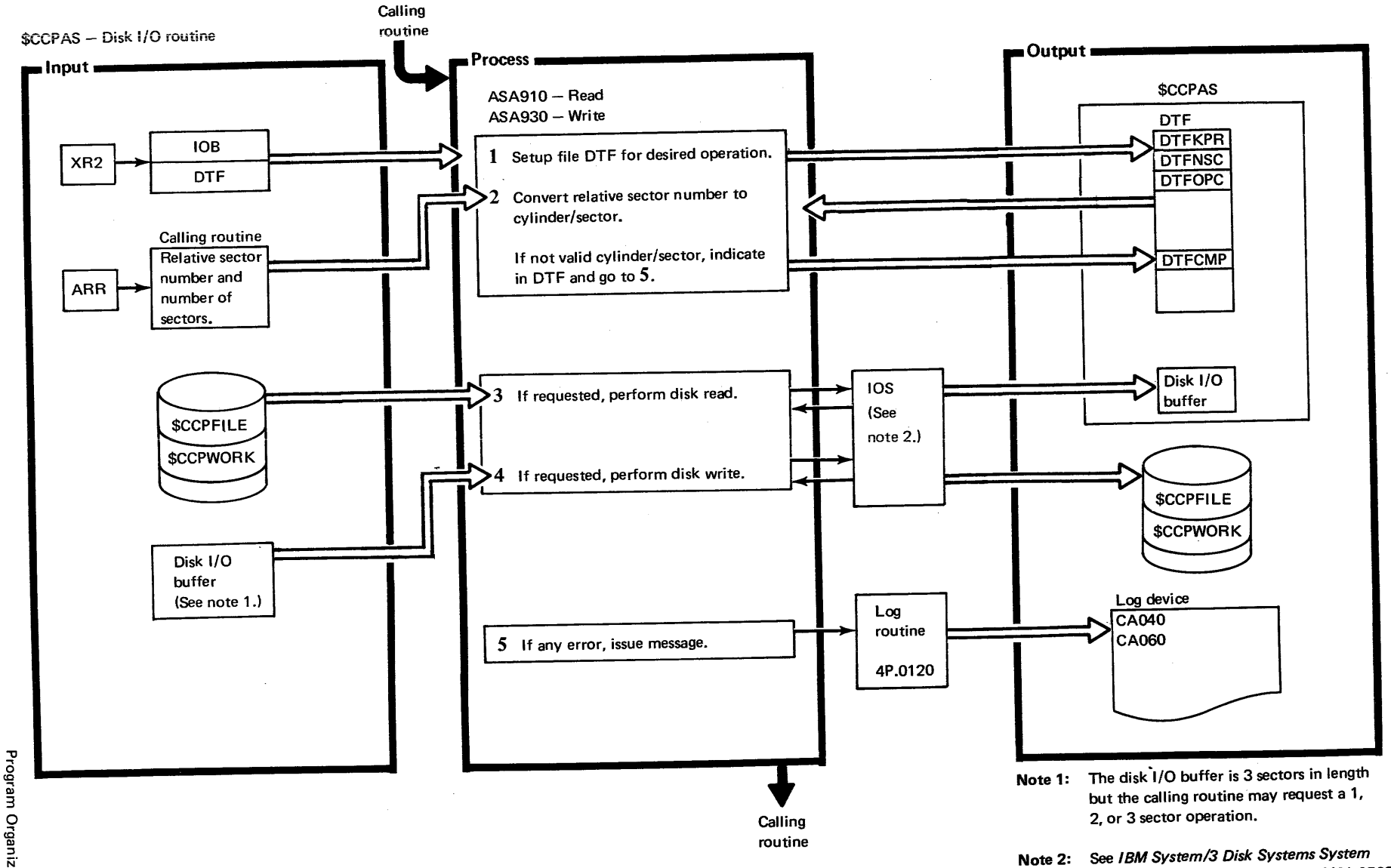


Note 1: Statements will not be logged if console is both sysin and syslog or if LOG is OFF.

Note 2: Asterisks and messages will not be logged if LOG is OFF.

Note 3: See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.

Diagram 4P.0120. $CCPAS Log Routine



Note 1: The disk I/O buffer is 3 sectors in length but the calling routine may request a 1, 2, or 3 sector operation.

Note 2: See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.

Diagram 4P.0130. $CCPAS Disk I/O Routine

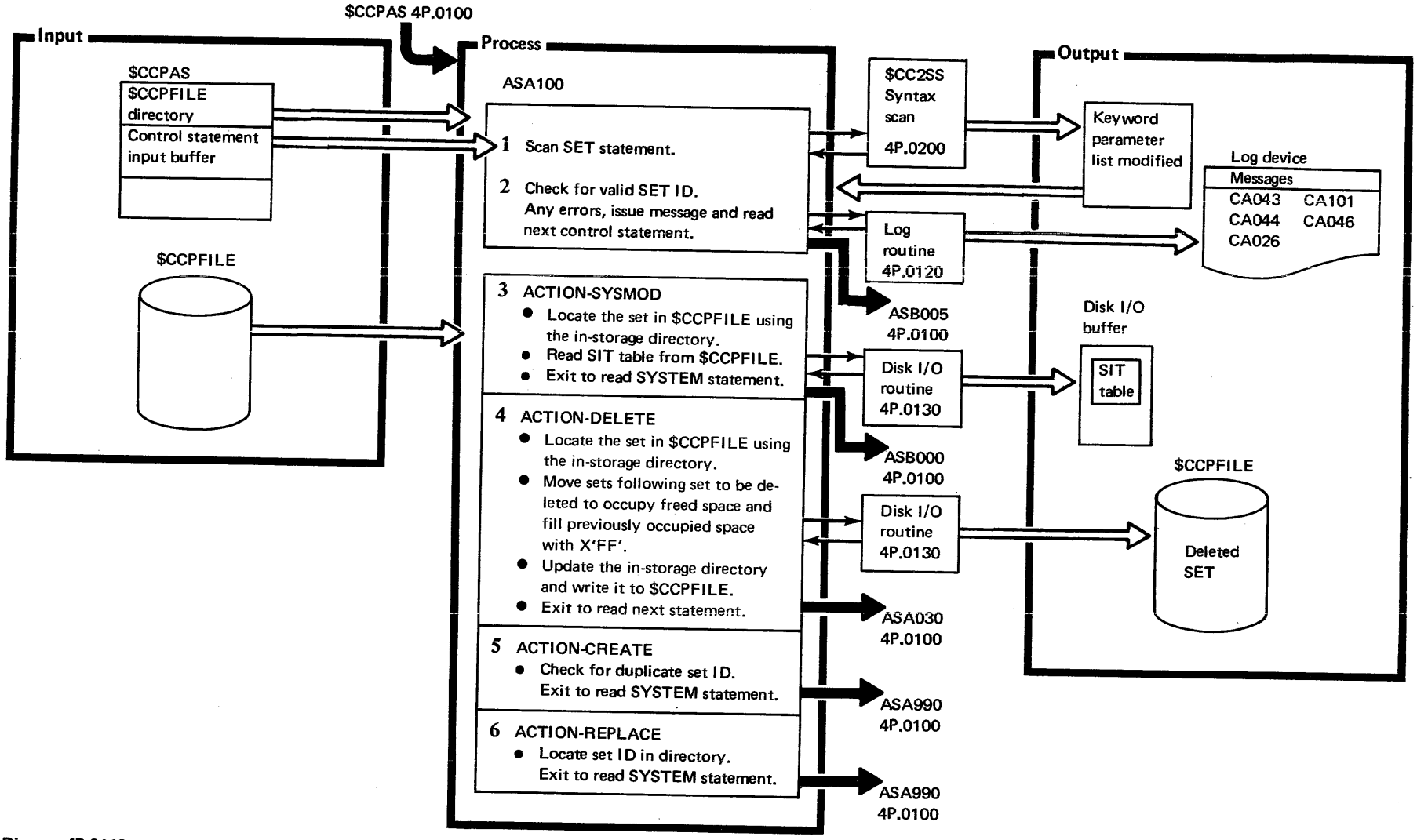
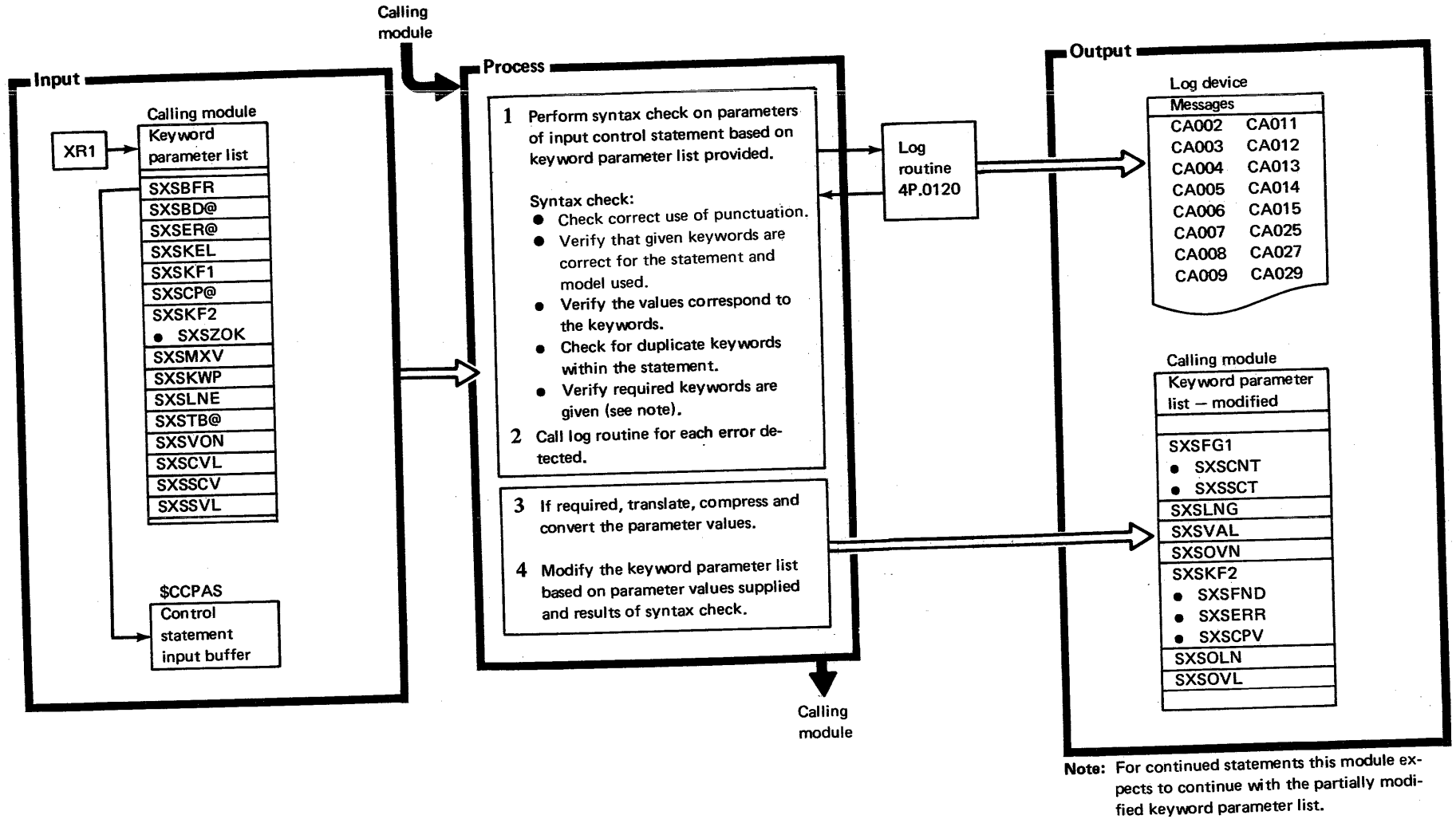
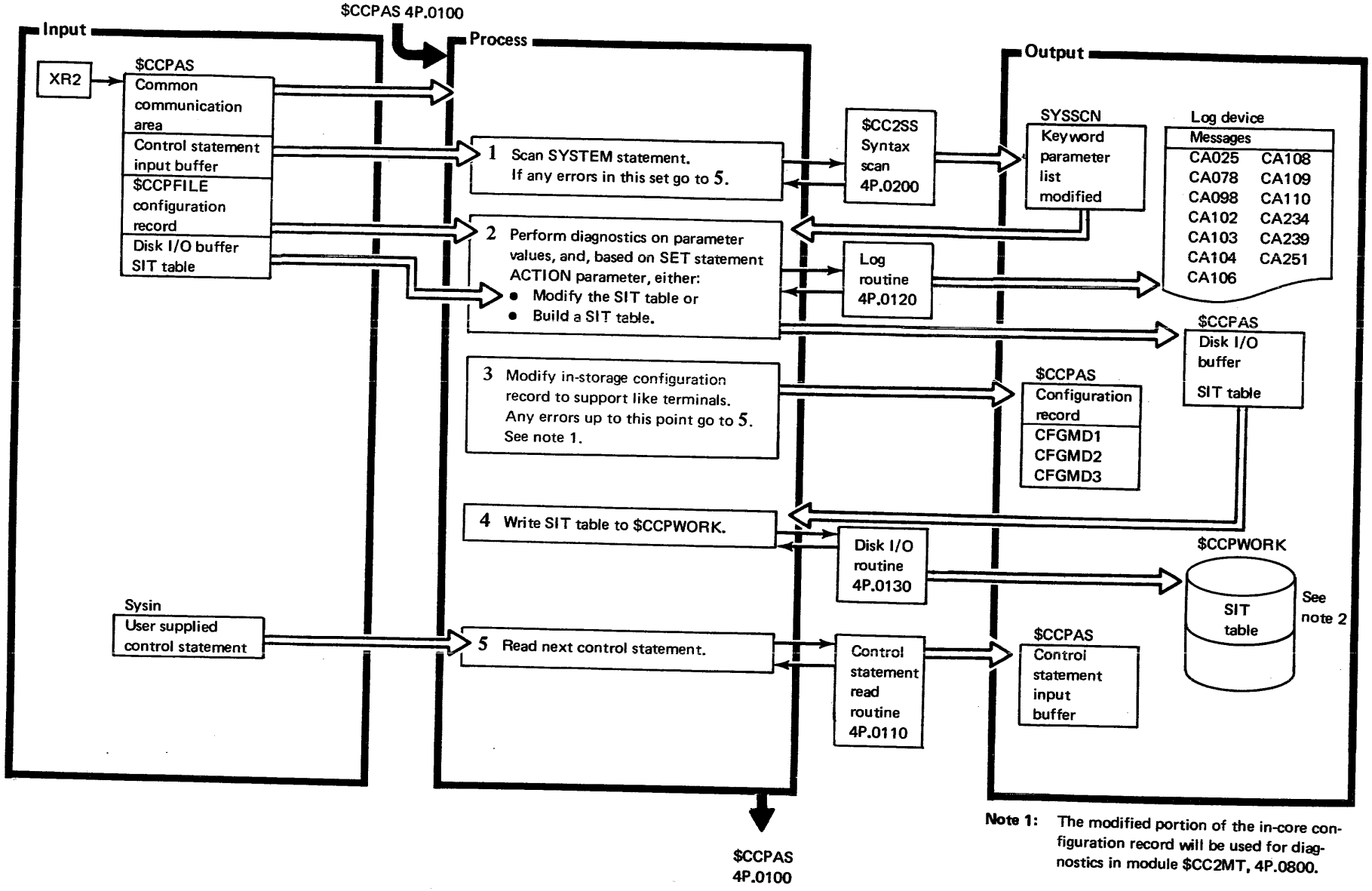


Diagram 4P.0140. SET Statement Processor



Note: For continued statements this module expects to continue with the partially modified keyword parameter list.

Diagram 4P.0200. $CC2SS



Note 1: The modified portion of the in-core configuration record will be used for diagnostics in module $CC2MT, 4P.0800.

Note 2: If the SIT table is built, additional information will be added to the SIT table by modules $CC2TA and $CC2HK.

Diagram 4P.0300. $CC2SY

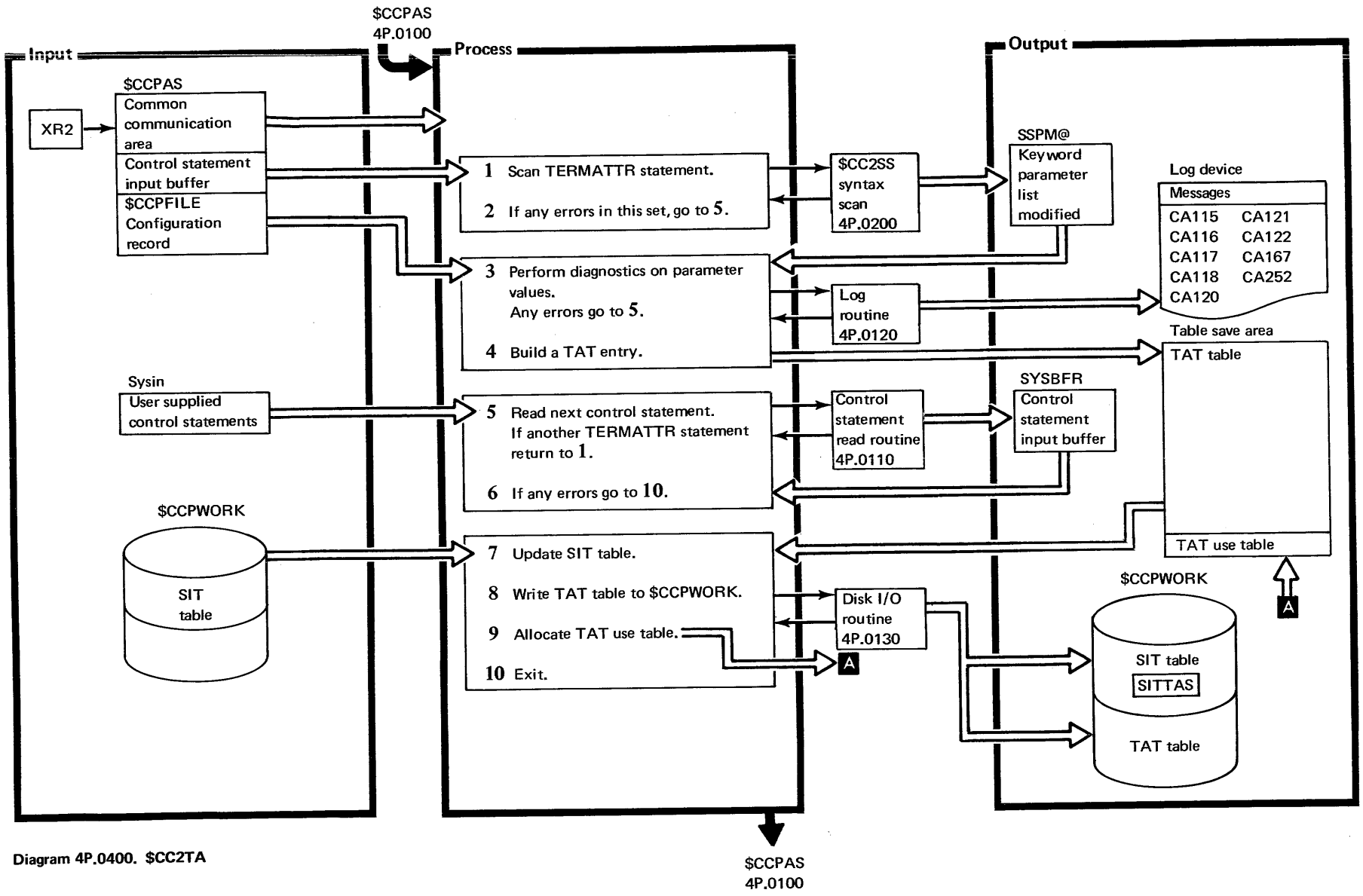
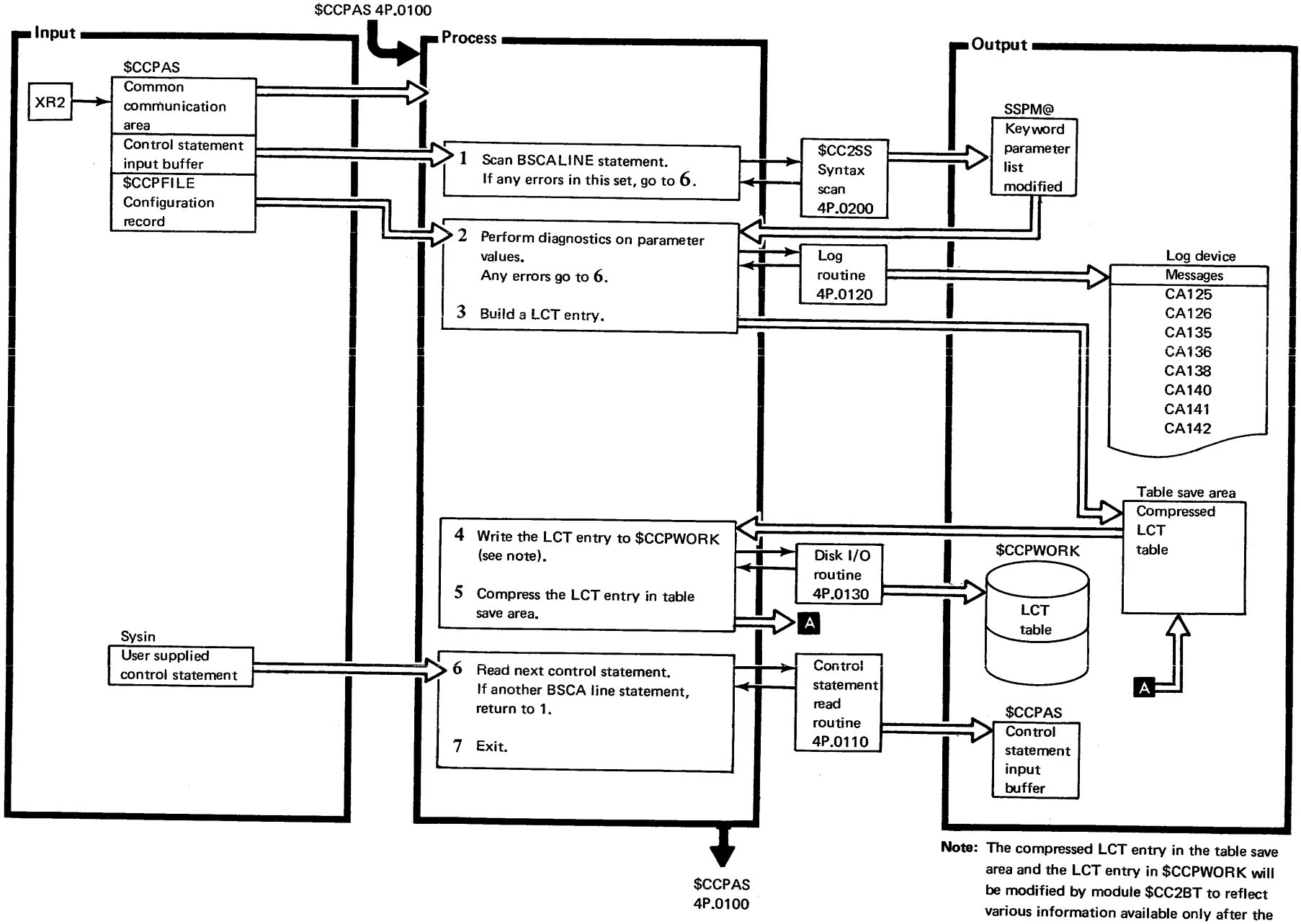


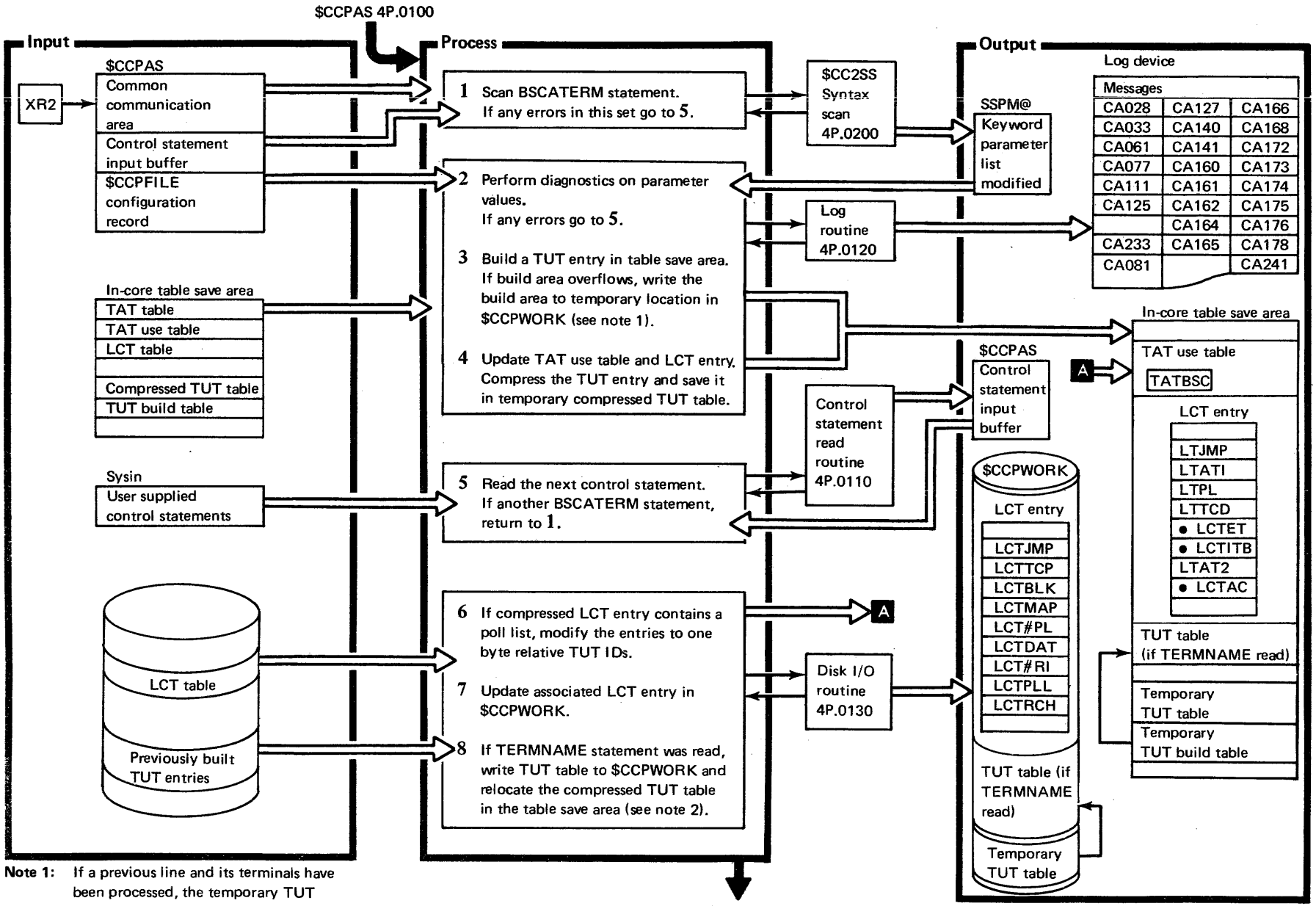
Diagram 4P.0400. $CC2TA

Page of SY21-0531-2
Issued 24 June 1977
By TNL: SN21-5530



Note: The compressed LCT entry in the table save area and the LCT entry in $CCPWORK will be modified by module $CC2BT to reflect various information available only after the BSCATERM statement has been processed. See chart 4P.0600.

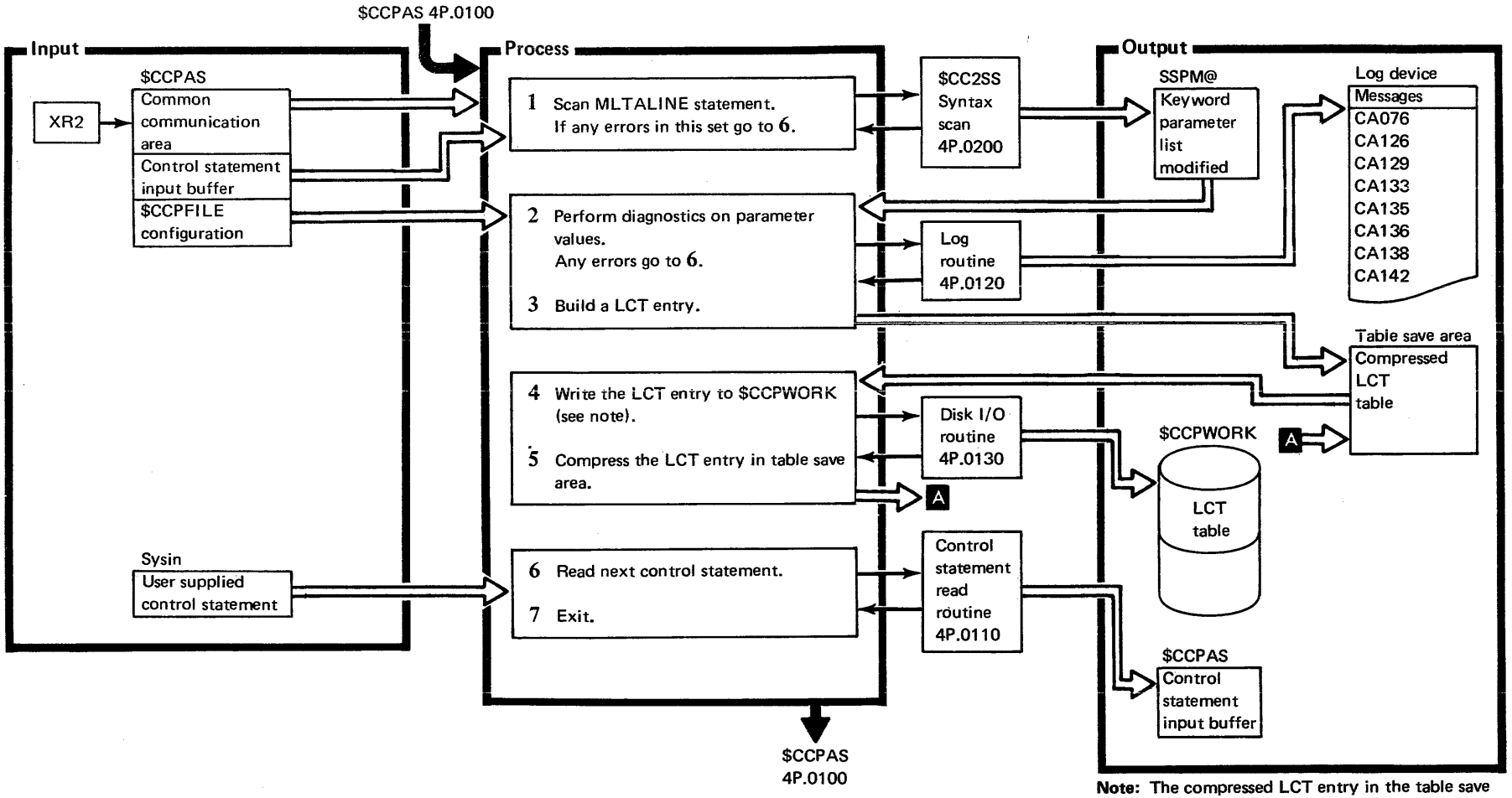
Diagram 4P.0500. $CC2BL



Note 1: If a previous line and its terminals have been processed, the temporary TUT tables will contain entries when entering this module. The entries may have been built by this module or by module $CC2MT.

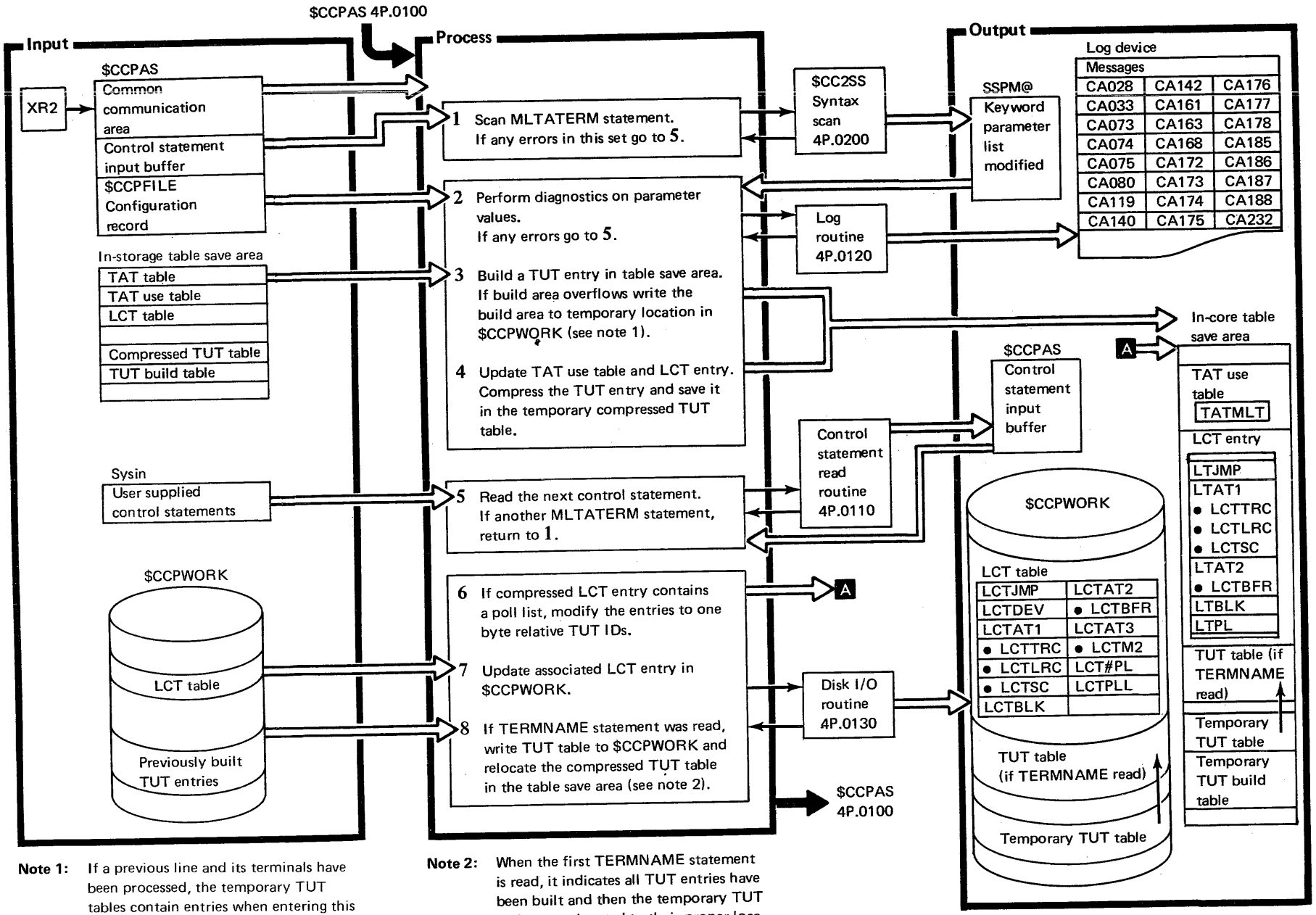
Note 2: When the first TERMNAME statement is read, it indicates all TUT entries have been built and the temporary TUT tables are relocated to their proper locations.

Diagram 4P.0600. $CC2BT



Note: The compressed LCT entry in the table save area and the LCT entry in $CCPWORK will be modified by module $CC2MT to reflect various information available only after the MLTATERM statement has been processed. See Chart 4P.0800.

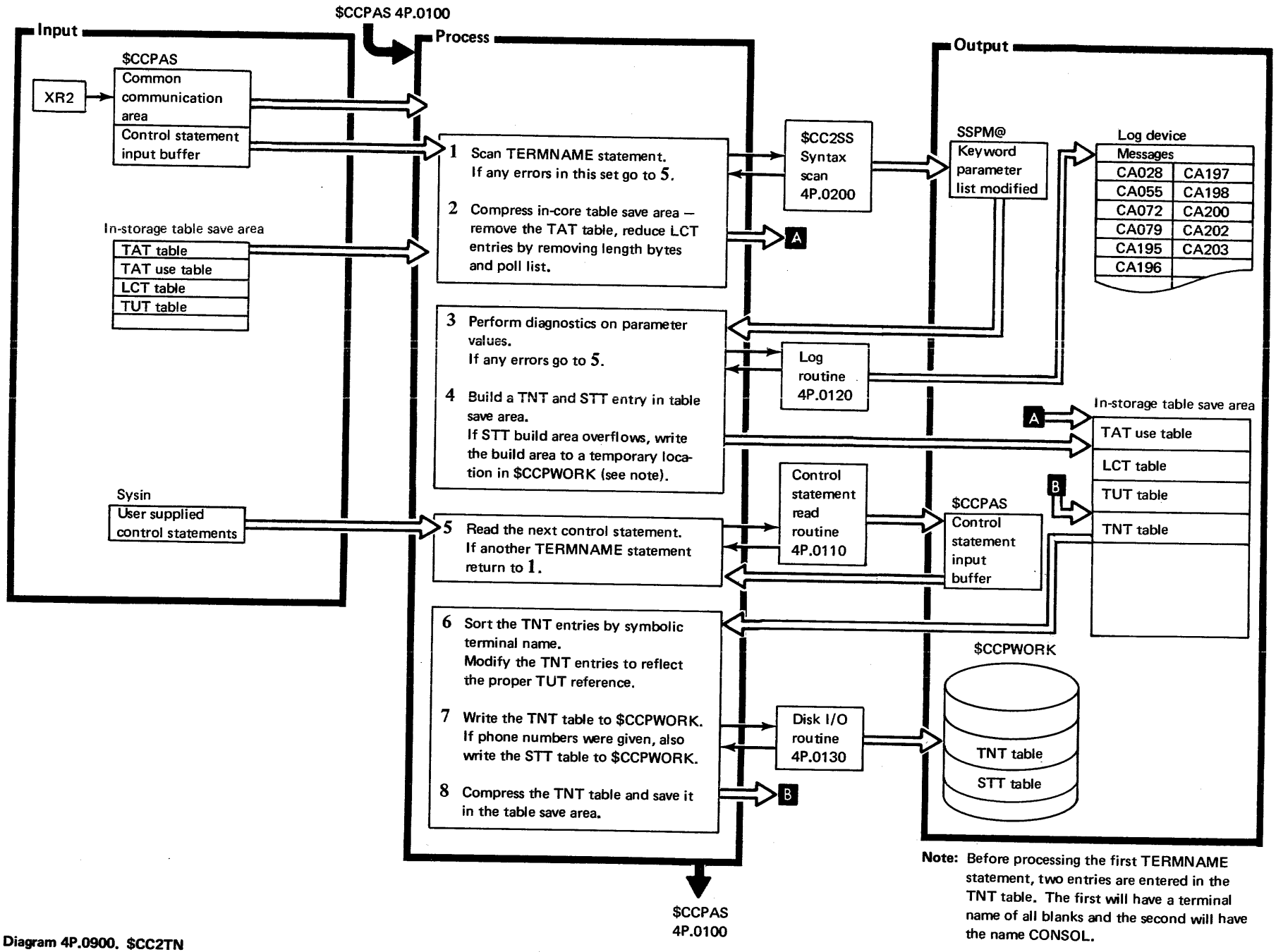
Diagram 4P.0700. $CC2ML (Models 8, 10, and 12 Only)



Note 1: If a previous line and its terminals have been processed, the temporary TUT tables contain entries when entering this module. The entries may have been built by this module or module $CC2BT.

Note 2: When the first TERMNAME statement is read, it indicates all TUT entries have been built and then the temporary TUT tables are relocated to their proper location.

Diagram 4P.0800. $CC2MT (Models 8, 10, and 12 Only)



Note: Before processing the first TERMNAME statement, two entries are entered in the TNT table. The first will have a terminal name of all blanks and the second will have the name CONSOL.

Diagram 4P.0900. $CC2TN

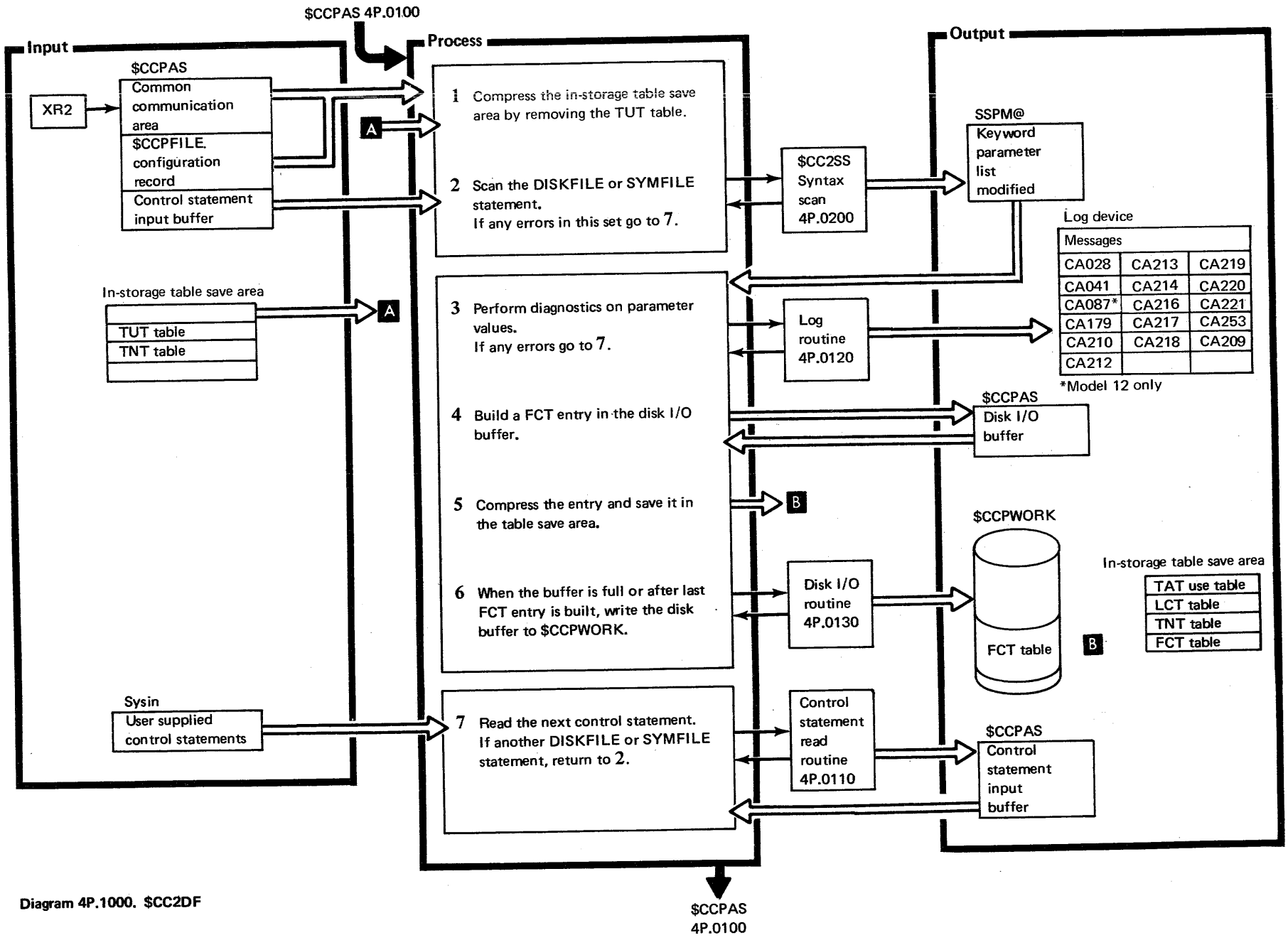


Diagram 4P.1000. $CC2DF

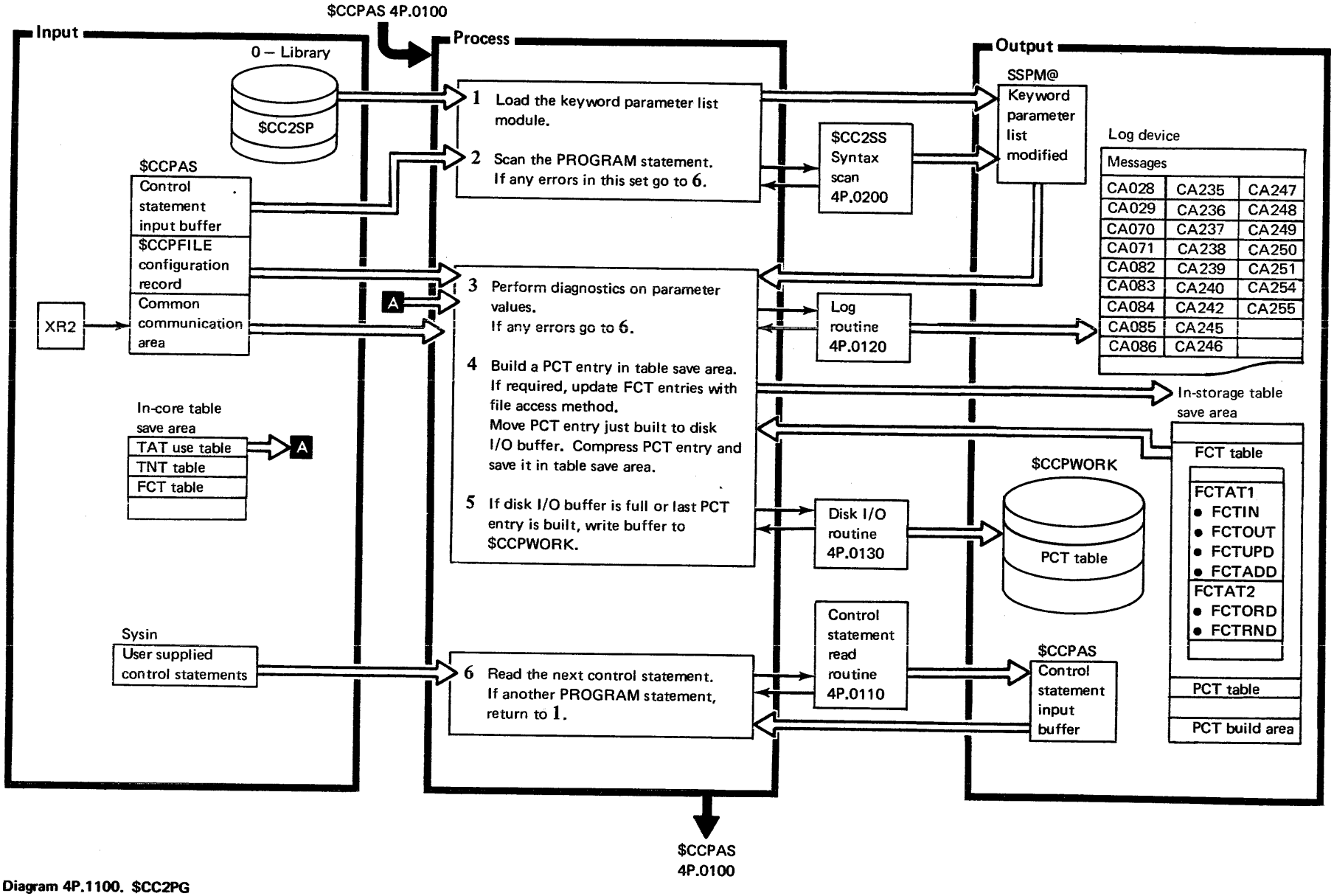


Diagram 4P.1100. $CC2PG

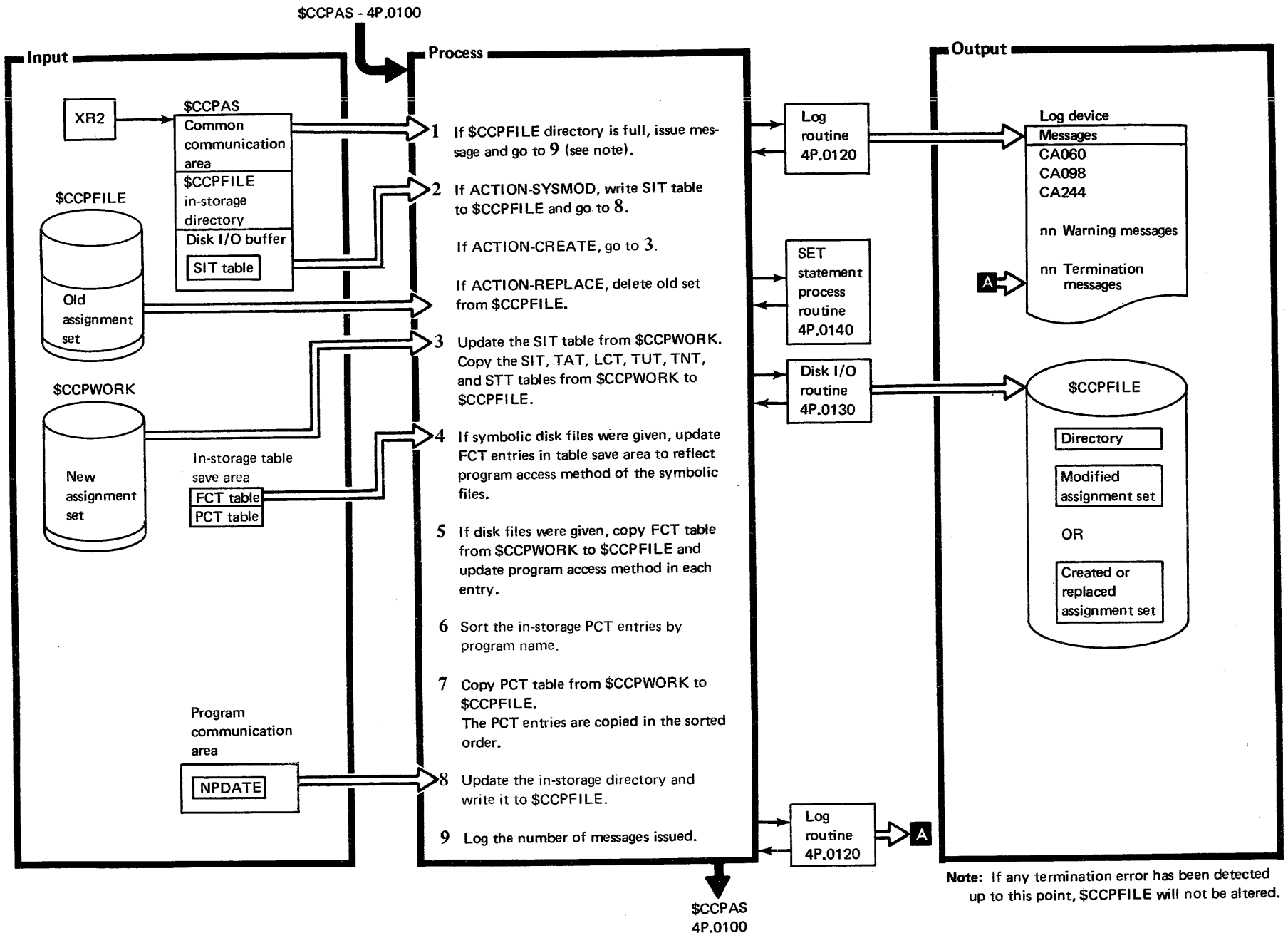


Diagram 4P.1200. $CC2HK

Chapter 5. Assignment List

Introduction

Assignment List ($CCPAL) is a stand alone program and requires 20K of main storage to execute. Assignment List runs under control of DSM and is distributed with CCP to:

- Display the Assignment File ($CCPFILE) configuration record.
- Display the Assignment File ($CCPFILE) directory.
- Display one particular set or all sets created by the Assignment Build Program ($CCPAS).
- Display and/or reset program request counts.

System requirements of Assignment List are:

- 5404 Processing Unit (64K of MOSFET memory)
  or
  5410 Model A15 Processing Unit (24K of main storage)—Model 10
  or
  5412 Model B17 Processing Unit (48K FET memory)—Model 12
- 5444 Model 2 Disk Storage Drive—Model 10
  or
  5447 Model 1 Disk Storage Drive—Model 4
  or
  3340 Model C2 Direct Access Storage Facility—Model 12
- 5471 Printer-Keyboard
- 5203 or 5213 or 1403 Printer

Method of Operation

Program Initialization (Diagram 5M.0100 and Figures 5-1 and 5-2)

After it is loaded, $CCPAL opens $CCPFILE, saves the $CCPFILE configuration record (ALCFG) and directory (ALDIR), loads the System Print routine ($$SYP1) into main storage, opens the printer, and sets the date (0-$$SYD2) constant.

Functional Determination (Diagram 5M.0100)

$CCPAL reads an input record containing the list specifications into main storage, logs the record on the system log device, and scans the record for syntax errors and the desired function. If a syntax error exists, an error message is logged and the next record is read. If no syntax errors exist, the requested function is performed.

$CCPFILE Configuration Display (Diagrams 5M.0100 and 5P.0100)

The $CCPFILE configuration is a one-page printout of the $CCPFILE configuration record. $CCPFILE can be printed only if CCP Generation (see Chapter 2) has been completed successfully. If the $CCPFILE is requested but CCP Generation was not successful, $CCPAL issues an error message and goes to end of job.

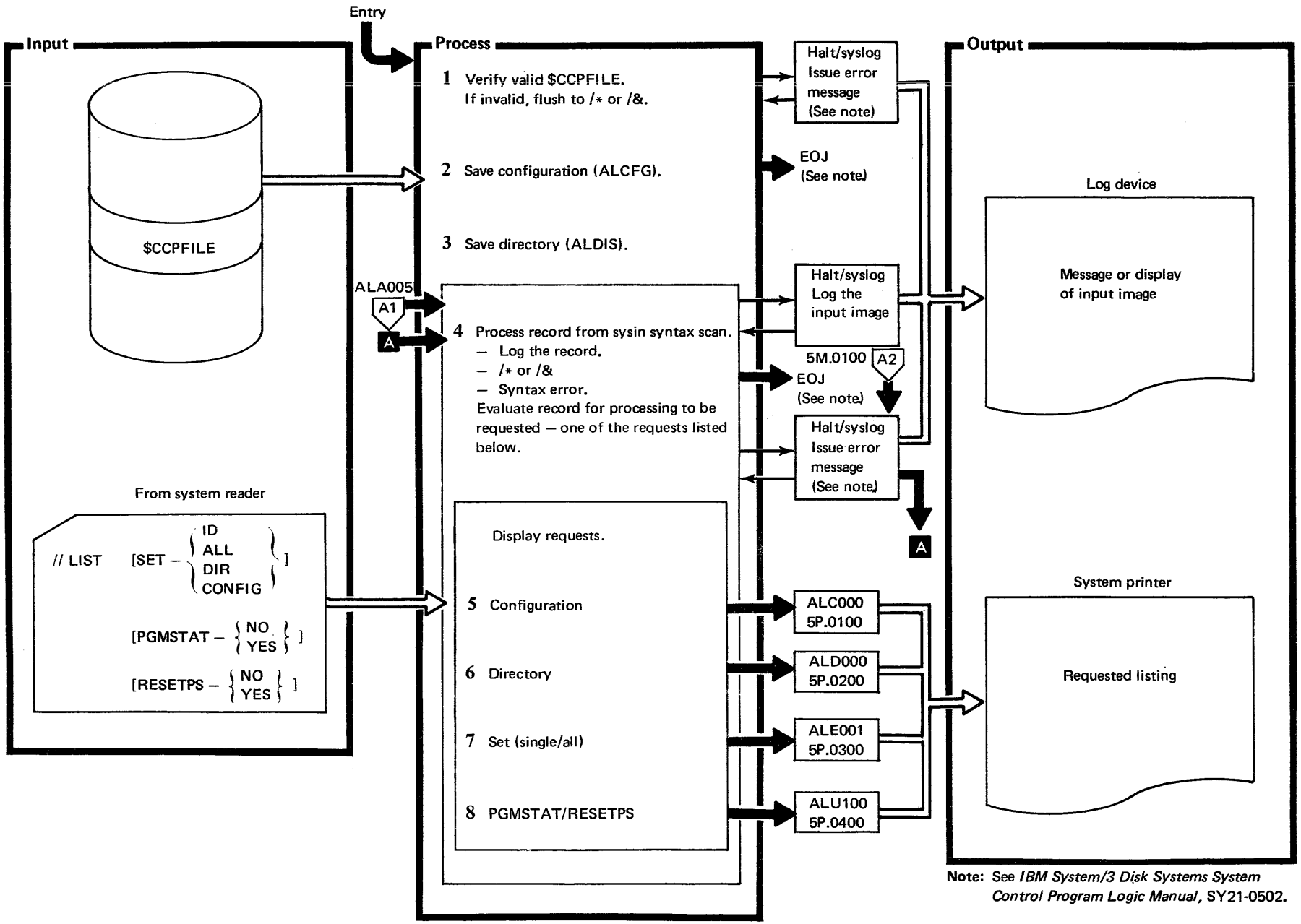
The $CCPFILE configuration display shows all the supported function of the $CCPFILE being displayed (these functions were determined during CCP Generation — see Chapter 2). Print lines in the display contain 80 print positions or less.

$CCPFILE Directory Display (Diagrams 5M.0100 and 5P.0200)

The $CCPFILE directory display is a one-page printout showing:

- The number of assignment sets in the $CCPFILE and the number of entries available.
- Data obtained from last CCP run using the $CCPFILE. (This is displayed only if there was a previous run.)
- Main storage size of the system for which the $CCPFILE is defined.
- Maximum number of dumps the $CCPFILE can contain.
- Start and end addresses of memory dump and trace areas. If the start and end addresses of the trace area are equal, no trace area will be shown.
- Execution time default set ID.
- Date of last $CCPFILE update.
- Each set ID in the $CCPFILE and the set length in sectors.

The output format of the print line will not exceed 80 print positions.



Note: See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.

Diagram 5M.0100. Overall Function of $CCPAL

Program Organization

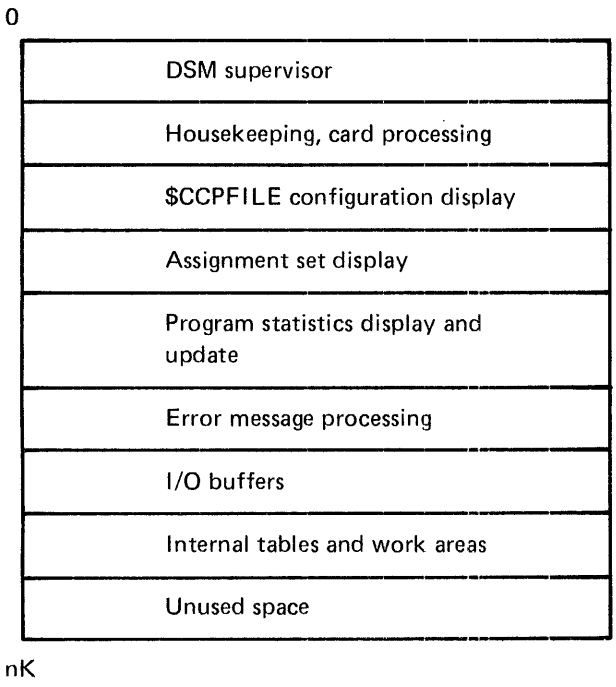


Figure 5-1. $CCPAL Main Storage Layout, General

$CCPAL Set Display (Diagrams 5M.0100 and 5P.0300)

The SET-ID and SET-ALL specifications cause display of one or all assignment sets in $CCPFILE.

Following is a list of the seven tables displayed:

- SIT (System Information Table)
- TAT (Terminal Attributes Table)
- LCT (Line Control Table)
- TUT (Terminal Used Table)
- TNT (Terminal Name Table)
- FCT (File Control Table)
- PCT (Program Control Table)

Some information displayed will not appear on any of the Assignment control statements but have been included as useful information. These values were calculated or defaulted by the Assignment Build Program.

Program Statistics (PGMSTAT) and Reset Program Request Counts (RESETPS) – (Diagrams 5M.0100 and 5P.0400)

$CCPAL contains code to perform two functions dealing with program statistics:

- Display programs statistics.
- Reset program request counts.

These functions are requested by the list specification statement. If both functions are specified, the request counts will be displayed before being reset to zero. If the request counts are to be reset and displayed, an additional comment will be added to the display indicating that the request counts have been reset. Unless PGMSTAT–YES is included on the specification statement, the request counts will not be displayed. $CCPAL will perform these functions for a particular set or all sets as specified on the specification statement.

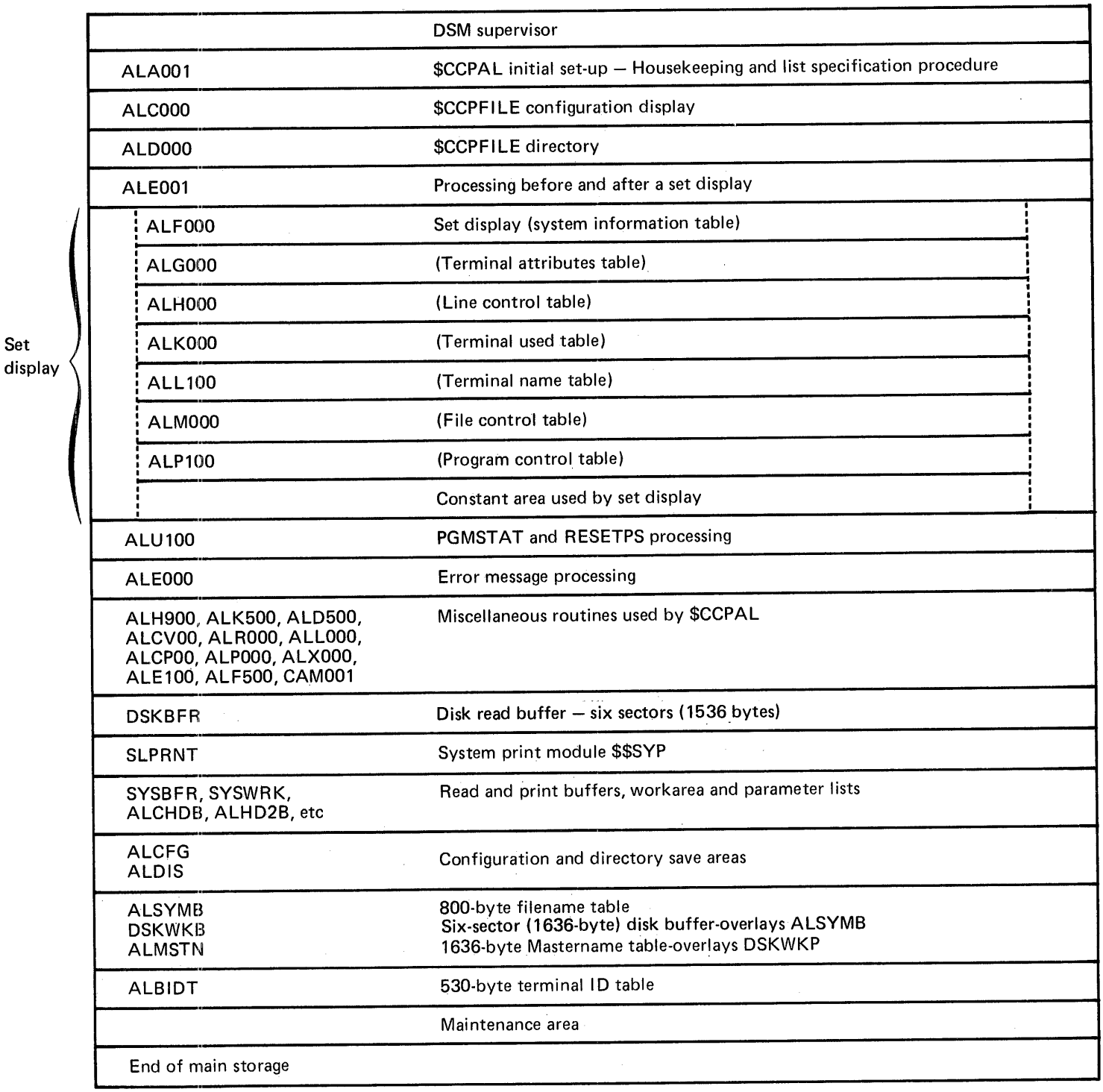


Figure 5-2. $CCPAL Main Storage Layout, Detail

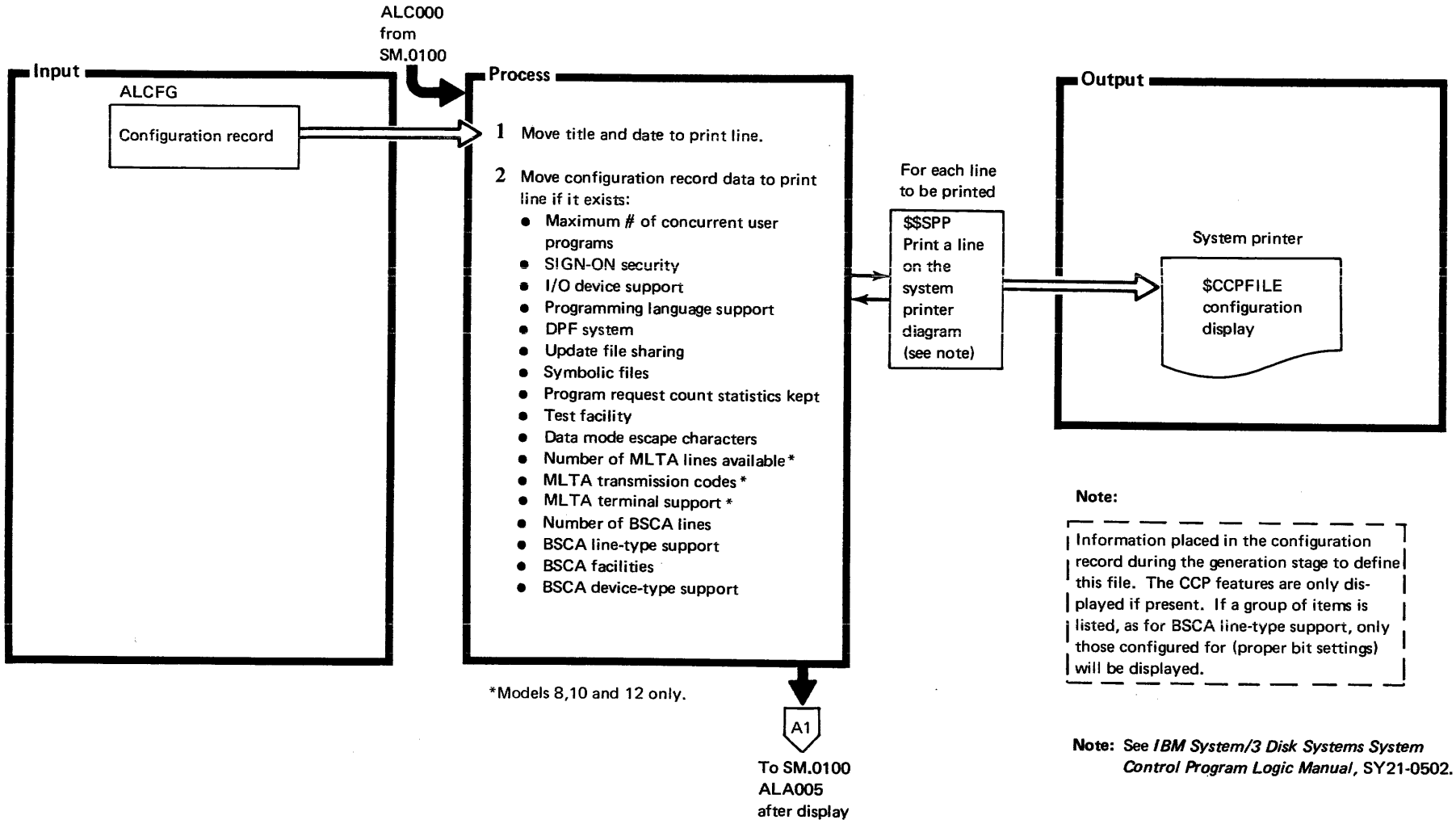


Diagram 5P.0100. Display of $CCPFILE Configuration

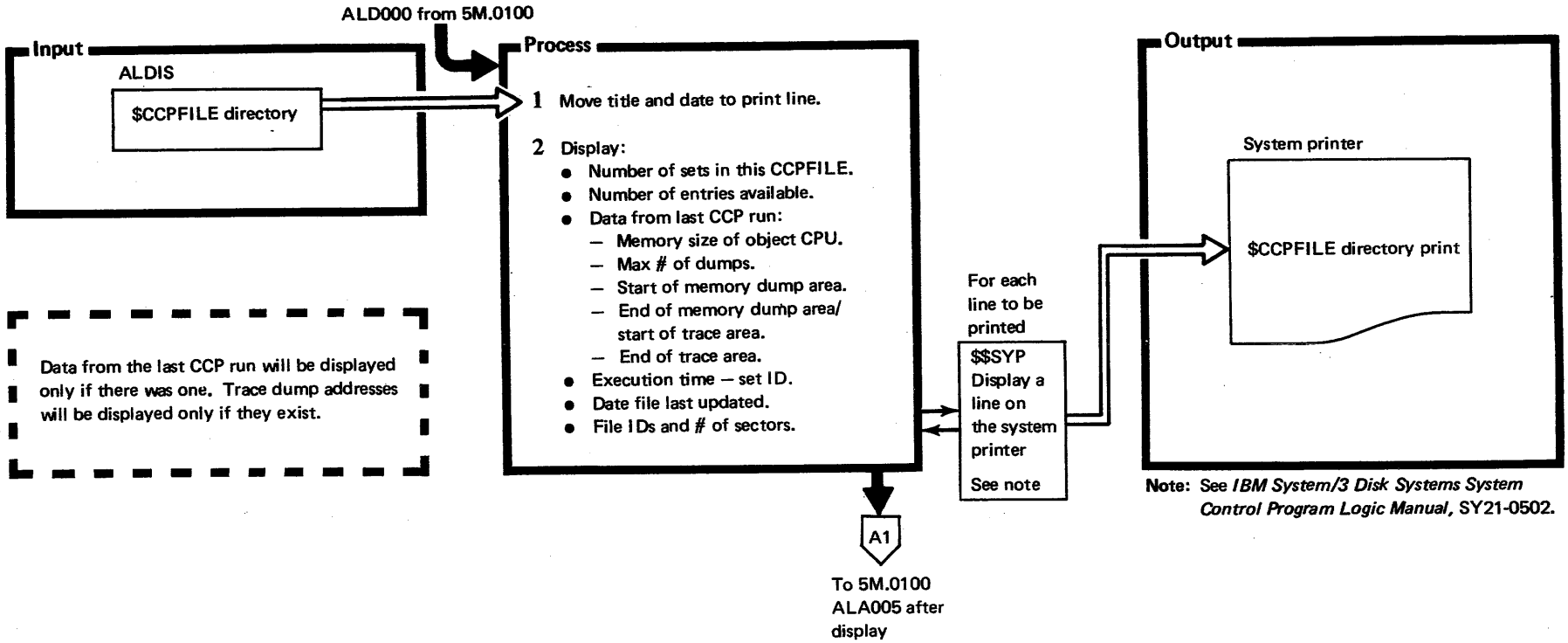
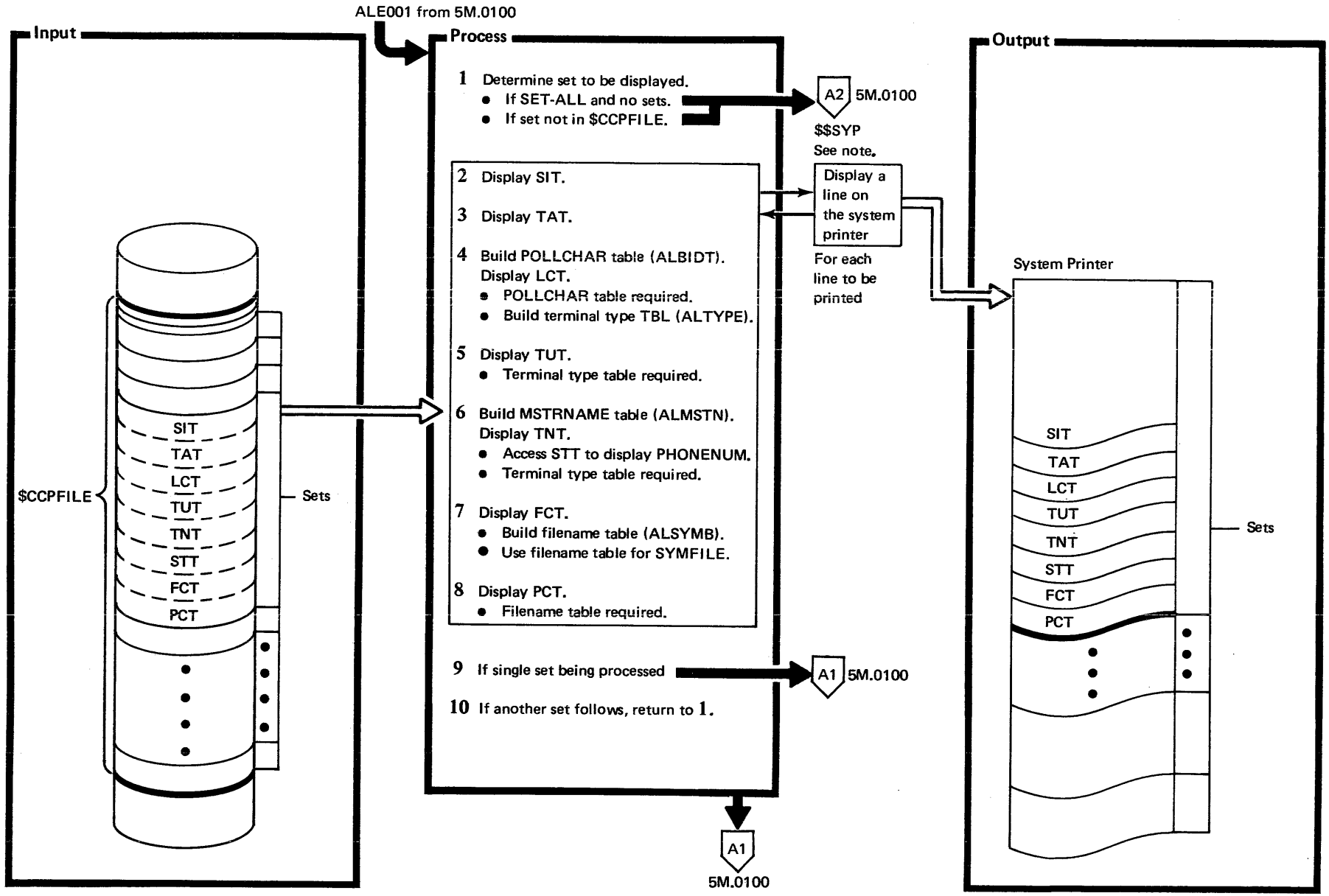
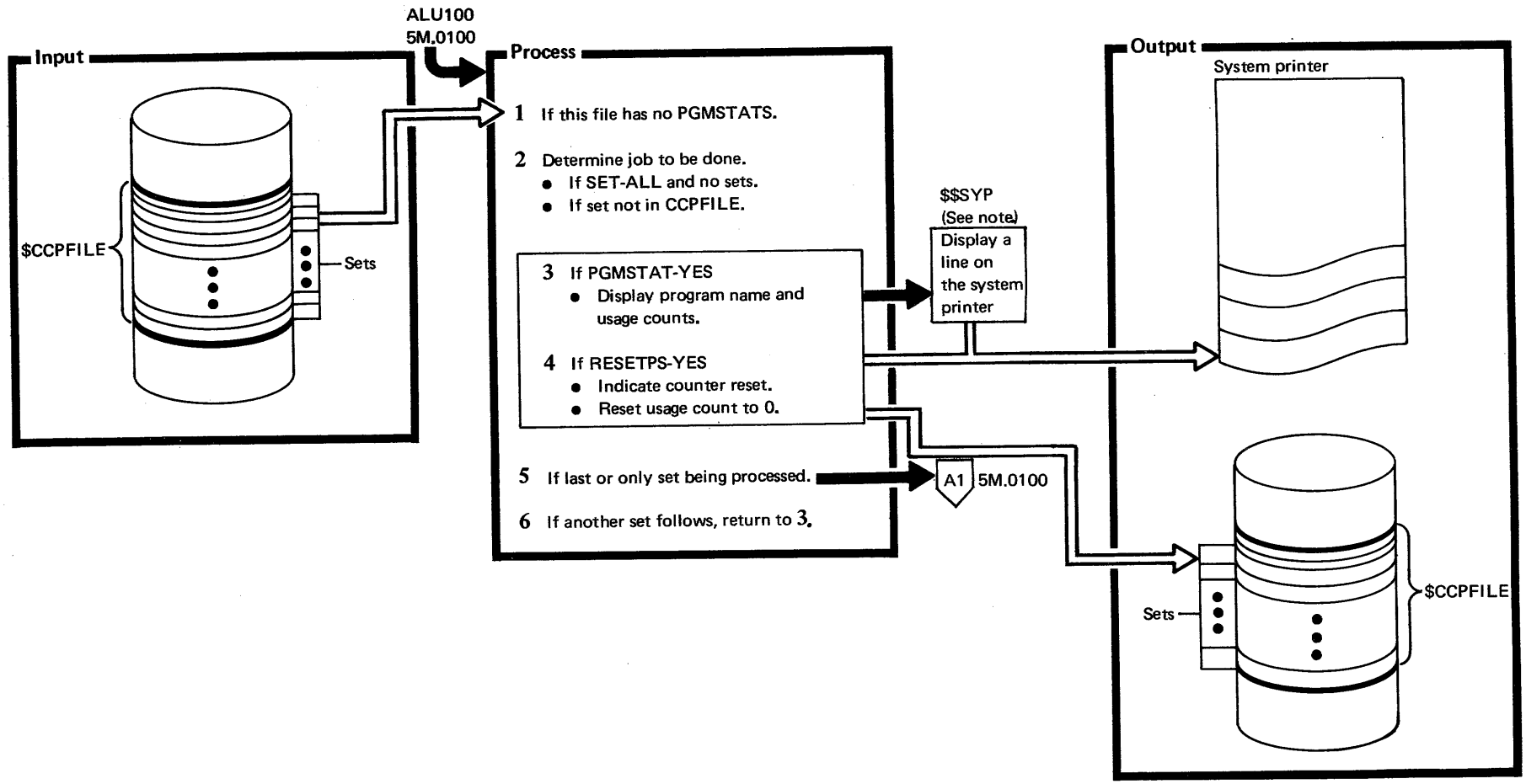


Diagram 5P.0200. Display of $CCPFILE Directory



Note: See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.

Diagram 5P.0300. Set Display Processing



Note: See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.

Diagram 5P.0400. PGMSTAT and RESETPS Processing

Chapter 6. User Security Information Build (Models 8, 10, and 12 Only)

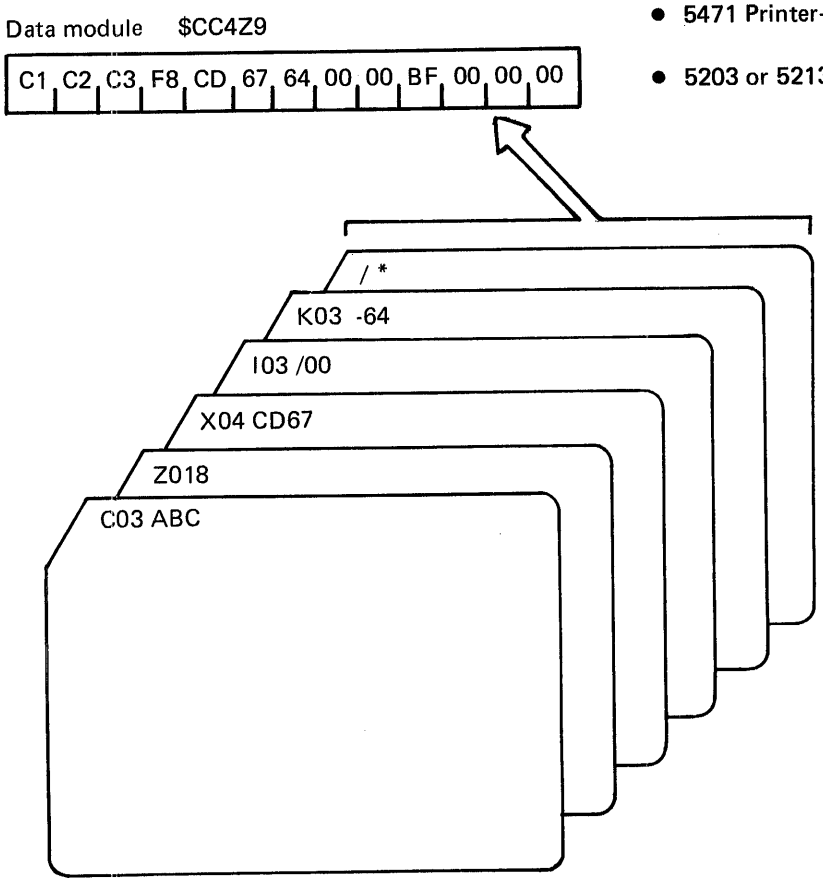
Introduction

The User Security Information Build Program ($CCPAU) is a stand-alone CCP support program that runs under control of DSM. $CCPAU is used to initialize and/or modify the contents of the user's security data module, $CC4Z9 (Figure 6-1).

$CC4Z9 is a dummy module created at CCP generation by the $ESEC generation statement. $CC4Z9 consists of a module name and an area defined by the LUSI keyword. It is this area that is filled by $CCPAU. (The user has the option of creating code instead of using $CCPAU to produce $CC4Z9.) At execution time $CC4Z9 is used to analyze user passwords as required.

$CCPAU can be run in either level of a DPF system, but it cannot be run while CCP is running. $CCPAU requires the following minimum system configuration for execution:

- 5404 Processing Unit (64K of MOSFET memory)
  or
  5410 Model A15 Processing Unit (24K of main storage)—Model 10
  or
  5412 Model B17 Processing Unit (48K FET memory)—Model 12
- 5444 Model 2 Disk Storage Drive—Model 10
  or
  5447 Model 1 Disk Storage Drive—Model 4
  or
  3340 Model C2 Direct Access Storage Facility—Model 12
- 5471 Printer-Keyboard or 5424 MFCU
- 5203 or 5213 or 1403 Printer



Sample input data records for program $CCPAU. Output to data module $CC4Z9.

Note: Length of 'LUSI' = 13
Length of input data = 10
The last 3 bytes are padding

Figure 6-1. Module Layout of $CC4Z9

Method of Operation

OCL STATEMENTS

The following OCL is required to run $CCPAU.

// LOAD $CCPAU,uu

// FILE UNIT-xx,NAME-$CCPFILE,PACK-CCPOBJ

// RUN

where uu is the disk unit that $CCPAU is loaded from and xx is the unit that $CCPFILE is on, as many data input records as are necessary to create the data for $CC4Z9 must follow the // RUN statement.

DATA INPUT RECORDS

Before allowing the user to enter data input records, program $CCPAU inspects the field NCCONF in the System Communication Region (SYSCOM) for a value of X'60'. If either bit is on, $CCPAU will not be allowed to continue and will be terminated.

In addition to comment records (those records with an * in column 1), seven different types of data input records may be used as source data for the security data module $CC4Z9. These data records may be entered in any order and with any frequency up to the maximum length specified at CCP generation time.

The seven types of data input records which may be used to generate the contents of $CC4Z9 are described below:

1. Type C, character string. Any character may be used as input.

   Column 1 = C.

   Columns 2-3 = the number of data characters to follow.

   Column 4 = blank.

   Column 5 on = as many columns of character data as was specified in columns 2 and 3.

   Sample input record:

   - Input = C05 ABCDE, contents of data buffer = C1C2C3C4C5.

2. Type Z, zoned decimal. Only zoned decimal data, signed or unsigned, may be used.

   Column 1 = Z.

   Columns 2-3 = the number of data characters, counting the sign if one is used, to follow.

   Column 4 = blank.

   Column 5 on = as many columns of zoned decimal data including the sign, as was specified in columns 2 and 3.

   Sample input records:

   - Input = Z04 1234, contents of data buffer = F1F2F3F4.
   - Input = Z05 +1234, contents of data buffer = F1F2F3F4.
   - Input = Z05 -1234, contents of data buffer = F1F2F3F4.

3. Type X, hexadecimal data. Data entered will be treated as hexadecimal and must be specified in even increments.

   Column 1 = X.

   Columns 2-3 = the number of data bytes to follow. This number must be an even integer.

   Column 4 = blank.

   Column 5 on = as many bytes of hexadecimal data as was specified in columns 2 and 3.

   Sample input records:

   - Input = X04 1234, contents of data buffer (2 bytes) = 1234.
   - Input = X06 A1B1C1, contents of data buffer (3 bytes) = A1B1C1.

4. Type I, one-byte integer records, signed or unsigned.

   Column 1 = I.

   Columns 2-3 = the number of data bytes, including the sign if used, to follow. This value cannot exceed 03 for unsigned or 04 for signed values.

Column 4 = blank.

Column 5 on = the number of bytes of data, including the sign if used, that was specified in columns 2 and 3. This value may not exceed 255 decimal.

Sample input records:

- Input = I01 1, contents of data buffer (1 byte) = 01.
- Input = I02 + 6, contents of data buffer (1 byte) = 06.
- Input = I03 +16, contents of data buffer (1 byte) = 10.
- Input = I03 -64, contents of data buffer (1 byte) = BF.

5. Type J, two-byte integer records, signed or unsigned.

Column 1 = J.

Column 2-3 = the number of data bytes, including the sign if used, to follow. This value cannot exceed 05 for unsigned or 06 for signed values.

Column 4 = blank.

Column 5 on = the number of bytes of data, including the sign if used, that was specified in columns 2 and 3. This value may not exceed 65535 decimal.

Sample input records:

- Input = J01 1, contents of data buffer = 0001 (2 bytes).
- Input = J02 +6, contents of data buffer = 0006 (2 bytes).
- Input = J03 +16, contents of data buffer = 0010 (2 bytes).
- Input = J03 -64, contents of data buffer = 00BF (2 bytes).
- Input = J05 65534, contents of data buffer = FFFE (2 bytes).

6. Type K, three-byte integer records, signed or unsigned.

Column 1 = K.

Column 2-3 = the number of bytes of data, including the sign if used, to follow. This value cannot exceed 08 for unsigned or 09 for signed values.

Column 4 = blank.

Column 5 on = the number of bytes of data, including the sign if used, that was specified in columns 2 and 3. This value may not exceed 16,777,215 decimal.

Sample input records:

- Input = K01 1, contents of data buffer = 000001 (3 bytes).
- Input = K02 +6, contents of data buffer = 000006 (3 bytes).
- Input = K03 +16, contents of data buffer = 000010 (3 bytes).
- Input = K03 -64, contents of data buffer = 0000BF (3 bytes).
- Input = K08 16777215 contents of data buffer = FFFFFF (3 bytes).

Type L, four-byte integer records, signed or unsigned.

Column 1 = L.

Column 2-3 = the number of data bytes, including the sign if used, to follow. This value cannot exceed 10 for unsigned or 11 for signed values.

Column 4 = blank.

Column 5 on = the number of bytes of data, including the sign if used, that was specified in columns 2 and 3. This value may not exceed 4,294,967,295 decimal.

Sample input records:

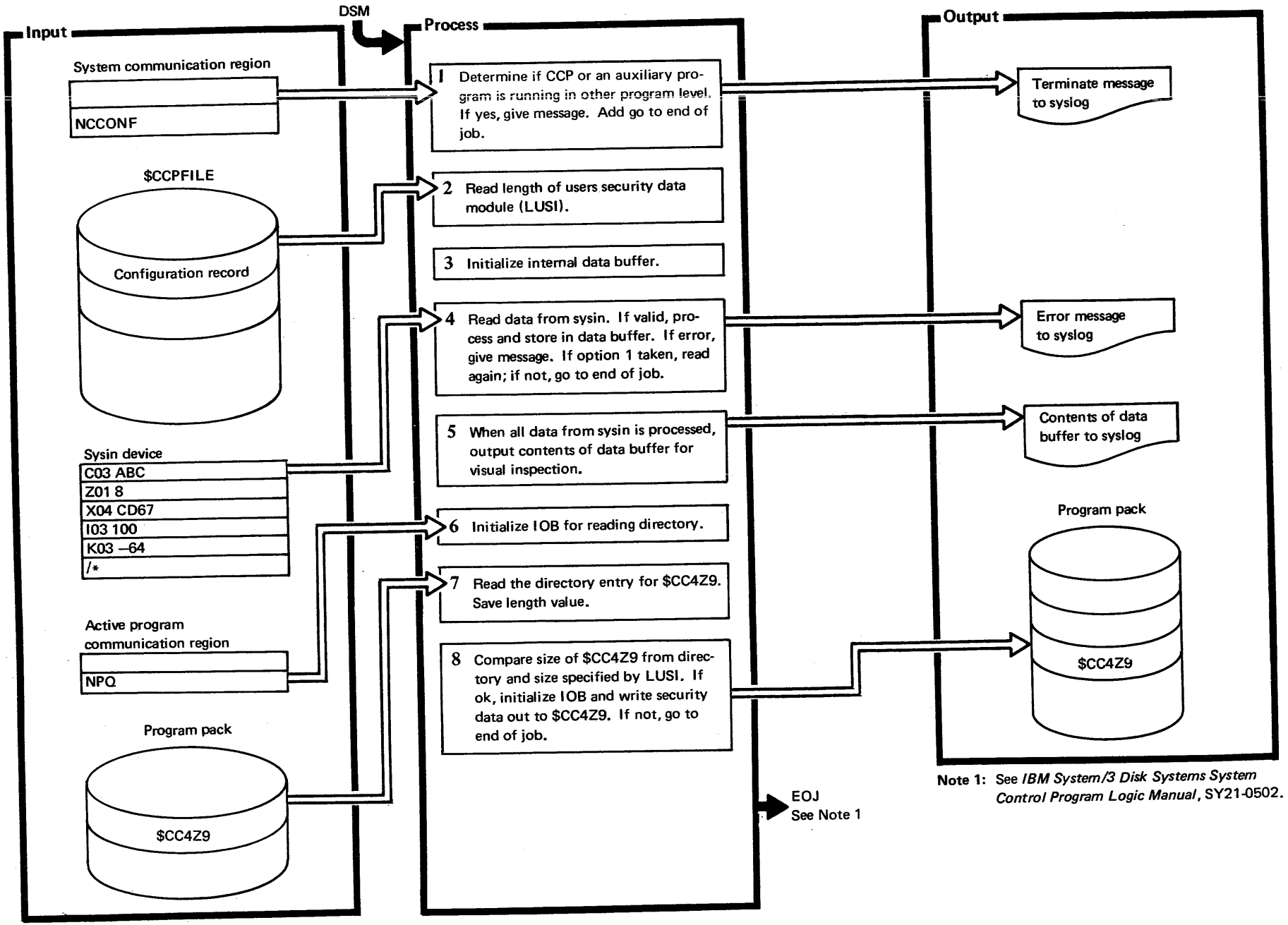
- Input = L01 1, contents of data buffer = 00000001 (4 bytes).
- Input = L02 +6, contents of data buffer = 00000006 (4 bytes).
- Input = L03 +16, contents of data buffer = 00000010 (4 bytes).

Input = L03 -64, contents of data buffer = 000000BF (4 bytes).

Input = L10 4294967295, contents of data buffer = FFFFFFFF (4 bytes).

As each data record is entered from the system input device, syntax checks are performed and warning messages are issued whenever syntax errors are detected. Data records that do not contain errors are added to the data buffer within $CCPAU until their total size equals the size of data module $CC4Z9 or until a '/*' record is read. If the amount of data entered is insufficient or if enough data to fill $CC4Z9 has been accumulated prior to reading a '/*' record, appropriate warning messages are issued to the operator. Options are afforded which allow more data to be entered, to start over with a clear data buffer, or to write the data as is to module $CC4Z9. In all cases the data to be written to module $CC4Z9 is printed on the system log device before it is actually written to $CC4Z9. In the event that too little data was entered but the option to write the data to $CC4Z9 was taken, the unfilled portion $CC4Z9 will contain binary zeros.

Diagram 6M.0100 is an overview of data flow in $CCPAU.



Note 1: See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.

Diagram 6M.0100. Overall Data Flow of $CCPAU

Program Organization

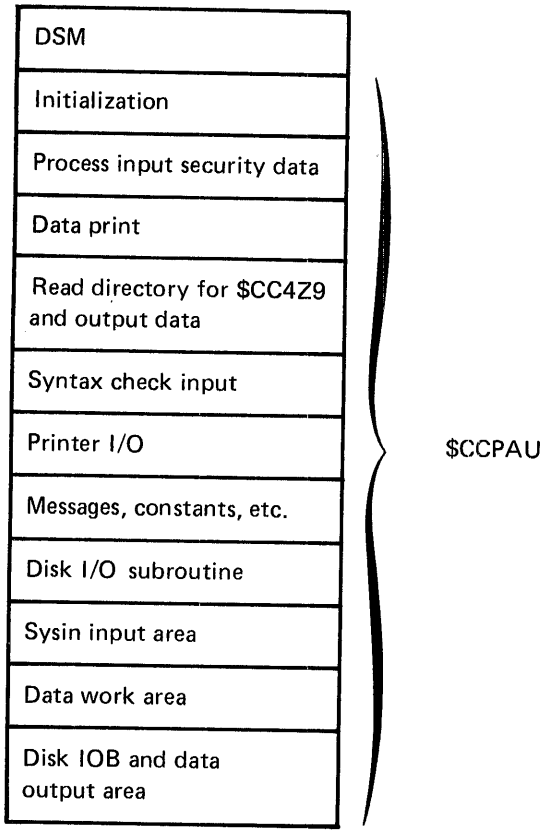


Figure 6-2. Storage Layout of $CCPAU

$CCPAU Module Description

ENTRY POINT: AU0000
CHART: MU
FUNCTIONS: Reads user's security data from SYSIN and, after syntax checking, writes it out to data module $CC4Z9.
INPUT: Security data records via SYSIN.
OUTPUT:
- Listing of security data to SYSLOG.
- Security data to $CC4Z9.

EXIT, NORMAL: System end of job routine (see note).
EXIT, ERROR: System end of job routine (see note).

Note: DSM EOJ is described in *IBM System/3 Disk Systems System Control Program Logic Manual,* SY21-0502.

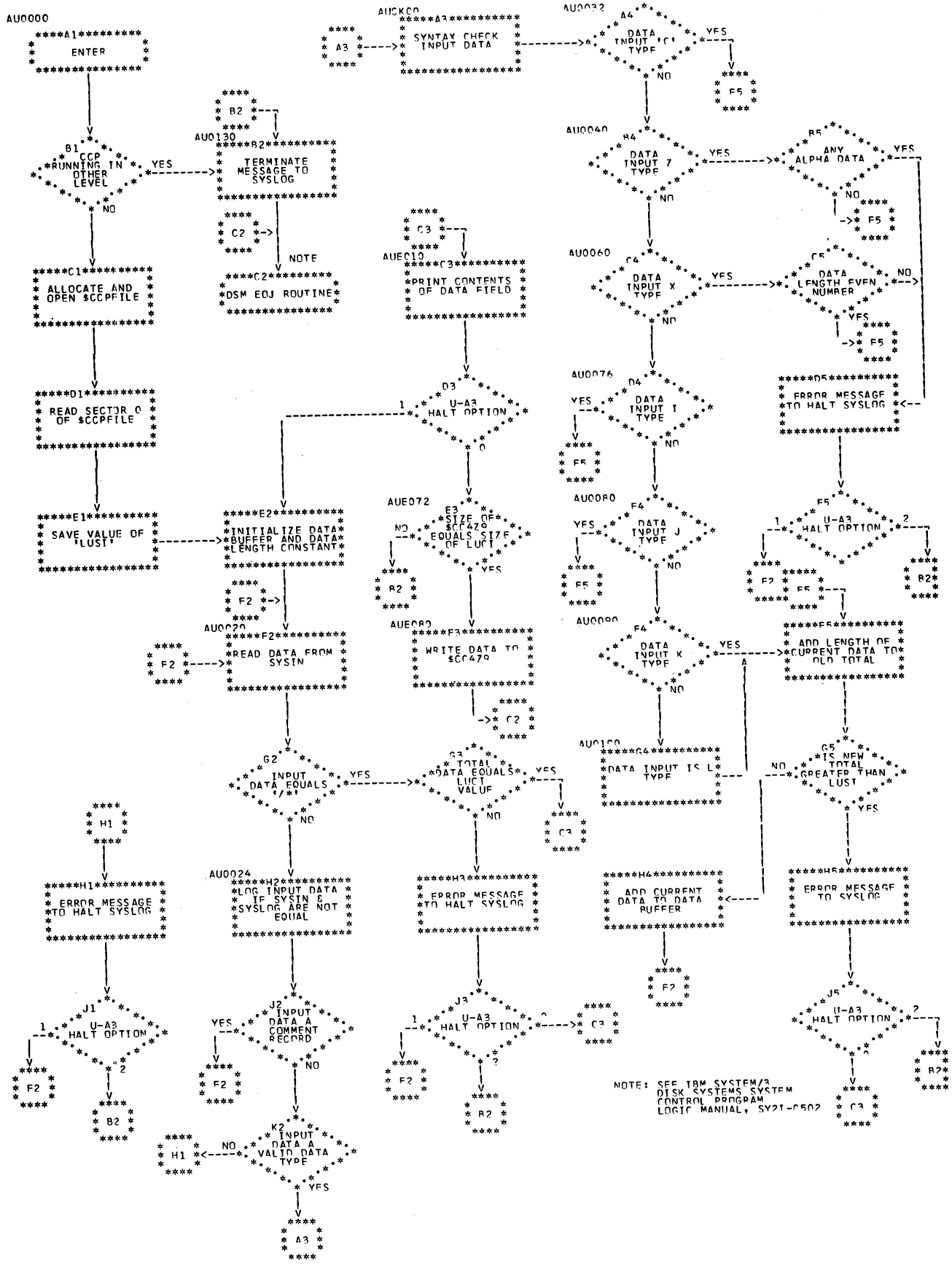


Chart MU. User Security Information Build ($CCPAU) (Models 8, 10, and 12 Only)

Chapter 7. Format Generator

Display Format Generator

INTRODUCTION

The Display Format Generator reads records generated from display format specification sheets (using $CCPDF for single or multiple format builds), and creates a 3270 display format. Display format specifications can be read from the system input device or the source library. The Display Format Generator prints information for the user about the format as it is built. When the build is completed, the Display Format Generator calls the Overlay Linkage Editor Librarian Phase ($OLBO) to catalog the display format into the object library. ($OLBO is described in *IBM System/3 Overlay Linkage Editor and Checkpoint/Restart Program Logic Manual,* SY21-0530.)

Two types of specification statements are prepared by the user to define a display format: display control statements and field definition statements. At least one display control statement is required and it must precede all field definition statements. The display control statement provides information about the entire display format. Each field in the display format must be defined on a field definition statement. The field definition statements define each field by name, length, location, and type. They also tell where data for each field is defined.

The Display Format Generator runs offline; that is, it does not run under the control of CCP although the display format it generates is used by the Display Format Control Routine (DFCR) which does run under CCP. The Display Format Generator can run in either level of a DPF system. It can run while CCP is running in the other level unless DFGR puts a format on the pack from which CCP was loaded.

The Display Format Generator runs under System/3 DSM and requires 22K of main storage for execution. The Display Format Generator requires a 2-track work file on disk called $WORK. The system in which the Display Format Generator is running must contain the Overlay Linkage Editor Librarian modules. If reading display format specifications from the source library, a $SOURCE file is needed. If a $SOURCE file statement is not specified, a 5-track $SOURCE file is automatically allocated. However, whether the $SOURCE file is specified or automatically allocated, this file has to be large enough to contain all display format specifications.

Method of Operation

FUNCTIONS

The Display Format Generator performs the following functions (see Diagram 7M.0100 for an illustrated overview of the Display Format Generator using single or multiple format builds read from sysin or the source library):

1. Reads display format specification statements.

2. Produces a printout of the specification statements, analyzes the specifications for errors, and logs diagnostic error messages as required.

3. Builds the display format as a 2-part table structure:
   a. Field Descriptor Table (FDT) – a table of descriptive field information including the symbolic name of the field.
   b. 3270 Data Stream – containing output data if provided, and 3270 device-dependent control information required for formatting all fields defined.

4. Provides a printout of field names, in the order in which they must appear in the output record area if data from the Field Descriptor Table is required by the application program using the Display Format Facility.

5. Produces a printout of all fields defined for input and the order in which they will appear in the input record area.

6. Calculates and prints the following:
   a. Length of the output record area required in the DFF program
   b. Length of the input record area required in the DFF program
   c. Decimal length of the Field Descriptor Table
   d. Decimal length of the 3270 output data stream
   e. Decimal length of the 3270 input data stream

7. Places the display format in a work file ($WORK) on disk and then invokes the Overlay Linkage Editor Librarian Phase ($OLBO) to catalog the display format in an object library on disk.

COMPONENTS – FOR SINGLE AND MULTIPLE FORMAT BUILDS, READ FROM SYSIN OR THE SOURCE LIBRARY

The Display Format Generator consists of four components:

- $CCPDF – DFGR Build Common Area (LOMMON), reads from SYSIN or the source library and opens files.
- $CC2CR – DFGR reads from the $SOURCE file.
- $CC2CF – DFGR prints and diagnoses display specifications.
- $CC2CP – DFGR prints and copies to disk.

The following describes these components and their functions.

$CCPDF

$CCPDF (Diagram 7M.0200) receives initial control when loaded by the user, then performs the following functions:

1. Builds the Overlay Linkage Editor communications area (LOMMON).

2. Automatically allocates $SOURCE file if one is not specified.

3. Opens $WORK and $SOURCE files.

4. Copies data from SYSIN or the source library to $SOURCE file.

5. Finds and loads $CC2CR.

$CC2CR

For implementation of single or multiple format builds, see the *IBM System/3 Communications Control Program Programmer's Reference Manual,* GC21-7579.

$CC2CR (Diagram 7M.0300) receives control from $CCPDF, $CC2CF, $CC2CP, or $OLBO, then performs the following functions:

1. Finds and loads $CC2CF after $CC2CR.

2. Reads display format specification statements from the $SOURCE file.

$CC2CF

$CC2CF (Diagram 7M.0400) receives control from $CC2CR, then performs the following functions:

1. Logs and processes display format specification statements.
2. Builds the display format in main storage.
3. Builds the DFGR common area containing pointers and information about the display format being generated.
4. Finds and loads $CC2CP if no terminating errors occurred.
5. Finds and loads $CC2CR if a terminating error did occur.

$CC2CP

$CC2CP (Diagram 7M.0500) receives control from $CC2CF, then performs the following functions:

1. Logs information to the user about the display format being generated.
2. Copies the display format from main storage to a work file on disk ($WORK).
3. Finds and loads the Overlay Linkage Editor Library Phase ($OLBO) if no terminating errors occurred.
4. Finds and loads $CC2CR if a terminating error did occur.

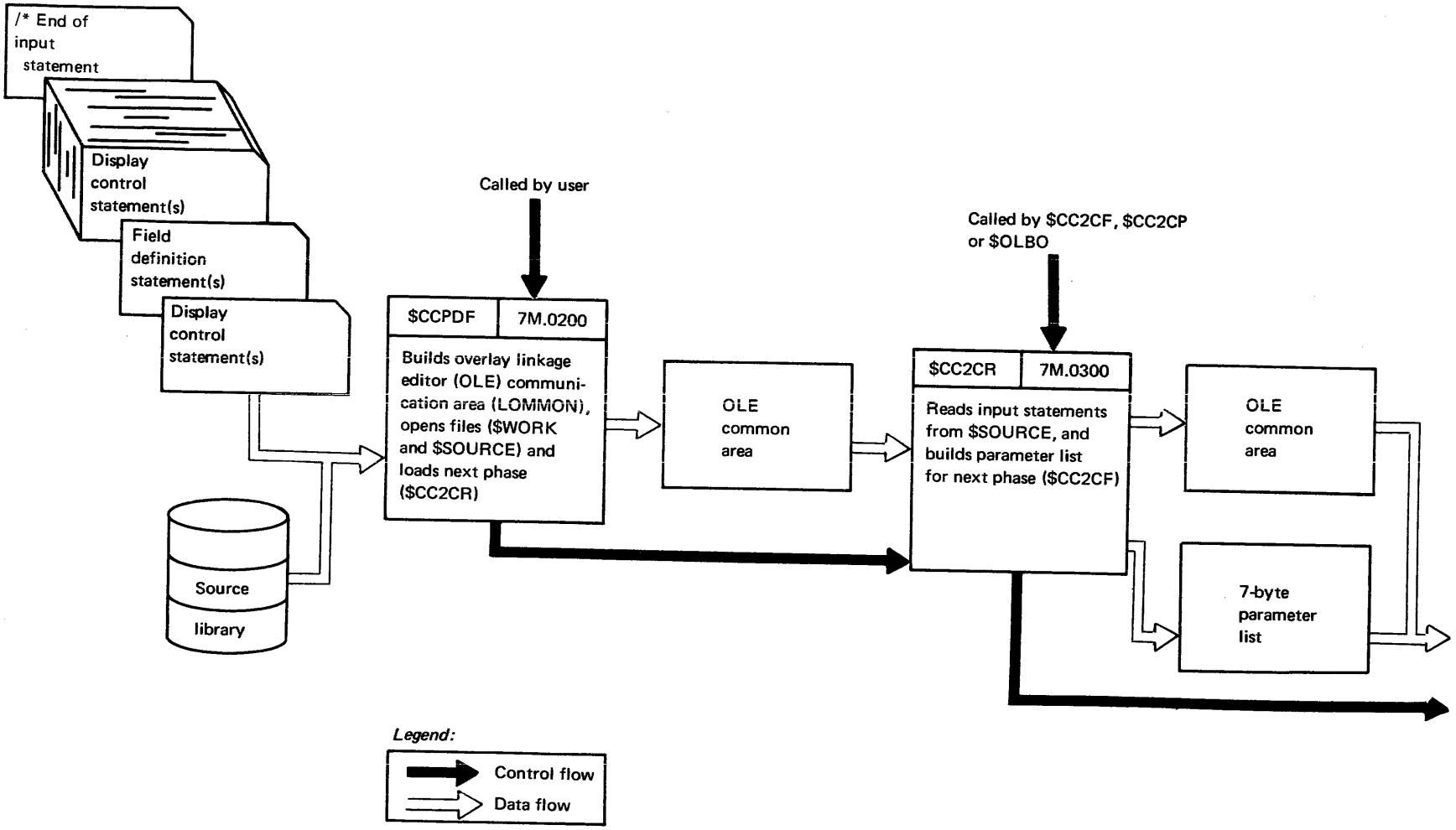


Diagram 7M.0100 (Part 1 of 2). Overall Flow of DFGR (using single or multiple format builds)

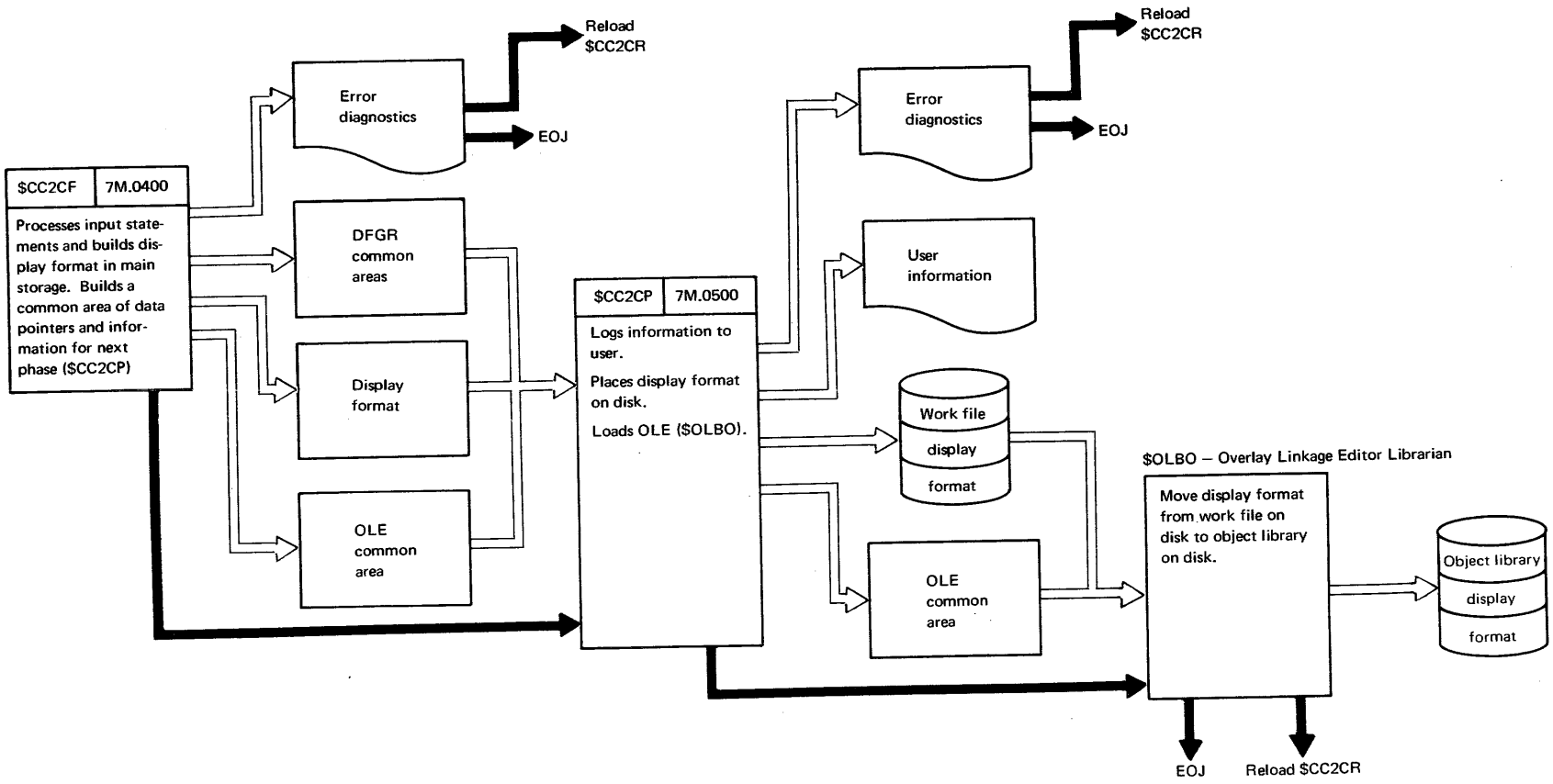
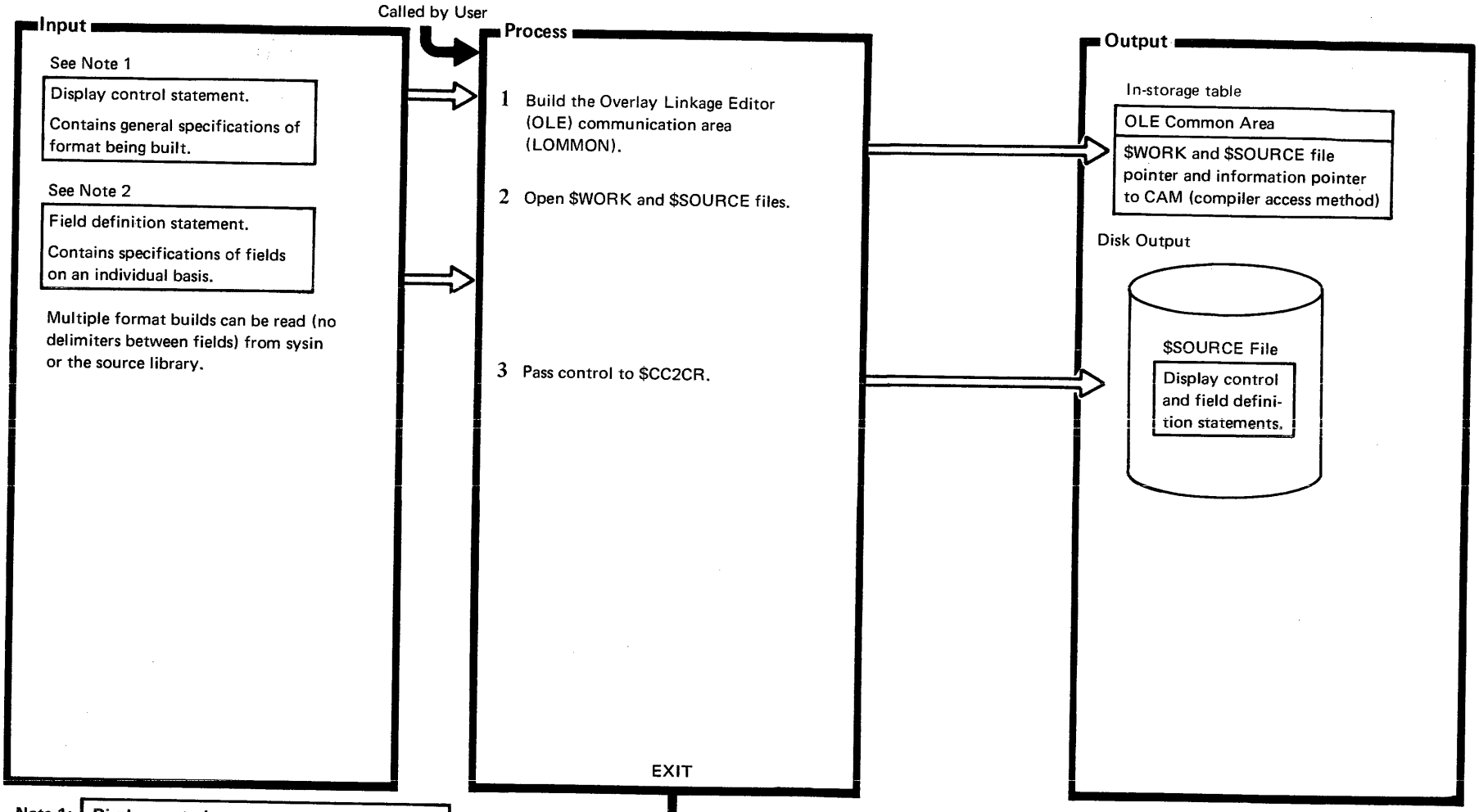


Diagram 7M.0100 (Part 2 of 2). Overall Flow of DFGR (using single or multiple format builds)



Note 1:

Display control statement contents	
Required	**Optional**
Display name Display size	Initial cursor position Clear before write Device Disk unit for output Printer line control

Note 2:

Field definition statement contents	
Required	**Optional**
Field name Field location Field length Field class/type	Field data source Field data

Diagram 7M.0200. $CCPDF

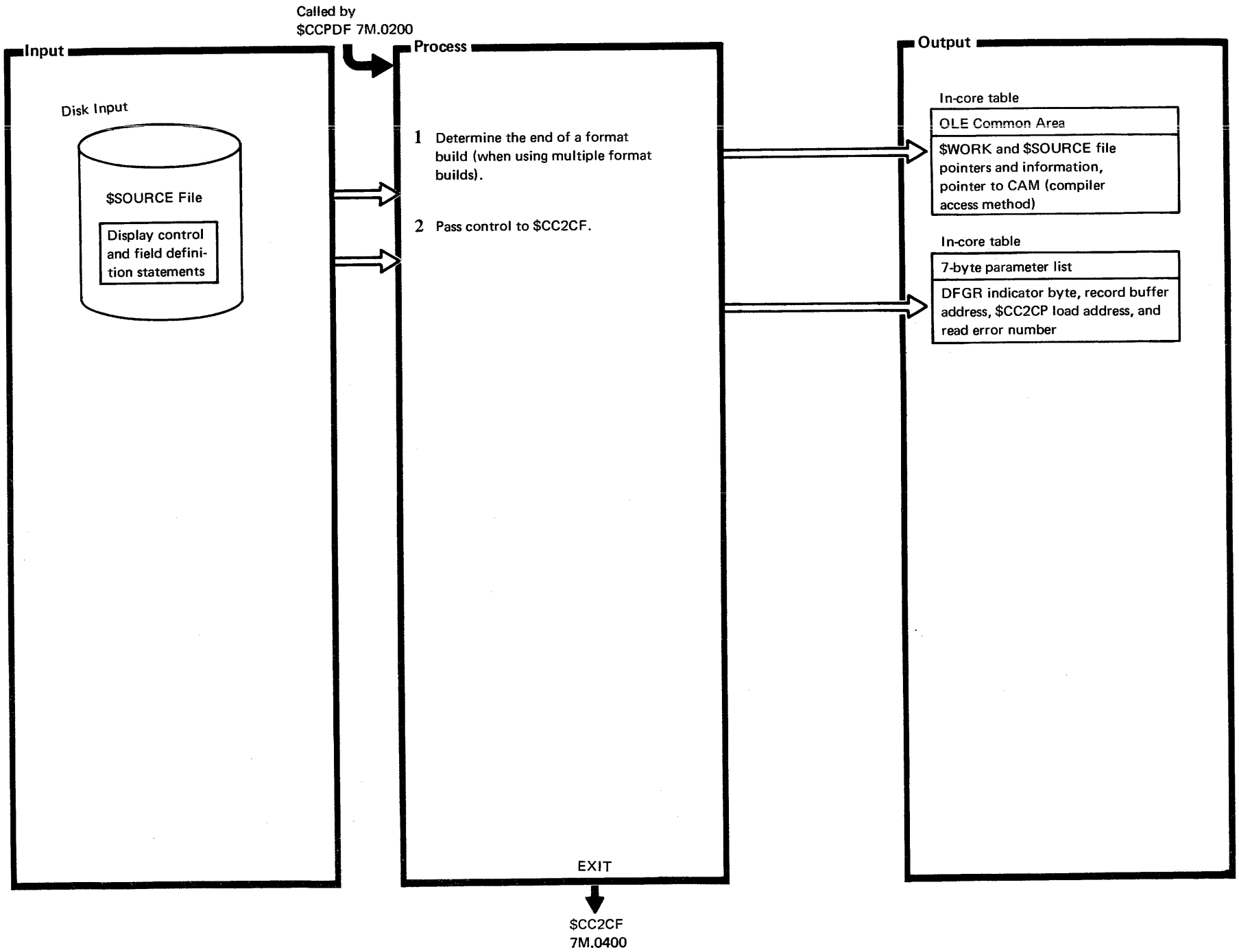
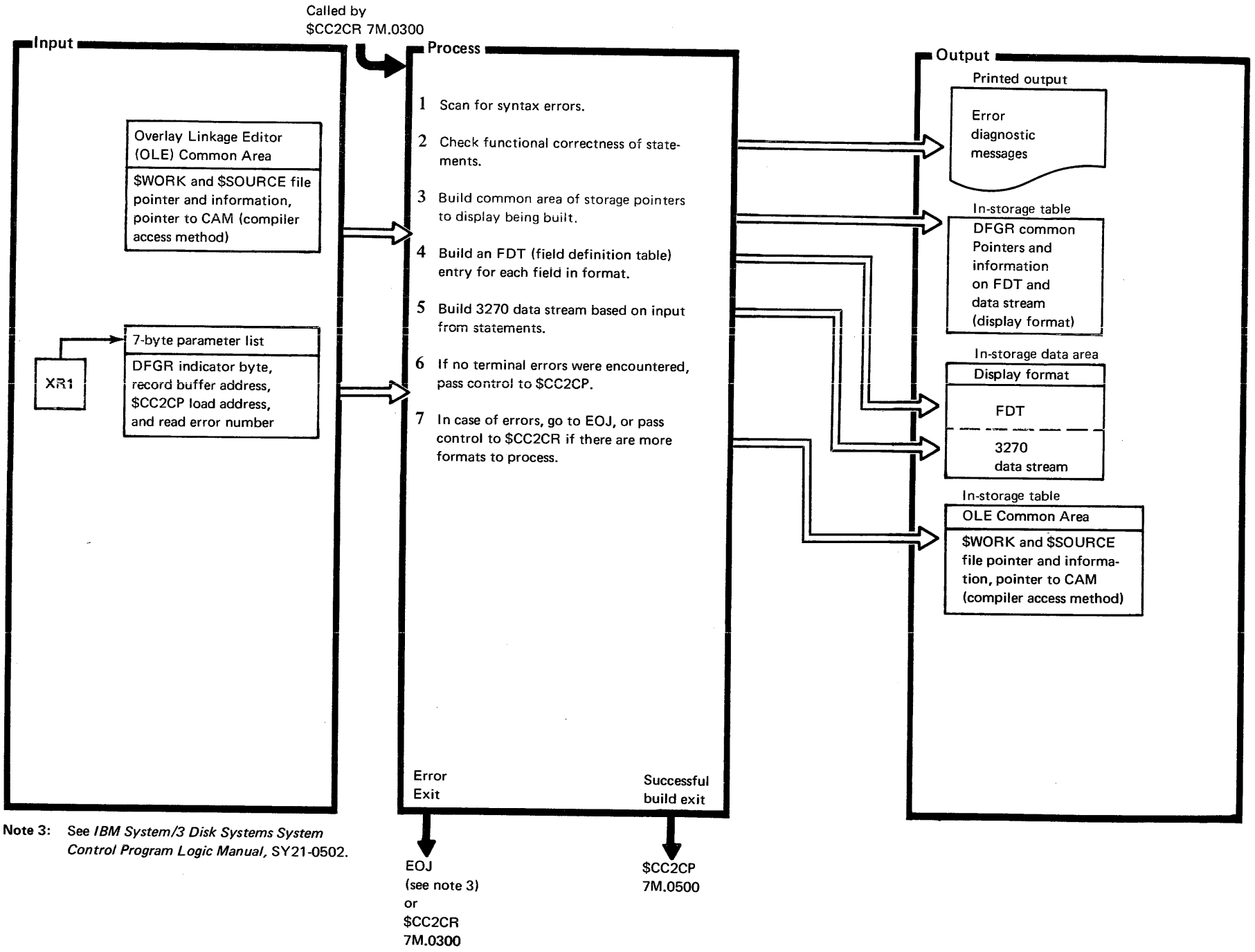


Diagram 7M.0300. $CC2CR



Note 3: See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.

Diagram 7M.0400. $CC2CF

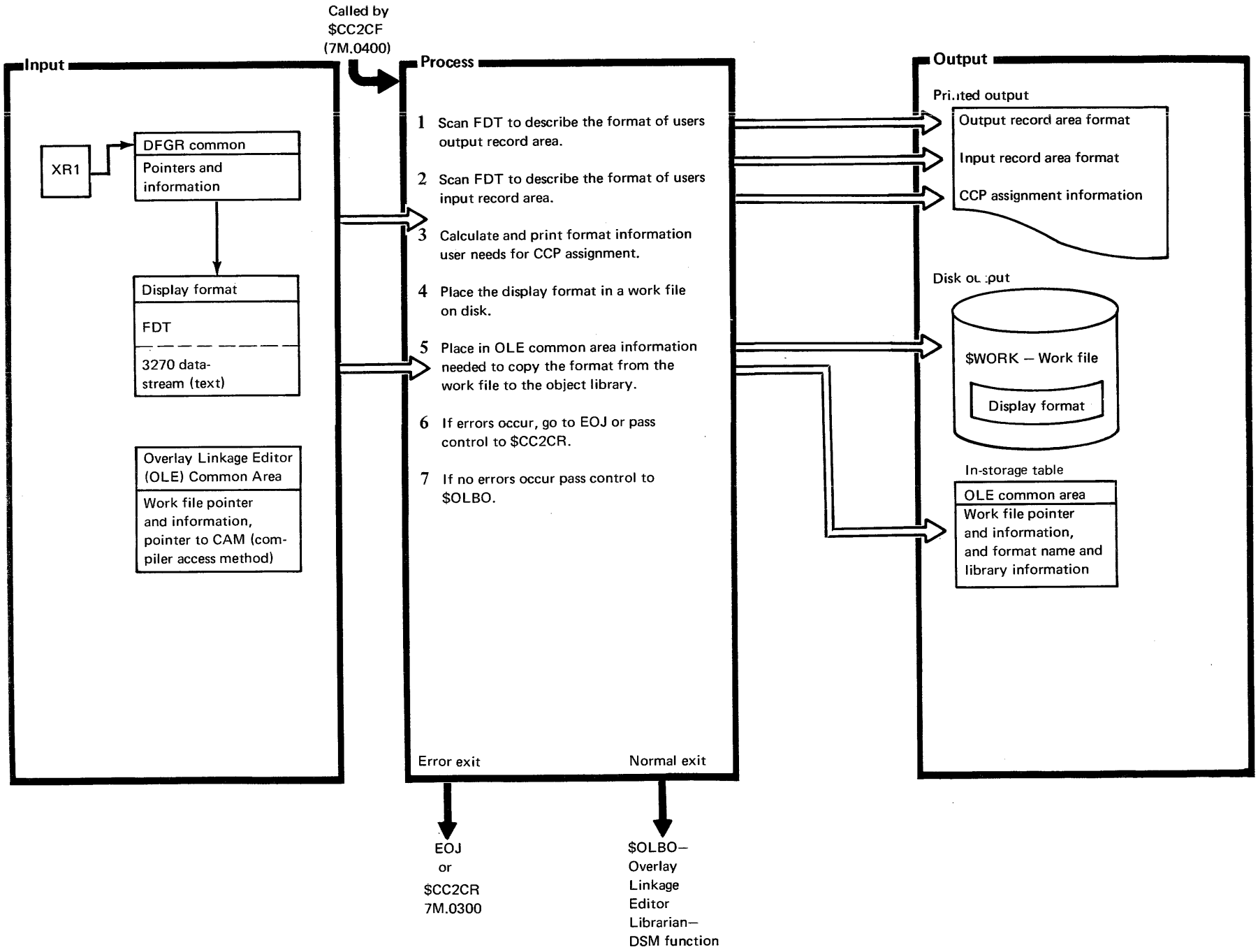


Diagram 7M.0500. $CC2CP

Program Organization

This section describes in detail the organization of the Display Format Generator routines. It includes both single and multiple format builds, read from sysin or the source library. Included in the following discussions are:

- Module descriptions derived from listing prologs.
- Main storage maps showing the layout of main storage at the time the routine being described has control (Figure 7-1).
- Flowcharts showing the internal logic of the routine being described.

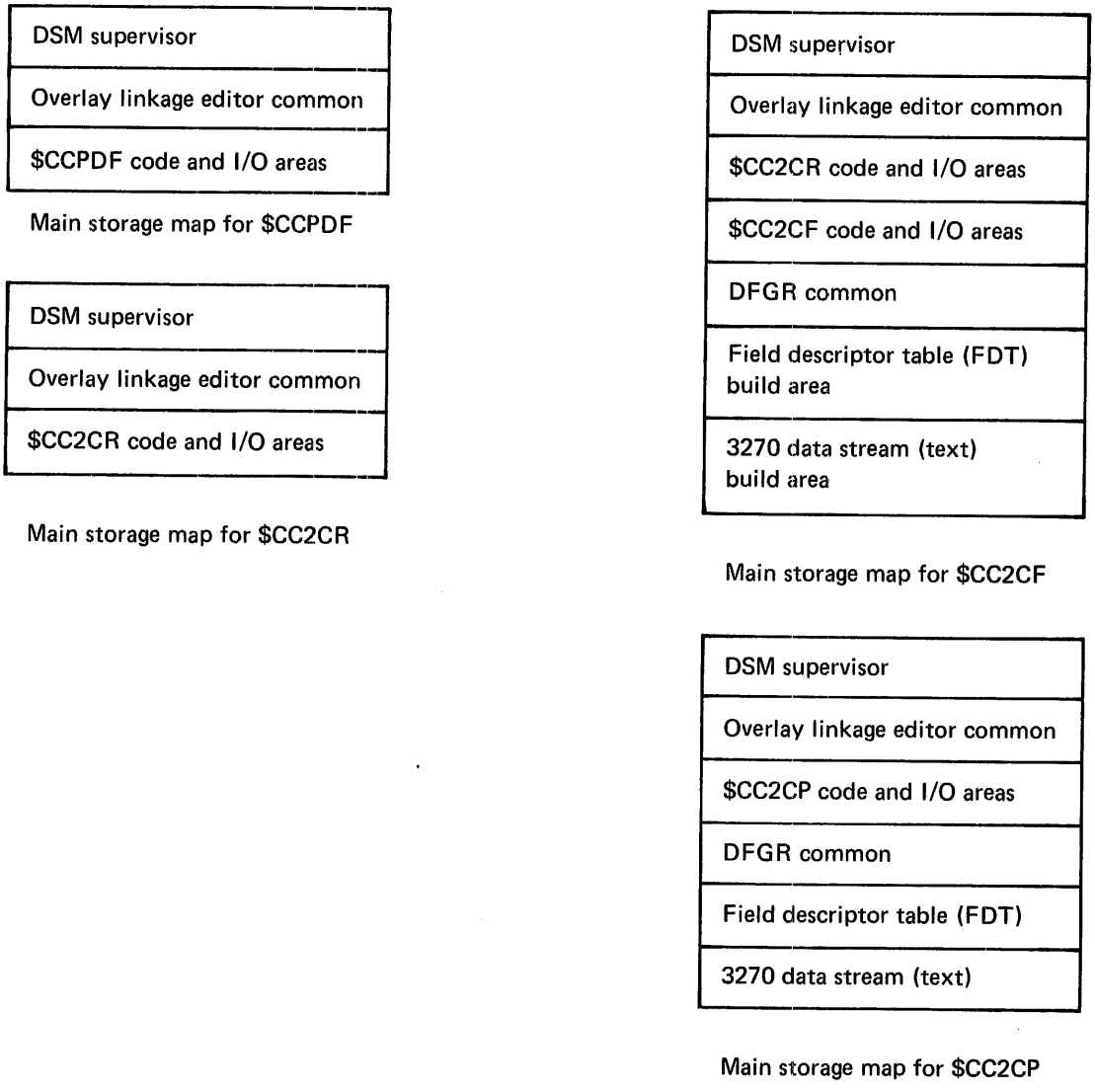


Figure 7-1. Storage Layout of DFGR

Display Format Generator Build Common Area (LOMMON), Read from Sysin, the Source Library, and Open Files ($CCPDF)

ENTRY POINT: CRDINP
CHART: MV
FUNCTIONS:
- Builds the overlay linkage editor communications area (LOMMON).
- Automatically allocates $SOURCE file if one was not specified.
- Opens $WORK and $SOURCE files.
- Reads the display format specifications from sysin or the source library, and copies to the $SOURCE file.
- Loads the next phase of the DFGR ($CC2CR).

INPUT: Display format specifications via system or the source library. (Sysin routines are described in *IBM System/3 Disk Systems Control Program Logic Manual,* SY21-0502.)
OUTPUT:
- Overlay linkage editor communications area (LOMMON).
- $SOURCE file containing display control and field definition statements.

EXITS:
- Normal: Loads and passes control to the next phase of DFGR ($CC2CR).
- Error: A terminal error causes a halt, followed by EOJ. Halt/syslog and system EOJ are described in *IBM System/3 Disk Systems System Control Program Logic Manual,* SY21-0502.

Display Format Generator Read Single or Multiple Formats from the $SOURCE File ($CC2CR)

ENTRY POINT: BLDINP
CHART: MW
FUNCTIONS:
- Reads display format specifications from the $SOURCE file.
- Loads the next phase of the DFGR ($CC2CF) following $CC2CR.

INPUT: Display format specifications via $SOURCE. (Data is moved into the $SOURCE file from SYSIN, or from the source library via $CCPDF.)
OUTPUT:
- Overlay linkage editor communications area (LOMMON).
- A 7-byte parameter list containing information used by the next phase ($CC2CF).

EXITS:
- Normal: Loads and passes control to the next phase of DFGR ($CC2CF).
- Error: A terminal error causes an error number to be placed in the parameter list. Control is then passed to the next phase ($CC2CF).

Display Format Generator Print and Diagnose Display Specifications ($CC2CF)

ENTRY POINT: BLDFMT
CHART: MY
FUNCTIONS:
- Logs the display format specifications.
- Diagnoses any errors in the specifications.
- Builds the display format in main storage.
- Builds the DFGR common area.
- Calls the next phase of the DFGR ($CC2CP).

INPUT: A 7-byte parameter list containing information used by this phase, and a display format specification read by a previous phase.
OUTPUT:
- Overlay linkage editor communications area (LOMMON).
- A 2-part table which makes up the display format. The first part, the Field Descriptor Table (FDT), contains the names of the fields making up the display format along with other field descriptive information. The second part of the table contains the 3270 data stream for the display format.
- DFGR COMMON, which contains information about, and pointers to, the display format being processed.
- Diagnostic messages describing any error encountered while processing the display format specifications.

EXITS:
- Normal: Loads and passes control to the next phase of DFGR ($CC2CP).
- Error: If a terminal error occurred, a message is given to the operator (halt/syslog) with the options to go to EOJ or build the next format (pass control to $CC2CR). Halt/syslog and system EOJ are described in *IBM System/3 Disk Systems System Control Program Logic Manual,* SY21-0502.

Display Format Generator Print and Copy to Disk ($CC2CP)

ENTRY POINT: $CC2CP
CHART: MZ
FUNCTIONS:

- Logs information about the display format being generated.
- Copies the display format from main storage to a work file ($WORK) on disk.
- Call the Overlay Linkage Editor Librarian Phase to place the format in the object library on disk. ($OLBO is described in *IBM System/3 Overlay Linkage Editor and Checkpoint/Restart Programs Logic Manual,* SY21-0530.)

INPUT:

- The display format in main storage.
- The DFGR common area, which contains information about the display format being generated.
- On entry to $CC2CP, index register one points to DFGR common area, which contains pointers to the format.

OUTPUT:

- Printed output describing the display format just built.
- A work file ($WORK) containing the display format.
- Overlay Linkage Editor common area containing information needed to catalog the display format in an object library.

EXITS:

- Normal: Loads and passes control to the Overlay Linkage Editor Librarian Phase ($OLBO).
- Error: If a terminal error occurs, a message is given to the operator (halt/syslog) with the options to go to EOJ or build the next format (pass control to $CC2CR). Halt/syslog and system EOJ are described in *IBM System/3 Disk Systems System Control Program Logic Manual,* SY21-0502.

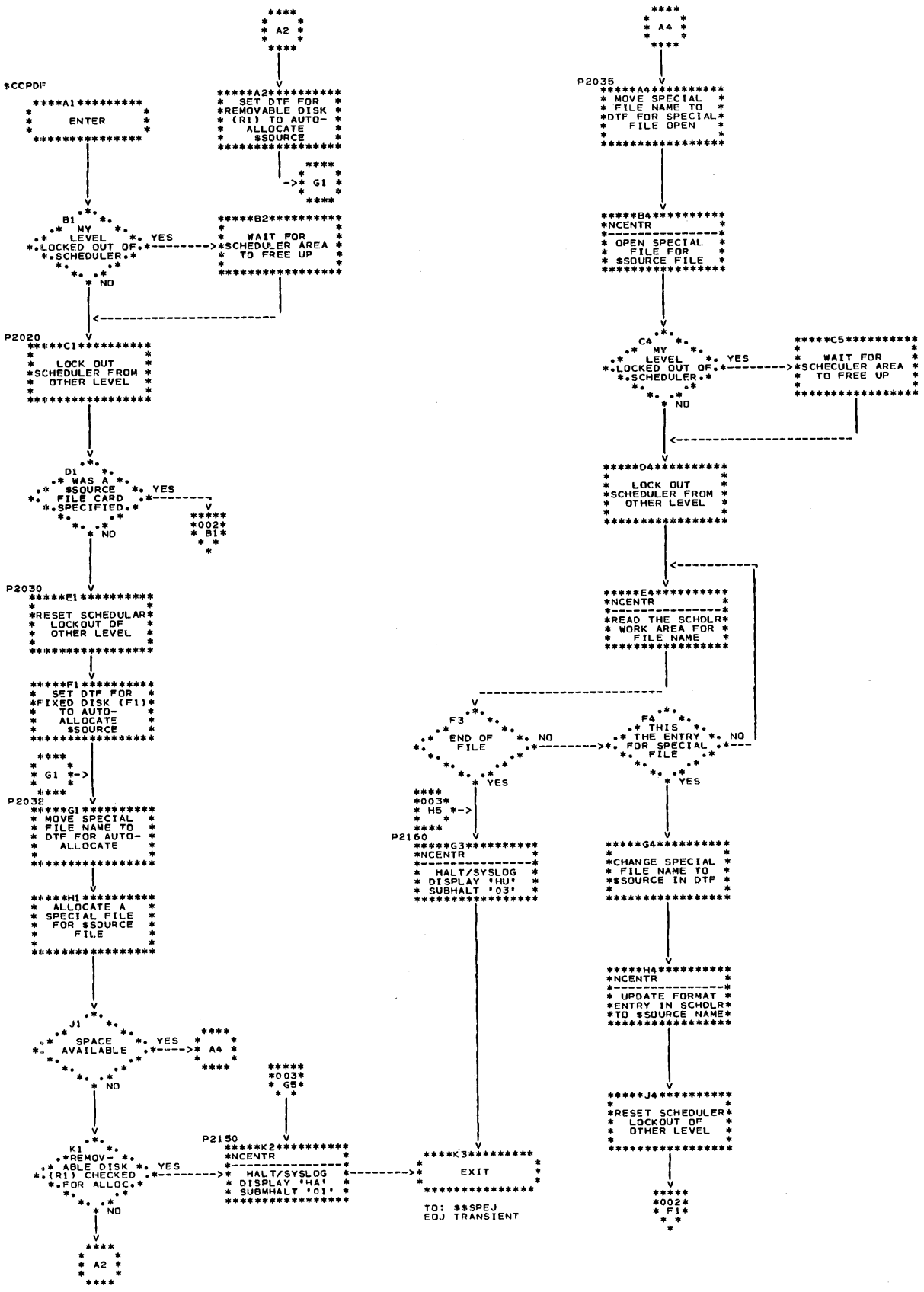


Chart MV (Part 1 of 3). DFGR Build Common Area (LOMMON) ($CCPDF)

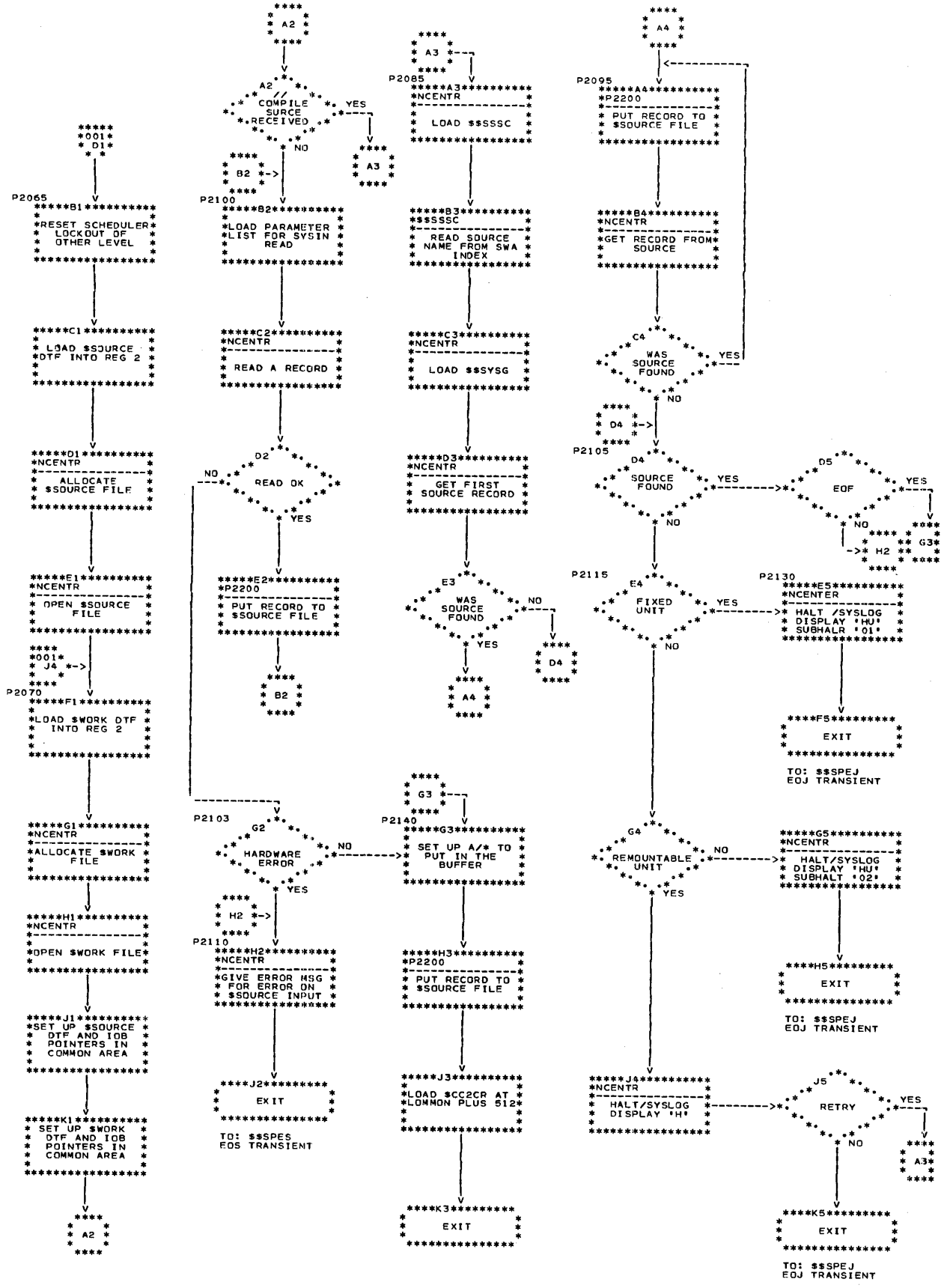


Chart MV (Part 2 of 3). DFGR Build Common Area (LOMMON) ($CCPDF)

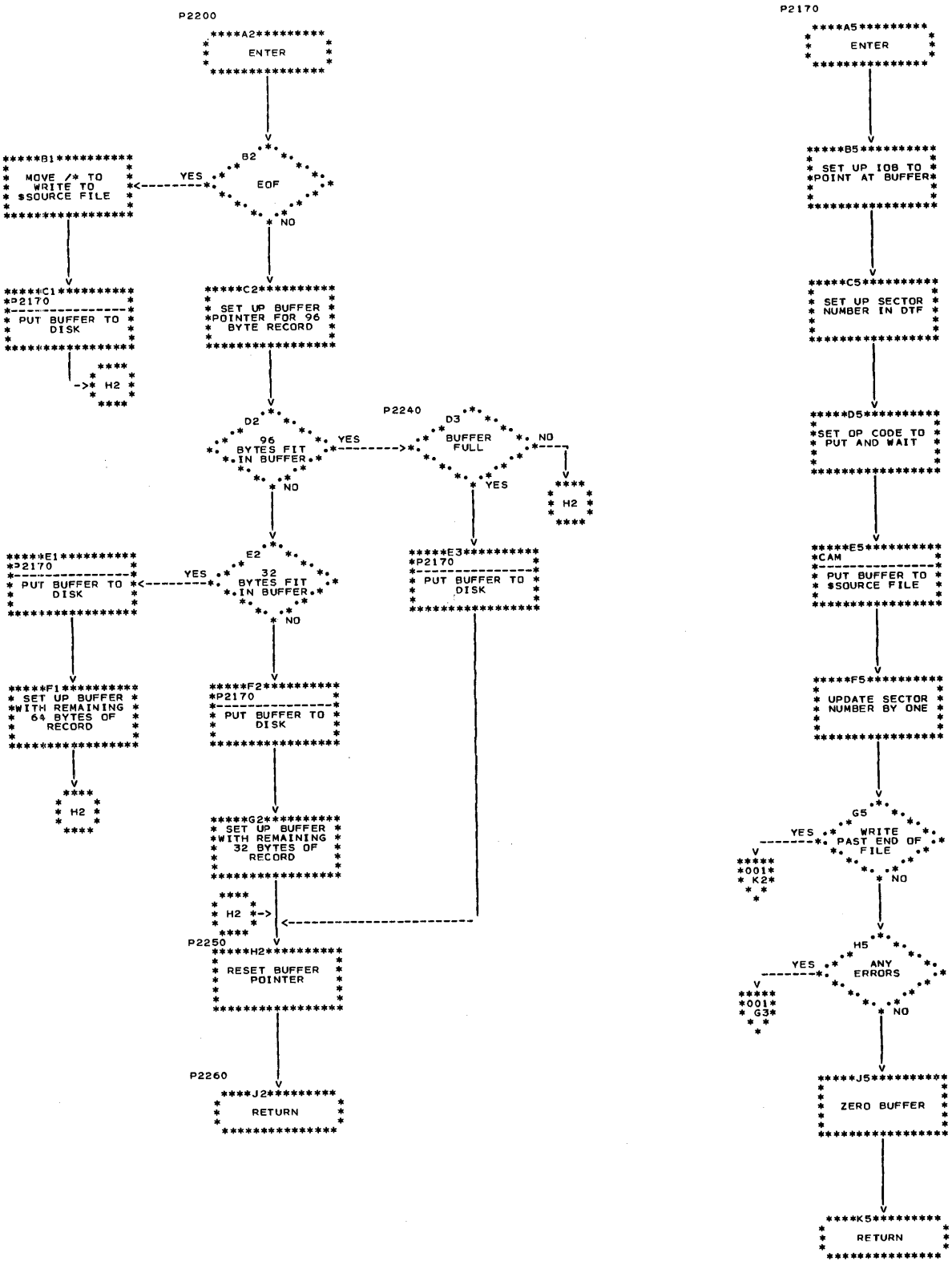


Chart MV (Part 3 of 3). DFGR Build Common Area (LOMMON) ($CCPDF)

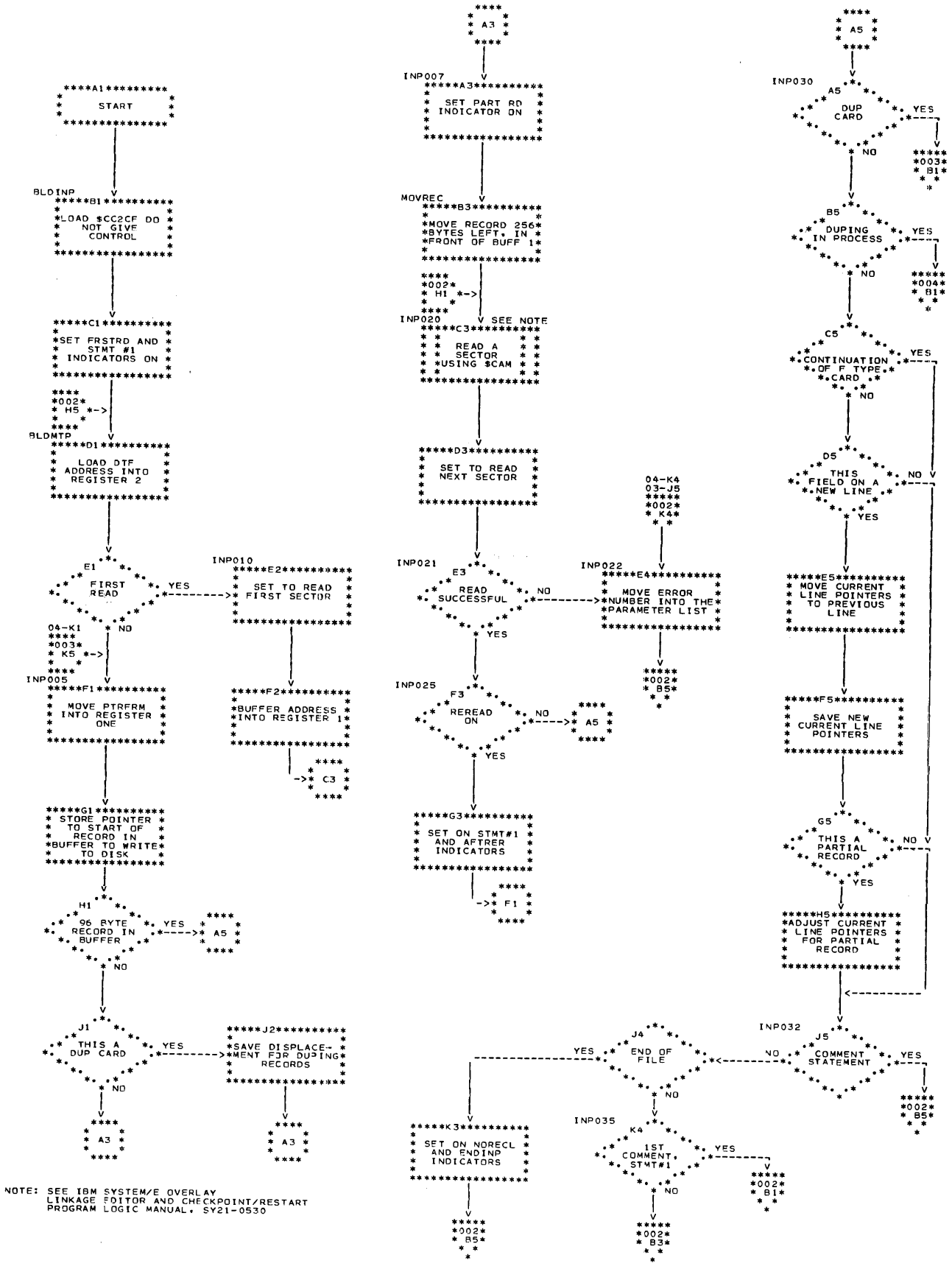


Chart MW (Part 1 of 4). DFGR Read Multiple from Sysin or Source Library ($CC2CR)

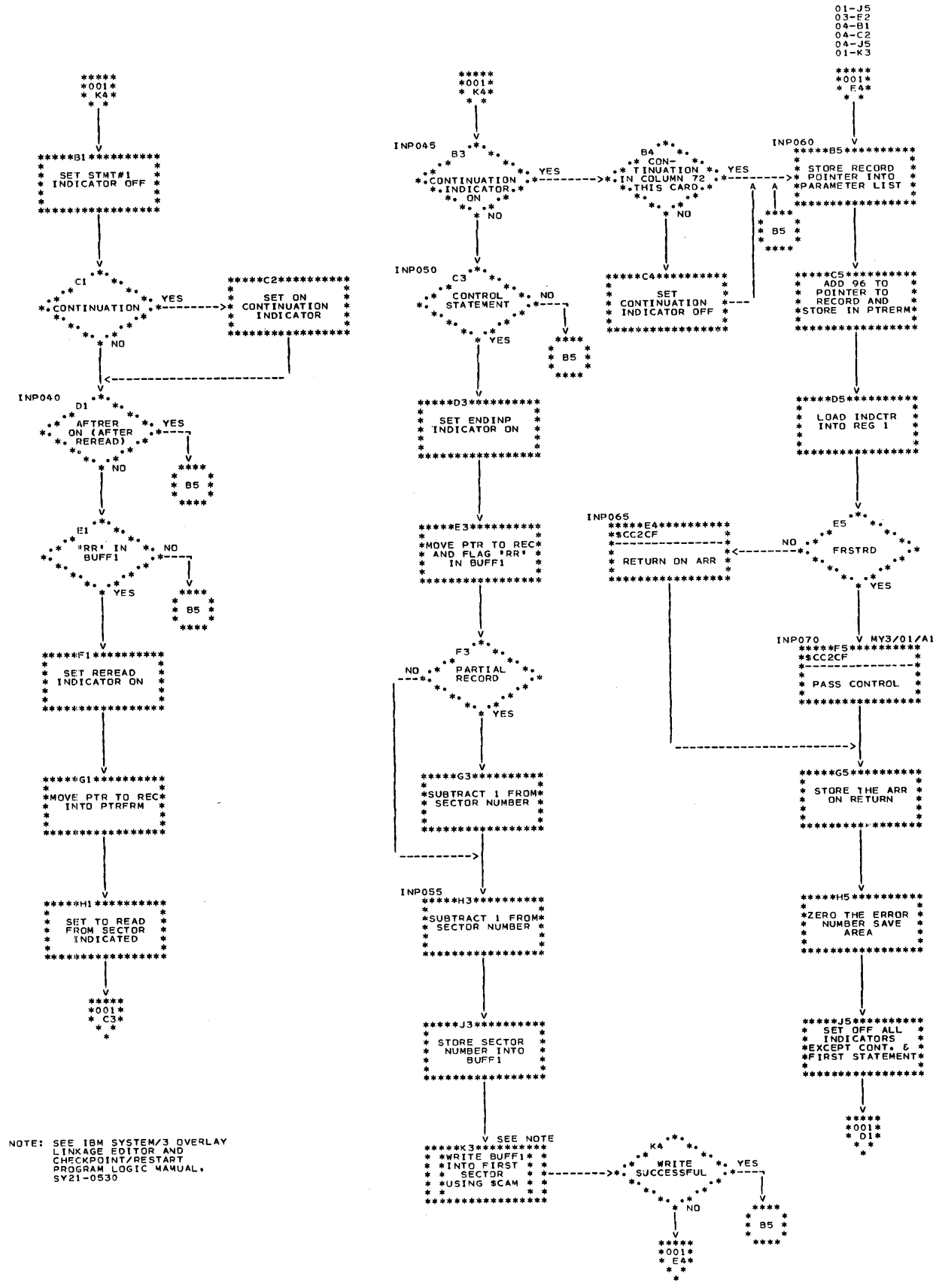


Chart MW (Part 2 of 4). DFGR Read Multiple from Sysin or Source Library ($CC2CR)

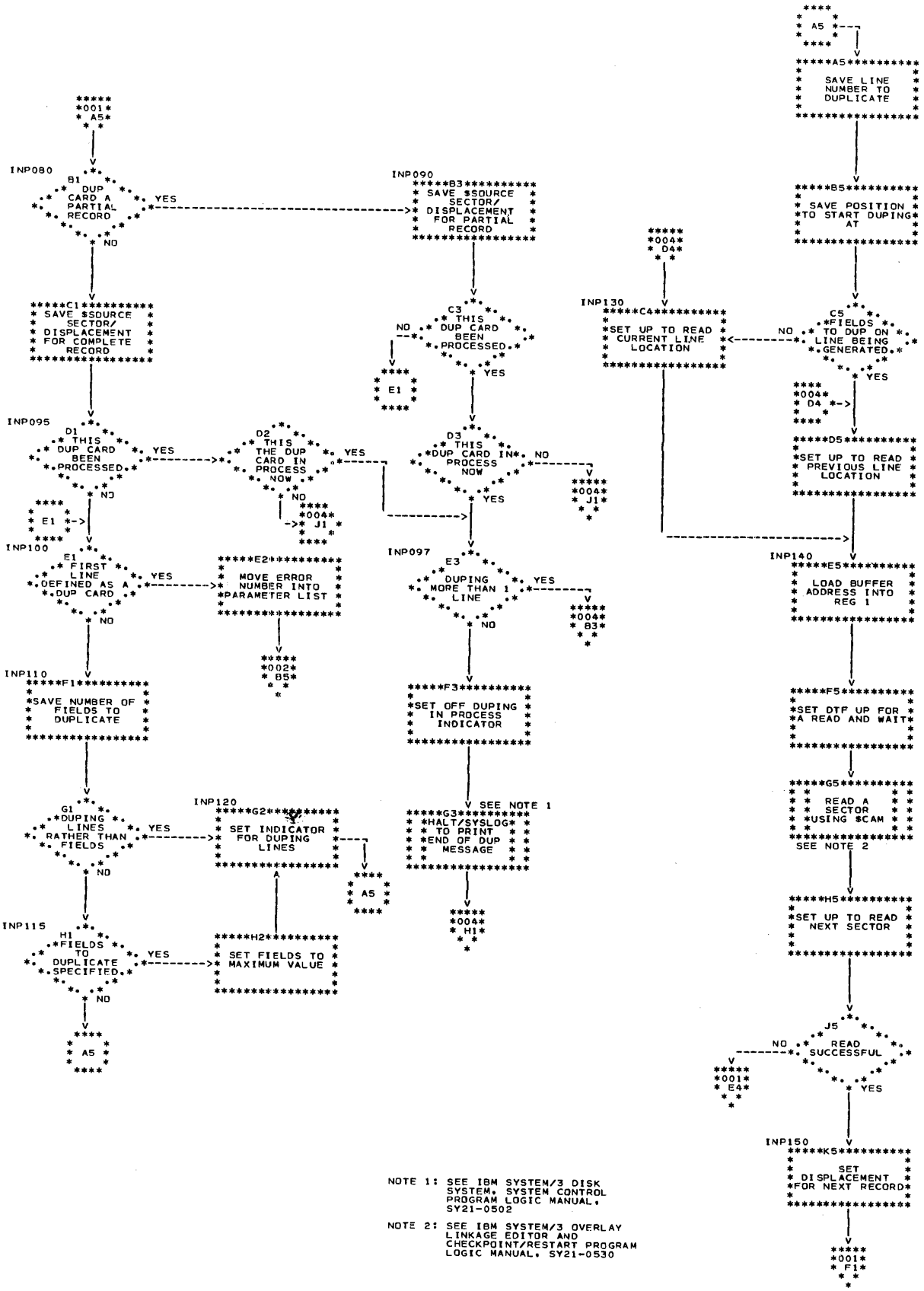


Chart MW (Part 3 of 4). DFGR Read Multiple from Sysin or Source Library ($CC2CR)

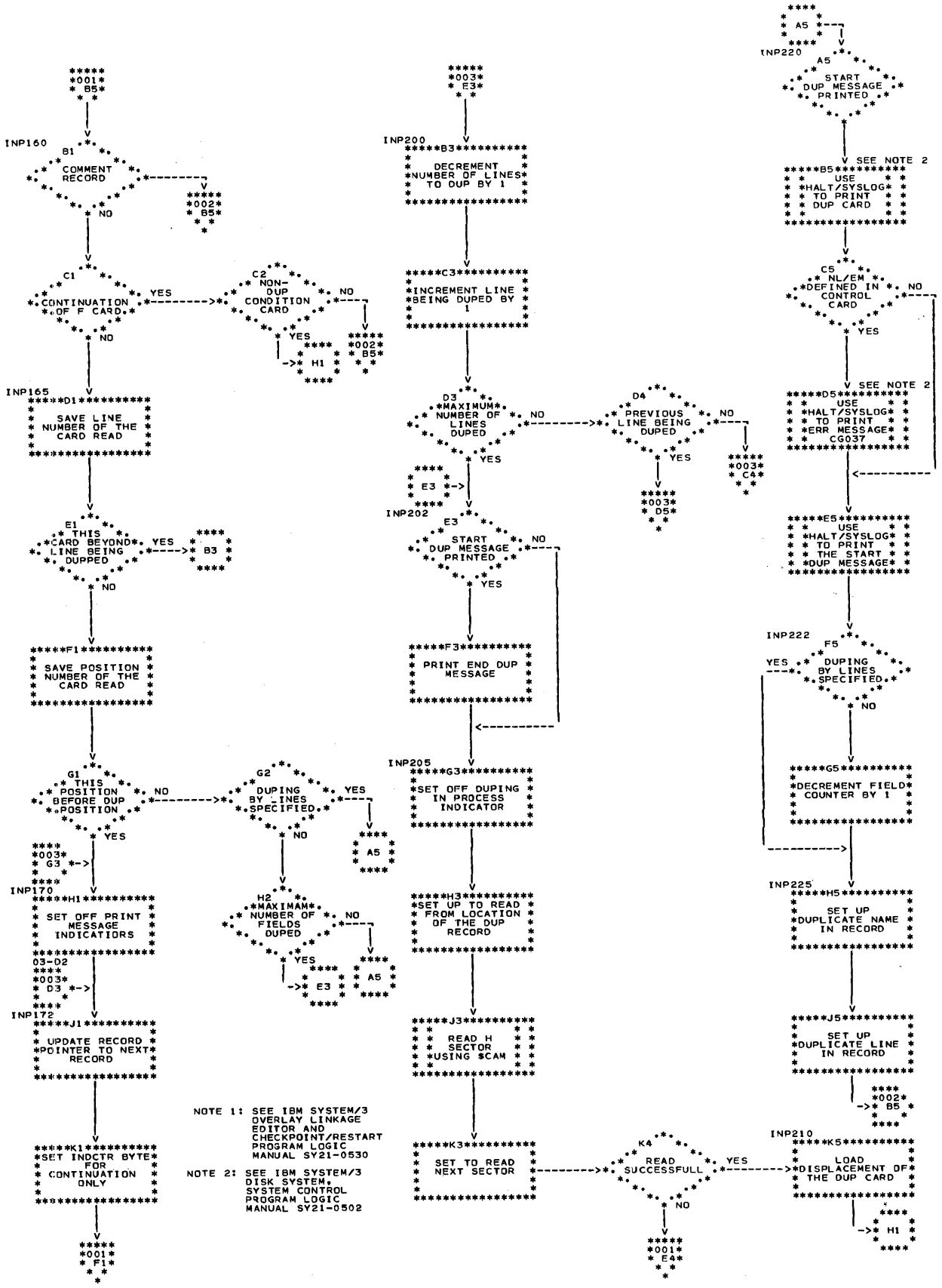


Chart MW (Part 4 of 4). DFGR Read Multiple from Sysin or Source Library ($CC2CR)

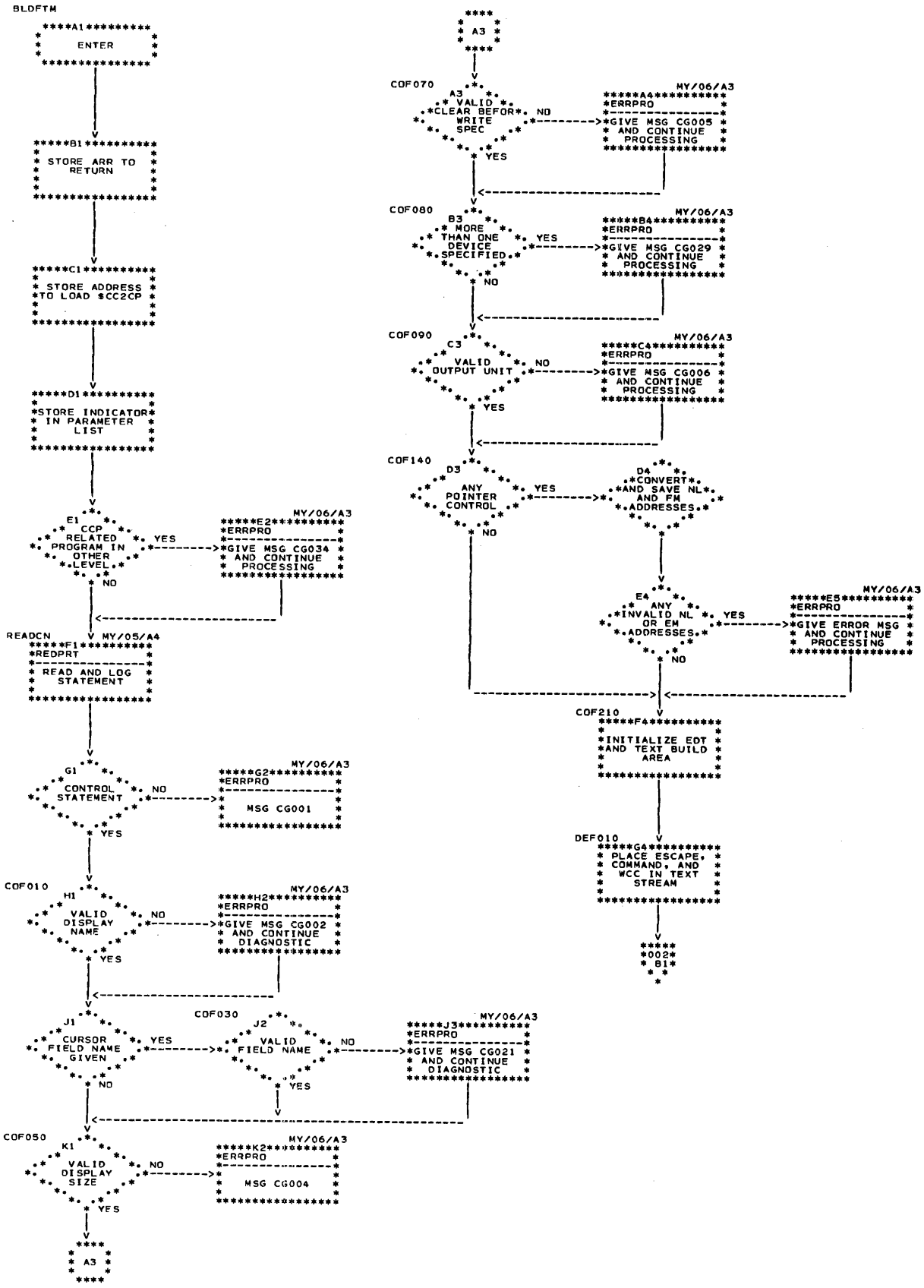


Chart MY (Part 1 of 6). DFGR Print and Diagnose Display Specifications ($CC2CF)

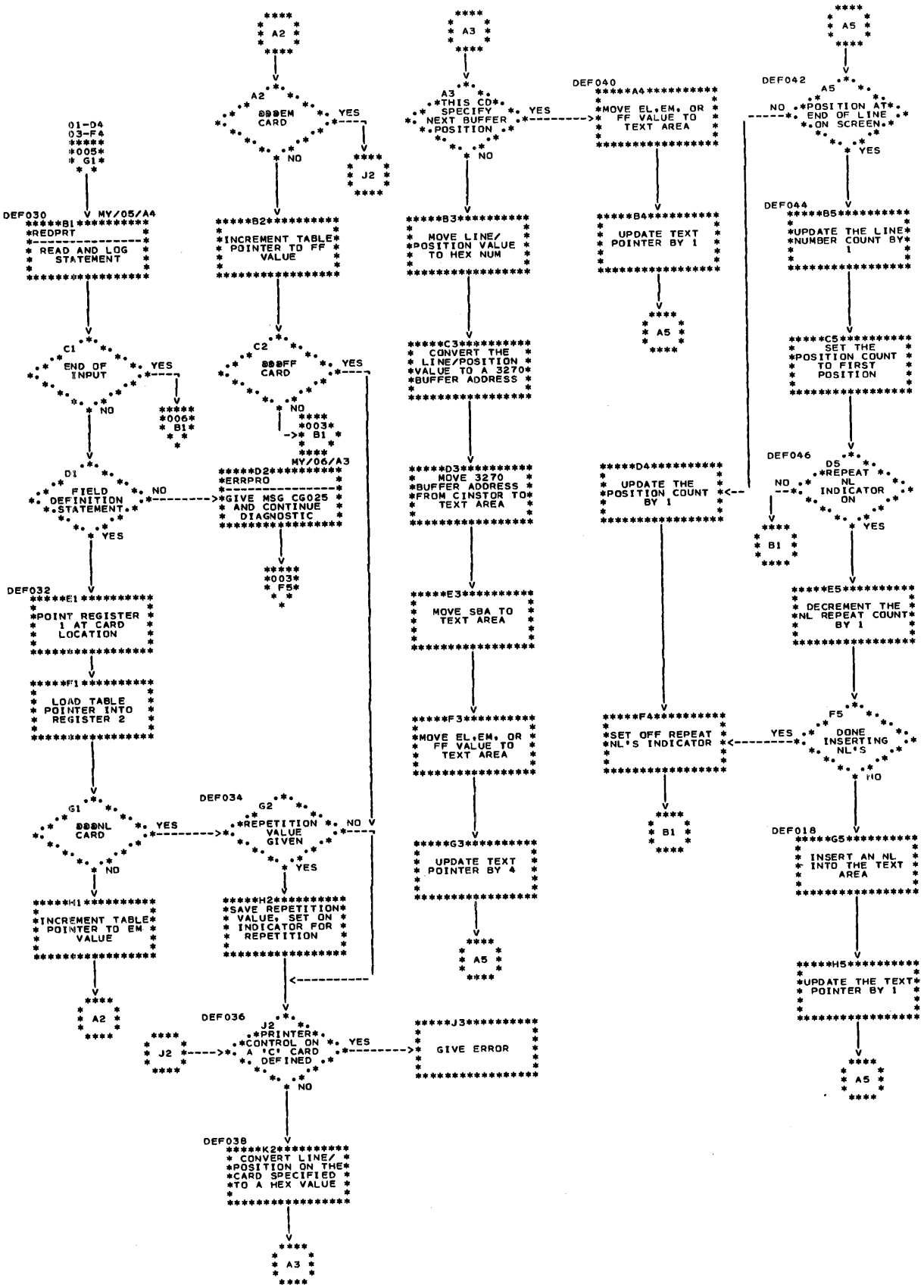


Chart MY (Part 2 of 6). DFGR Print and Diagnose Display Specifications ($CC2CF)

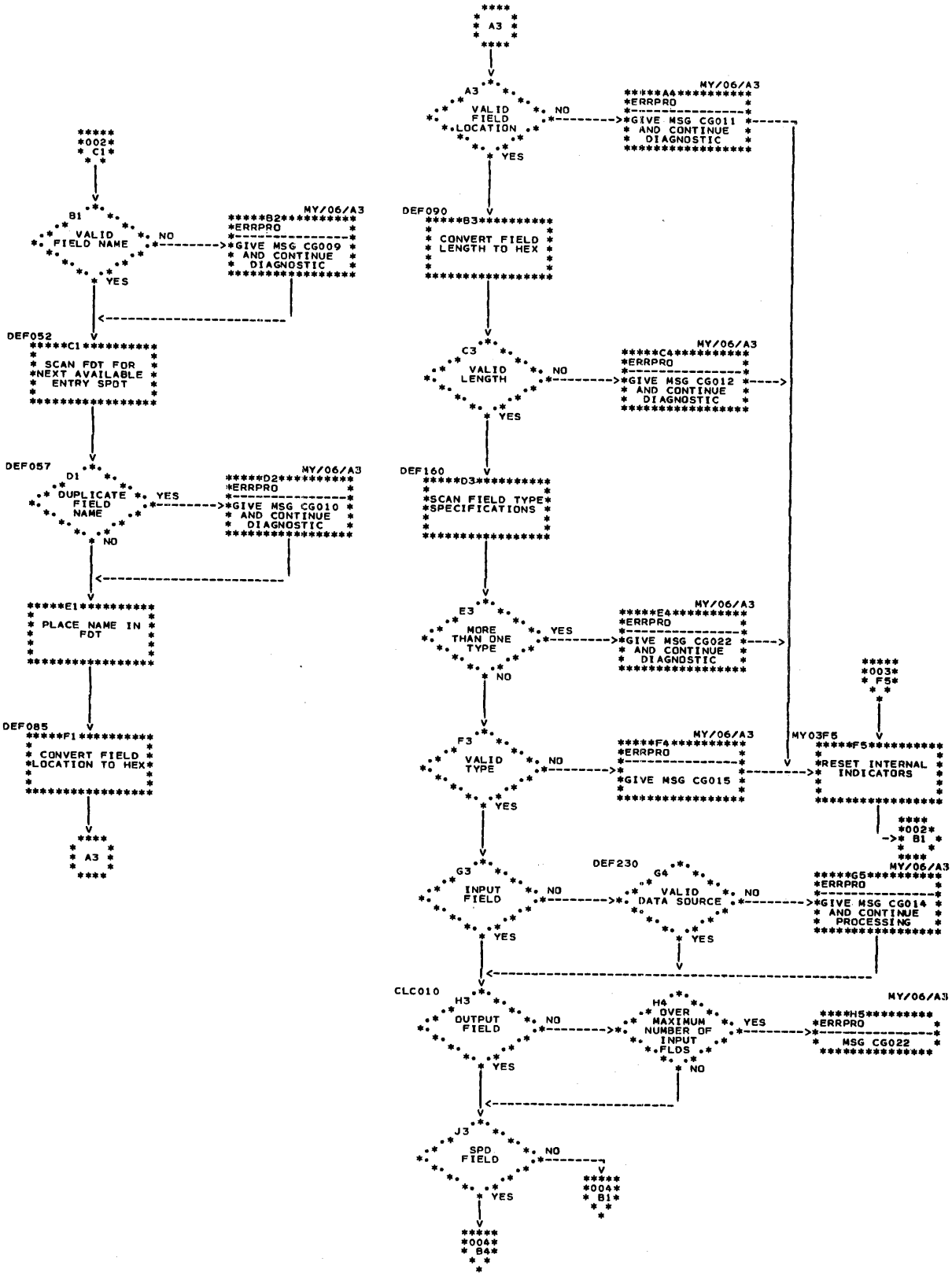


Chart MY (Part 3 of 6). DFGR Print and Diagnose Display Specifications ($CC2CF)

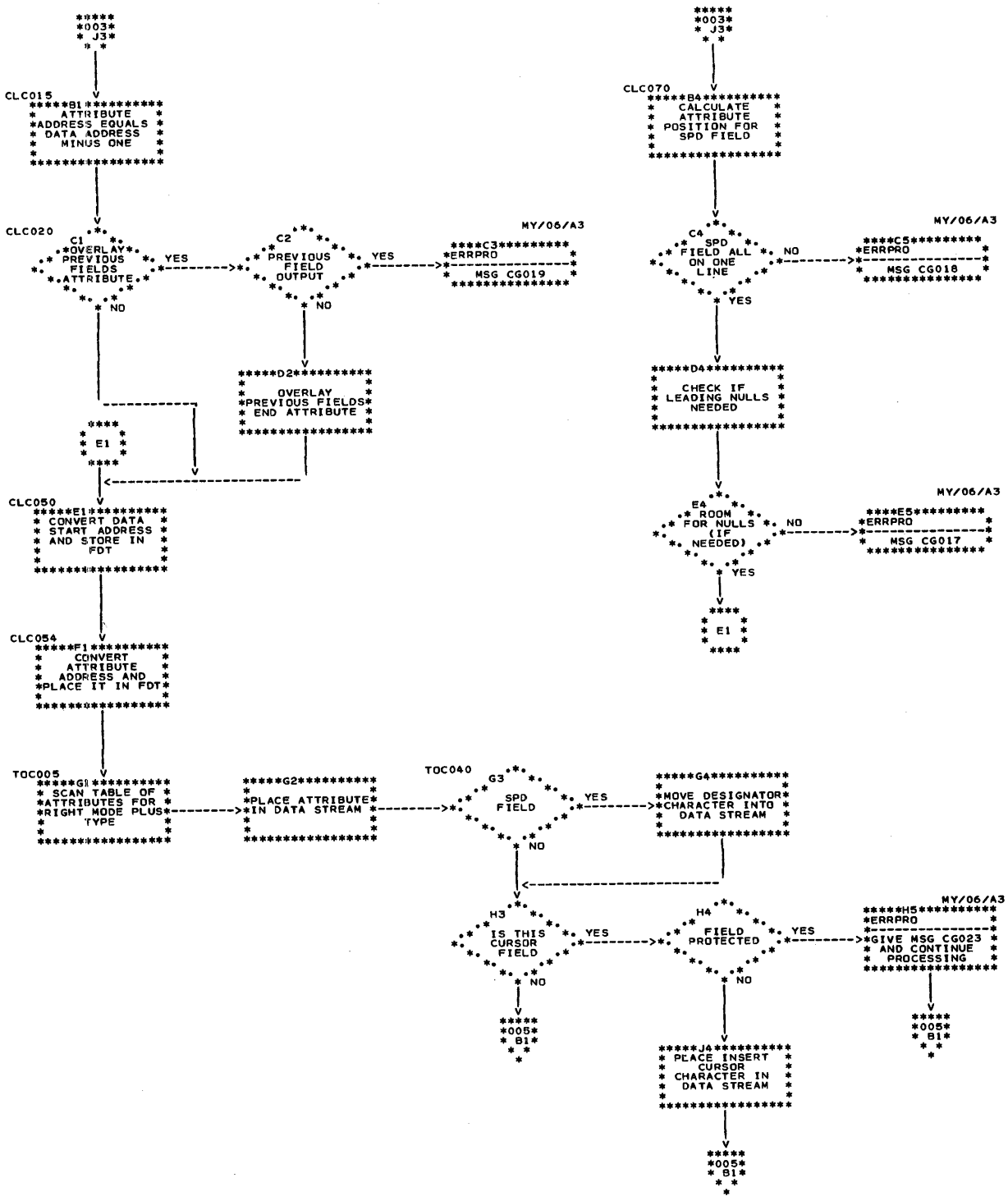


Chart MY (Part 4 of 6). DFGR Print and Diagnose Display Specifications ($CC2CF)

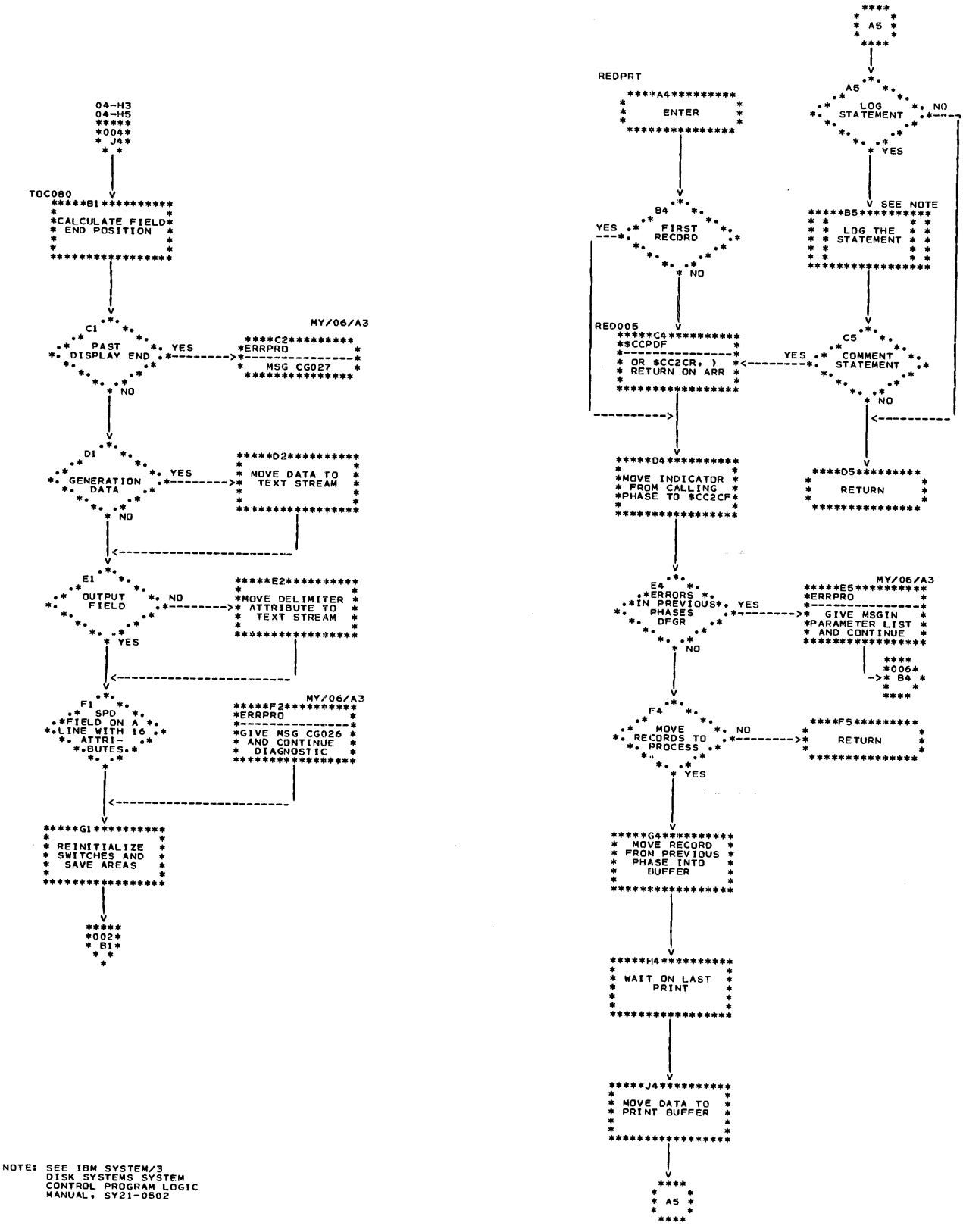


Chart MY (Part 5 of 6). DFGR Print and Diagnose Display Specifications ($CC2CF)

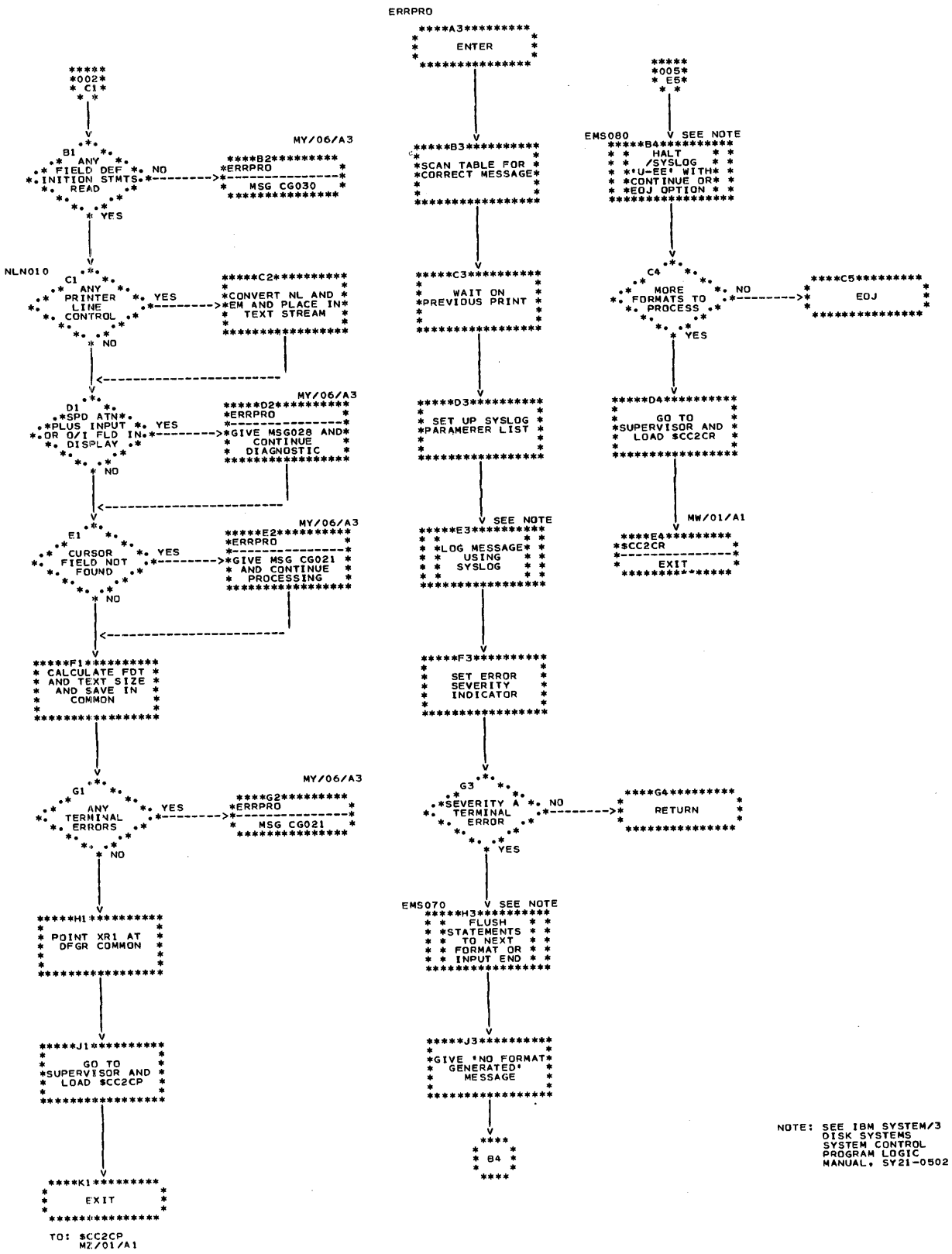


Chart MY (Part 6 of 6). DFGR Print and Diagnose Display Specifications ($CC2CF)

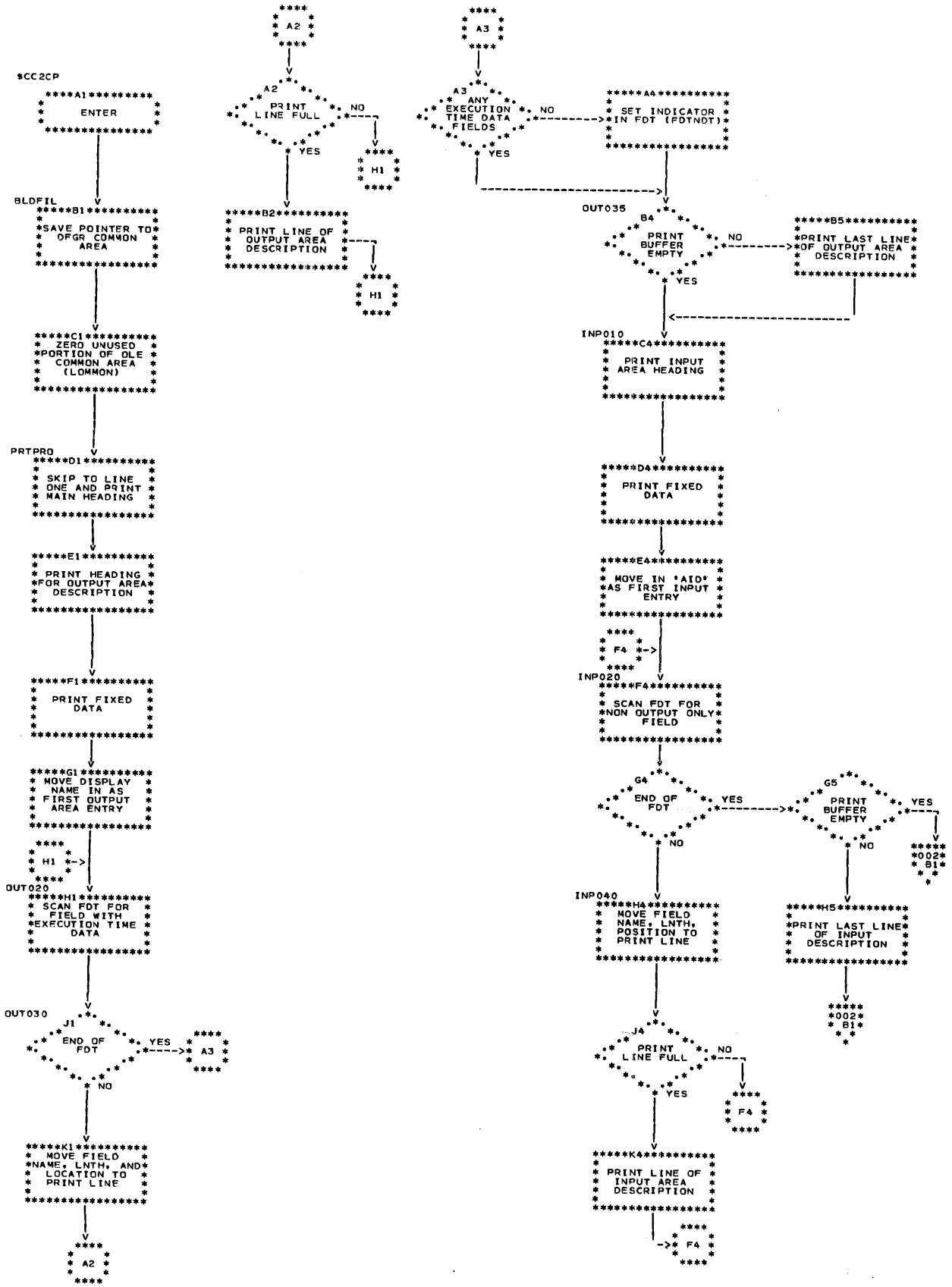


Chart MZ (Part 1 of 2). DFGR Print and Copy to Disk ($CC2CP)

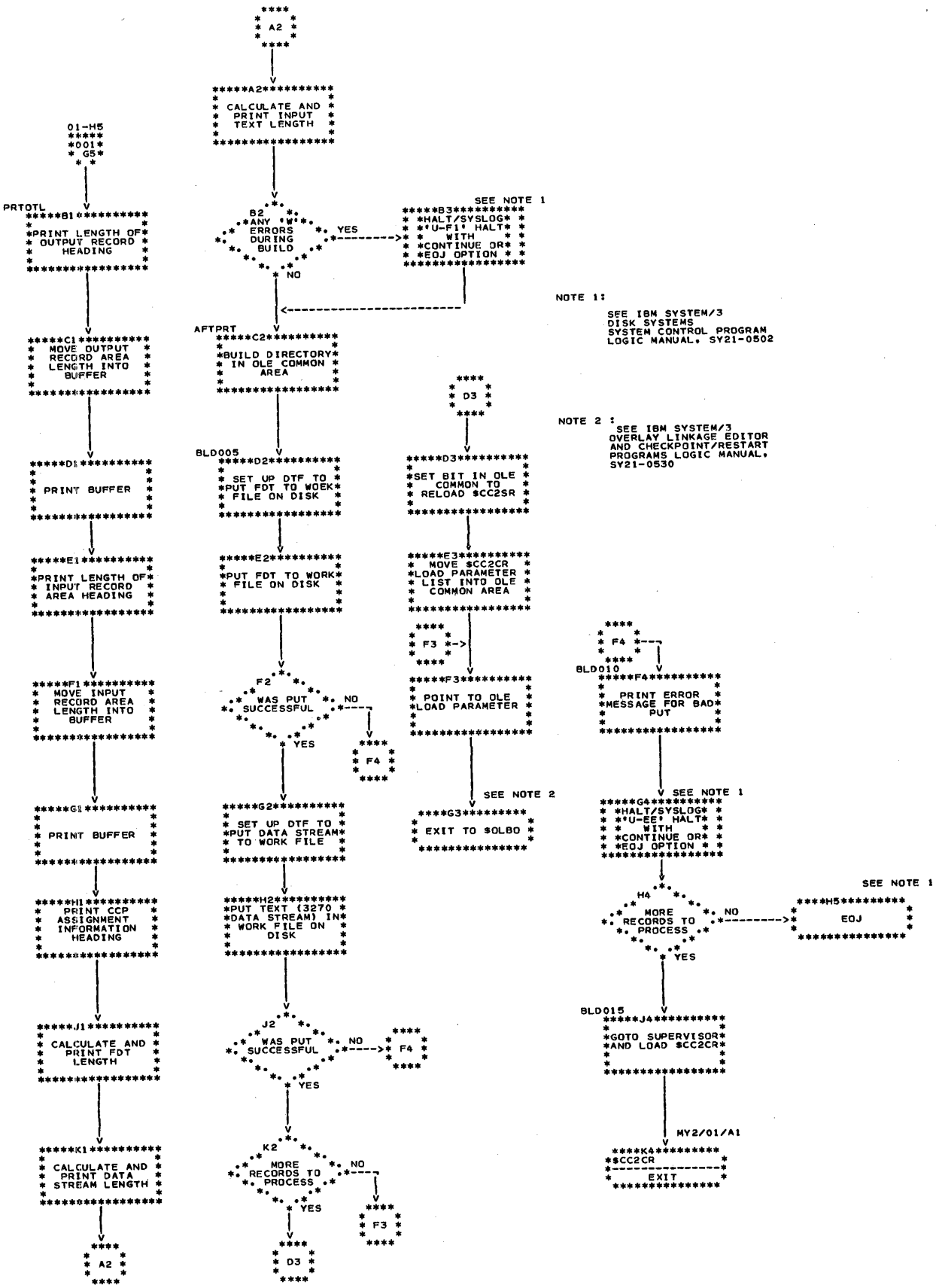


Chart MZ (Part 2 of 2). DFGR Print and Copy to Disk ($CC2CP)

Display Format Test Routine

INTRODUCTION

The Display Format Test Routine ($CCPDT) is a stand-alone program which is used to display DFF formats on an IBM 3270 Information Display System. It runs under the control of DSM and MLMP. Communication between $CCPDT and the 3270 is maintained using the Multiline/ Multipoint (MLMP) Binary Synchronous Communication (BSC) data link.

$CCPDT uses the formats generated by the Display Format Generator Routine (DFGR). The formats must be in the object library of the production pack from which $CCPDT is loaded.

METHOD OF OPERATION

OCL Statements

An initialized MLTERFIL must be on F1, for logging control station terminal statistics. The OCL required to initialize MLTERFIL is:

```
// LOAD $$BSFI,unit

// FILE NAME-MLTERFIL,UNIT-F1,PACK-pack,
   TRACKS-1,LOCATION-track number (optional),
   RETAIN-P

// RUN
```

The following OCL is required to run $CCPDT:

```
// LOAD $CCPDT,unit

// BSCA LINE-2

// RUN
```

$CCPDT uses SYSIN as the input device. When $CCPDT is loaded, the following option menu is logged on SYSLOG:

```
OPTION MENU
A. SEND FORMAT TO TERMINAL
B. READ FORMAT FROM TERMINAL AND PRINT
   IT
C. GO TO EOJ
D. SEND FORMAT TO TERMINAL, POLL, PRINT
   INPUT
E. SEND FORMAT TO SYSTEM PRINTER
```

Enter all options desired through SYSIN in any one of two forms below:

```
O FFFFFF PPPP AAAA

O FFFFFF *
```

$CCPDT uses BSCA-1. If the user wants to use the BSCA LINE-2 instead of BSCA LINE-1, include the optional BSCA LINE-2 statement in the OCL. To use the Integrated Communications Adapter (ICA) on the Model 8, the user must enter the // BSCA LINE-2 statement within the OCL. On Model 4, a prompt is issued that asks if this if BSCA.

Data Input Records

$CCPDT provides the user with two forms of input records and five options:

Starting Position	
	1. O FFFFFF PPPP AAAA
1	O = A, B, C, D, or E (these are described in a following paragraph)
3	F = Format name, which must begin with the characters $Z. The name must be at least three characters long but not more than six characters long.
10	P = Poll characters
15	A = Address characters
	2. O FFFFFF *
1	O = A, B, C, D, or E (these are described in a following paragraph)
3	F = Format name, which must begin with the characters $Z. The name must be at least three characters long but not more than six characters long.
10	* = Default poll and address characters

Note: If the poll and address characters are not specified, $CCPDT will use the previous specified poll and address character. In this way the user can send multiple formats to the same terminal. If the user does not specify the poll and address characters at least once, $CCPDT will default to the following characters:

Poll characters = ⍽⍽⍽⍽ (40404040)
Address characters = ——⍽⍽ (60604040)

Options

A. Send Format to Terminal
$CCPDT reads the selected format. The format is then transmitted to a terminal. This option allows the user to view the format. This option does not allow input or output data to be received or transmitted.

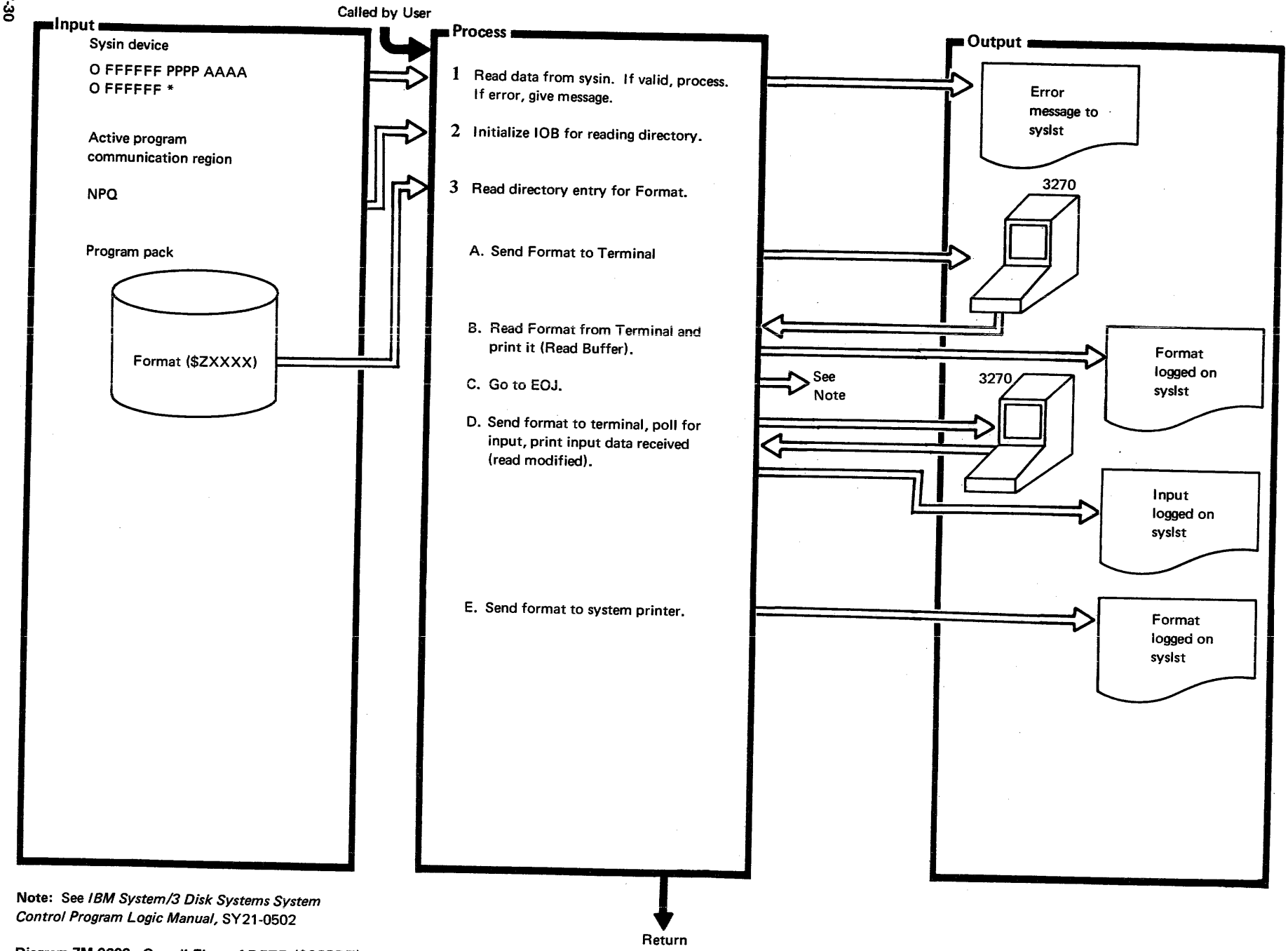
B. Send Format to Terminal, Read It Back and Print
This option allows the user to test for invalid characters within the text stream. $CCPDT reads the entire format from the terminal (read buffer) and prints it. The user should compare the format on the terminal sent to it by option A and the data portion (text stream) of the format printed. For an example, see the *IBM System/3 Communications Control Program Programmer's Reference Manual,* GC21-7579.

C. Go to End of Job
This option should be entered after the user has entered all other options to be used. $CCPDT will then go to end of job.

D. Send Format to Terminal, Poll for Input and Print the Input Data Received
$CCPDT sends the format to the terminal and MLMP polls the terminal for the input the user enters on the terminal keyboard. When attention is requested by the terminal operator by pressing either the ENTER, PA or PF key $CCPDT receives the modified data and prints it on the system printer. This is a means of verifying the modified data against what is being transmitted to $CCPDT.

E. Send Format to System Printer
$CCPDT reads the selected format and prints the entire format on the system printer. This allows the user to view the 3270 text stream and provides hard copy for documentation.

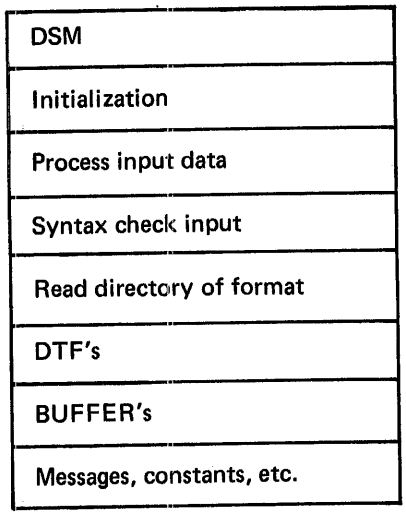
Any format that the user wants to test must be in the object library or the production pack. Before a format is sent to a terminal or the printer, it is scanned for nulls ('00'). In fields within formats defined as having execution time data containing nulls, the nulls are replaced by asterisks. Also, in overlay formats with input fields containing nulls, these nulls are replaced by asterisks. Since nulls do not appear on the terminal, by replacing the nulls with asterisks the user is able to see the null fields.



Note: See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502

Diagram 7M.0600. Overall Flow of DFTR ($CCPDT)

PROGRAM ORGANIZATION



Main storage map for $CCPDT

$CCPDT Module Description

ENTRY POINT: DTOOOO
FUNCTIONS: Displays DFF formats on a 3270 Information Display System
INPUT: Option record via sysin
OUTPUT: Format sent to terminal. Format sent to syslst. Input data from format logged on system printer.
EXIT, NORMAL: System end of job routine. DSM EOJ is described in *IBM System/3 Disk Systems System Control Program Logic Manual,* SY21-0502.

Printer Format Generator

INTRODUCTION

The Printer Format Generator reads records generated from printer format specification sheets (using $CCPPF for single or multiple format builds), and creates a 3270 printer format. Printer format specifications can be read from the system input device or the source library. The Printer Format Generator prints information for the user about the format as it is built. When the build is completed, the Printer Format Generator calls the Overlay Linkage Editor Librarian Phase ($OLBO) to catalog the printer format into the object library. ($OLBO is described in *IBM System/3 Overlay Linkage Editor and Checkpoint/Restart Programs Logic Manual,* SY21-0530.)

Two types of specification statements are prepared by the user to define a printer format: printer control statements and field definition statements. At least one printer control statement is required and it must precede all field definition statements. The printer control statement provides information about the entire printer format. Each field in the printer format must be defined on a field definition statement. The field definition statements define each field by name, length, and location. They also tell where data for each field is defined.

The Printer Format Generator runs offline; that is, it does not run under the control of CCP, although the printer format it generates is used by the Display Format Control Routine (DFCR) that does run under CCP. The Printer Format Generator can run in either level of a DPF system. It can run while CCP is running in the other level unless PFGR will put a format on the pack from which CCP was loaded.

The Printer Format Generator runs under System/3 DSM and requires 18K of main storage for execution. The Printer Format Generator requires a 2-track work file on disk called $WORK. The system in which the Printer Format Generator is running must contain the Overlay Linkage Editor Librarian modules. If reading display format specifications from the source library, a $SOURCE file is needed. If a $SOURCE file statement is not specified, a 5-track $SOURCE file is automatically allocated. However, whether the $SOURCE file is specified or automatically allocated, this file has to be large enough to contain all printer format specifications.

METHOD OF OPERATION

Functions

The Printer Format Generator performs the following functions (see Diagram 7M.0700 for an illustrated overview of the Printer Format Generator using single or multiple format builds read from sysin or the source library):

1. Reads printer format specification statements.

2. Produces a printout of the specification statements, analyzes the specifications for errors, and logs diagnostic error messages as required.

3. Builds the printer format as a 2-part table structure:
   a. Field Descriptor Table (FDT)—containing descriptive field information including the symbolic name of the field.
   b. 3270 Data Stream—containing output data if provided, and 3270 device-dependent control information required for formatting all fields defined.

4. Provides a printout of field names, in the order in which they must appear in the output record area if data from the Field Descriptor Table is required by the application program using the Display Format Facility.

5. Calculates and prints the following:
   a. Length of the output record area required in the DFF program.
   b. Decimal length of the Field Descriptor Table.
   c. Decimal length of the 3270 output data stream.

6. Places the printer format in a work file ($WORK) on disk and then invokes the Overlay Linkage Editor Librarian Phase ($OLBO) to catalog the printer format in an object library on disk.

Components—for Single and Multiple Format Builds, Read from Sysin or the Source Library

The Printer Format Generator consists of four components:

- $CCPPF — PFGR Build Common Area (LOMMON), reads from sysin or the source library and opens files.
- $CC2CS — PFGR reads from the $SOURCE file.
- $CC2CG — PFGR prints and diagnoses display specifications.
- $CC2CQ — PFGR prints and copies to disk.

The following describes these components and their functions.

$CCPPF

$CCPPF (Diagram 7M.0800) receives initial control when loaded by the user, then performs the following functions:

1. Builds the Overlay Linkage Editor communications area (LOMMON).
2. Automatically allocates $SOURCE file if one is not specified.
3. Opens $WORK and $SOURCE files.
4. Copies data from sysin or the source library to $SOURCE file.
5. Finds and loads $CC2CS.

$CC2CS

For implementation of single or multiple format builds, see the *IBM System/3 Communications Control Program Programmer's Reference Manual,* GC21-7579.

$CC2CS (Diagram 7M.0900) receives control from $CCPPF, $CC2CG, $CC2CQ, or $OLBO, then performs the following functions:

1. Finds and loads $CC2CG after $CC2CS.
2. Reads printer format specification statements from the $SOURCE file.

$CC2CG

$CC2CG (Diagram 7M.1000) receives control from $CC2CS, then performs the following functions:

1. Logs and processes printer format specification statements.
2. Builds the printer format in main storage.
3. Builds the PFGR common area containing pointers and information about the printer format being generated.
4. Finds and loads $CC2CQ if no terminating errors occurred.
5. Finds and loads $CC2CS if a terminating error did occur.

$CC2CQ

$CC2CQ (Diagram 7M.1100) receives control from $CC2CG, then performs the following functions:

1. Logs information to the user about the printer format being generated.
2. Copies the printer format from main storage to a work file on disk ($WORK).
3. Finds and loads the Overlay Linkage Editor Library Phase ($OLBO) if no terminating errors occurred.
4. Finds and loads $CC2CS if a terminating error did occur.

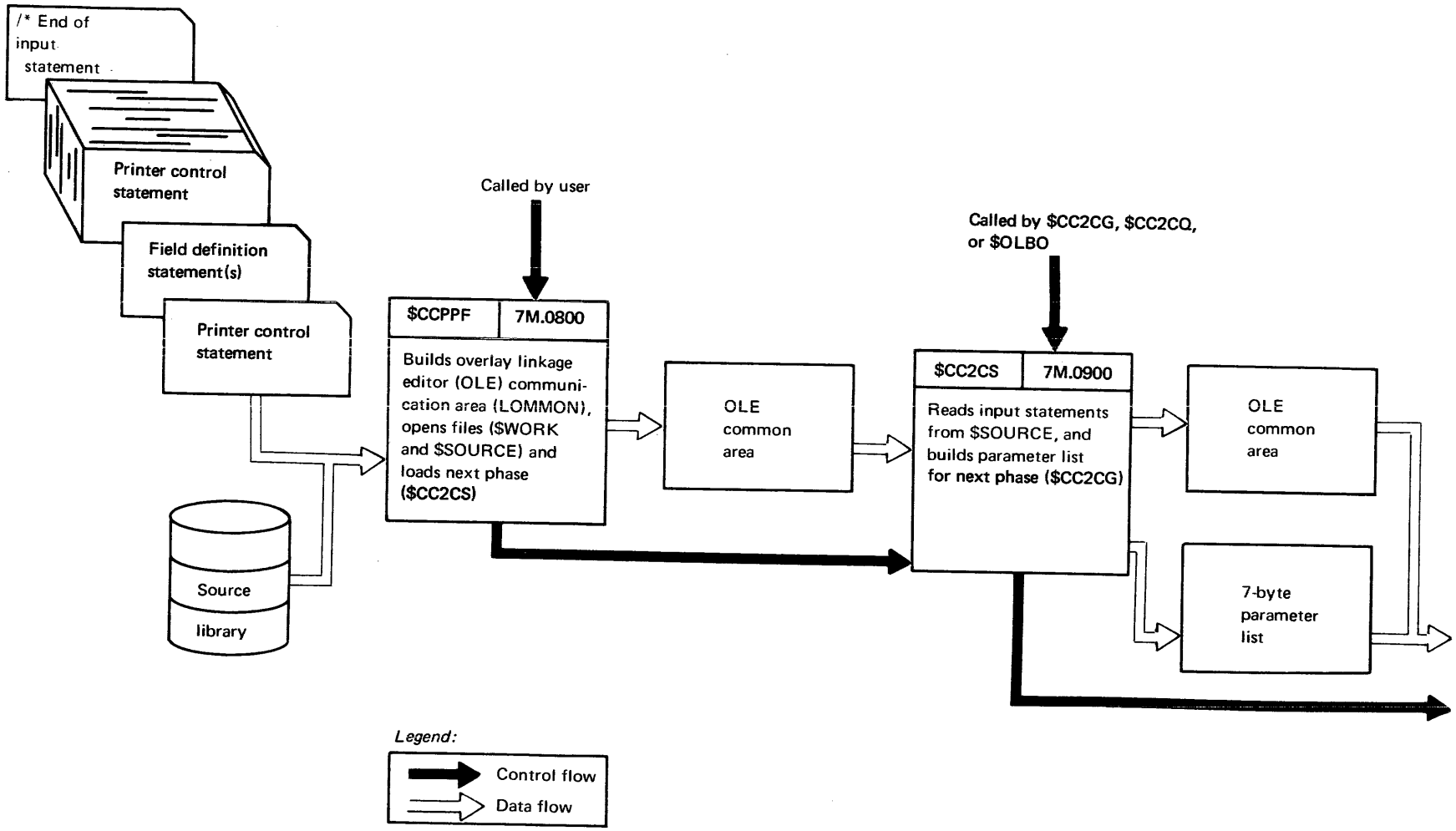


Diagram 7M.0700 (Part 1 of 2). Overall Flow of PFGR (using single or multiple format builds)

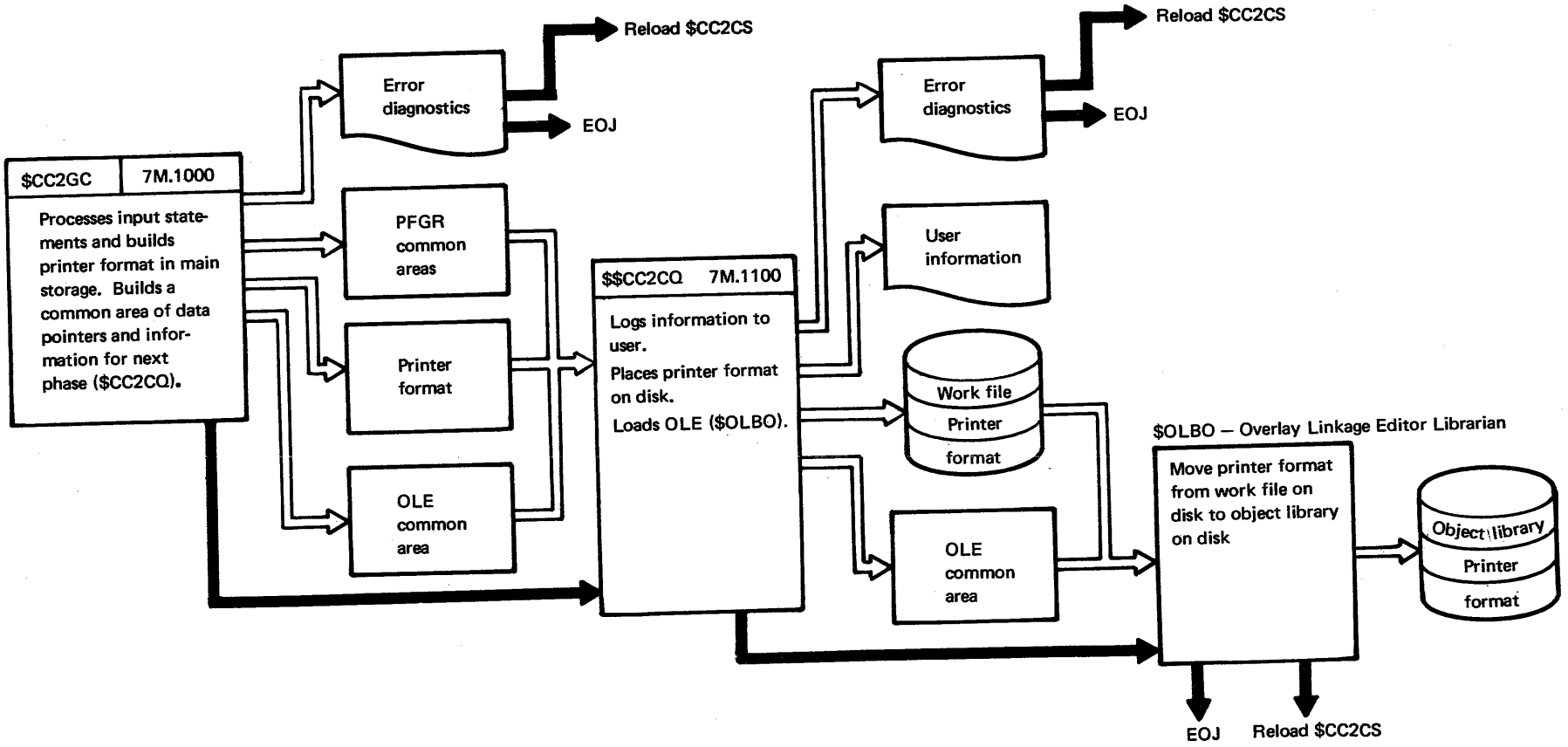
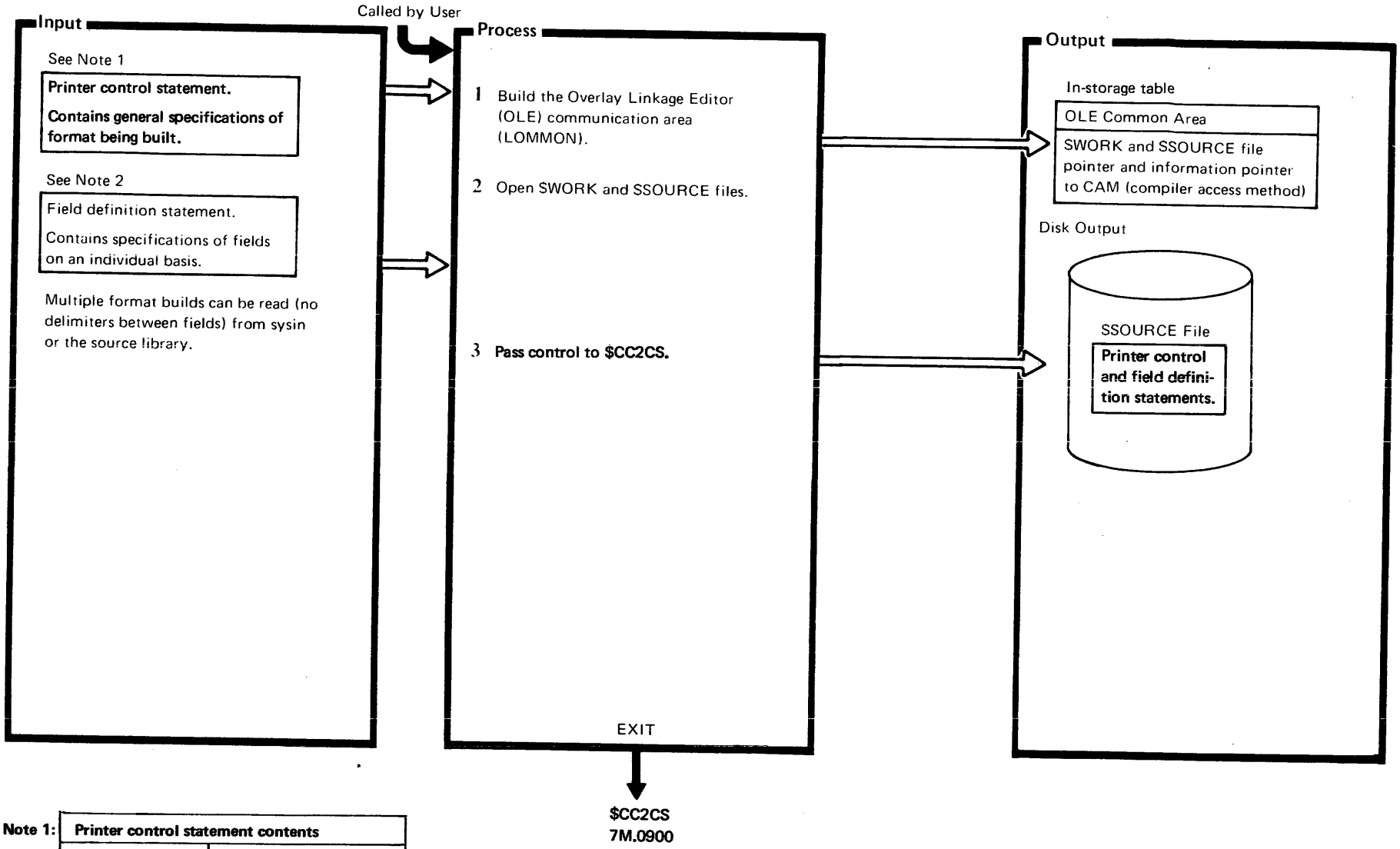


Diagram 7M.0700 (Part 2 of 2). Overall Flow of PFGR (using single or multiple format builds)



$CC2CS
7M.0900

Note 1:

Printer control statement contents	
Required	**Optional**
Printer name Printer size Line length	Platen length Lines per page Multiple pages Vertical forms feed Disk unit for output Katakana Print/no-print

Note 2:

Field definition statement contents	
Required	**Optional**
Field name Field location Field length	Field data source Field data Repeat last character continuation

Diagram 7M.0800. $CCPPF

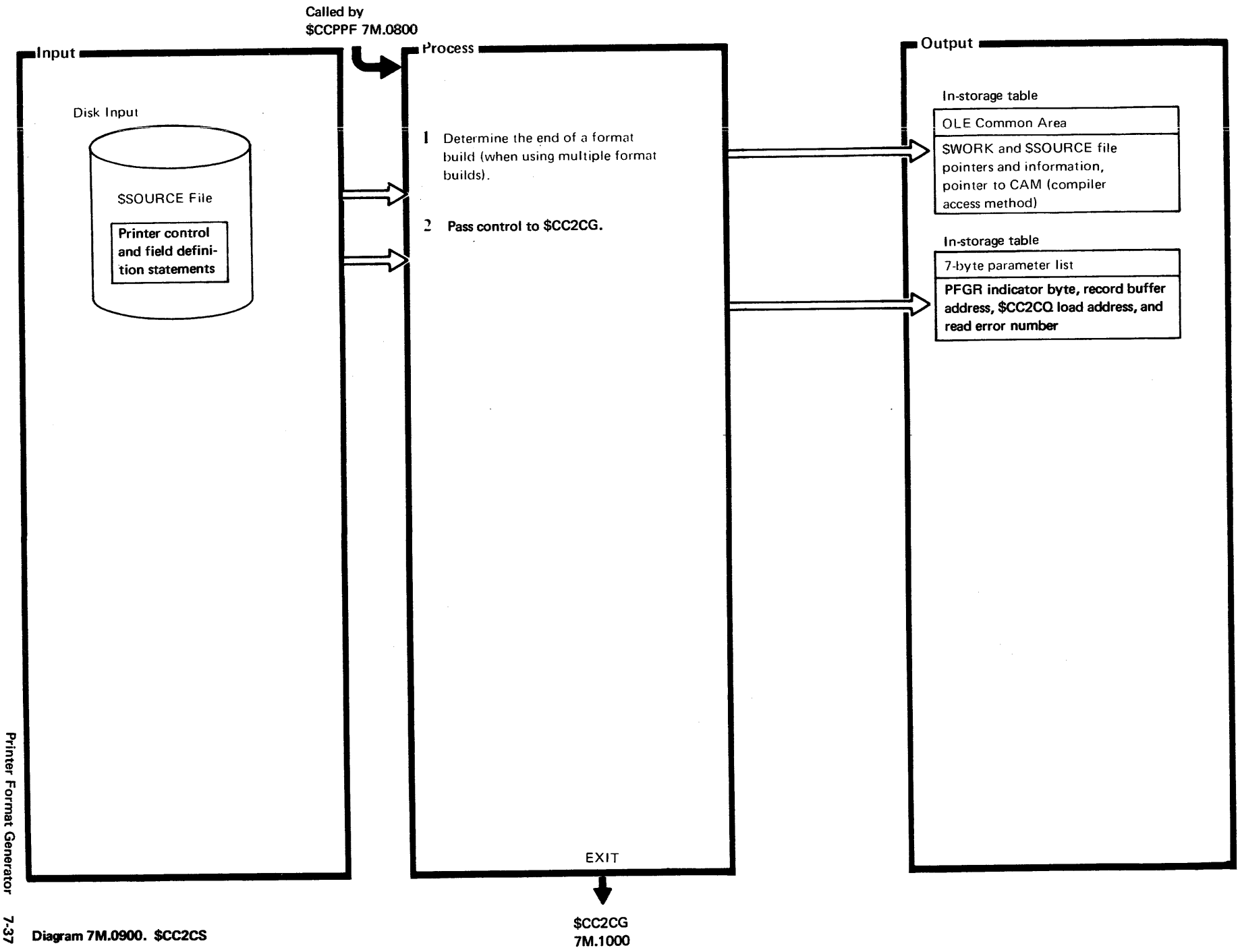
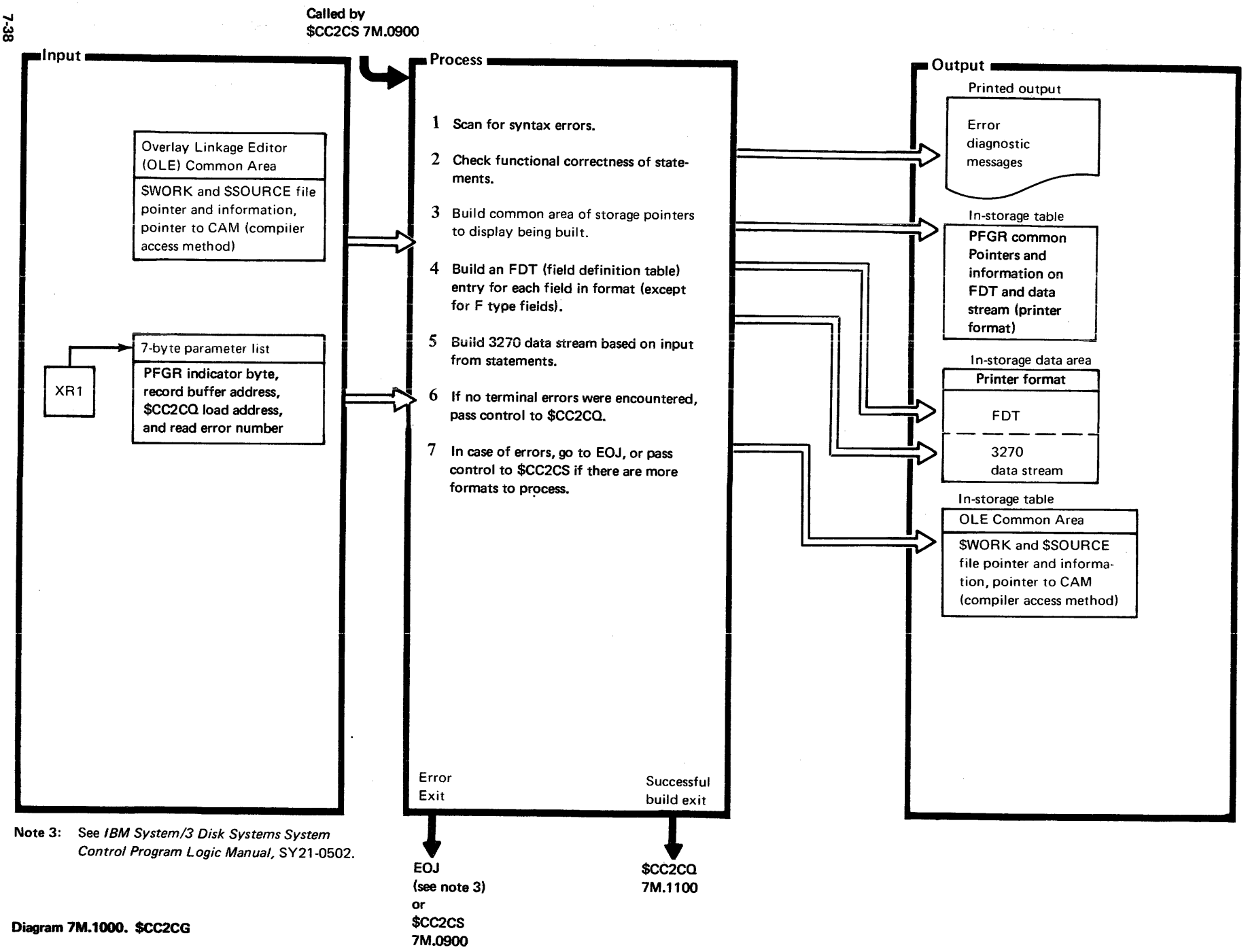


Diagram 7M.0900. $CC2CS



Note 3: See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.

Diagram 7M.1000. $CC2CG

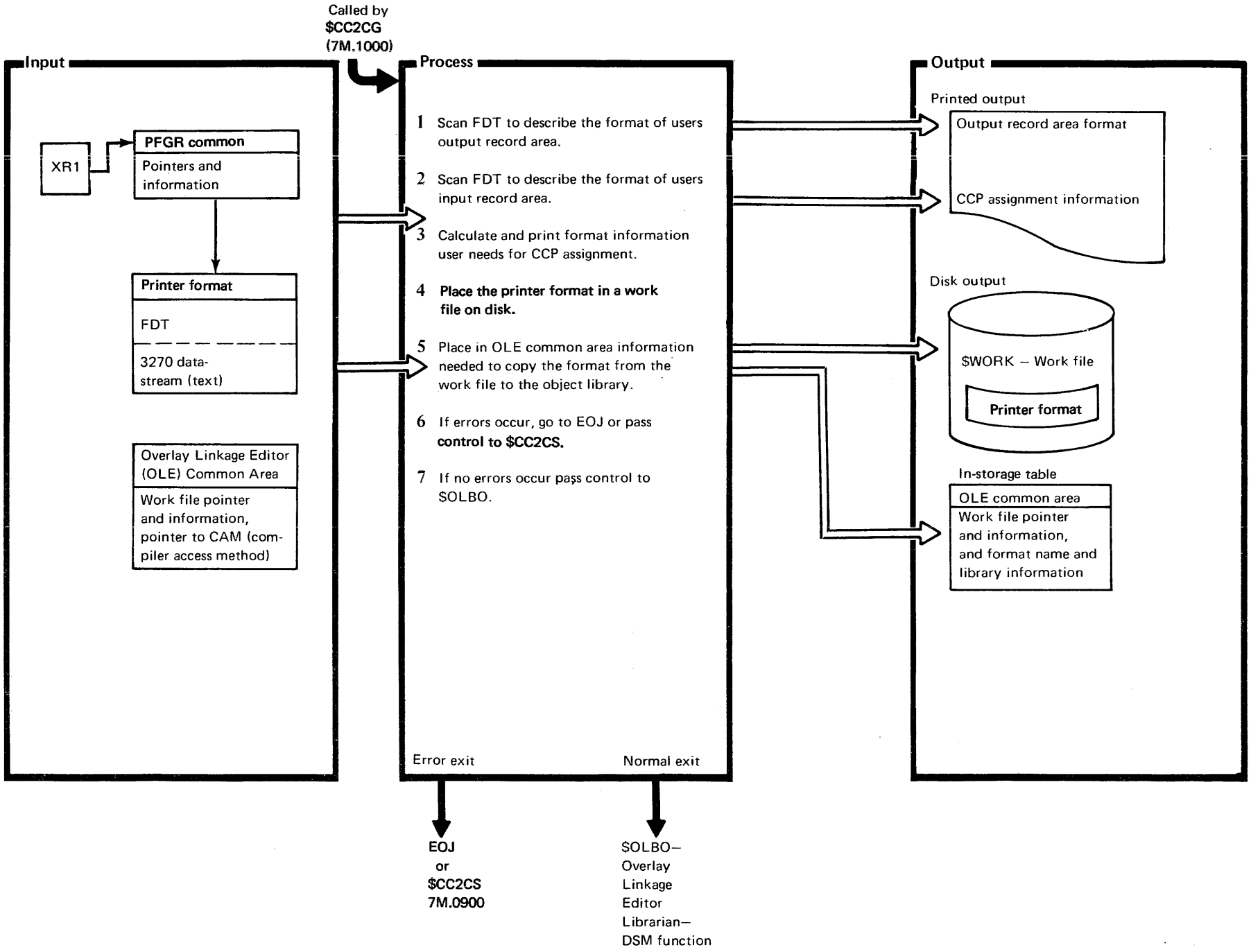


Diagram 7M.1100. $CC2CQ

PROGRAM ORGANIZATION

This section describes in detail the organization of the Printer Format Generator routines. It includes both single and multiple format builds, read from sysin or the source library. Included in the following discussions are:

- Module descriptions derived from listing prologs.
- Main storage maps showing the layout of main storage at the time the routine being described has control (Figure 7-2).
- Flowcharts showing the internal logic of the routine being described.

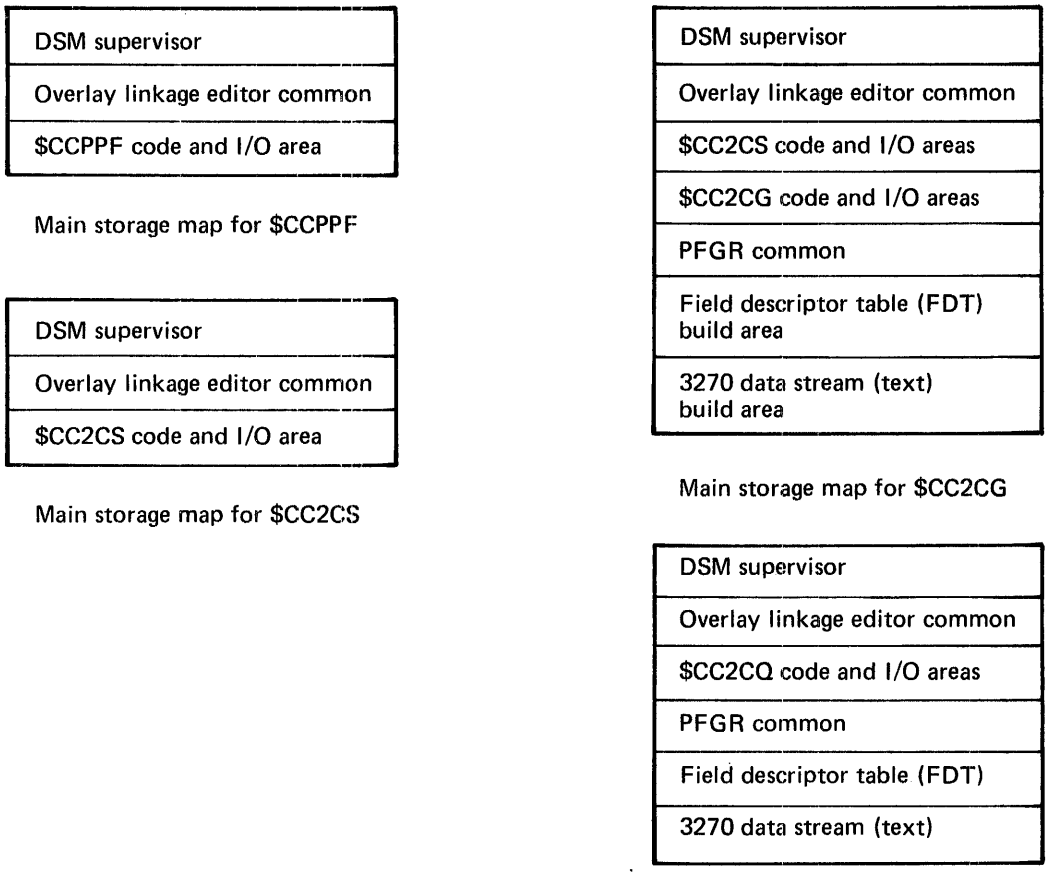


Figure 7-2. Storage Layout of PFGR

Printer Format Generator Build Common Area (LOMMON), Read from Sysin, the Source Library, and Open Files ($CCPPF)

ENTRY POINT: CRDINP
CHART: M1
FUNCTIONS:
- Builds the overlay linkage editor communications area (LOMMON).
- Automatically allocates $SOURCE file if one was not specified.
- Opens $WORK and $SOURCE files.
- Reads the printer format specifications from sysin or the source library, and copies to the $SOURCE file.
- Loads the next phase of the PFGR ($CC2CS).

INPUT: Printer format specifications via system or the source library. (Sysin routines are described in *IBM System/3 Disk Systems Control Program Logic Manual,* SY21-0502.)
OUTPUT:
- Overlay linkage editor communications area (LOMMON).
- $SOURCE file containing printer control and field definition statements.

EXITS:
- Normal: Loads and passes control to the next phase of PFGR ($CC2CS).
- Error: A terminal error causes a halt, followed by EOJ. Halt/syslog and system EOJ are described in *IBM System/3 Disk Systems System Control Program Logic Manual,* SY21-0502.

Printer Format Generator Read Single or Multiple Formats from the $SOURCE File ($CC2CS)

ENTRY POINT: BLDINP
CHART: M2
FUNCTIONS:
- Reads printer format specifications from the $SOURCE file.
- Loads the next phase of the PFGR ($CC2CG) following $CC2CS.

INPUT: Printer format specifications via $SOURCE. (Data is moved into the $SOURCE file from sysin, or from the source library via $CCPPF.)
OUTPUT:
- Overlay linkage editor communications area (LOMMON).
- A 7-byte parameter list containing information used by the next phase ($CC2CG).

EXITS:
- Normal: Loads and passes control to the next phase of PFGR ($CC2CG).
- Error: A terminal error causes an error number to be placed in the parameter list. Control is then passed to the next phase ($CC2CG).

Printer Format Generator Print and Diagnose Display Specifications ($CC2CG)

ENTRY POINT: BLDFMT
CHART: M3
FUNCTIONS:
- Logs the printer format specifications.
- Diagnoses any errors in the specifications.
- Builds the printer format in main storage.
- Builds the PFGR common area.
- Calls the next phase of the PFGR ($CC2CQ).

INPUT: A 7-byte parameter list containing information used by this phase, and a printer format specification read by a previous phase.
OUTPUT:
- Overlay linkage editor communications area (LOMMON).
- A 2-part table which makes up the printer format. The first part, the Field Descriptor Table (FDT), contains the names of the fields making up the display format along with other field descriptive information. The second part of the table contains the 3270 data stream for the printer format.
- PFGR COMMON, which contains information about, and pointers to, the printer format being processed.
- Diagnostic messages describing any error encountered while processing the printer format specifications.

EXITS:
- Normal: Loads and passes control to the next phase of PFGR ($CC2CQ).
- Error: If a terminal error occurred, a message is given to the operator (halt/syslog) with the options to go to EOJ or build the next format (pass control to $CC2CS). Halt/syslog and system EOJ are described in *IBM System/3 Disk Systems System Control Program Logic Manual,* SY21-0502.

Printer Format Generator Print and Copy to Disk ($CC2CQ)

ENTRY POINT: $CC2CQ
CHART: M4
FUNCTIONS:

- Logs information about the printer format being generated.
- Copies the printer format from main storage to a work file ($WORK) on disk.
- Call the Overlay Linkage Editor Librarian Phase to place the format in the object library on disk. ($OLBO is described in *IBM System/3 Overlay Linkage Editor and Checkpoint/Restart Programs Logic Manual,* SY21-0530.)

INPUT:

- The printer format in main storage.
- The PFGR common area, which contains information about the printer format being generated.
- On entry to $CC2CQ index register 1 points to PFGR common area, which contains pointers to the format.

OUTPUT:

- Printed output describing the printer format just built.
- A work file ($WORK) containing the printer format.
- Overlay Linkage Editor common area containing information needed to catalog the printer format in an object library.

EXITS:

- Normal: Loads and passes control to the Overlay Linkage Editor Librarian Phase ($OLBO).
- Error: If a terminal error occurs, a message is given to the operator (halt/syslog) with the options to go to EOJ or build the next format (pass control to $CC2CS). Halt/syslog and system EOJ are described in *IBM System/3 Disk Systems System Control Program Logic Manual,* SY21-0502.

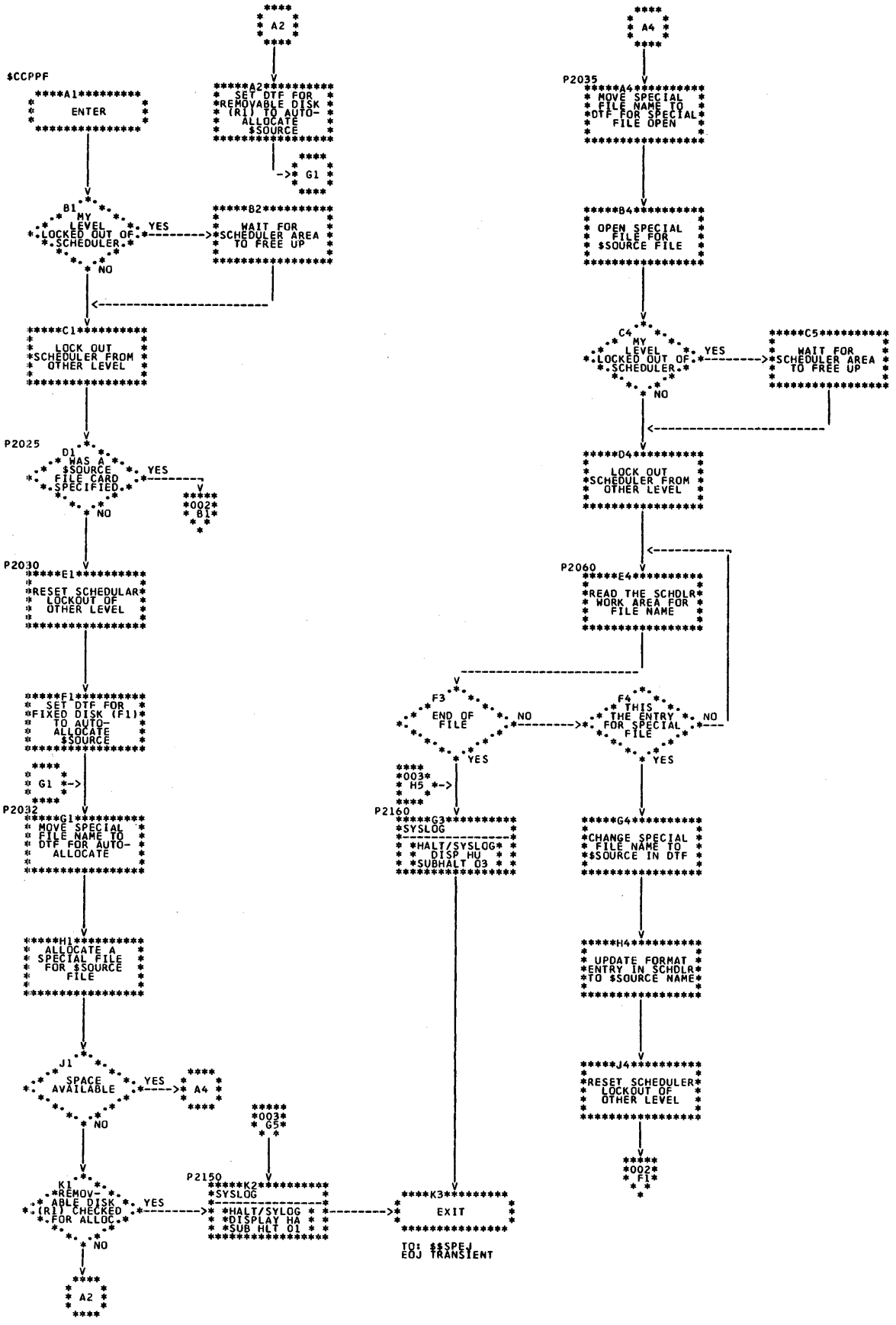


Chart M1 (Part 1 of 3). PFGR Build Common Area (LOMMON), $CCPPF

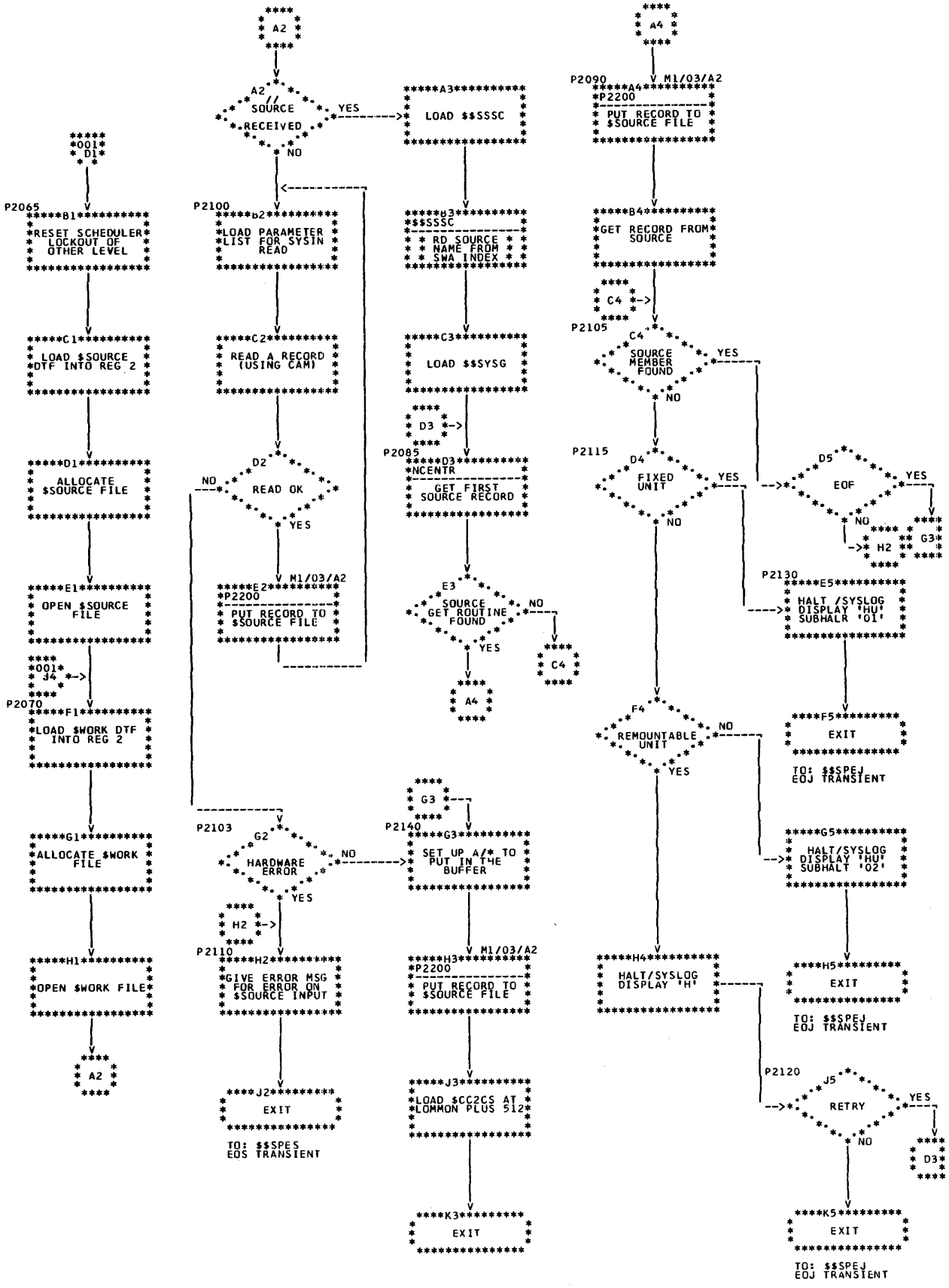


Chart M1 (Part 2 of 3). PFGR Build Common Area (LOMMON), $CCPPF

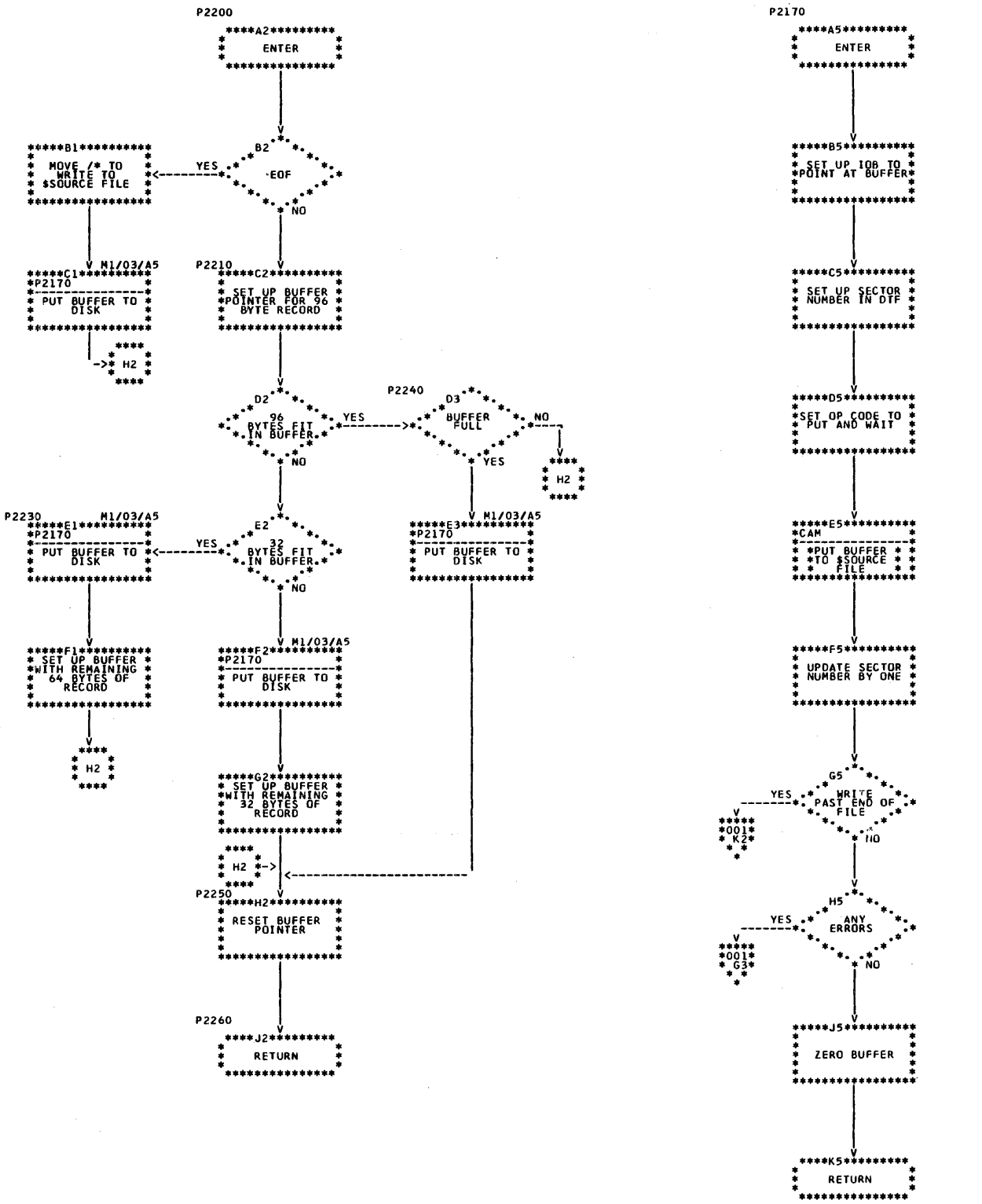
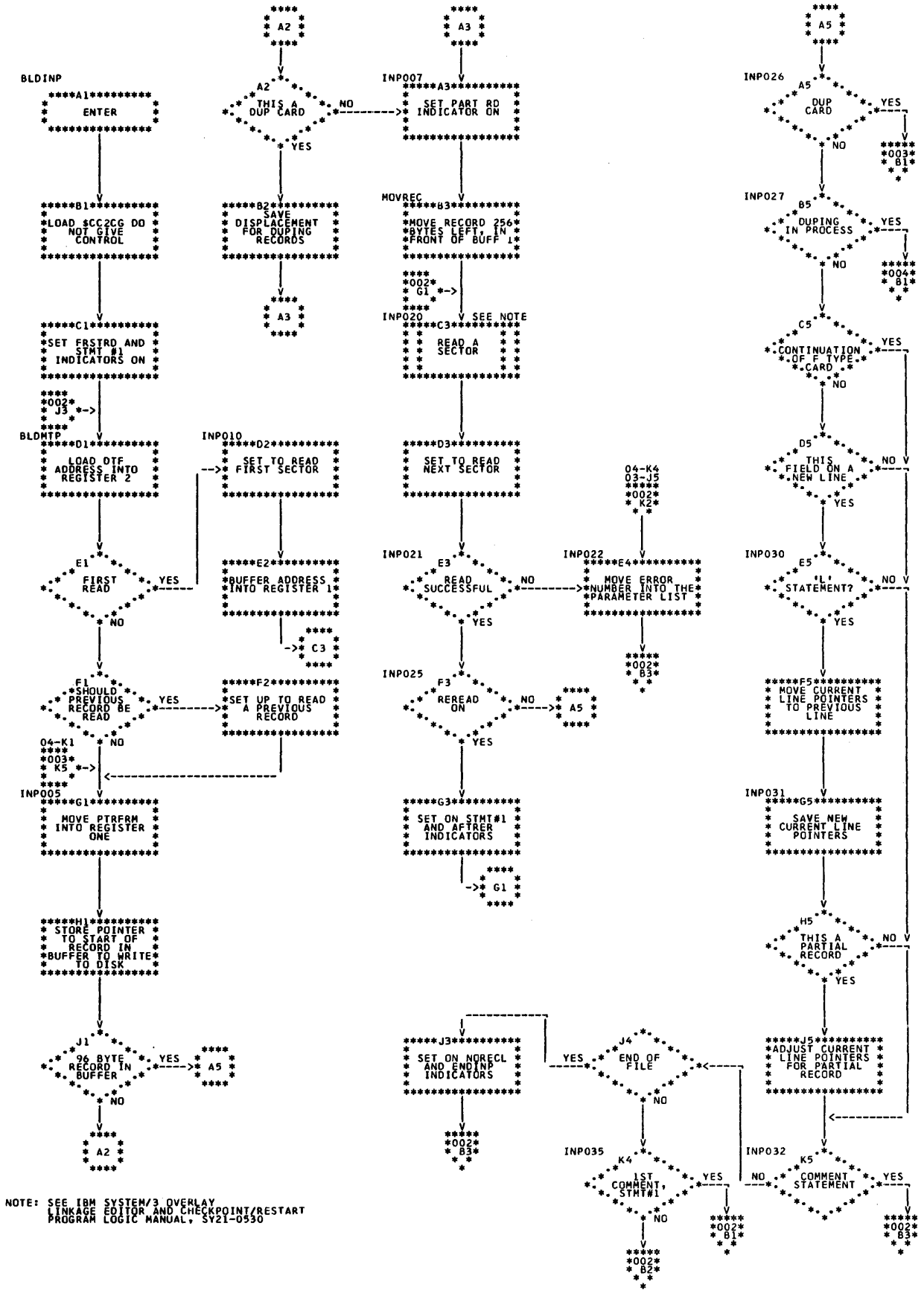


Chart M1 (Part 3 of 3). PFGR Build Common Area (LOMMON), $CCPPF



NOTE: SEE IBM SYSTEM/3 OVERLAY LINKAGE EDITOR AND CHECKPOINT/RESTART PROGRAM LOGIC MANUAL, SY21-0530

Chart M2 (Part 1 of 4). PFGR Read Multiple from Sysin or Source Library ($CC2CS)

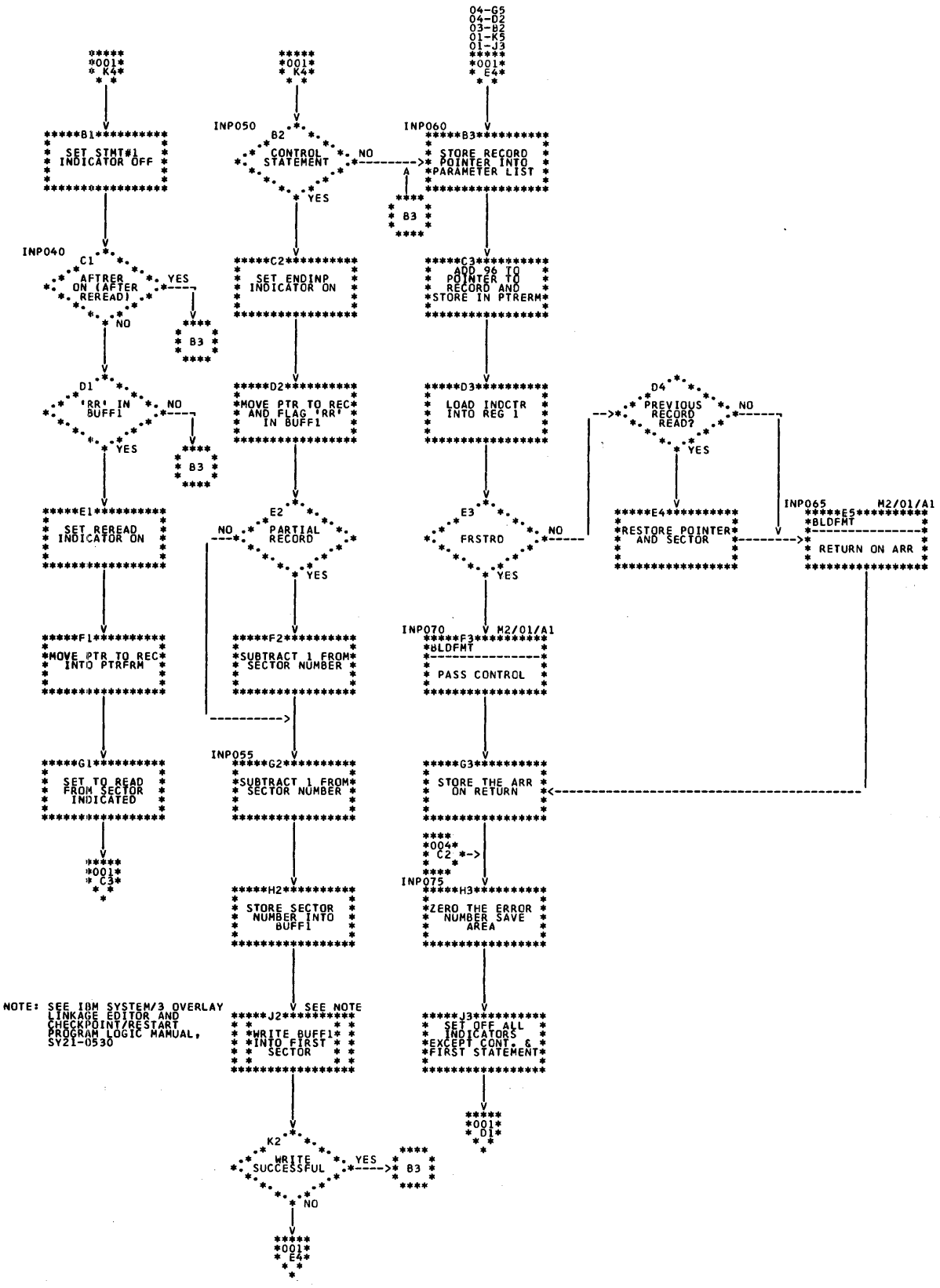


Chart M2 (Part 2 of 4). PFGR Read Multiple from Sysin or Source Library ($CC2CS)

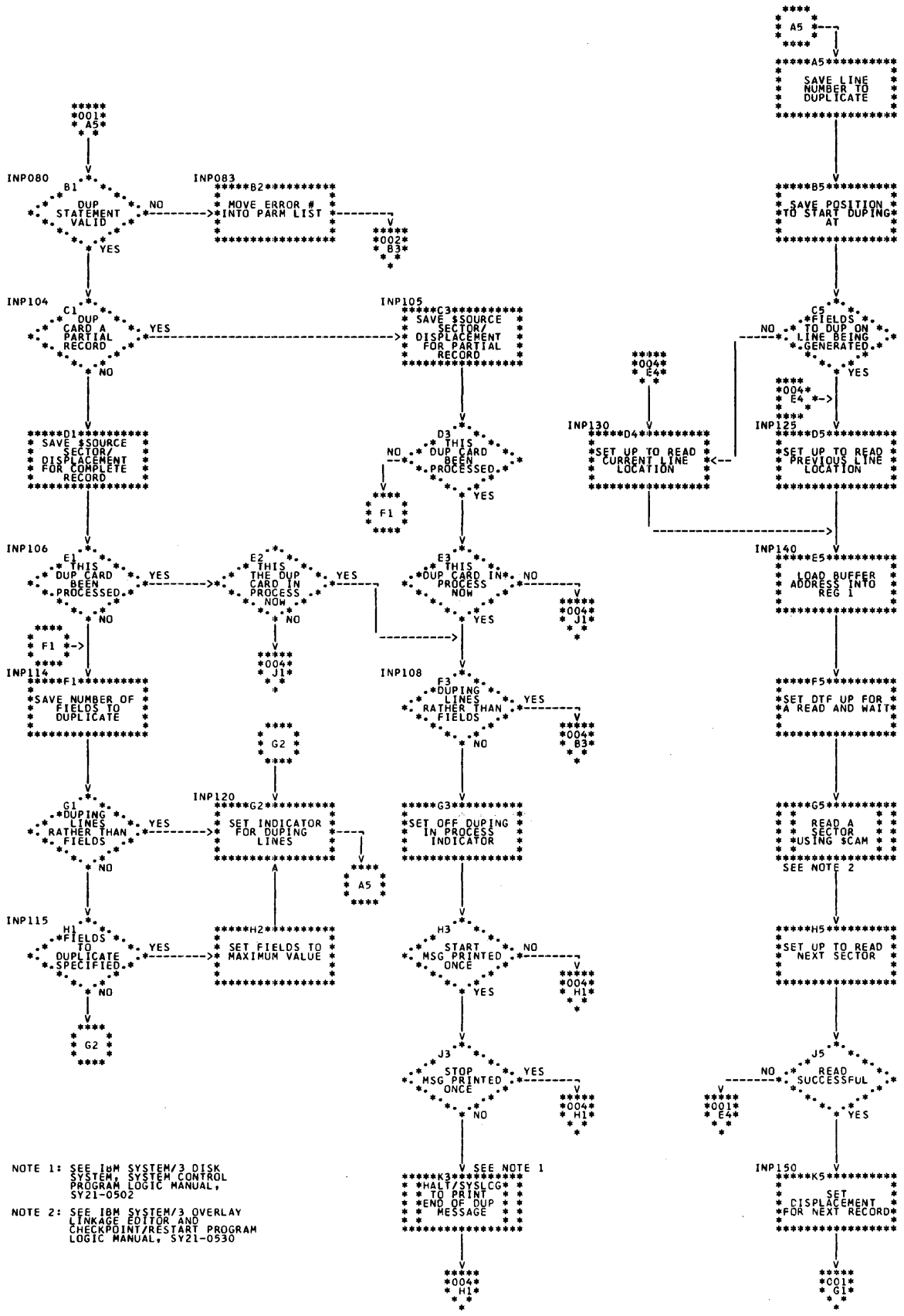


Chart M2 (Part 3 of 4). PFGR Read Multiple from Sysin or Source Library ($CC2CS)

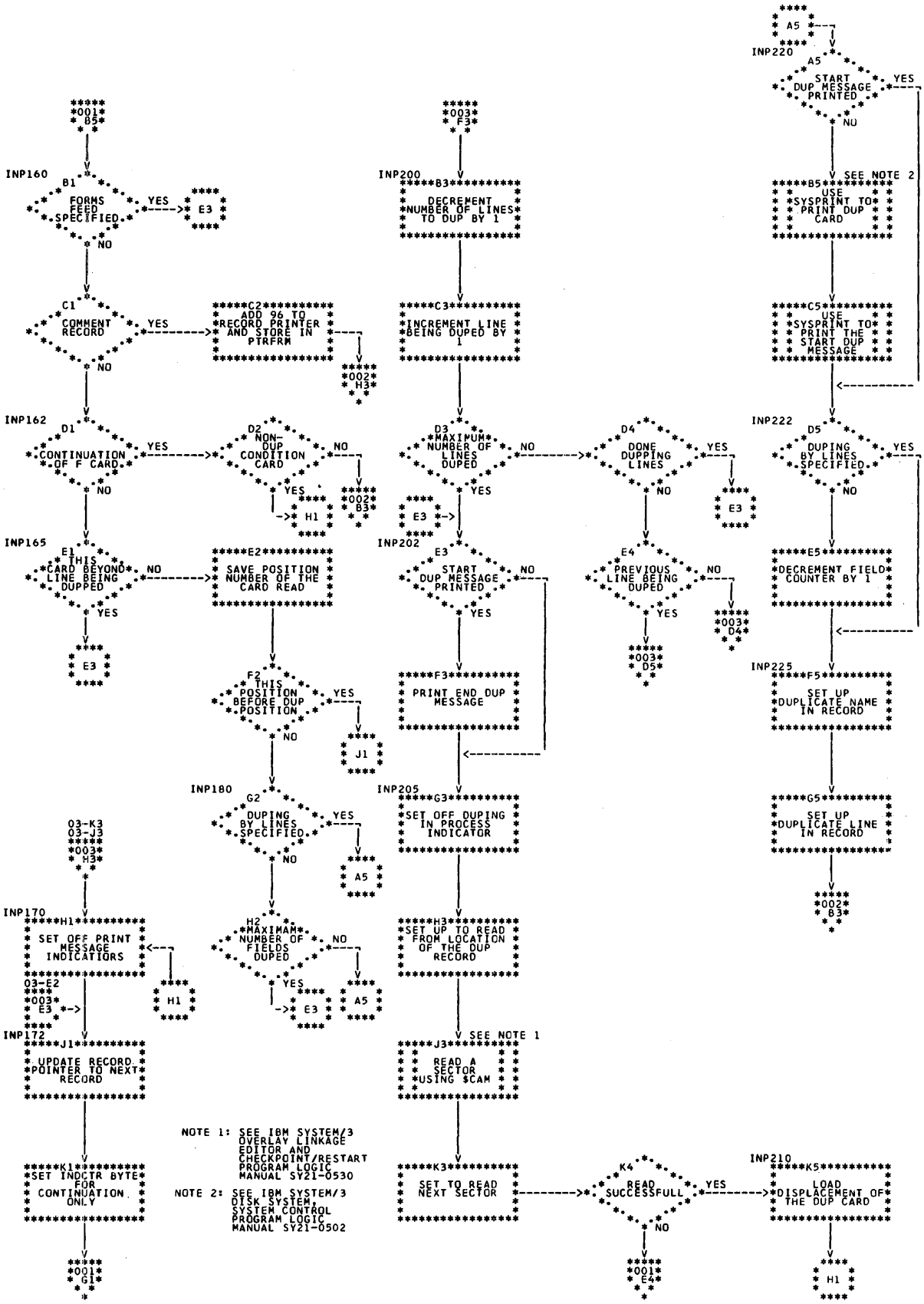


Chart M2 (Part 4 of 4). PFGR Read Multiple from Sysin or Source Library ($CC2CS)

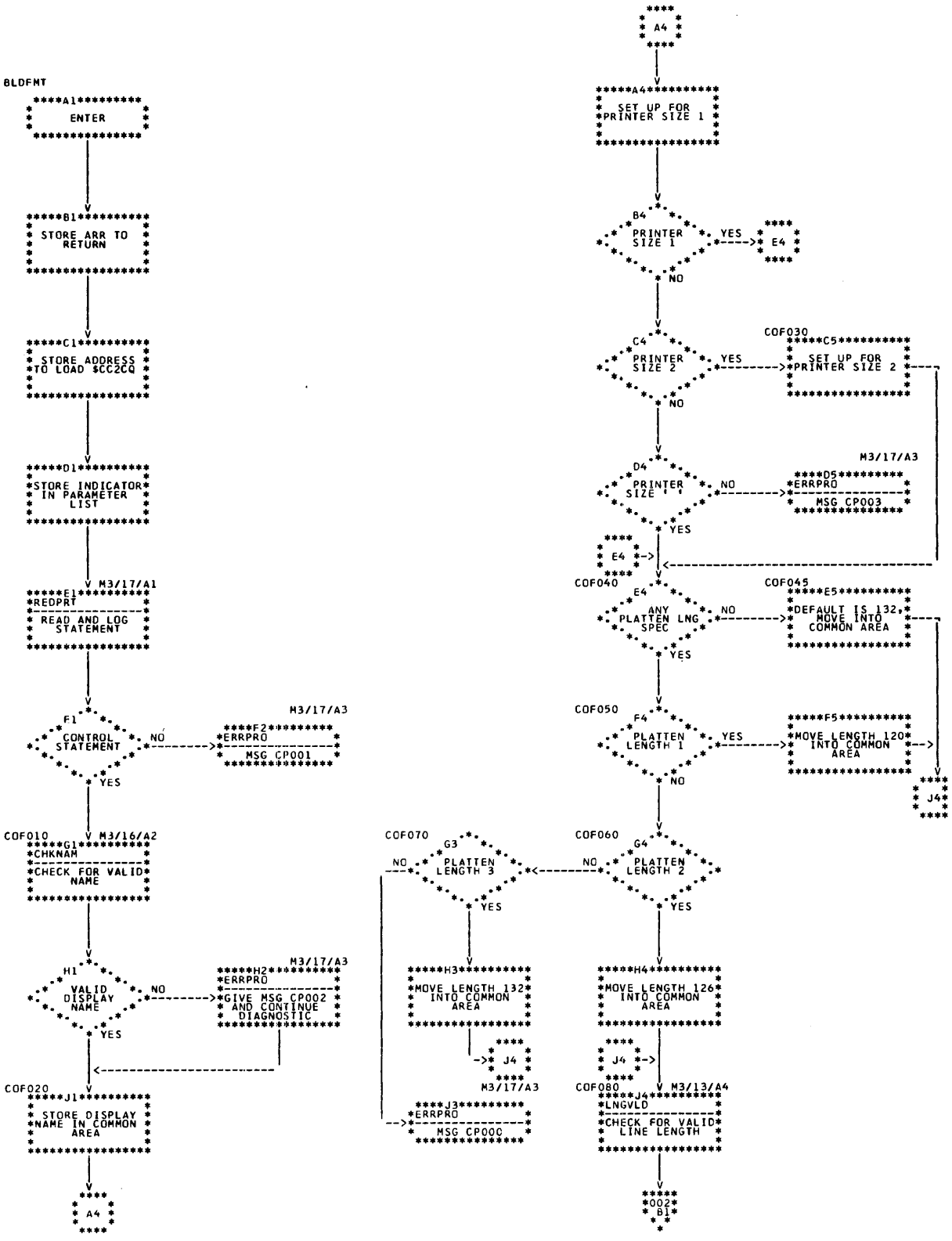


Chart M3 (Part 1 of 18). PFGR Print and Diagnose Display Specifications ($CC2CG)

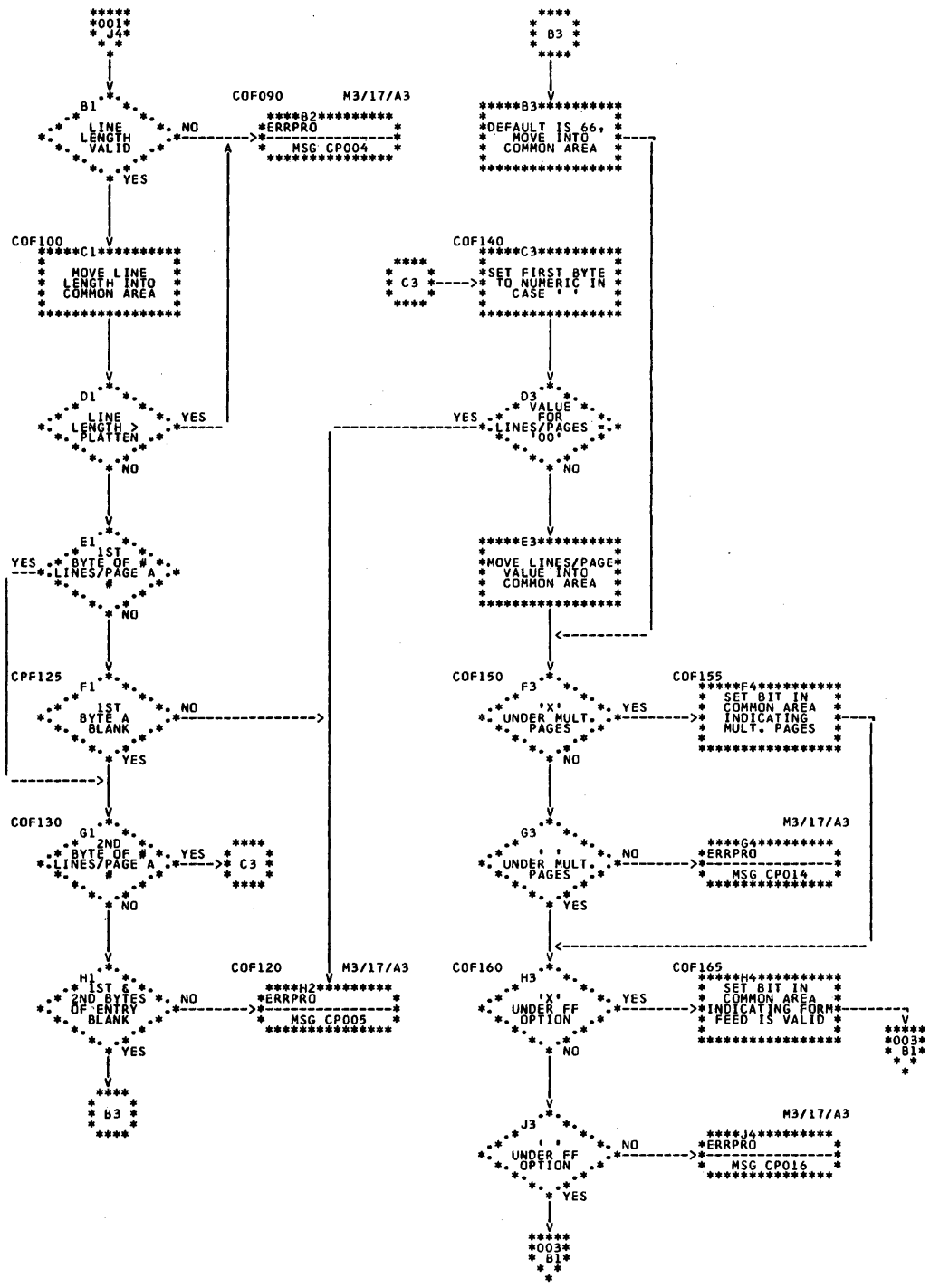


Chart M3 (Part 2 of 18). PFGR Print and Diagnose Display Specifications ($CC2CG)

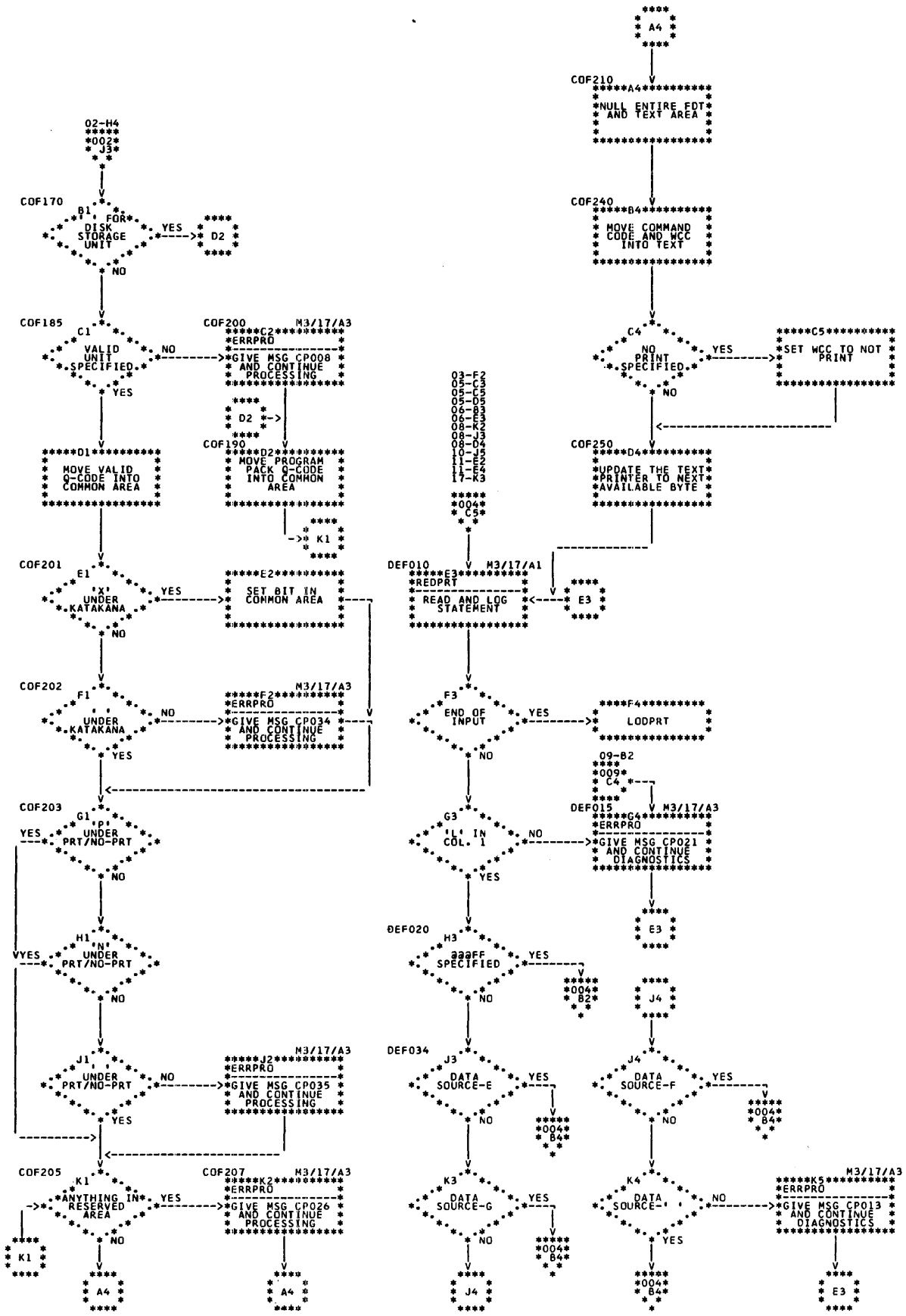


Chart M3 (Part 3 of 18). PFGR Print and Diagnose Display Specifications ($CC2CG)

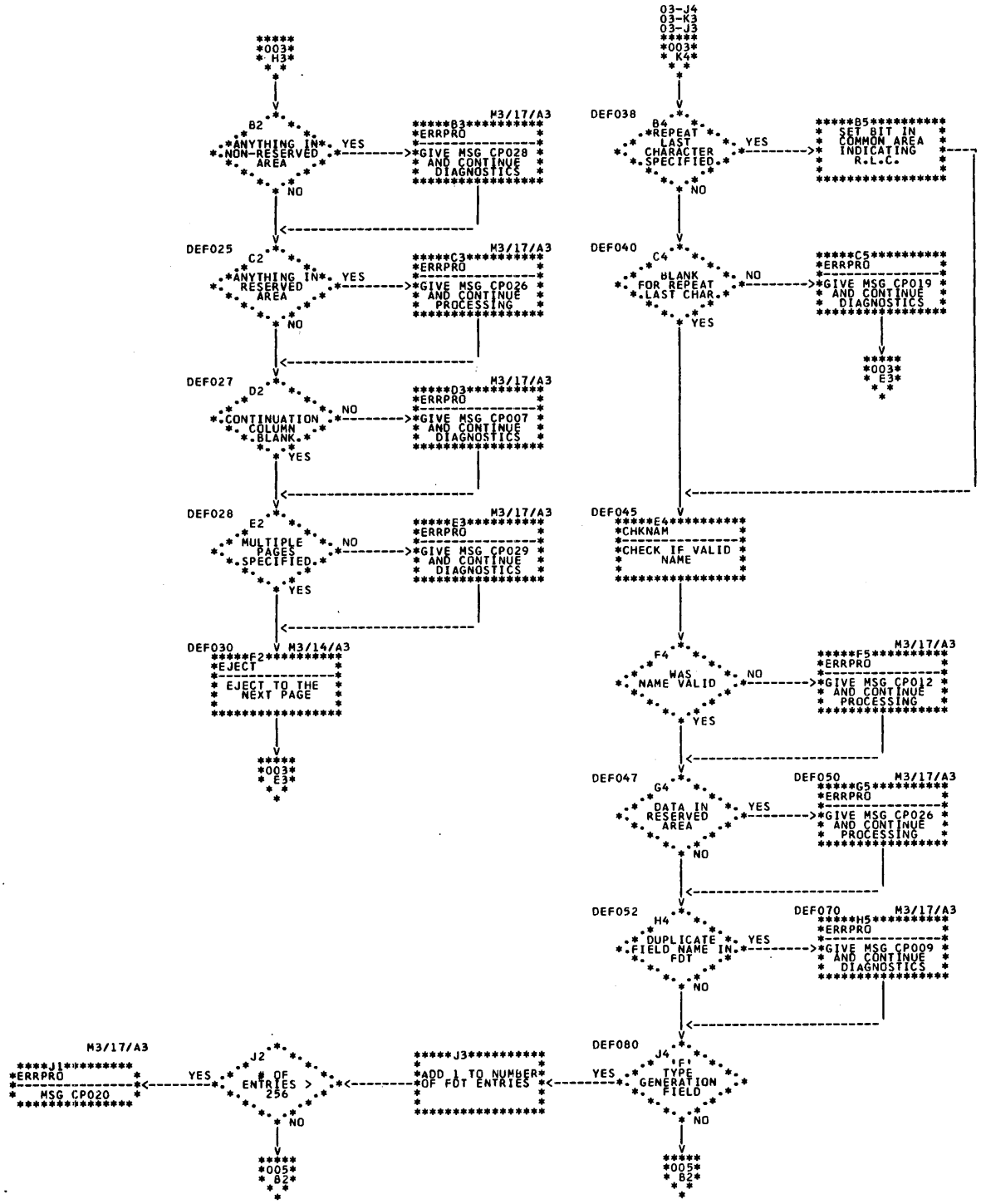


Chart M3 (Part 4 of 18). PFGR Print and Diagnose Display Specifications ($CC2CG)

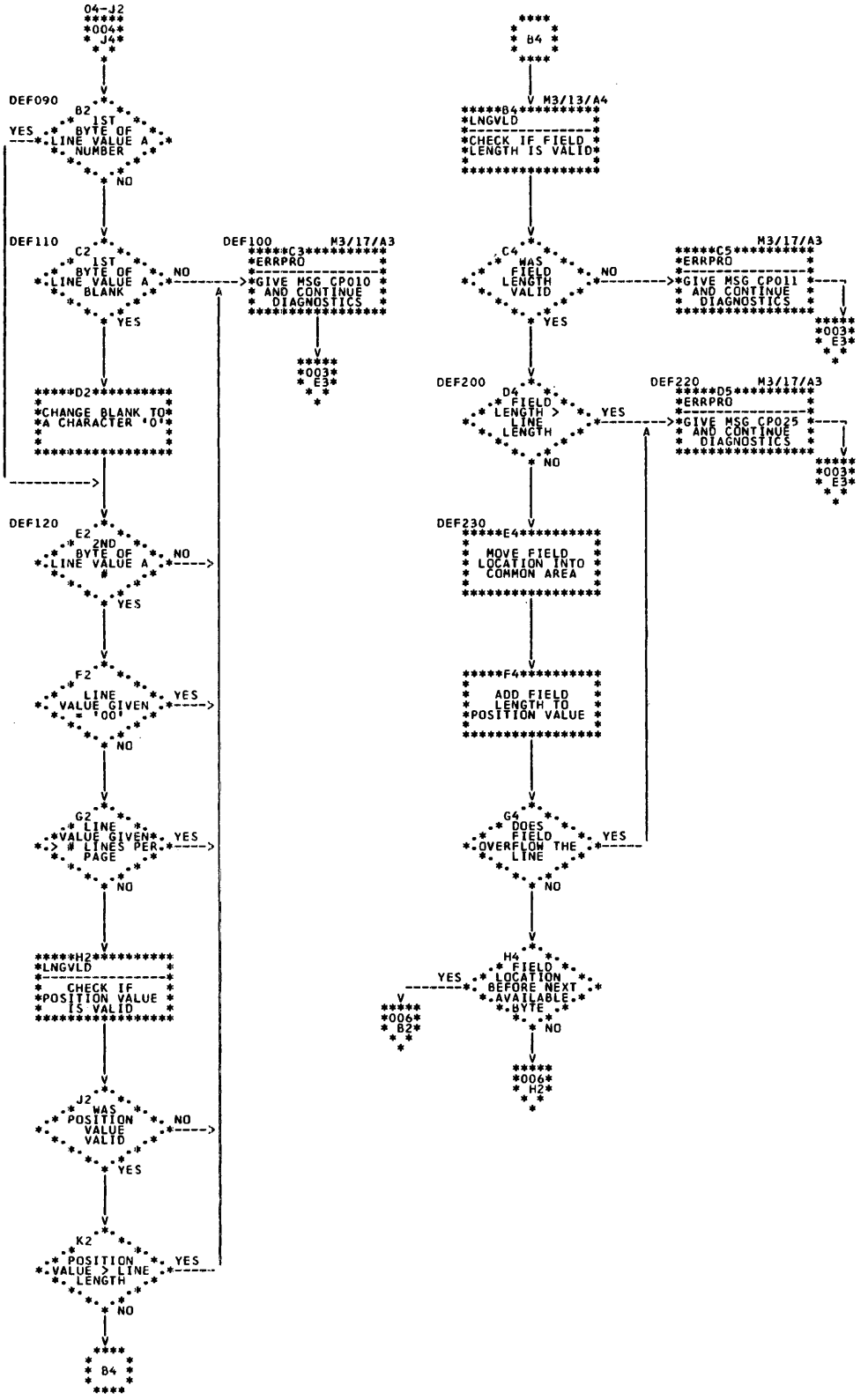


Chart M3 (Part 5 of 18). PFGR Print and Diagnose Display Specifications ($CC2CG)

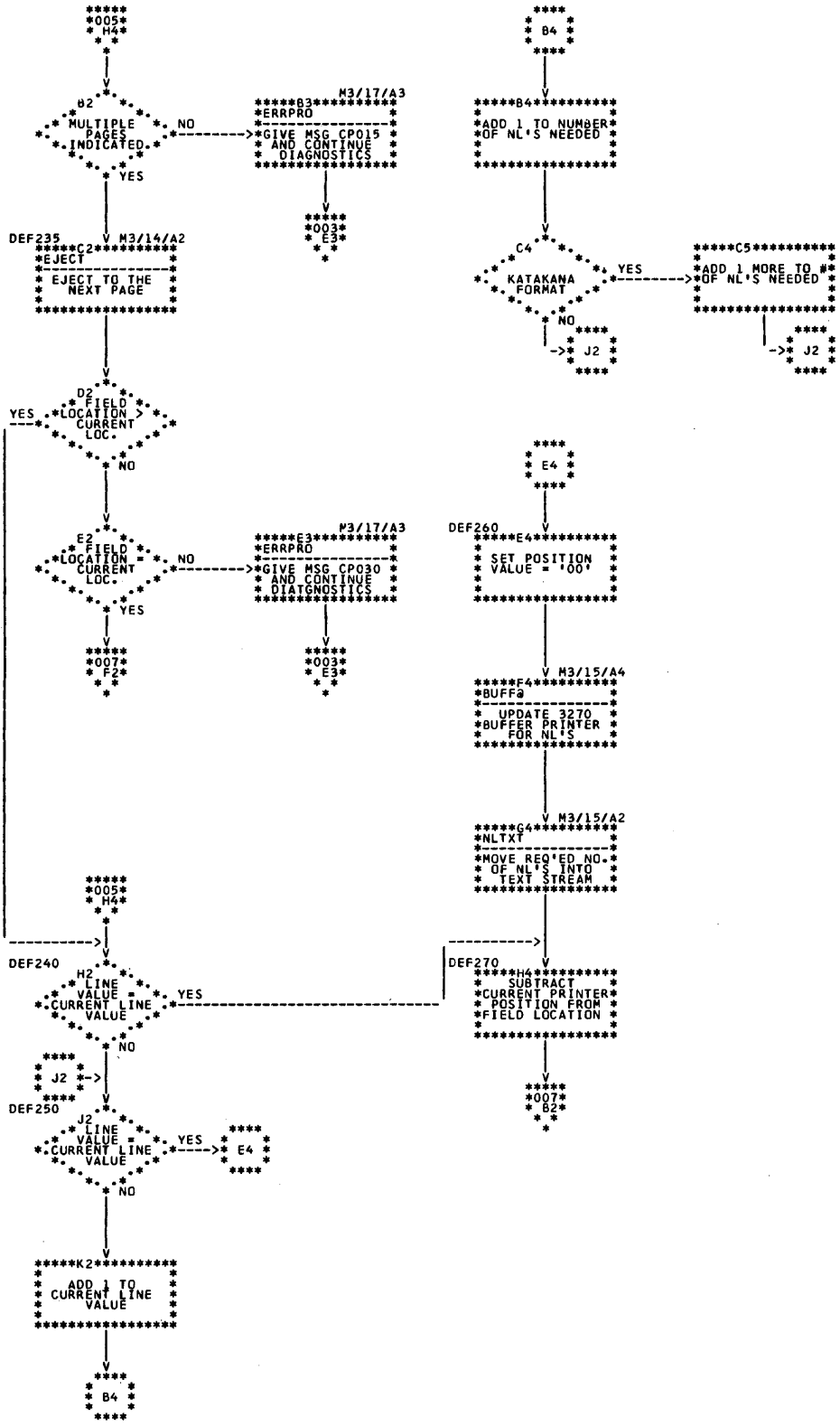


Chart M3 (Part 6 of 18). PFGR Print and Diagnose Display Specifications ($CC2CG)

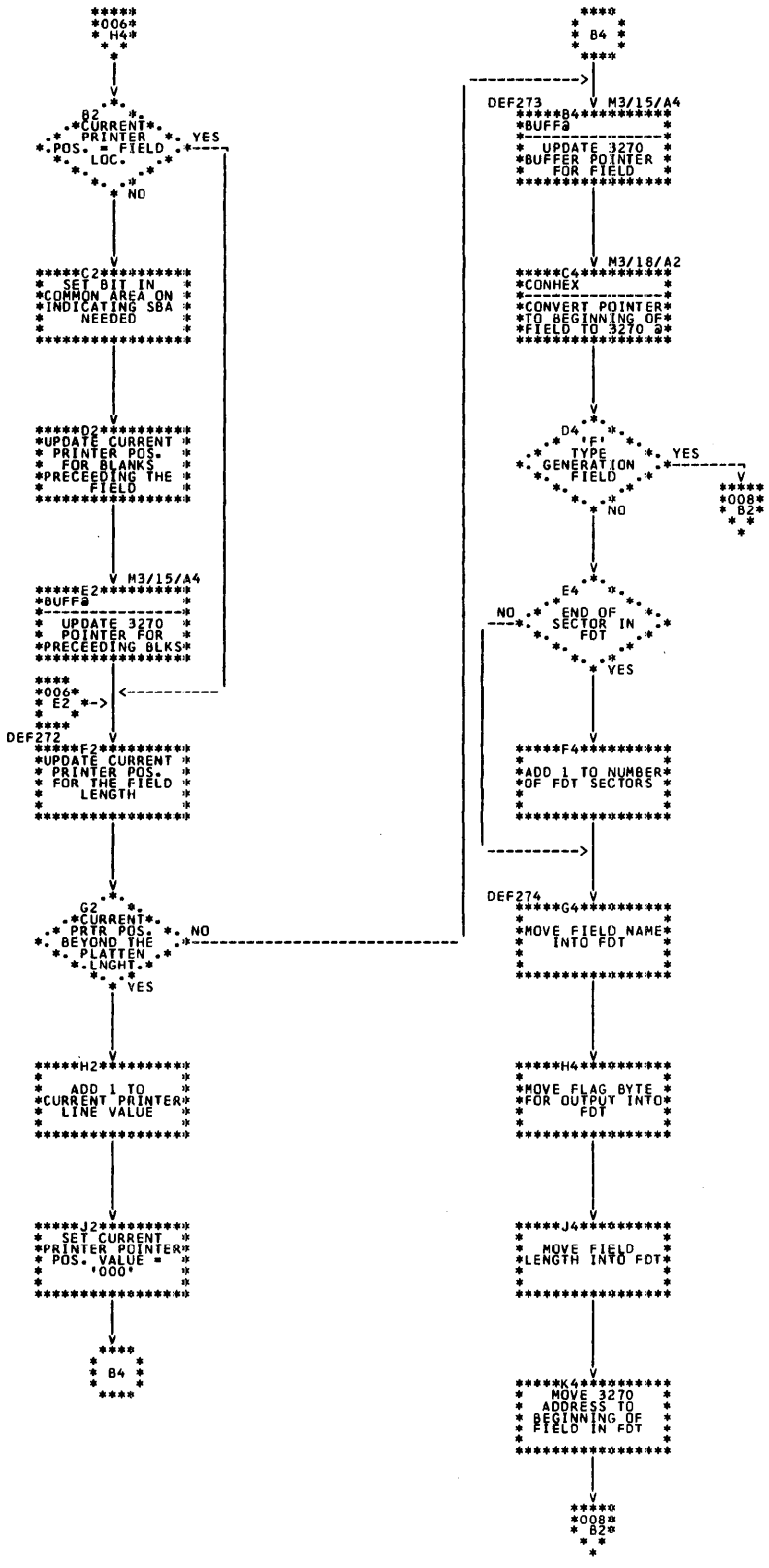


Chart M3 (Part 7 of 18). PFGR Print and Diagnose Display Specifications ($CC2CG)

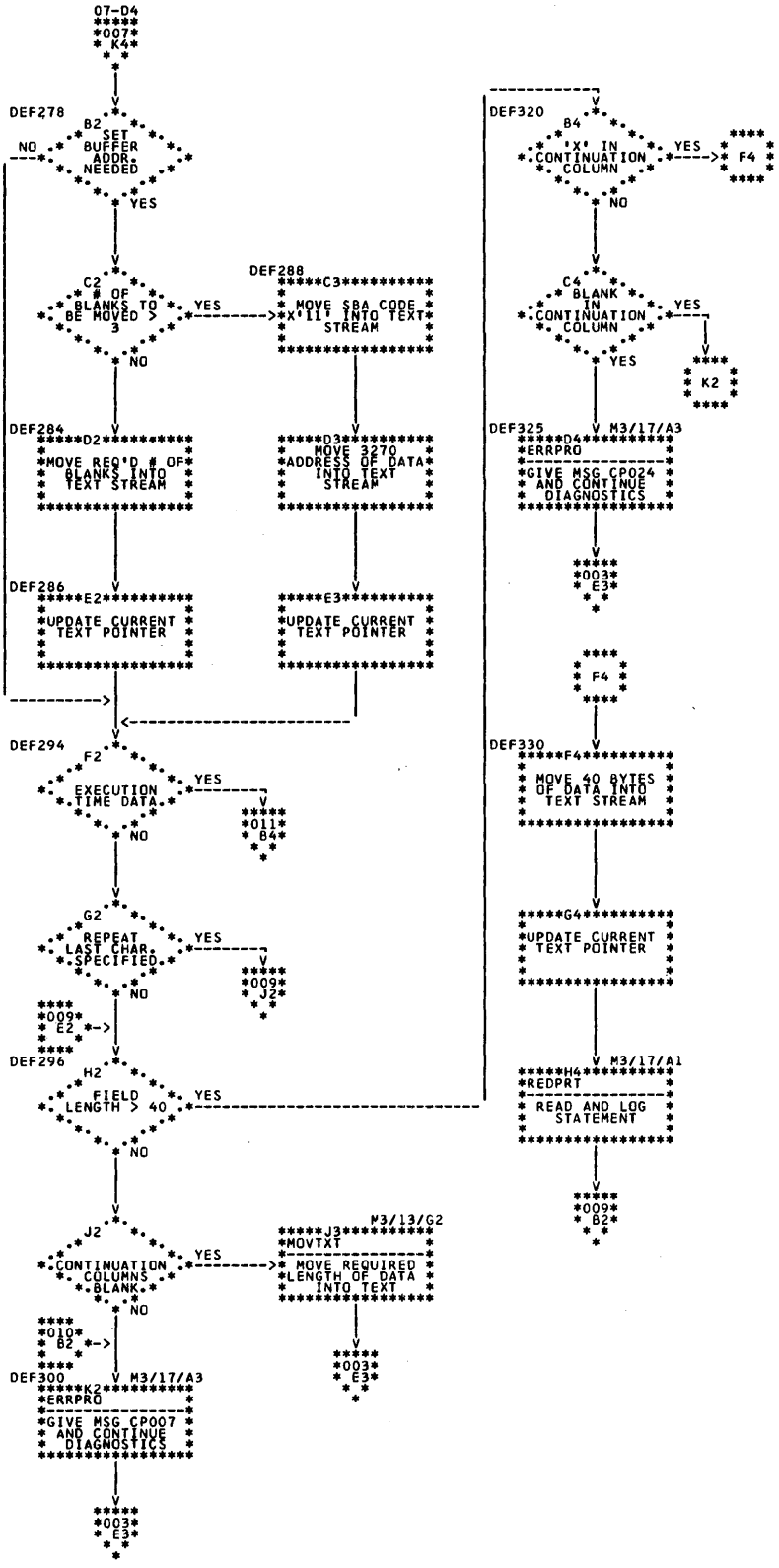


Chart M3 (Part 8 of 18). PFGR Print and Diagnose Display Specifications ($CC2CG)

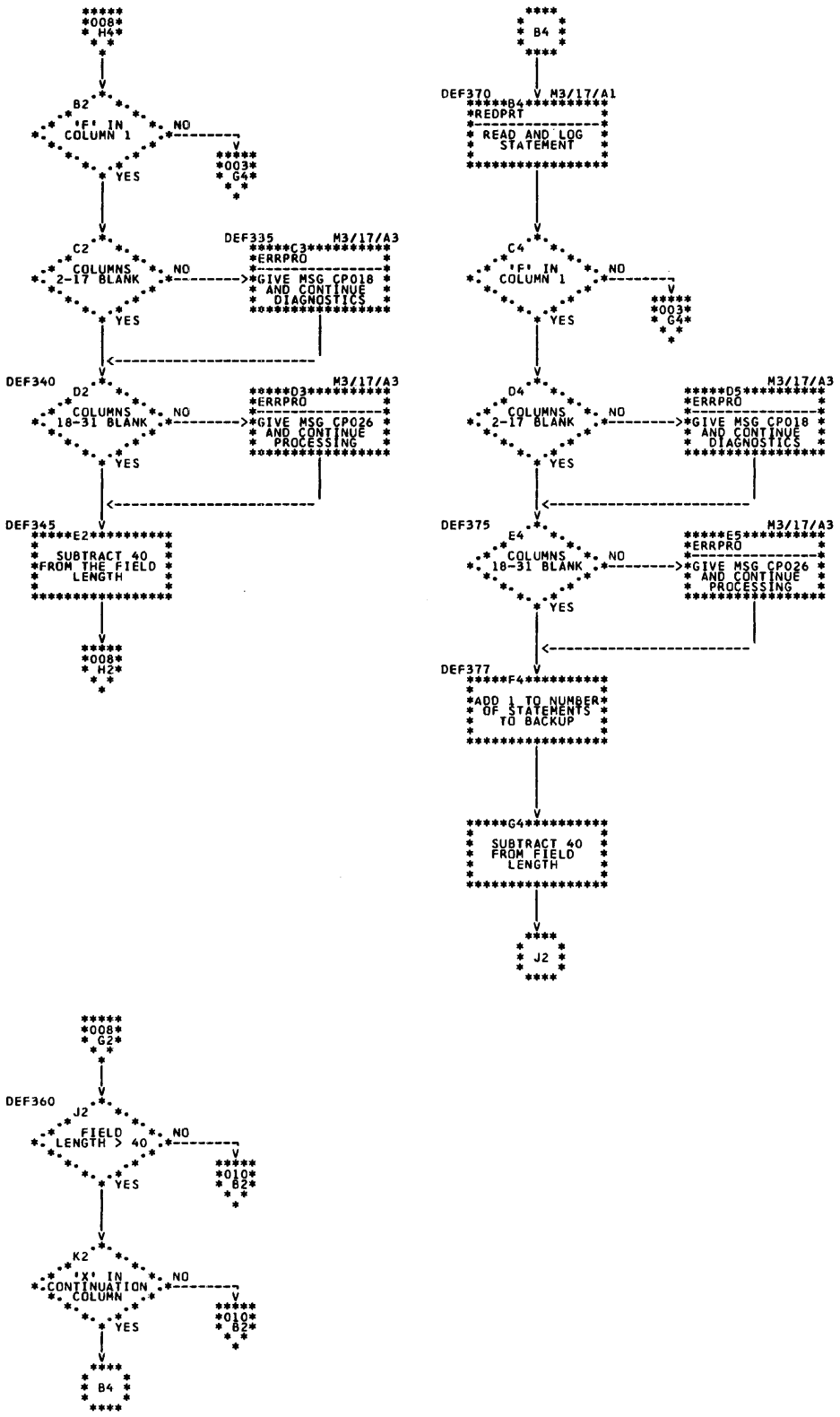


Chart M3 (Part 9 of 18). PFGR Print and Diagnose Display Specifications ($CC2CG)

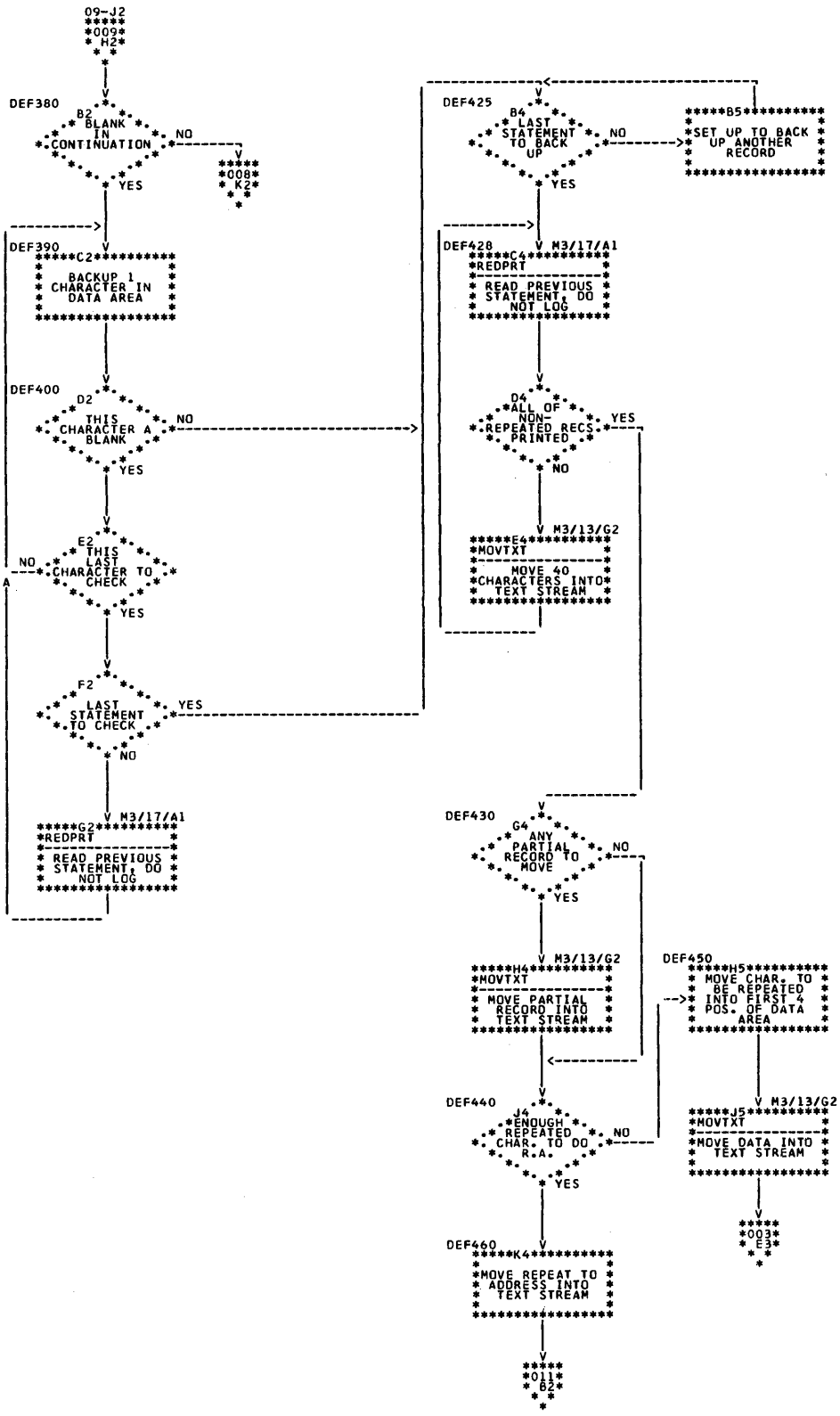


Chart M3 (Part 10 of 18). PFGR Print and Diagnose Display Specifications ($CC2CG)

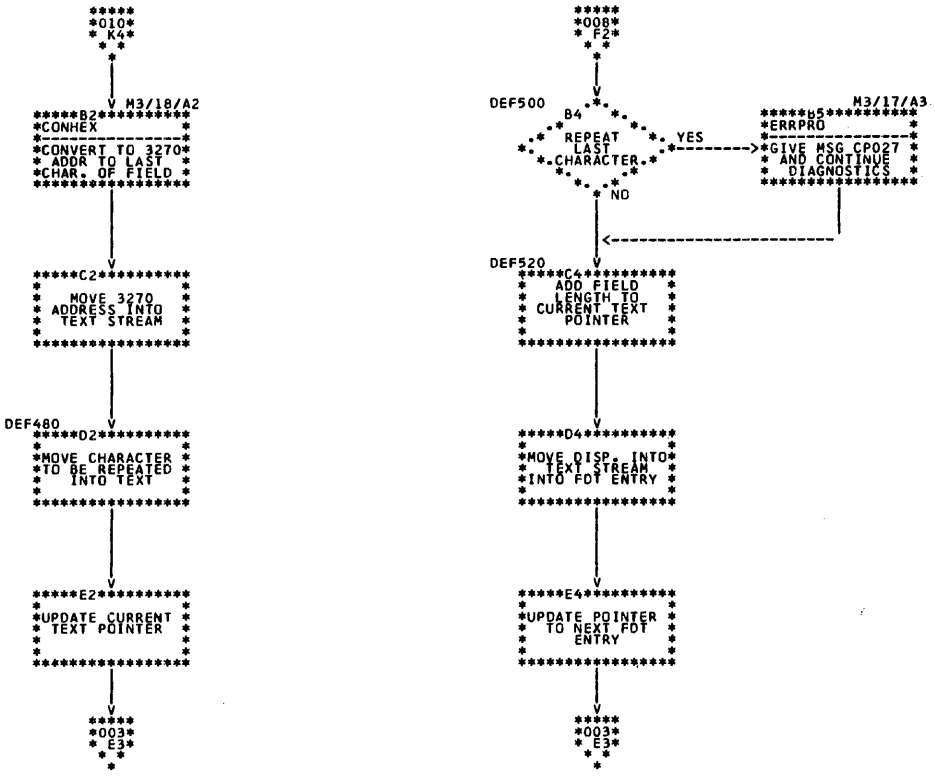


Chart M3 (Part 11 of 18). PFGR Print and Diagnose Display Specifications ($CC2CG)

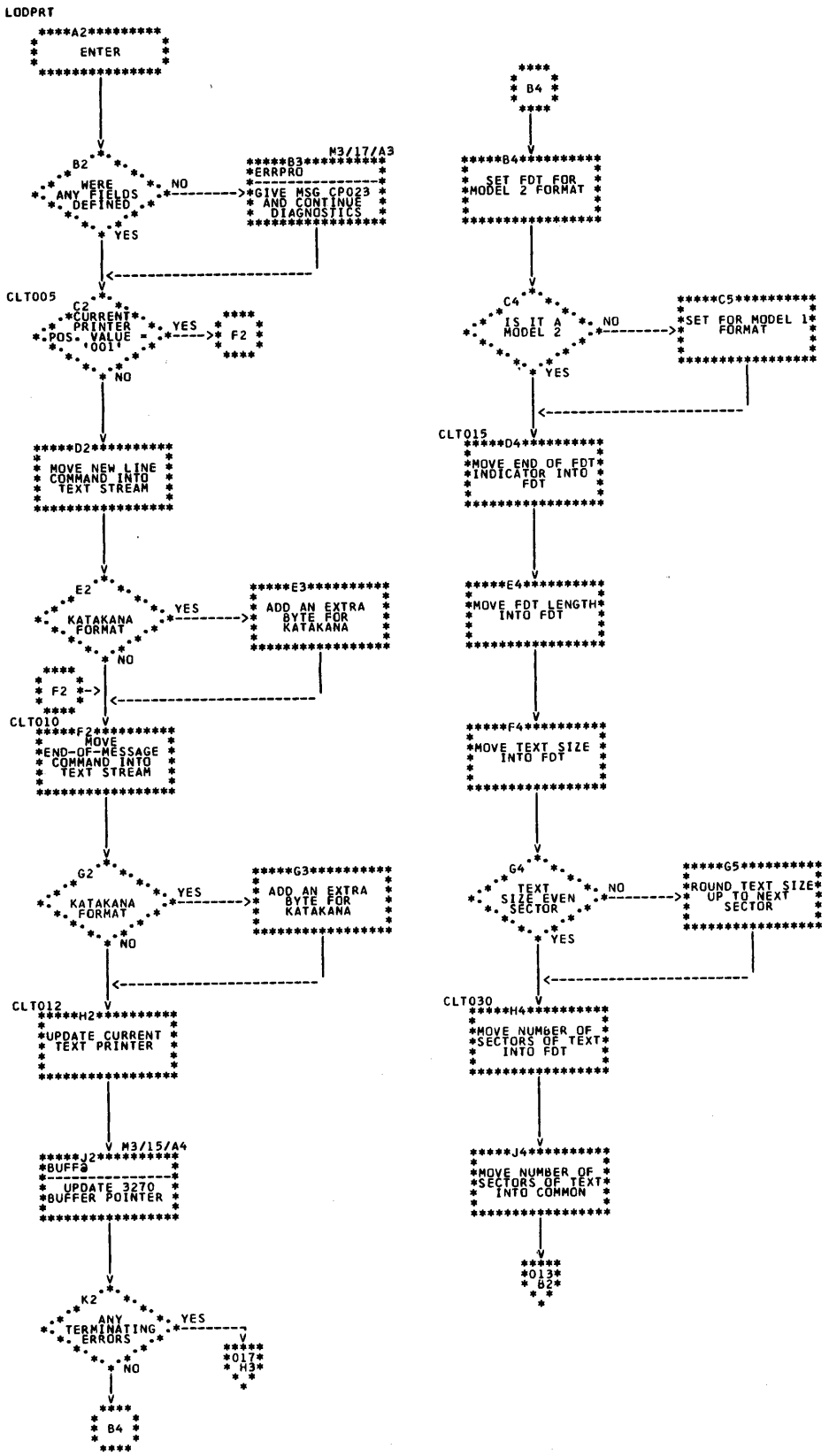


Chart M3 (Part 12 of 18). PFGR Print and Diagnose Display Specifications ($CC2CG)

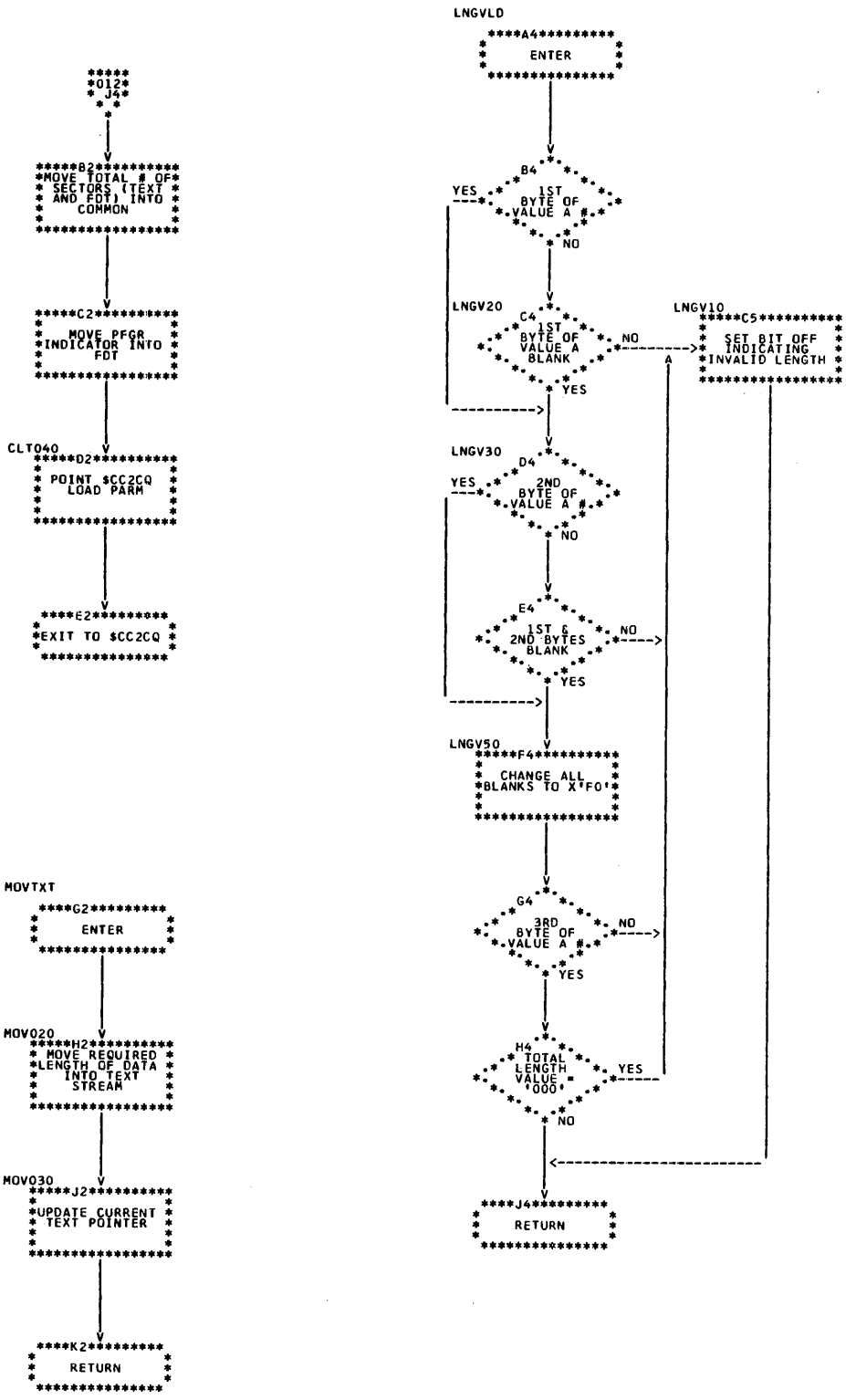


Chart M3 (Part 13 of 18). PFGR Print and Diagnose Display Specifications ($CC2CG)

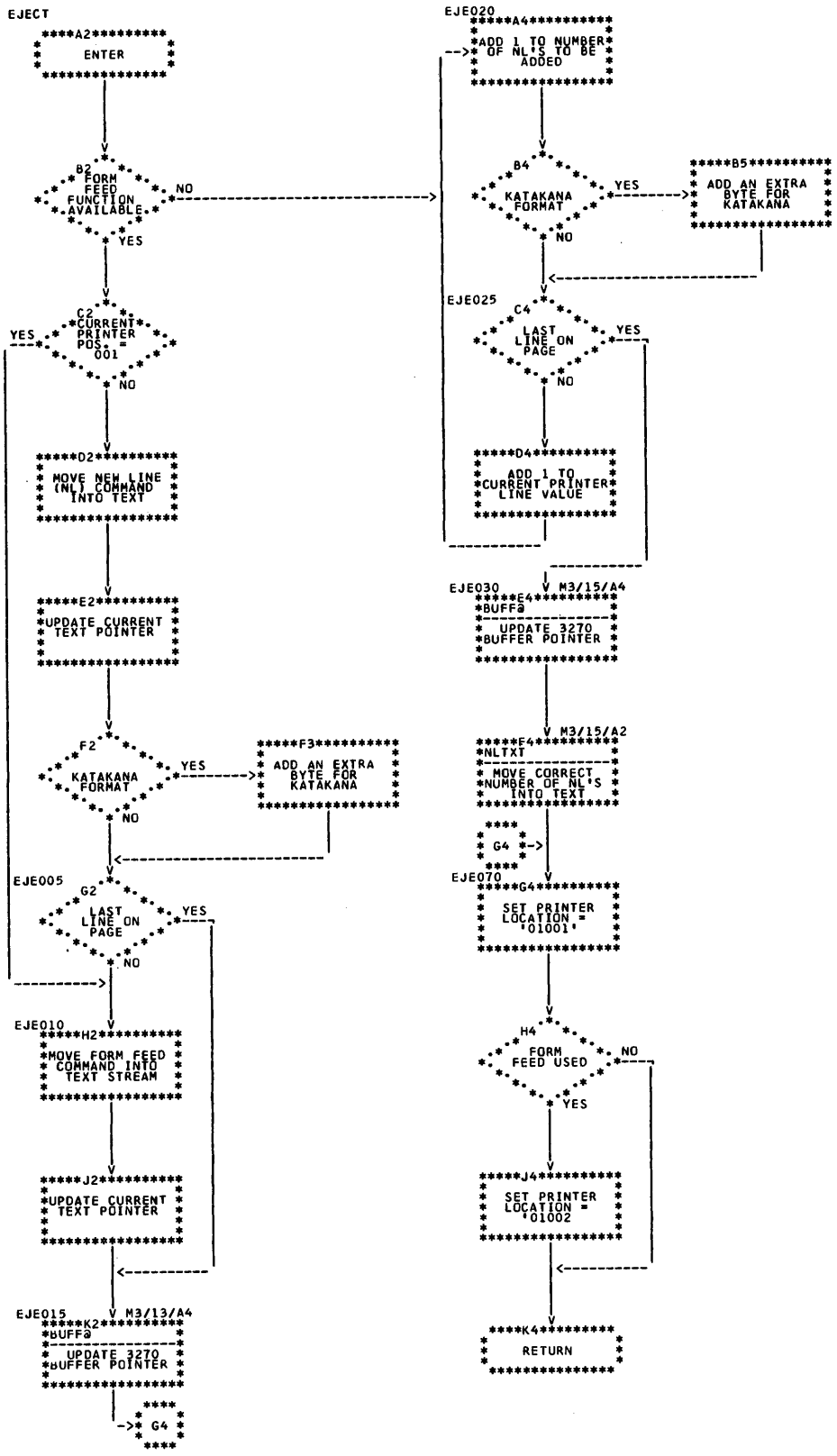


Chart M3 (Part 14 of 18). PFGR Print and Diagnose Display Specifications ($CC2CG)

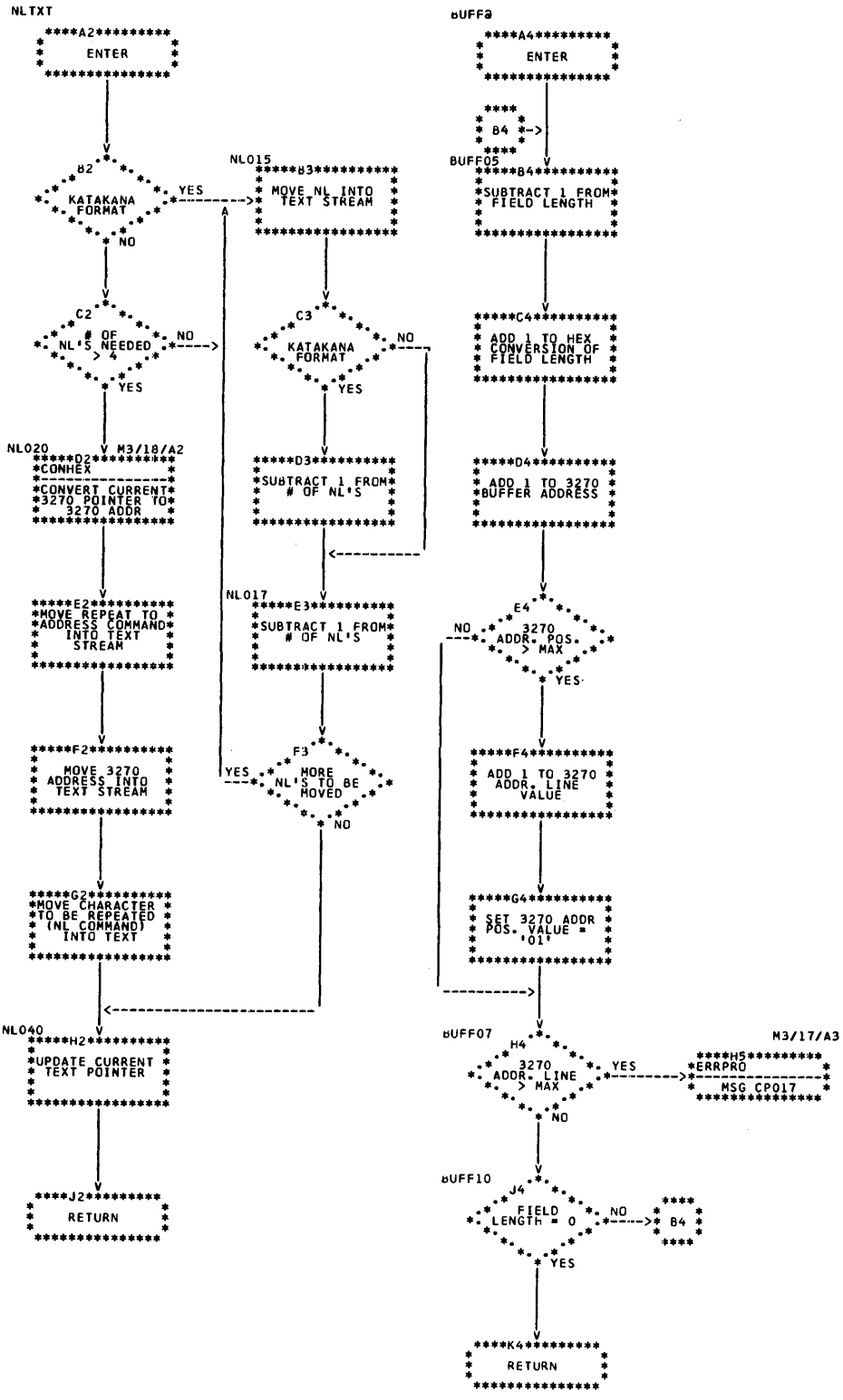


Chart M3 (Part 15 of 18). PFGR Print and Diagnose Display Specifications ($CC2CG)

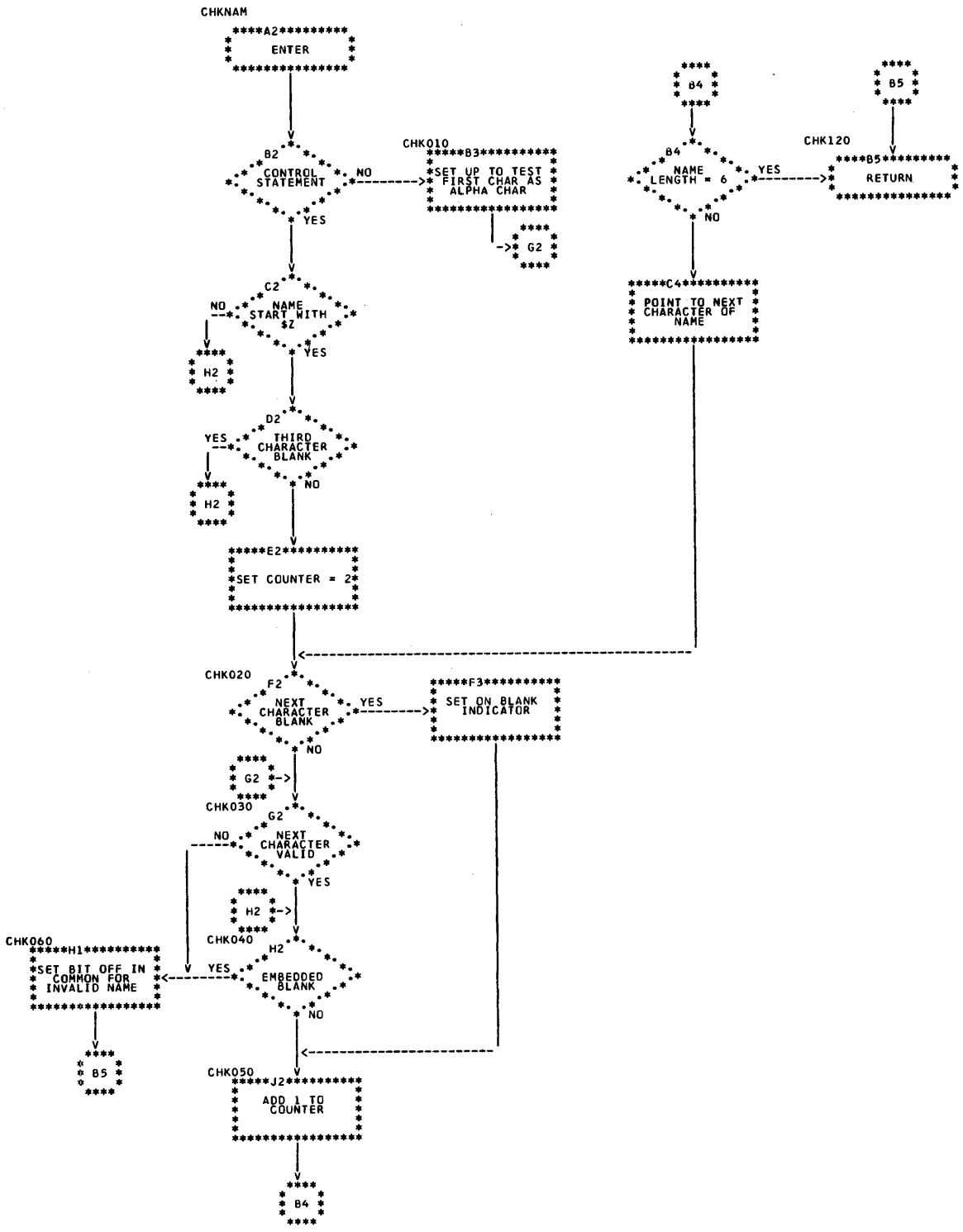


Chart M3 (Part 16 of 18). PFGR Print and Diagnose Display Specifications ($CC2CG)

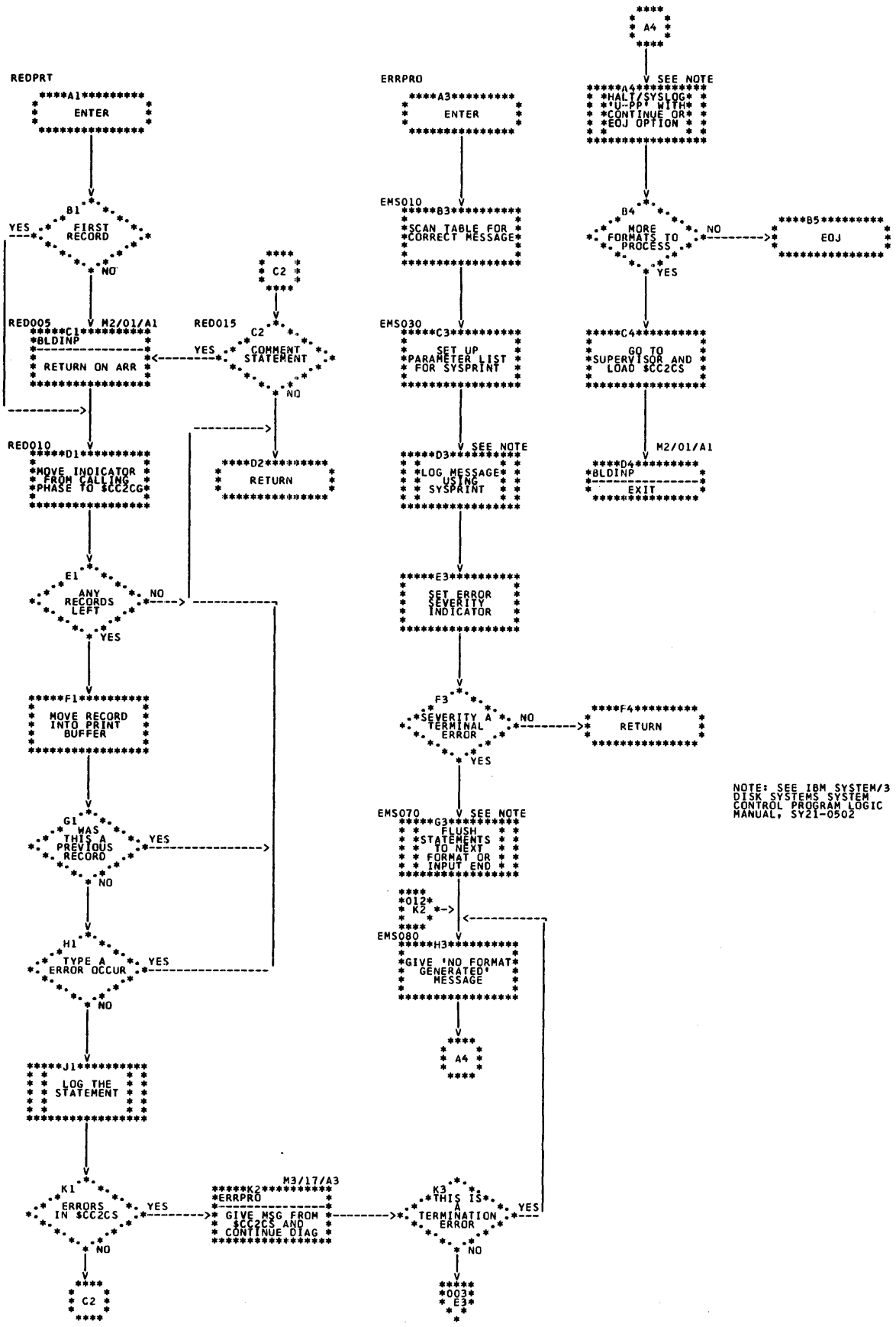


Chart M3 (Part 17 of 18). PFGR Print and Diagnose Display Specifications ($CC2CG)

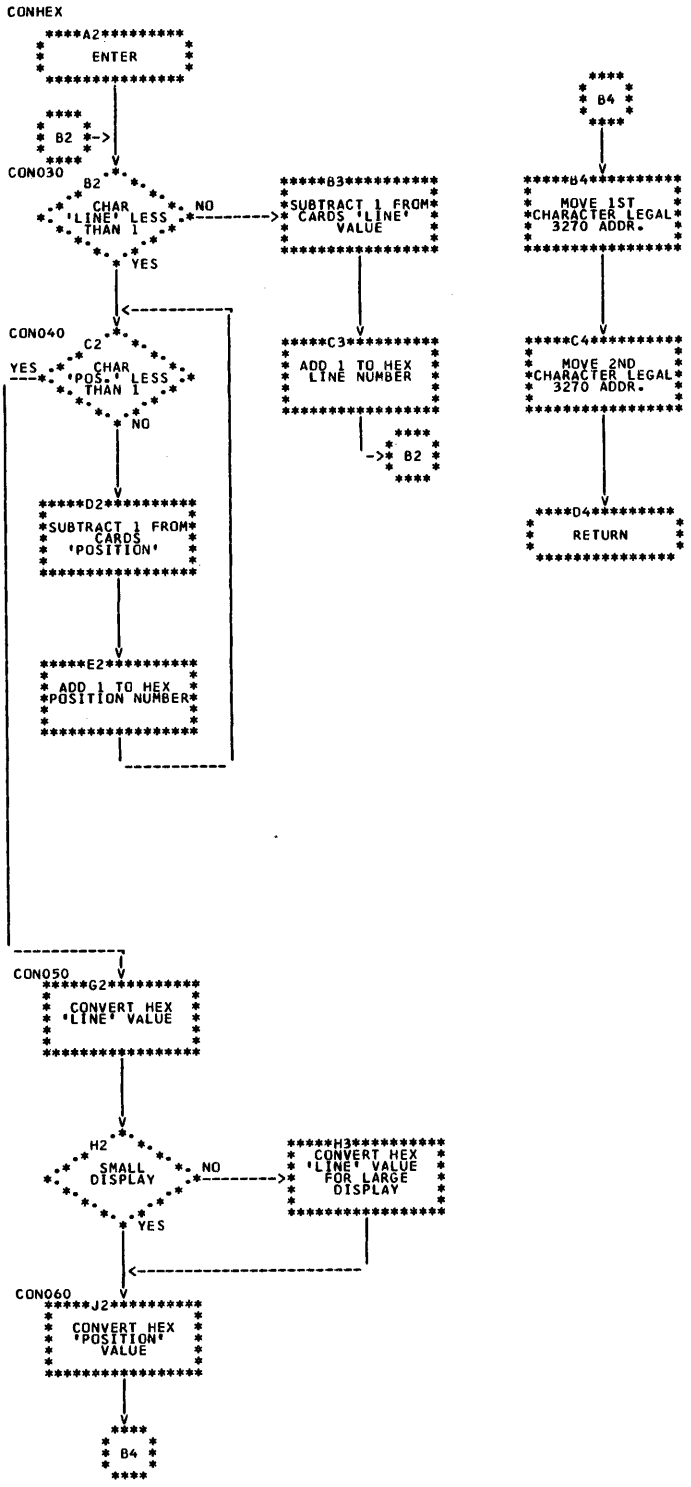


Chart M3 (Part 18 of 18). PFGR Print and Diagnose Display Specifications ($CC2CG)

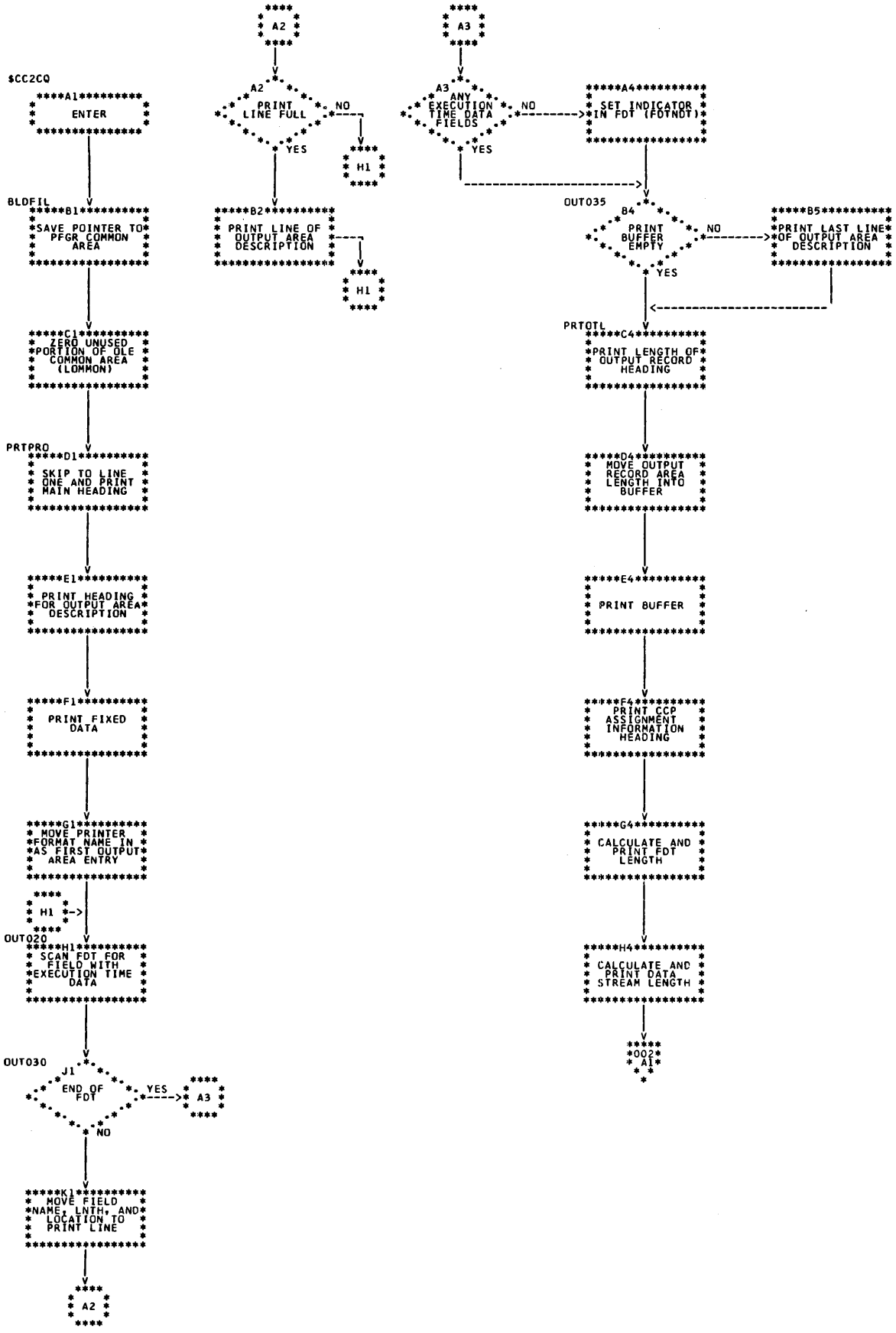


Chart M4 (Part 1 of 2). PFGR Print and Copy to Disk, $CC2CQ

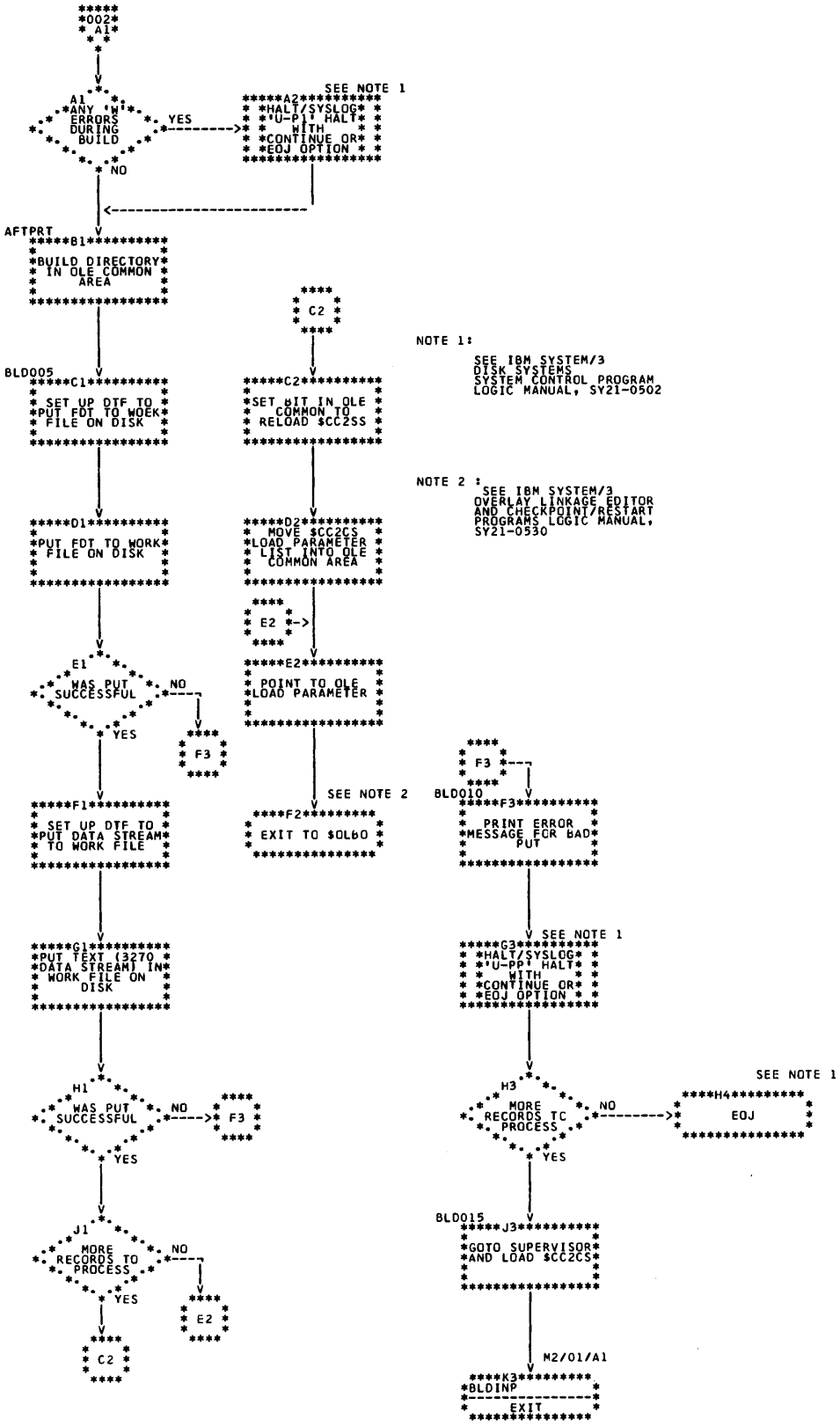


Chart M4 (Part 2 of 2). PFGR Print and Copy to Disk, $CC2CQ

Introduction

Startup, the third stage of CCP, combines various tables of required control information with executable code to form a usable CCP system. Startup is the final stage of CCP preparatory work and the beginning of execution of the resident CCP control program. Once initiated, Startup initializes CCP and turns control over to stage 4 (the Execution Stage), which controls the teleprocessing network.

Startup draws together the results of Generation/Installation (stage 1) and Assignment Build (stage 2), information contained in the DSM library directories, and the system operator's input to form necessary control information on disk and in main storage (Figure 8-1). Some of this control information is required in a form processable by DSM; other segments are formatted for MLTA and/or BSCA IOCS. Still other information is designed exclusively for CCP control code.

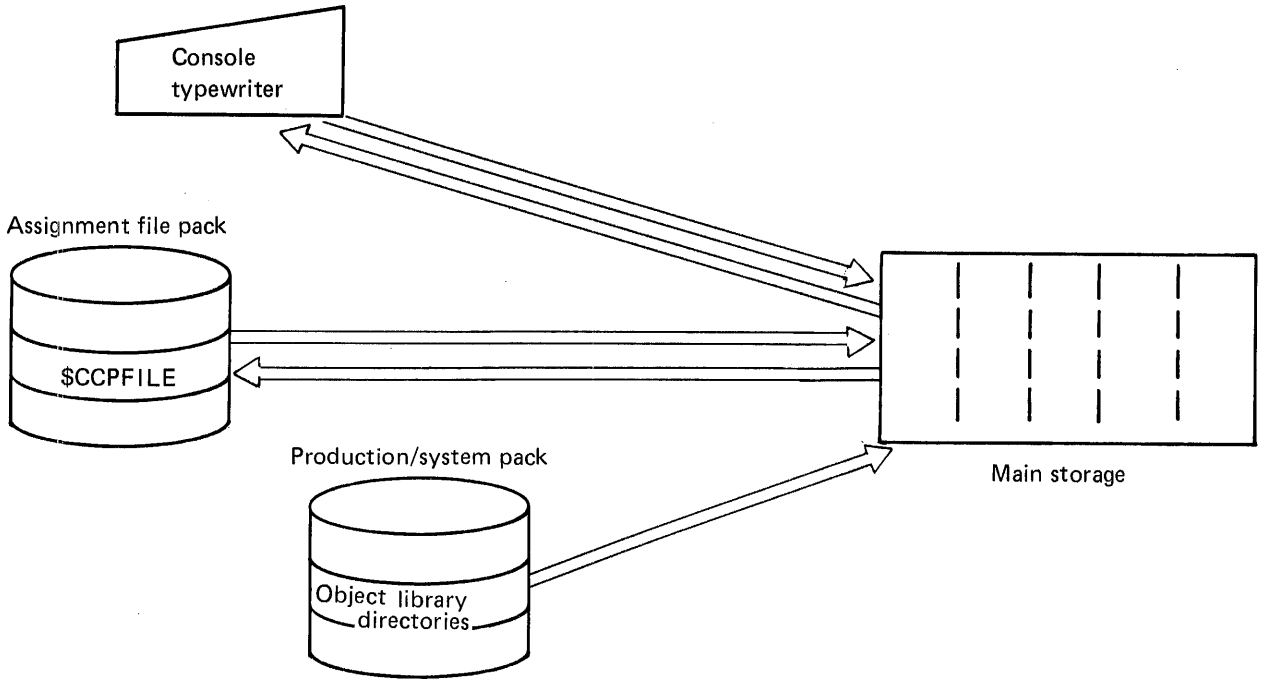


Figure 8-1. Basic Picture of Startup ($CCPFILE, Object Library, Storage, Console)

Following is a list of major functions of Startup in the order they are executed (not necessarily by phase):

- Receive specifications for this run from the system operator.
- Locate subsequent Startup phases.
- Build DFF format index.
- Cross reference CCP transients.
- Relocate CCP transients.
- Remove, for any facility, all indicators of suppression of that facility that may remain from a previous execution of CCP.
- Allow system operator to suppress facilities.
- Locate user programs and initialize Program Control Table.
- Load generated base module $CC4 and initialize it.
- Load Terminal Attributes Table.
- Build DTF/LCBs and related areas for TP control.
- Build Terminal Unit Blocks.
- Build Terminal Name Table.
- Build user Task Control Blocks.
- Build Contents Directory Entries.
- Build index to Program Control Table.
- Open disk files and compress DTFs.
- Build Sector Enqueue Blocks.
- Build File Specification Blocks.
- Execute F.E. REP card processor.
- Load DFF and $CC4#3 into PL1 if remap.
- Locate first level CCP transients.
- Allow terminals to be set offline.
- Allocate the remainder of main storage.
- Open MLTA lines (Models 8, 10, and 12 only)
- Open BSCA lines.
- Reset the DSM entry points for CCP use.

If any error situation is encountered during Startup which will prevent the normal completion of an initialized CCP system, an attempt is made (where practical) to continue for the purpose of further diagnostics. Ultimately, though, a termination message is issued to the system operator, and control is returned to DSM instead of being passed to the CCP resident control program.

Startup is interactive. The system console typewriter serves as the vehicle of communication between Startup and the system operator. It must be available to CCP when starting up. There are two modes by which the operator can specify information to Startup: keyword mode and prompt mode. Keywords may normally be entered when a question is asked (question mark at the end of the sentence). Answers to prompting statements are given directly as each is called for.

OPERATIONAL CONSIDERATIONS

OCL Restrictions

A maximum of 40 FILE statements for disk files can be entered. This is a DSM restriction. (The Assignment Build Program limits the combined sum of DISKFILE and SYMFILE statements to 100.) If more than 40 files have been defined to CCP Assignment Build, those lacking OCL at Startup must be suppressed during phase $CC3FS or a terminal Startup error is issued by phase $CC3DL. Extraneous OCL (those FILE statements for which there are not matching FCT entries) is ignored by $CC3DL when FILE statements are scanned.

On the Models 8, 10, and 12, the // BSCA OCL statement can also be entered but causes a terminal error to be issued by $CC3TB if two BSCA DTFs are built: this OCL causes all BSCA DTFs to refer to the same designated line after the DTFs are opened where CCP requires one DTF for each BSCA line opened. If only one (either line) BSCA DTF is built by $CC3TB, the BSCA OCL is accepted and the values #DF1LN and #DF2LN in $CCCOM are set accordingly. After the DTF is opened, it reflects the OCL-designated line.

User Security Information (Models 8, 10 and 12 Only)

Startup assumes that the user has run $CCPAU if user-written sign on security routines are used. No check is made on the data in module $CC4Z9 when $CC3LD moves it into the user security work area within $CC4.

SYSTEM REQUIREMENTS

Startup requires the following minimum system configuration:

- 5404 Processing Unit (64K of MOSFET memory)
  or
  5410 Model A15 Processing Unit (24K of main storage)—Model 10
  or
  5412 Model B17 Processing Unit (48K FET memory)—Model 12

- 5444 Model 2 Disk Storage Drive—Model 10
  or
  5447 Model 1 Disk Storage Drive—Model 4
  or
  3340 Model C2 Direct Access Storage Facility—Model 12

- 5471 Printer-Keyboard

- 5203 or 5213 or 1403 Printer

- Teleprocessing line (BSCA, MLTA, or local display adapter)

Note: Startup can be initiated in a program level of 18K bytes or larger. However, if at least 20K is not available to CCP, Startup terminates with an error message. The 20K limit does not guarantee a successful Startup with regard to main storage availability; the limit is based on the size of the smallest possible $CC4 module, assignment set, and user program area.

Method of Operation

OPERATIONAL OVERVIEW

Startup is initiated when the system operator enters OCL from the sysin device requesting the loading of $CCP. $CCP is the initial (or root) module leading to the execution stage of CCP. When $CCP is given control, its first function is to allocate the console. This serves two purposes: (1) it lets DSM check for console support, and (2) it prevents data management in the other level of a DPF system from using the console.

DPF System Determinations

Following the initial identification message, SU001, $CCP determines whether DSM is generated for DPF by testing bit 2 of byte NCSCH1 in the System Communication Area and sets a flag bit for DPF (not set for non-DPF) in the Startup Save Area (byte SSAMS1, bit SSADPF) to guide the error exit logic of any module encountering a permanent disk error.

If DSM supports DPF, the CCP program level is determined by storing the contents of program level 1 IAR and comparing it with an adcon of the address following the store instruction. Again a bit is set or not set in the Startup Save Area (byte SSAMS1, bit SSALVL) to guide the error exit logic if a disk error is encountered.

If the other level of a DPF system is active (bit 0 of byte NPSCH4 in the other Program Level Communication Area is tested), Startup terminates with an error message (SU010).

To prevent the other level from starting while Startup is active, a U– halt loop (halt, branch back to halt) is moved into the other level's transient work area, ending at NPHTRN. The address of the halt is set as the operand of the branch instruction and the address of the branch operand is set as the operand of a future load IAR instuction.

At this point the main storage bounds for CCP are determined. If CCP is in level 1, the ending address for level 1 is set to the beginning address of level 2. This level 2 address may or may not have been set by a PARTITION statement. If not, level 2 is rendered unusable until CCP has shut down. If CCP is in level 2, bit 5 of System Communication Area byte NCSCH is tested for the presence of a PARTITION statement. If one had been entered, the main storage bounds are not altered. If one has not been entered, the beginning address of level 2 is set to the ending address of level 1. This latter action inhibits level 1 until CCP is shut down.

The other level IAR is now loaded to set U–into the stick lights. EJ is returned into the lights just prior to the point in the temporary code in the $CC4 transient area (Chart OA) where control is turned over to the Command Processor. The address of the EJ halt is found by decrementing by 3 the address of the other level standby routine NPSTBY.

Initial Determinations for Any System

In preparation for verifying the presence, if necessary, of MLTERFIL on disk unit F1 and validating, if necessary, the keywords (PASSWORD, TRACEMLMP, and TRACEMLTA) prior to the specification of the unit for $CCPFILE, module $CC1BF is located by a Find operation. The first sector of $CC1BF, the CCP configuration record, is read into main storage and moved temporarily to the area of storage that will later be occupied by the CCP configuration record when it is read from the specified $CCPFILE. The user must not alter the contents of the unique $CC1BF belonging to a particular CCP generation/installation. It represents the generated level of support. Unpredictable results can occur if $CC1BF is altered.

If any MLTA support is generated in CCP, or if BSCA control station support is generated, disk unit F1 is searched by a VTOC read for MLTERFIL.

DSM and CCP are checked for both supporting DPF or both not supporting DPF (bit SSADPF in byte SPLVLK for DSM, bit CFGDPF in byte CFGFA1 for CCP). To support DPF, CCP generates code which manipulates the other level IAR. This code is not present if CCP was generated without DPF specified.

$CCPFILE Processing

The CCP program pack is searched for $CCPFILE using a VTOC read. If $CCPFILE is not found, Startup asks the system operator on the console typewriter for the $CCPFILE disk unit. A VTOC read is used to read the VTOC on the specified 5444 unit. (Console operations in Startup are performed using the Write-to-Operator Subroutine, $CCWTO.)

The basic check for integrity of the information in $CCPFILE is a test for X'EBD7' (X'FB04' on Model 4) in the first two bytes of the file. The test allows a check of whether the file is initialized or whether the first sector has been written over inadvertently.

Multiple $CCPFILEs may exist, but only one per pack can be addressed. The particular type of VTOC read used in $CCP considers the first $CCPFILE encountered to be the only $CCPFILE on that pack. The $CCPFILE designated for a particular run may reside on any 5444 unit, but it must have been initialized by the $CC1BF module on the program pack. If the specified unit is R2 or F2, the DSM system configuration record is checked (bits 6 and 7 of byte DISK) to be sure DSM supports that unit. (The DSM configuration record has previously been read by using a parameter list with a RIB of X'8A'.)

All disk reads, and writes ofinformation in $CCPFILE are done directly with an IOB and require the disk cylinder/sector address for each disk operation. Since the directory sector of $CCPFILE uses one byte per table to indicate the number of sectors that table contains, the disk address of each table must be calculated. This is done by stepping the address of the first $CCPFILE sector (from the VTOC read) to the address of the first System Information Table then looping through the directory entries, incrementing the disk address value to the next valid disk address for each sector in a table stepped over. After having calculated its set of table addresses, the operator-selected set ID is recognized and the directory of $CCPFILE is compressed to 17 bytes of prologue, the directory entry of the selected set, and the table of disk addresses calculated for that set. The result is the compressed directory used throughout Startup.

Keyword Mode Processing

When the first valid keyword (except LOG-unit) from the system operator is accepted in $CCP, bit SPKMOD of byte SPSW1 is set on to signal that the operator may not subsequently enter a YES/NO type reply. The same technique is used in $CC3FS, where an initial keyword replay causes bit FSKMOD of byte FSFLGD to be set on for the same purpose. The only case where the operator can escape keyword mode is by using CHANGE—YES or CHANGES—YES (Y may be substituted for YES) in a keyword specification after message SU011.

Pressing the END key with no input data selects a value by default for most items. However, in some cases where diagnostics are issued, a new value is imperative and the operator cannot get past that point until an explicit selection is made.

$CCP and $CC3FS contain the same keyword processor routine. This routine uses the scan subroutine $CCSCN with its Scan Control Block. The keyword table for this routine is (and must be) in alphameric order within keyword length.

A reasonableness check on the amount of main storage available to the dynamic TP buffer and user program area is made in $CCP by checking that neither (when specified) exceeds the size of the program level minus 10K. (The minimum setting of user program area (MINUPA) is 5K.) This guarantees space for Startup to execute in. The absolute minimum value for dynamic TP buffer area is the calculated value (by Assignment Build) of the maximum command length (field SITMCL in the System Information Table).

Disk I/O Errors

All phase 3 modules except $CC3CE, $CC3CX, $CC3RO, $CC3FB, $CC3QB, and $CC3UB perform disk I/O. A disk error in any Startup phase causes a U— halt (1234 halt on Model 4) in the CCP level. Pressing HALT RESET/START causes a UE subhalt (ABC 4 halt on Model 4). If a main storage dump is taken at this point, the value of the ARR, XR1, and XR2 will be found stored just beyond the halt-subhalt routine. In most phases ($CC3CR, $CC3DF, $CC3DL, $CC3FX, $CC3IP, $CC3LD, $CC3PX, $CC3RT, $CC3TA, $CC3TB, $CC3TC, $CC3US) these stored values follow a branch to location 4 with a RIB of X'84' (EOJ). In the other phases the location of these fields varies from phase to phase. The value of the stored ARR points just beyond the code invoking the disk operation which encountered the error. XR1 points to the IOB in use. If HALT RESET/START is pressed following the UE subhalt, the CCP changes to DSM, made in a DPF system (other level U— halt, alteration of branch to DSM halt check routine discussed later), are restored and control is returned to DSM via RIB of X'84' (EOJ).

Termination of Startup by the Operator

Startup may be cancelled by typing in CANCEL as a reply to messages SU011, SU025, SU045, SU105 if in prompt mode, SU300, SU375, SU918, and in one special case after message SU040. The last case is possible only if the assignment set specified by keyword (prior to the selection of the unit for $CCPFILE) did not exist. When instructed to select a set from the list, CCP may be cancelled.

It should be noted that Startup is constructed such that if the PROGRAM LOAD button is pressed at any time during Startup (and prior to the first terminal communication), no damage is done and there is nothing from which the operator must take special action to recover. CCP may be re-initiated after stopping it at any point during Startup. Use of the DPF interrupt control is not supported by Startup. Use of this control during Startup may cause both program levels to become inoperative.

Prior to entering the Execution Stage (stage 4), Startup sets on two bits (bits 1 and 2 of byte NCCONF), in the system communication area. These bits indicate that CCP is executing and are used to prevent initiation of certain programs in the other level.

PHASE OVERVIEWS

Diagram 8M.0100 is an overview of the Startup phase-to-phase control flow.

Loader Routine for Startup Phases

$CC3RO is the routine that loads subsequent phases of Startup. $CC3RO is loaded into the last three main storage sectors of the program level. The first sector begins with a jump instruction followed by the two-byte ID of the phase to be loaded and the corresponding ten-byte parameter list. The last sector contains 20 ten-byte parameter lists which identify subsequent Startup phases. In addition, the last four-bytes contain the two-byte ID of the current phase and the address of the beginning of the startup save area.

Phase $CC3CR: Initialize Disk Addresses of Transients

$CC3CR locates all $CC4xx object modules on the program pack and builds a table of their cylinder/sector addresses. These modules are then read as data and searched for the transfer control (XCTL) function (one $CC4xx module calling another $CC4xx module). Where this function is encountered, the actual cylinder/sector address of the called module is plugged into the calling module's parameter list. The calling module is then written to disk. This operation results in speedier operation of CCP during execution (stage 4).

Phase $CC3RT: Relocate CCP Transients

$CC3RT relocates addresses in CCP transients such that each transient can execute wherever its transient area falls during this execution of CCP (Figure 8-2). $CC3RT verifies that no transient is too large for the transient area.

The relocation operation is performed as follows:

The volume label of the program pack is read to obtain the extents of the object library directory.

The entire directory is scanned for object modules whose names begin with $CC4. If none are found, an error message is issued on the console, an error switch is set on, and an exit is taken. When such a module name is found, the relocation eligibility list in module $CC3RI (link edited with $CC3RT) is searched for a match on ID (last two characters of module name). If a match is found, the ID, cylinder/sector address, number of text sectors, and the displacement of the RLD within the last text sector are placed in a table.

The table of transients to be relocated is sorted on cylinder/sector address to minimize disk arm motion in reading and rewriting transients.

The second sector of module $CC4 (first sector of CCP transient area) is read in and retained. At the beginning of that area are the transient area address offsets and the table of resident code addresses needed for transient relocation.

The table of transients to be relocated is stepped through and each transient is read. The transient area ID within the transient is used to calculate the beginning address at which this transient is to be loaded later by the CCP control program.

After all relocatable addresses have been initialized, the transient is written back to disk provided the number of RAT (Relocation Adcon Table) entries exactly matches the number of RLD entries. If they do not match, an error message is issued on the console explaining which set of entries had the greater number, an error switch is set on (byte SSAERR in startup save area is set to X'FF') to prevent Startup from successfully completing, and the module in error is not written back to disk. The next transient is then relocated.

There are three types of RAT entries. Each is assembled as an absolute value:

0xxx — Address of beginning of transient + 0xxx.

Cxxx — Address of $CCCOM + offset of xxx.

80XX — Actual value is found in $CC4 transient area at offset of xx.

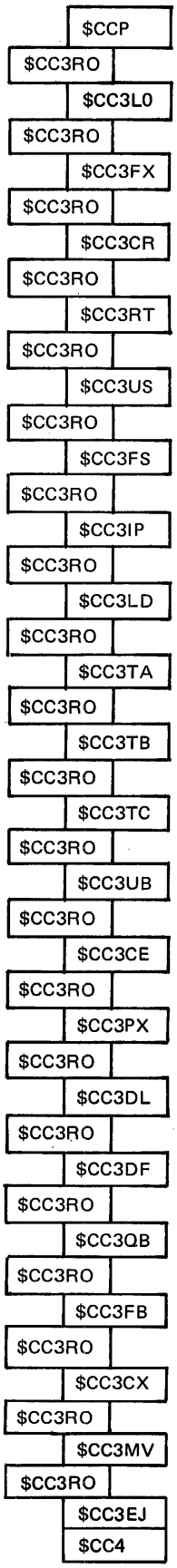


Diagram 8M.0100. Phase-to-Phase Control Flow

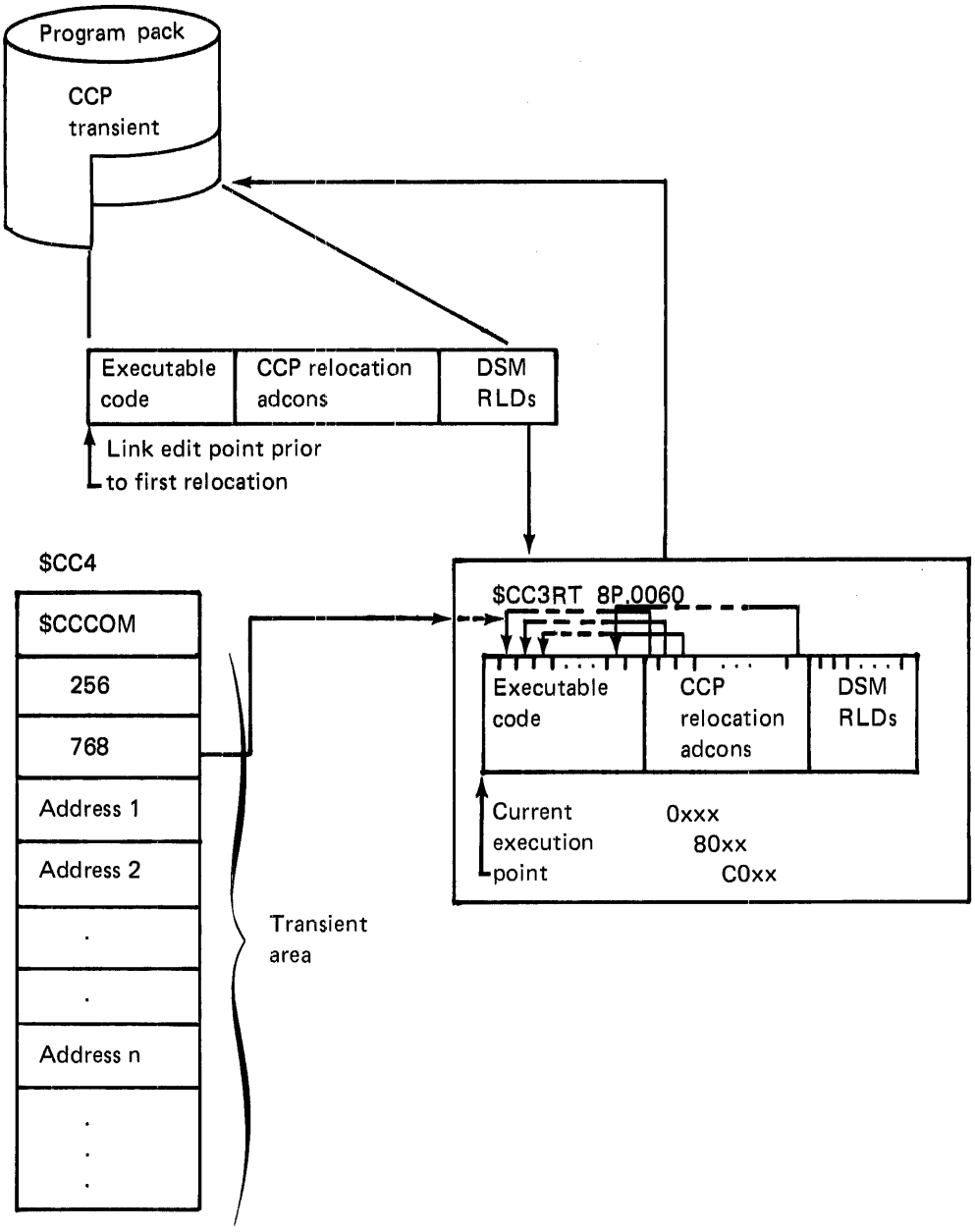


Figure 8-2 (Part 1 of 2). Overview of Transient Relocation

	Example of internal type relocation adcon		Example of $CCCOM type relocation adcon		Example of external type relocation adcon	
	– Original link-edit address was X'1400' – Program level begins at X'1400' – CCP transient area 1 begins at X'1500' – CCP transient area 2 begins at X'1700'		– Original link-edit address was X'1300' – Now program level begins at X'1400'		– Original link-edit address of transient was X'1300' – $CC4 link-edit address was X'1400' – Current program level begins at X'1500' – $CC4 transient area 1 contains |0100|0300|. . .|. . .|. . .|6A9C| 0 1 3 5 7 9 B	
	Instruction	*Relocation adcon*	*Instruction*	*Relocation adcon*	*Instruction*	*Relocation adcon*
Before relocation	|3502|1583|. . . OP Q Relocatable address Link-edit result	|0183|. . . 0xxx type relocation adcon	|3502|136F|. . . OP Q Relocatable address Link-edit result	|C06F|. . . C0xx type relocation adcon	|3502|130B|. . . OP Q Relocatable address Link-edit results	|800B|. . . 80xx type relocation adcon
After relocation for transient area 1	|3502|1683|. . .	|0183|. . .	Area 1 or 2 |3502|146F|. . .	|C06F|. . .	Area 1 or 2 |3502|7A9C|	|800B|. . .
After relocation for transient area 2	|3502|1883|. . .	|0183|. . .				

Figure 8-2 (Part 2 of 2). Overview of Transient Relocation

The internal type of RAT entry (0xxx) is handled by initializing the relocatable address field with the address of the transient area in which the transient is to execute, then adding the RAT entry. The result is an address within the transient itself.

The $CCCOM type of RAT entry (Cxxx) is handled by moving the beginning address of the program level ($CCCOM address) into the relocatable address field, then adding the RAT entry with the high order byte zone bits set off. The result is an address within $CCCOM.

The external type of RAT entry (always an odd number) is handled by first creating the address from which the actual value will come. This is accomplished by adding the RAT entry (with the high order byte zone bits set off) to the address of the I/O area within $CC3RT where the first sector of CCP transient area was read in. The value at the resulting address is moved into the relocatable address field. Then the link edit address of $CC4 is subtracted from the field and the address of the beginning of the program level is added to the field. The result is an address within $CC4, outside of $CCCOM, and adjusted to the correct address for the current CCP run.

Phase $CC3US: Clear All Suppression Indications

$CC3US successively reads in the File Control Table, the Program Control Table, the Terminal Unit Table, and the Line Control Table from $CCPFILE, sets off the suppress bit in each entry of each table, and rewrites the tables to disk.

Phase $CC3FS: Suppress Selected Facilities

Functions

Allows the operator to suppress any defined DISKFILE, SYMFILE, PROGRAM, BSCALINE, MLTALINE, TERMINAL, or the Program Request count.

Note: The distinction between suppression of a terminal and setting it offline is that suppression entirely deletes the Terminal Unit Block and marks the primary terminal name as unassigned, while setting a terminal offline simply sets off the TUBONL bit in TUB (TNTTUB is zero), byte TUBAT1. An offline terminal can be varied online by the system operator so the terminal can be used. A suppressed terminal is (to the Execution Stage) as if it never had been defined in Assignment Build. Therefore, it cannot be used.

Suppresses any facility requiring a facility which has been suppressed.

Cancels CCP if a startup cannot be completed due to suppressed facilities.

Allows CCP to be cancelled by the operator.

Resets all relative DISKFILE and SYMFILE pointers in each Program Control Table entry.

Operation

If the operator answered YES or Y to message SU045 or specified SUPPRESS by keyword, the opportunity will be given to suppress facilities. Otherwise this module goes directly to reset the PCT file pointers and exits.

If the facilities may be suppressed, a check is made (usually by searching for a dummy facility) for presence in the assignment set of SYMFILEs, BSCALINEs, and MLTALINEs, and the total number of TP lines is tallied. Flag bits are set so these facilities can be accepted or immediately rejected or bypassed later. In prompt mode, any facility not present will cause prompt to be bypassed.

Once the keyword or prompt mode is chosen, the alternate mode cannot be entered. The actual suppression is accomplished by reading the appropriate table from $CCPFILE, setting on the suppress bit in the designated entry, and rewriting the table to disk. In some cases, the same basic subroutines are used with different purposes and results by the use of flag bit settings.

In keyword mode, first the keyword is validated, then the keyword value normally takes the logic path that value would take as the reply to a prompt.

The suppression of a file, line, or terminal may cause a chain reaction of automatic suppression. Suppression of a DISKFILE will automatically suppress any program using that file and suppress any SYMFILE whose last (or only) unsuppressed reference is to the suppressed DISKFILE. The order of suppression is:

1. Suppress the designated DISKFILE.
2. Suppress any SYMFILE whose DISKFILEs are all suppressed.
3. Suppress any program using any suppressed DISKFILE or SYMFILE.
4. Check for all programs having been suppressed.
5. Check for all files having been suppressed.

If all files are now suppressed, a warning message is issued and in prompt mode will cause the next question to be on suppression of programs. SYMFILES are bypassed since they are all suppressed.

SYMFILEs are suppressed using almost the same logic as DISKFILE suppression. However, all files cannot be suppressed as a result of SYMFILE suppression.

Programs are straightforwardly suppressed with nothing else being affected by this action. CCP is terminated if all programs have been suppressed.

If BSCALINEs and/or MLTALINEs exist in the set, any of these may be suppressed. An error message is issued if the designated line number does not exist in the set. Following straightforward line suppression, all terminals on the line are also suppressed and the operator is informed by console message of each newly suppressed terminal. CCP is terminated if all TP lines have been suppressed.

Any terminal may be suppressed by specifying its two-character ID. If in prompt mode, CCP may be cancelled by typing in CANCEL when the opportunity to suppress terminals is given. A terminal is straightforwardly suppressed. If all terminals on the line have been suppressed, the line itself is suppressed and the operator is informed by console. A check is made for the suppression of all TP lines.

If the program request count facility was generated, it may be suppressed. Flag bit SSAPRQ of byte SSAMS1 is set off in the Startup Save Area so $CC3LD can bypass building the request count table. Bit #PUCNT of byte #FLGB is later set off in $CCCOM.

The end-of-phase routine in $CC3FS resets all relative DISKFILE and SYMFILE pointers in each PCT entry, then reduces any relative number to the correct value if a lesser numbered file of that type (DISKFILE or SYMFILE) has been suppressed.

Phase $CC3IP: Initialize PCT

$CC3IP searches the program pack object library directory (and possibly the system pack object library directory) for programs in the Program Control Table. The technique used is one of reading a number of sectors of PCT into main storage where the highest and lowest (alphameric) program names are captured and held as limits. The entire directory, 24 sectors at a time, is then passed by these limits and only those directory names falling between the limits (inclusively) are considered as candidates to cause a search of the PCT segment in main storage for a match. After the entire directory has been read, programs designated to be on that pack and whose directory entries were not found are suppressed. Another segment of the PCT is then read into main storage and the process is repeated with the same directory. When the PCT is completed, the first segment of the PCT is read again and the read/search procedure occurs with the system pack object library directory (unless the program pack is the system pack). A terminal error occurs if none of the programs in the PCT was found. This module also saves the length of the longest unsuppressed PCT entry for later setting into $CCCOM.

Phase $CC3LD: Initialize Resident Control Module

Functions

- Verifies that the combined size of the user program area and the dynamic TP buffer is reasonable.
- Initializes the moving pointer called the next build address and the pointer to the highest address (+1) which can be used for CCP control blocks.
- Verifies that sufficient main storage is available to load the generated resident CCP control module named $CC4.
- Loads $CC4 into main storage.
- Alters the halt check routine address (if the system supports DPF), pointing the halt branch to the CCP halt response intercept routine in the CCP transient area during the remainder of Startup.
- Saves the address into which the address of the first BSCA DTF will be placed.
- Validates the position and format of the initially NO OP'ed CCP branch instruction in the DSM console interrupt routine.
- Initializes many fields in $CCCOM.
- Locates the user security data module $CC4Z9 and moves the data from there into the resident user security work area allocated at CCP generation.
- Initializes the $CCPFILE directory sector.
- Initializes CCP trace and allocates main storage for the trace table.
- Initializes the service aid module CCSA.
- Initializes BSCA trace.
- Initializes MLTA trace.
- Allocates and zeros out the Program Request Count Table in main storage.
- Provides diagnostic messages on the console typewriter if necessary.

Operation

After some initial saving of addresses, the minimum user program area is added to the minimum dynamic TP buffer size and the result is checked for overflow (which would be an error designated as the exhaustion of main storage in attempting to load $CC4 – control would be returned to DSM). If the above result is not too large, it is checked against the size of the program level, with any excess buffer area yielding the same overflow error.

Having passed the above tests, the $CC4 load parameters are checked to see if loading $CC4 would overlay any of $CC3LD. If so, the overflow error is indicated.

The address of the byte just beyond the last byte of the generated module is calculated. This is where the control blocks will start. Then the addresses of the beginning of the minimum user program area and the minimum TP buffer area respectively are calculated, beginning at the top (high address) end of the program level. If the address of the beginning of the control blocks is lower than the address of the beginning of the dynamic TP buffer, $CC4 is loaded. Otherwise the overflow error is indicated as above.

$CC4 is entered at its entry point CPINIT (Chart OA), the generated code there being used as a subroutine to initialize some address for DPF and/or BSCA support. This is the routine which points the halt branch of a DPF system to a temporarily resident halt check intercept routine in case of a DSM halt during Startup (the user must choose option 2 or 3). If BSCA is supported, upon return to $CC3LD the address of the save area for the first BSCA DTF address is saved.
The specified location in the DSM console interrupt routine is checked for the CCP branch of the form X'C0800001', with any difference signalling an error. Transient $CC4SU will later plug this branch with a Q-byte of X'87' and an operand address of the CCP Console Interrupt Routine ($CC4IC).

If DFF is supported in this set, the disk device Q-byte (program pack or system pack) of DFF formats and the disk address of the format index built by $CC3FX are placed into $CCCOM.

Several miscellaneous items are initialized in $CCCOM, mainly values to be used or restored by Shutdown and the current values of TP buffer and user program area bounds.

If user security is used, the security data module $CC4Z9 is located, read a sector at a time, and moved in the largest possible blocks to the resident user security work area. If CCP password security is used, the password is right justified when moved into $CCCOM.

The directory sector of $CCPFILE is read in and initialized with the beginning and ending disk addresses for $CCPFILE, the beginning disk address for CCP disk-trace, the beginning disk address for the disk core dump area, the correct CPU size, and the maximum number of main storage dumps which may be written to disk during the current CCP run. If CCP trace is not used during a CCP run (TRACEBLK-0 is specified on the // SYSTEM statement), the beginning and ending disk addresses of the disk-trace area are equal. Any error messages pertaining to the lack of disk space for CCP trace reflect the actual net number of additional tracks needed.

If space is available, up to 9 main storage dumps may be placed into $CCPFILE. There must exist at least enough disk space for one complete dump from the machine currently running CCP. The error messages pertaining to lack of disk space for main storage dumps include an accrued number for any CCP trace requirements. The first sector of each potential dump area on disk is set to binary zeros on each startup.

If CCP trace is to be used, adequate main storage for loading is verified and allocated, and the trace module (CCTR) is loaded on a 16-byte boundary. The address in $CCCOM of the Pseudo Trace (@CTRAC) is modified to point to the trace table pointers. CCP trace-to-disk is automatically set on if a non-zero value was specified with the TRACEBLK keyword on the console keyboard when CCP was first initiated. Various pointers and values within the trace module are initialized. The disk area for disk trace is set to all X'FF'.

If CCP trace is used, an attempt is made to locate a service aid module (CCSA) on the program pack. If none is found, no further action is taken. If one is found, the service aid is loaded and the CCP trace branch to the service aid is initialized.

If the BSCA trace module (CCBS) is to be used, the module is loaded into main storage and the $CCCOM address (@BTRAC) is initialized.

If the MLTA trace module (CCML) is to be used, the module is loaded into main storage, the $CCCOM address (@MTRAC) is initialized, and the bit significant line trace mask for the lines to be traced is placed in the first byte of the trace module.

If program requests are to be counted, a space in main storage (2 bytes for each PCT entry) is zeroed and the table address is placed into $CCCOM @PUCNT. The Program Request Count Table is not reduced for suppressed programs.

The loading of $CC4 imposes several requirements upon Startup. More main storage is no longer available: the startup execution and work area is now limited to the size of the minimum user program area, which is 5K. With $CC3RO using 768 bytes approximately 4K is left for phase execution and work area. The control blocks must be built, which requires the keeping of a pointer to the next available address. This field is suffixed by NB@ in all phases using such a pointer. Whenever main storage must be allocated, a check must be made to assure main storage availability.

Following the loading of $CC4, $CC3LD branches to the $CC4 entry point CPINIT (Chart OA). Located there is a subroutine whose size and function depends on whether CCP supports DPF and/or BSCA. If neither is supported, an immediate return to $CC3LD (via saved ARR) is made. If DPF is supported, the EJ halt address of the other level is initialized, the halt branch operand at the end of the program Level Communications Area is altered to point to a halt response intercept routine generated in the CCP transient area, and the exit branch of this CCP routine is pointed to the resident DSM halt check routine. If a DSM halt should subsequently occur during Startup, the Startup routine senses the halt response for a 2 or 3. If either response was made, the halt branch is restored and the IAR of the other level is restored to the EJ halt. With any response, control is passed from the CCP routine to the halt check routine of DSM to perform normally. The DSM halt check routine address is restored along with the EJ halt in the other level at the end of the temporary code in the CCP transient area as Startup code is completed.

If BSCA is supported, CPINIT returns the address (in XR2) of where the address of the first BSCA DTF is to be placed. This field is checked later by the BSCA open routine in the CCP transient area to determine if any BSCA DTF was built by the current Startup.

Note: The remainder of Startup, up through $CC3CX, is concerned with building control blocks and data areas for CCP. However, there are two special codes to handle special formats of tables in $CCPFILE. The first is $CC3TA, which handles the Terminal Attribute Table. The table is read into main storage in its final form. The second, and much more complex, is $CC3US, $CC3FS, and $CC3TB, which handles possible sector spanning of the Line Control Table.

Note: The remainder of Startup, up through $CC3CX, is concerned with building control blocks and data areas for CCP. However, there are two special codes to handle special formats of tables in $CCPFILE. The first is $CC3TA, which handles the Terminal Attribute Table. The table is read into main storage in its final form. The second, and much more complex, is $CC3US, $CC3FS, and $CC3TB, which handles possible sector spanning of the Line Control Table.

Phase $CC3TB: Build Line Control Blocks

Functions

- Builds teleprocessing line control blocks and related fields such as logging area, polling list, line buffer, addressing list, exchange-ID list.
- Builds checklist for both MLTA and BSCA DTFs.

Operation

The LCT (Line Control Table) is read and the fixed position of each unsuppressed entry is saved because the TUT (Terminal Used Table) occupies the same input buffer at times. On Models 8, 10, and 12, a determination is made whether the LCT entry is for MLTA or BSCA and the appropriate routine is given control. Several pointers are initialized if the first DTF is being built. Line Control Blocks are built in the order the user has presented the BSCALINE and MLTALINE statements to Assignment Build.

When the BSCA LCB (Line Control Block) is built, the area is zeroed out and the items that can be immediately initialized are inserted. The ASCII translate buffer is allocated if appropriate and then the normal line buffer is allocated. The DFF output hold area size is initialized —it is either zero or the proper size. After the record separator byte is put into place, a point-to-point LCB is finished.

A switched line requires, in addition to the above: the LCB to be 4 bytes longer, the attribute byte set, any send-ID characters, and any receive-ID list. A dummy last receive-ID entry is made for usage by terminals with no exchange ID. If the LCT is entirely read, the phase ending routine gets control.

If a multipoint tributary LCB is being constructed, an indication of this is set into the attribute byte, the auto reponse routine address is initialized, and the TUT is read for the two poll characters. The LCB then is finished.

The major items of a control station line are constructed: addressing list, polling list, and terminal logging area. The addressing list is constructed first and there is one entry for each unsuppressed terminal. The addressing list is then scanned to determine the validity of a polling list entry. Any poll entry without a corresponding addressing list entry is bypassed. The terminal logging area size depends on the number of terminals. This accumulation of main storage requirements was done while building the addressing list. The fixed requirement is added, the address is inserted into the DTF, and the logging area itself is zeroed.

For MLTA on Models 8, 10, and 12 the one adapter DTF is constructed when the first line DTF needs to be built, then the line DTF/LCB. Line buffer length is calculated, the transfer vector address is determined, and the translate table transient ID numbers are calculated.

A point-to-point or switched MLTA line on Models 8, 10, and 12 can be completed by building one SDR entry and allocating the line buffer.

For a control station line, the polling list (relative terminal number) is saved, but in any case the TUT is read. The statistical data recording (SDR) area is built, one entry for each unsuppressed terminal. Built in a polling list work area are two-byte entries which will later be used in building the polling list.

When the entire TUT has been processed, the polling list is built using the entries placed in the work area when the SDR was built. If the terminal is a 1050, a polling entry is built for each input component. The special poll entry containing the polling list count and timeout values is placed last.

As each group of control blocks for a line is finished, the sector of the LCT from which the LCB was built is restored in main storage so the next LCT entry can be addressed.

After all LCBs have been built, a check is made for an invalid BSCA OCL statement and whether a valid BSCA OCL statement should cause the DFF buffer values in $CCCOM to be swapped.

On Model 4, in addition to the above, an internal TUT is forced for the console and, if an LCT was processed for BSCA-2, an internal LCT is forced.

Finally the DTF checklist is moved into place and the next build address is restored.

Phase $CC3DF: Build Short DTFs for Disk Files

$CC3DF builds a chain of disk DTFs and chain allocates this string. Then the chain is broken for each DTF to be opened and waited for during any required buffer priming. All priming is done in the buffer area. The chain must be broken and a wait for priming must occur because of potential conflicts in usage of the buffer priming area. Following the compression of the post-open CCP short DTFs, subsequent passes will be made until all files are opened and their DTFs compressed if more disk files are to be opened. Following the construction of any CCP-built master indexes, the DSM Program Level Communications Area (field NPDTF@) is set to X'FFFF', indicating to DSM there are no DTFs to close.

Should the user move a file from a 5444 to a 5445 (3340 main data area to simulation area on Model 12) or vice versa, the File Control Table (FCT) must be updated to reflect the actual file location. $CC3DF builds DTFs according to the FCT device type indication, but not the OCL UNIT parameter.

Phase $CC3MV: Load and Initialize $CC4DF and $CC4#3

Functions

- Borrows ATRs to map PL1.
- Loads and initializes $CC4DF and $CC4#3.
- Initializes intercept routine ($@9CIN) and interrupt handler ($@@BIN).

Operation

- Always saves NPBEG of PL1 in SYSCOM to ensure access to $CCCOM.
- If system size is greater than 64K, inserts addresses into $@9CIN, which contains a table of required addresses. The table contains offsets into $@9CIN where these addresses are to be inserted.
- If DFF remap is required, the starting address of CM is found in $CCCOM and beyond this point ATRs may be borrowed. The first four useable ATRs are set to map between 6K (minimum) and 8K (maximum) of PL1 coincident with PL2's NPBEG. $CC4DF is loaded so that its RLDs, if any, are in $CCCOM where a sector has been saved and is used later to overlay the RLDs. $CC4#3 is loaded so its RLDs are physically in the low storage location used by $CC4DF. There, RLDs are overlaid by the saved sector of $CC4DF.
- Both loaded modules are initialized using a table contained in each. The table contains an offset into common + X(100) where the needed address is located, and an offset into the module itself where the address is required.
- To best utilize main storage, the table in $CC4DF is overlaid with constants it requires to execute, and NPBEG of PL2 is adjusted to the closest sector boundary less or equal to $CC4#3's first executable statement.
- The interrupt handler is initialized by passing: The original values contained in the six ATRs used in mapping PL1, the required Q codes to address these ATRs, and a Q code to no-op two jumps in that module, $@@BIN. $CC4DP is initialized with appropriate ATR Q codes and addresses.
- The load parameter list $CC4DF and its beginning and ending addresses are saved in $CCOMM. The load address of $CC4#3 is entered into $CC4II.
- Control is returned to $CC3RO for the final place of Startup, $CC3EJ.

Note: 6K to 8K bytes of CM are unmapped to supply the ATRs to map the remap area in PL1. PL1 will lose 5.5K to 6K bytes to store $CC4DF and $CC4#3.

Phase $CC3EJ: Final Processing Within Startup

Functions

- Initializes the first level transient program list addressed by @TALST in $CCCOM with the cylinder/sector disk address and the number of text sectors minus one.
- Issues console error messages if required.
- Sets any specified terminal offline.
- Sets final bound addresses of dynamic TP buffer area and user program area.
- Opens MLTA DTFs.
- Indicates in the phase roller eyecatcher field that the Startup phases have completed and passes control to resident Startup code in the CCP transient area of $CC4 (second entry point not named).

Operation

The program pack volume label is read to obtain the extents of the object library directory.

A flag bit is set on in every entry of the first level program list in $CC4. This bit is later turned off if the entry is found in the directory.

The object library directory is searched for module names beginning with $CC4. If one is found, the program list is searched for a match on the last two characters of module name. When a match is found, the cylinder/sector disk address and the number of text sectors (reduced by one) are placed in the program list entry. Inserting the correct number of text sectors automatically turns off the not-found flag bit placed there earlier.

When the directory search is finished, any module still containing the flag bit set earlier is noted in a console message and error switch SSAERR is set on in the Startup Save Area.

If no errors have occurred during startup and if OFFLINE or SUPPRESS was specified by keyword or if YES or Y was answered to message SU045, the system operator is allowed to set any terminal ID offline. If the terminal was already offline, this is noted in a console message. Keyboard input is validated.

If main storage was not exhausted during startup, the dynamic TP buffer is set to begin following the last control block. If storage is available, the buffer will begin on a four-byte boundary; otherwise it will begin at whatever address is available following the control blocks. The upper bound of this storage pool is set at the lowest 256-byte boundary address equal to or greater than the beginning buffer address plus the value of MINTPBUF. Thus the user's value for MINTPBUF may be automatically increased by one to 255 bytes. The reason for the possible expansion is that the upper end is contiguous to the user program area, which requires 256-byte boundary alignment.

Console messages are issued notifying the operator the current size of the dynamic TP buffer area and user program area (the latter still at the minimum). If any unallocated storage remains, the operator is informed via a console message and allowed to direct one or more integral 256-byte blocks of this to additional user program area. Any storage not directed to user program area becomes part of the dynamic TP buffer area, determining the final bounds of the TP buffer and user program area. The value of #TPBUF is decremented by four because the Communications Manager maintains a four-byte parameter list in this pool.

If no terminal errors have occurred, a console message (OPENING COMMUNICATION LINES) is issued and any MLTA adapter and MLTA lines are opened.

The phase ID eyecatcher in the Phase Roller is set to X'0001' to indicate control is being given to the resident function.

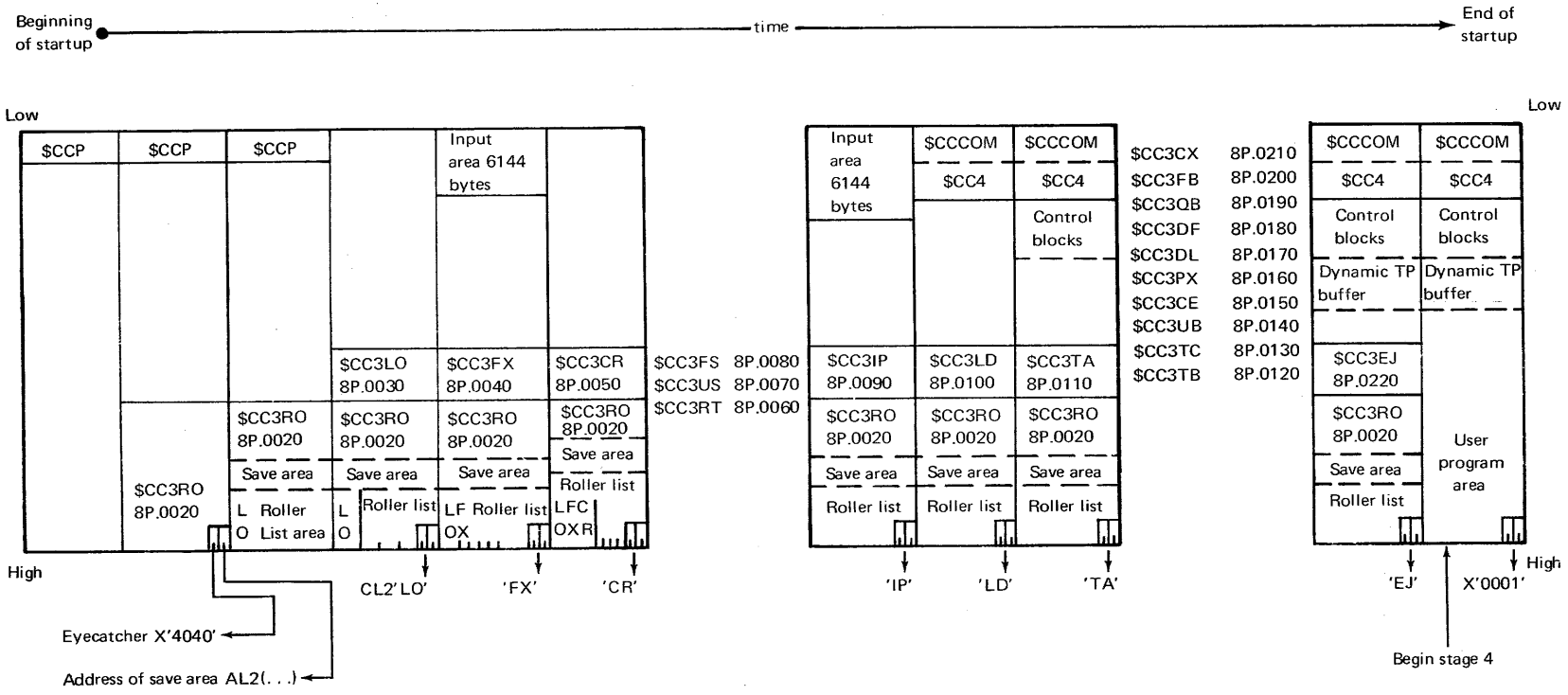
The second entry point address of $CC4 (previously saved from the ARR value of the first $CC4 routine exit) is incremented by two bytes to get beyond a two-byte field in which the halt check routine address is saved for a DPF system.

$CC4 is entered at this point in the CCP transient area containing temporary resident Startup code.

Other Functions

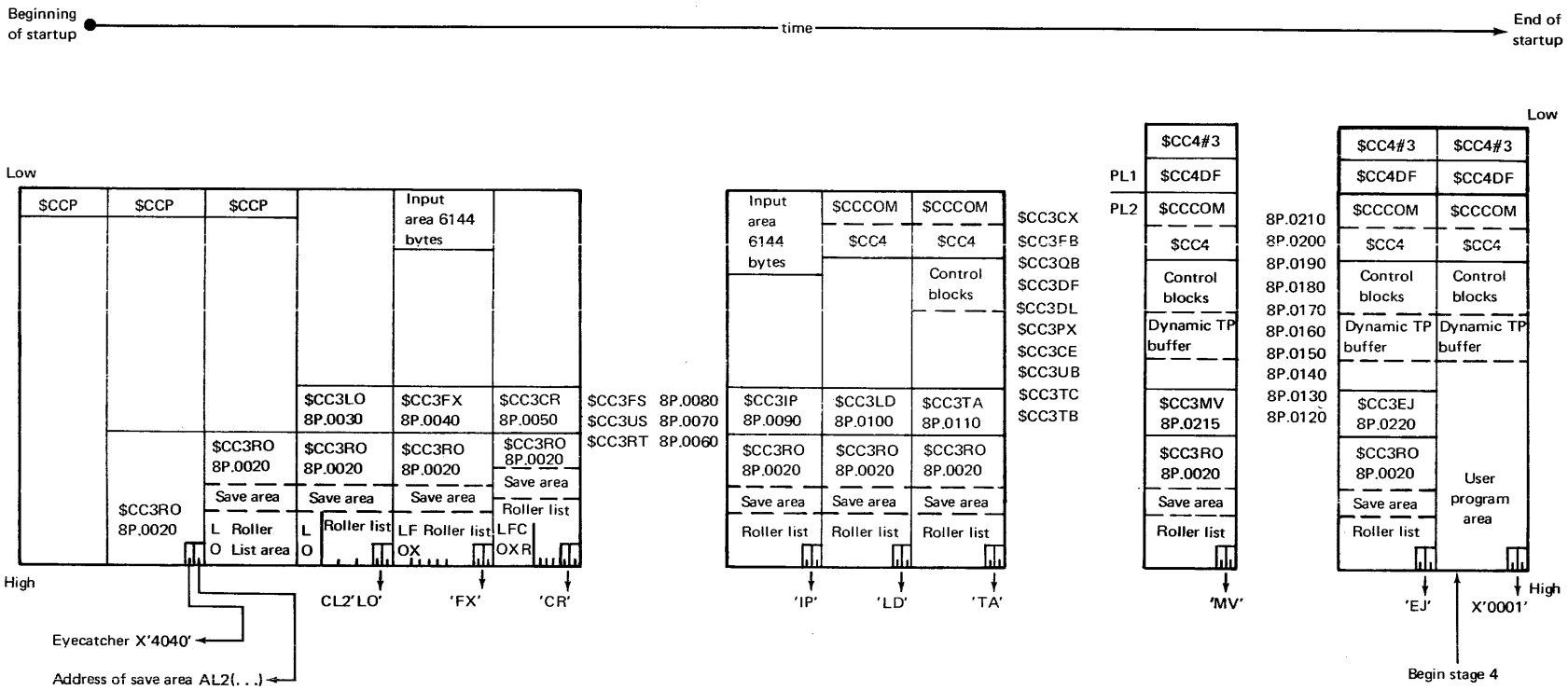
Several Startup functions appear in the temporary resident code in the $CC4 transient area. These are documented as part of the Execution Stage.

The last Startup functions appear in the transient $CC4SU. There the DSM Console Interrupt Handler is plugged to branch to the CCP Console Interrupt Intercept Routine, the keyboard is reset, the CCP STARTED message is issued on the console typewriter, a flag bit is set off allowing the TP MLTA interrupt handler to accept input from terminals, and an Invite-Input is issued to the console and to all online command terminals.



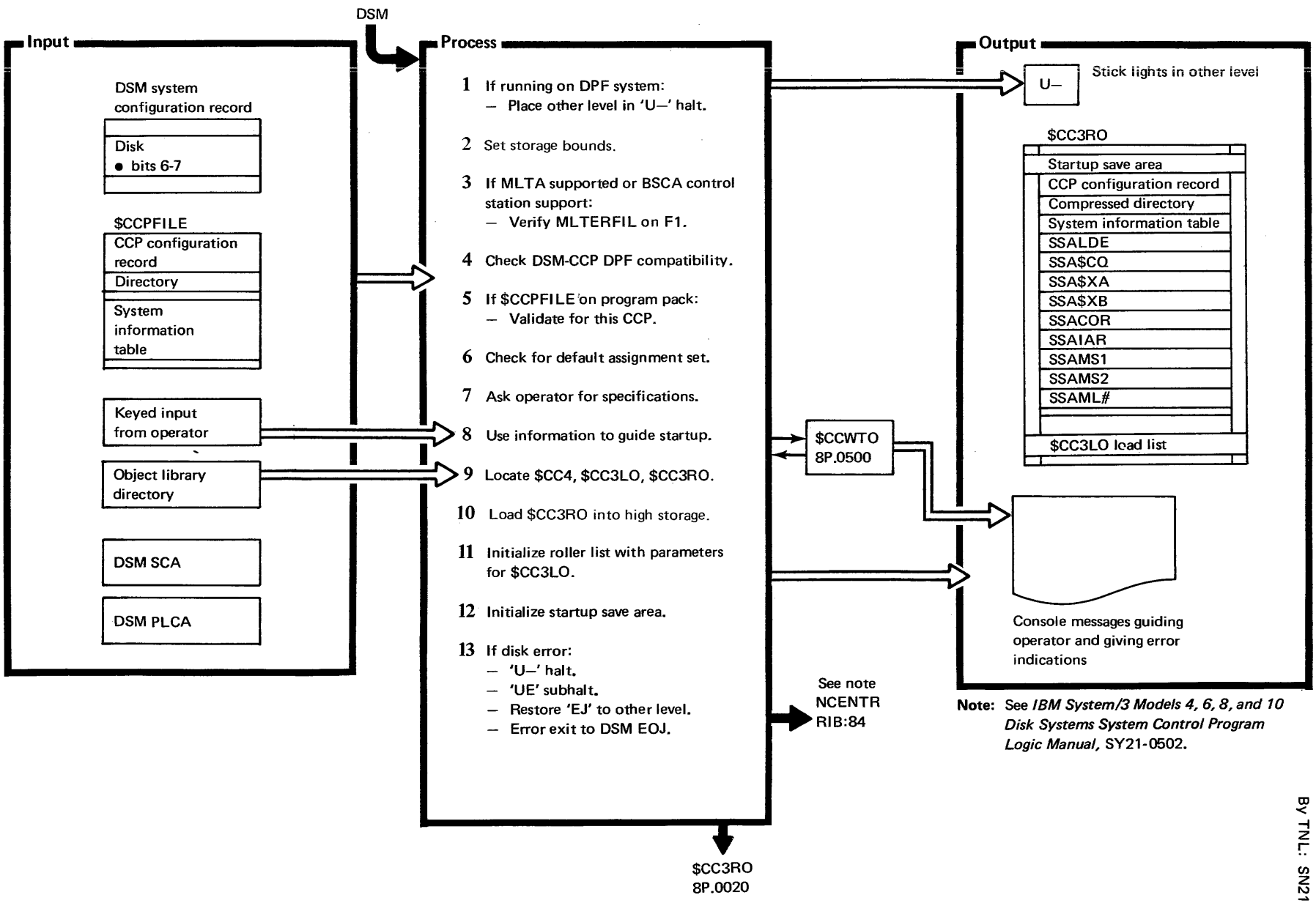
Last 4 bytes of program level

Figure 8-3. Storage Layout of Program Level During Startup



Last 4 bytes of program level

● **Figure 8-4. Storage Layout of Program Level During Startup for Model 12 with DFF Remap**



Note: See *IBM System/3 Models 4, 6, 8, and 10 Disk Systems System Control Program Logic Manual*, SY21-0502.

Diagram 8P.0010. $CCP

Page of SY21-0531-2
Issued 24 June 1977
By TNL: SN21-5530

This page intentionally left blank.

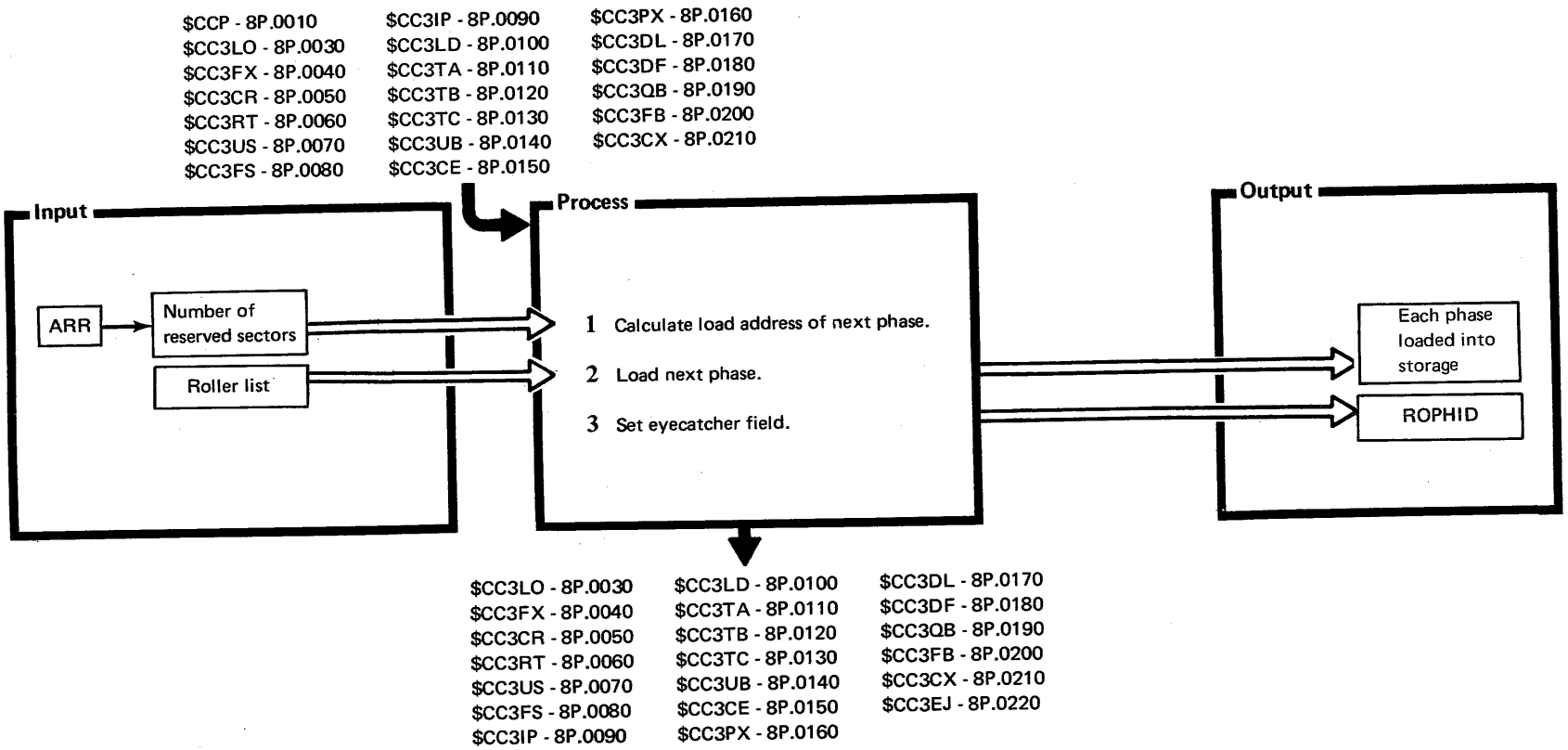
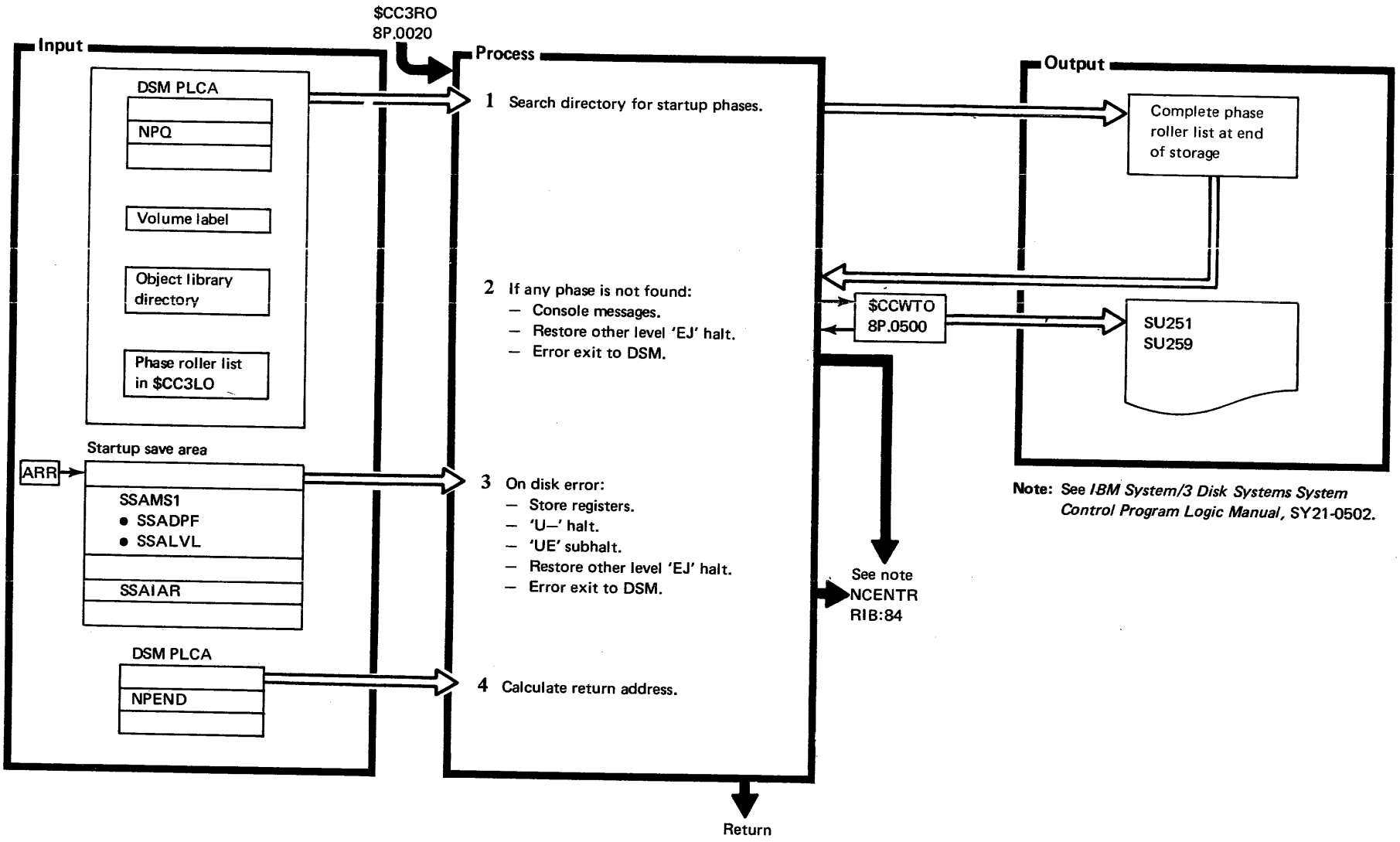
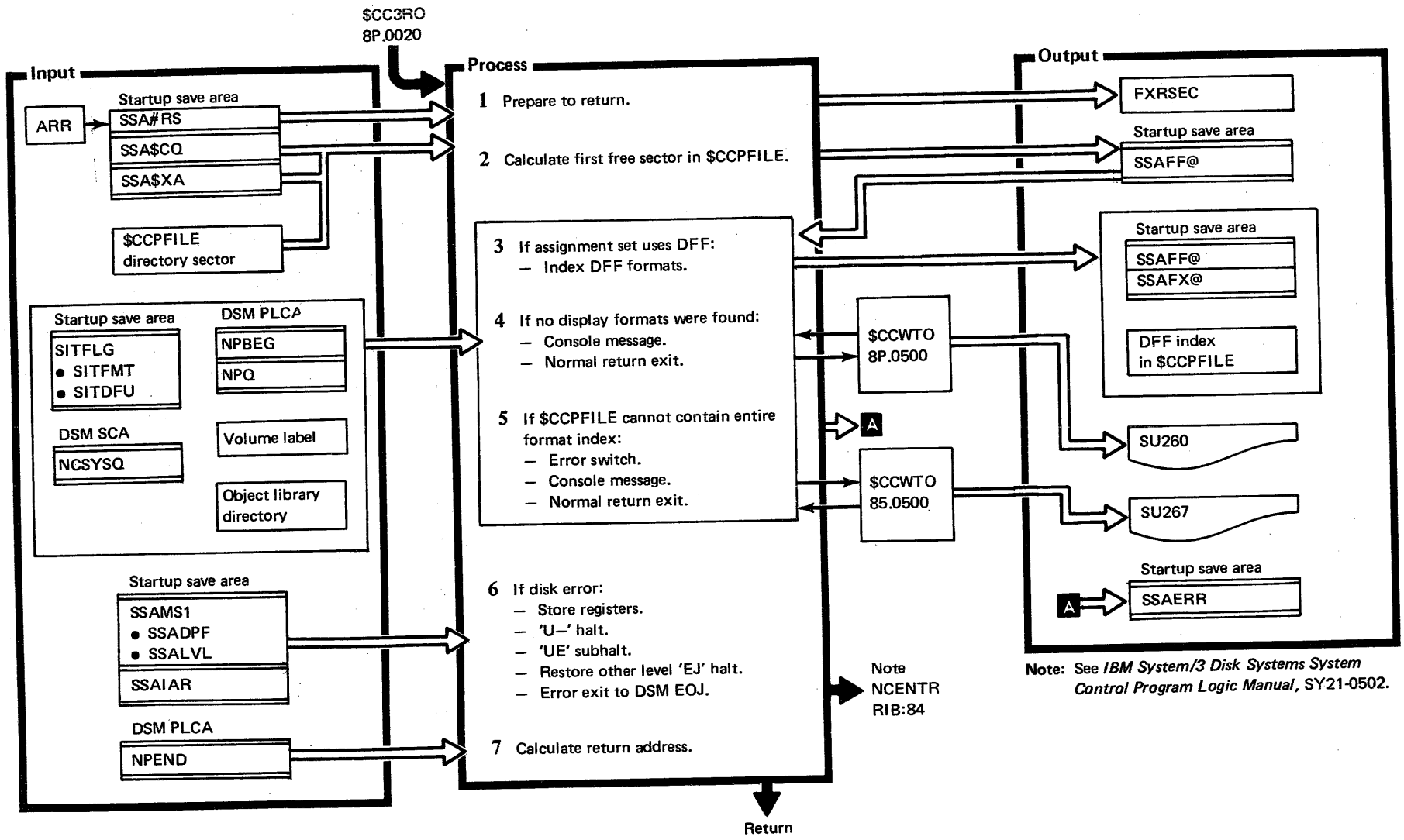


Diagram 8P.0020. $CC3RO



Note: See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.

Diagram 8P.0030. $CC3LO (Models 8, 10, and 12 Only)



Note: See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.

Diagram 8P.0040. $CC3FX

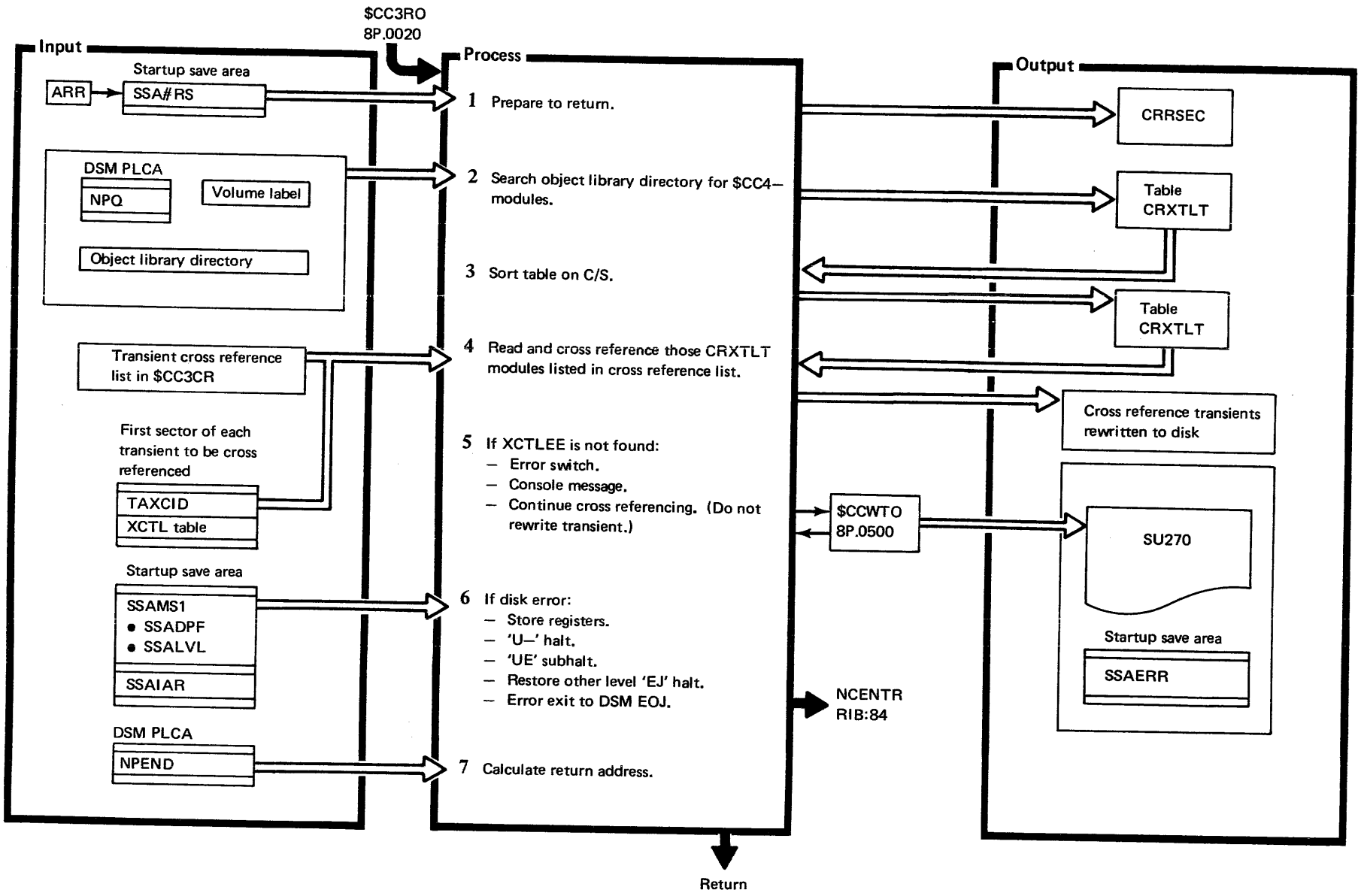


Diagram 8P.0050. $CC3CR

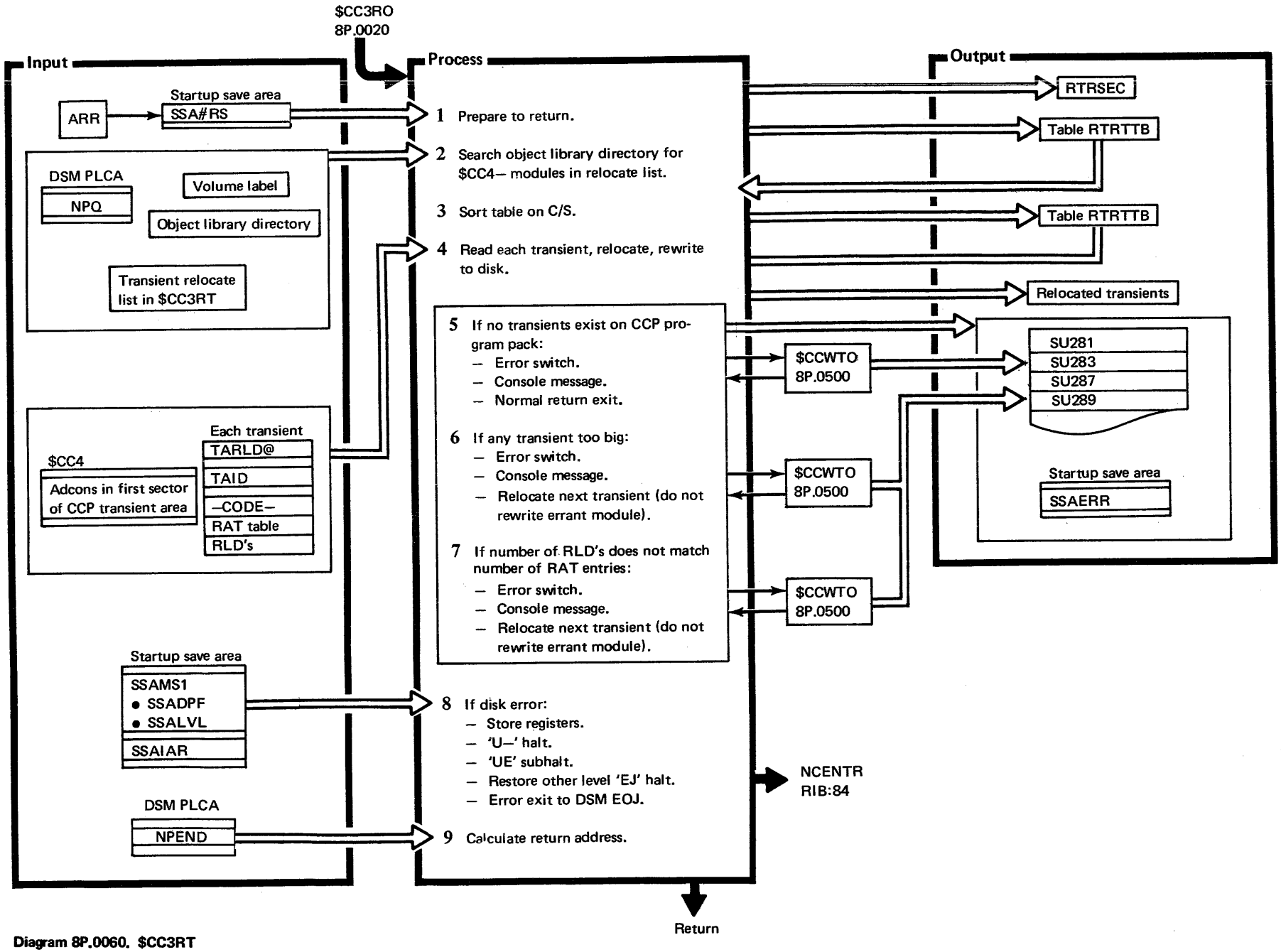


Diagram 8P.0060. $CC3RT

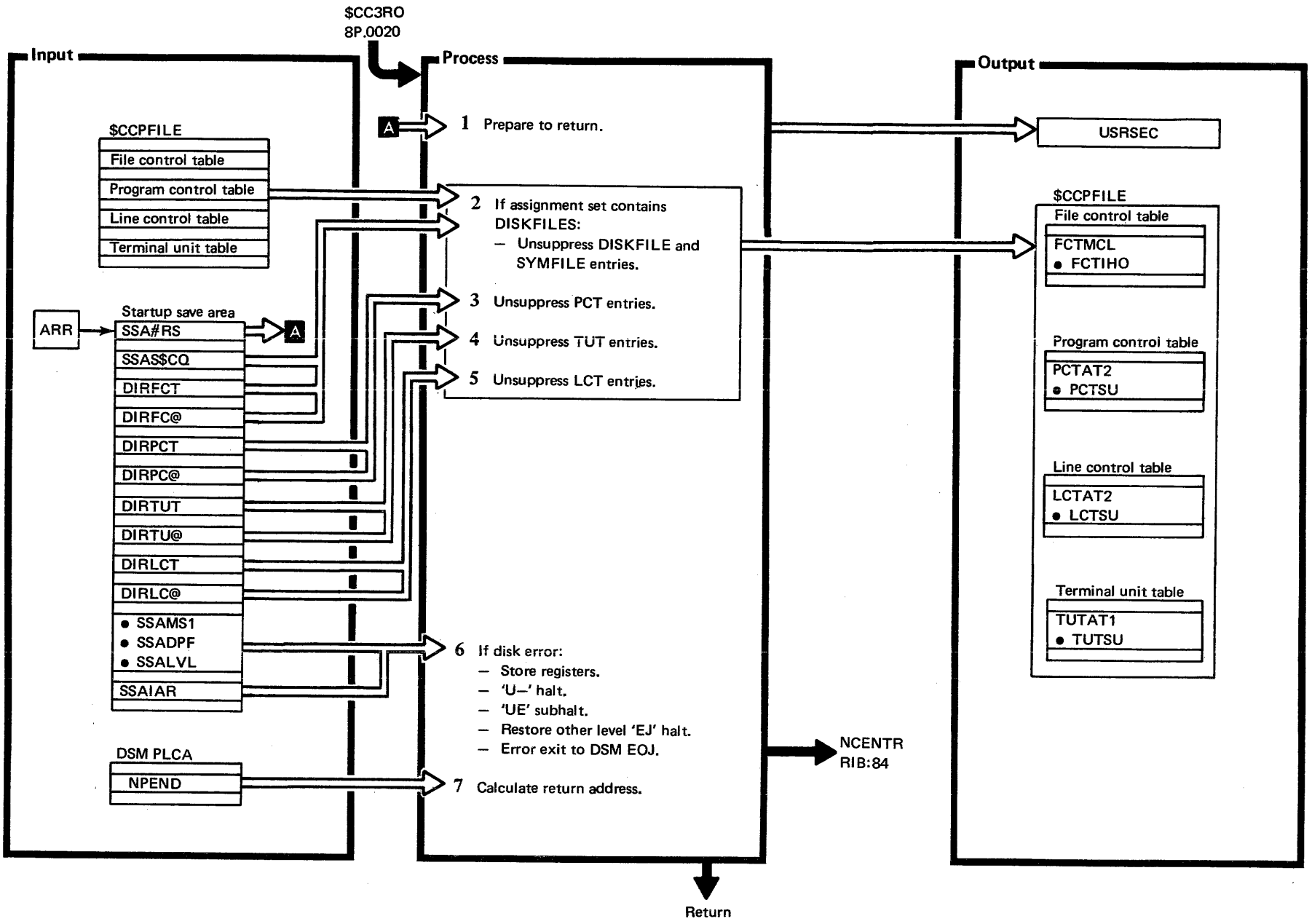


Diagram 8P.0070. $CC3US

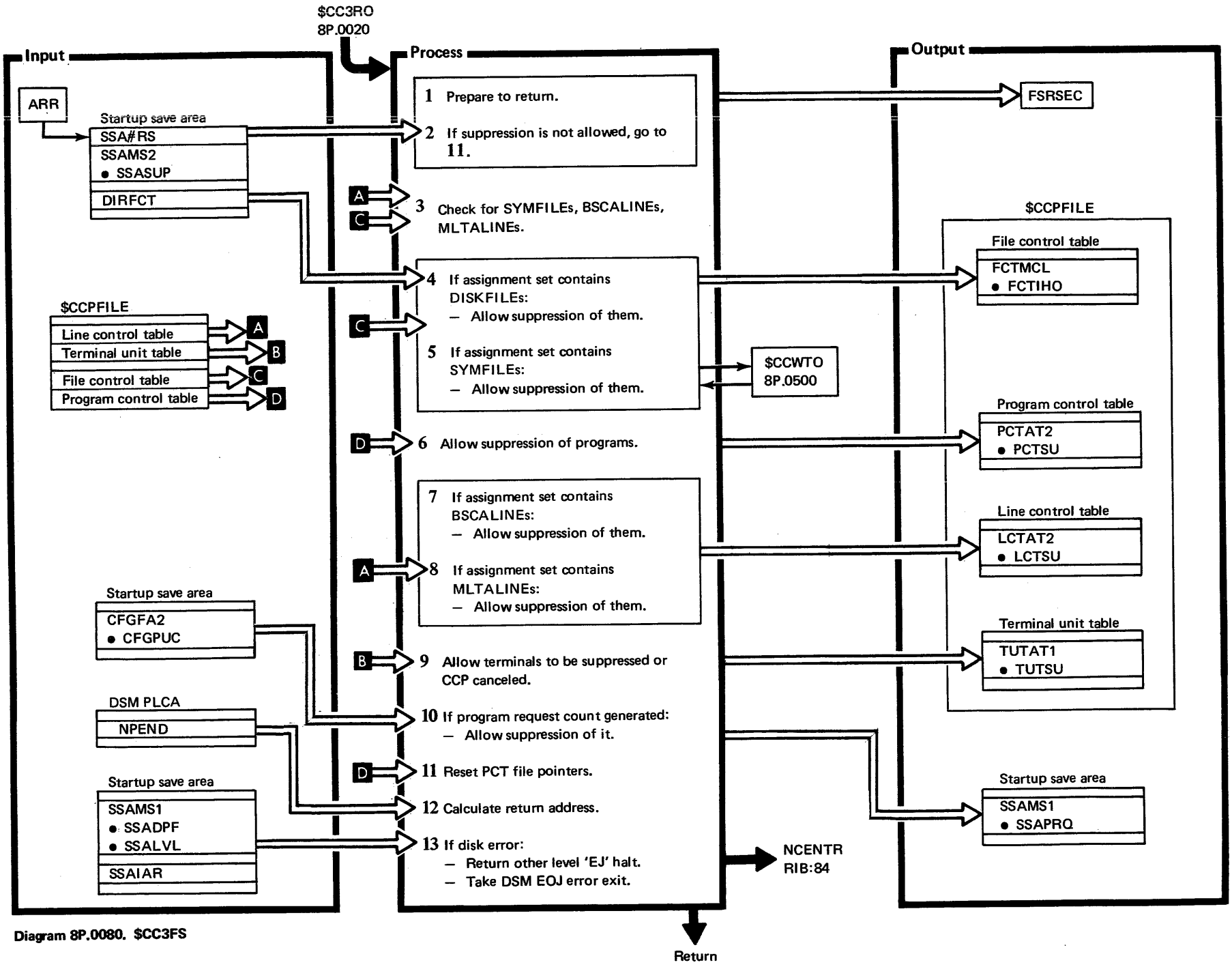


Diagram 8P.0080. $CC3FS

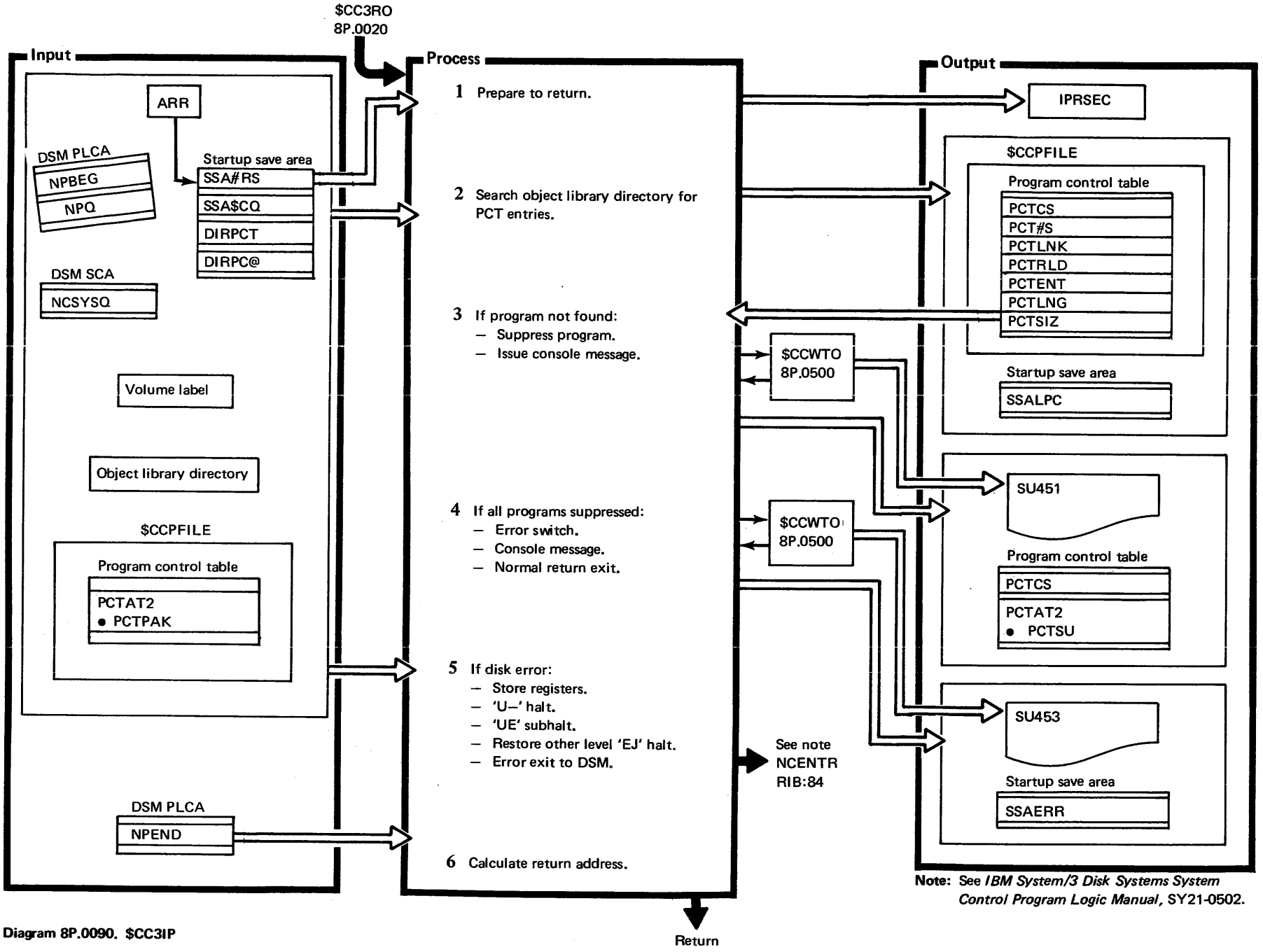


Diagram 8P.0090. $CC3IP

Note: See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.

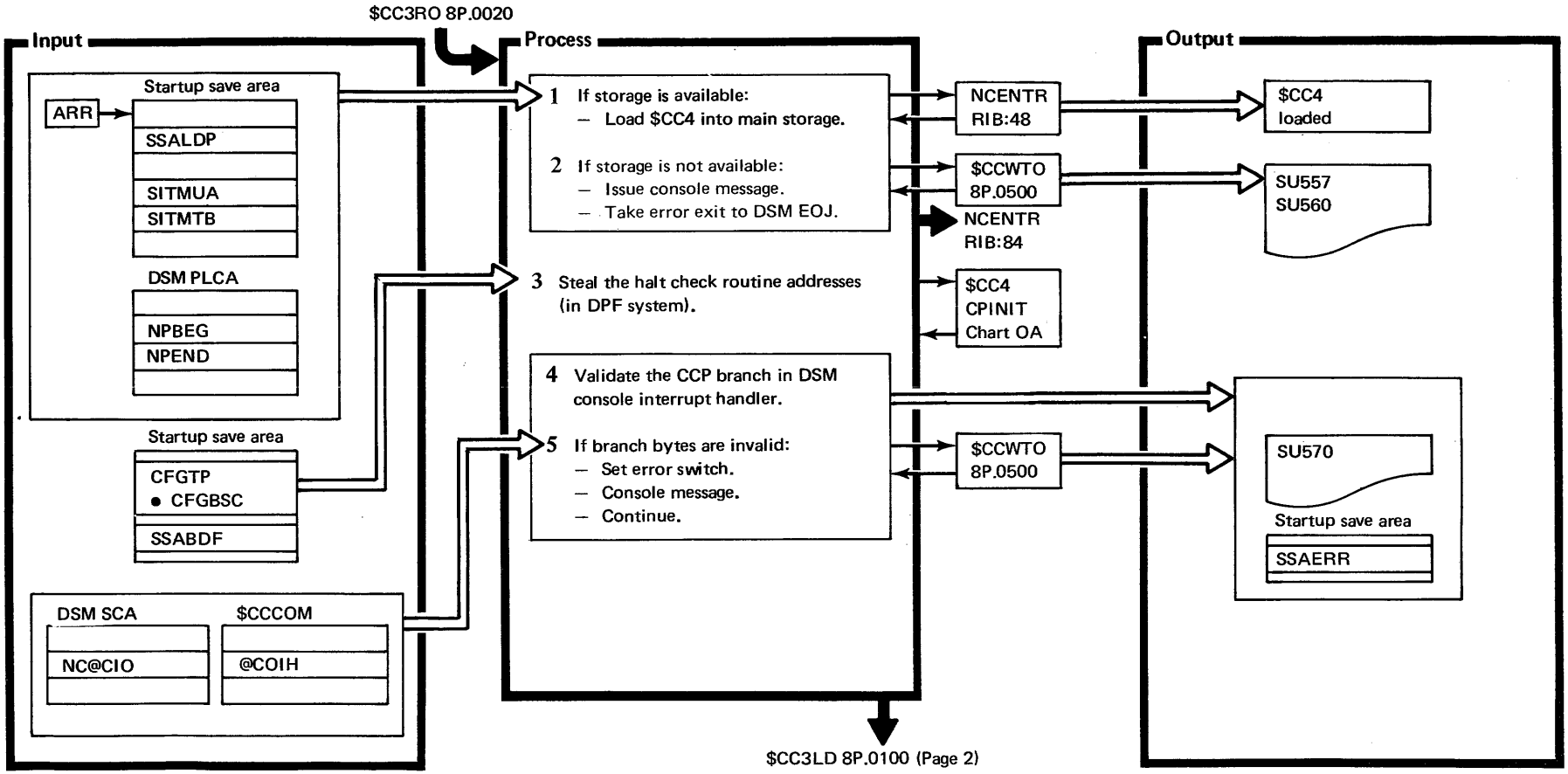
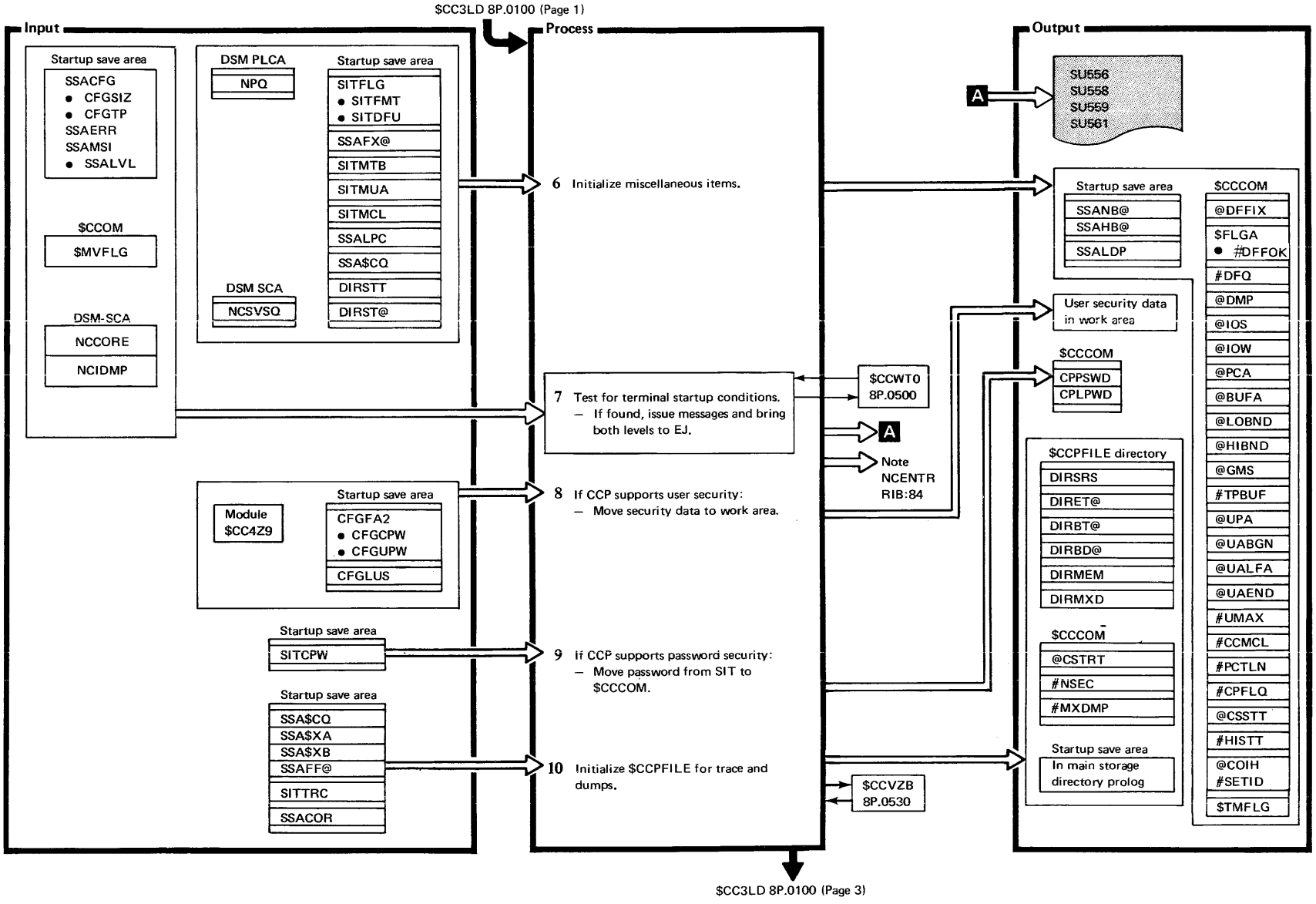
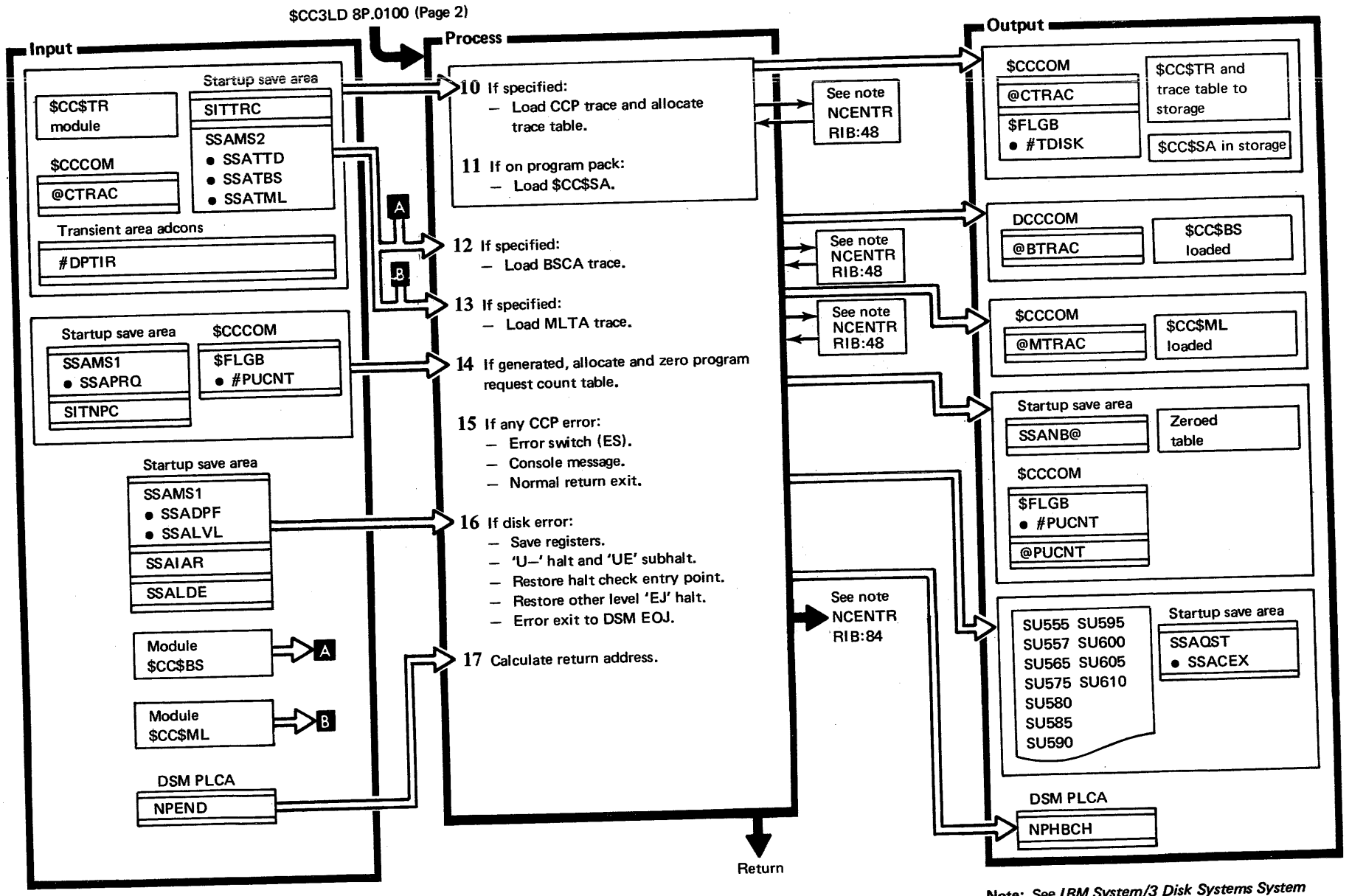


Diagram 8P.0100 (Part 1 of 3). $CC3LD

Page of SY21-0531-2
Issued 24 June 1977
By TNL: SN21-5530



● **Diagram 8P.0100 (Part 2 of 3). $CC3LD**



Note: See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.

Diagram 8P.0100 (Part 3 of 3). $CC3LD

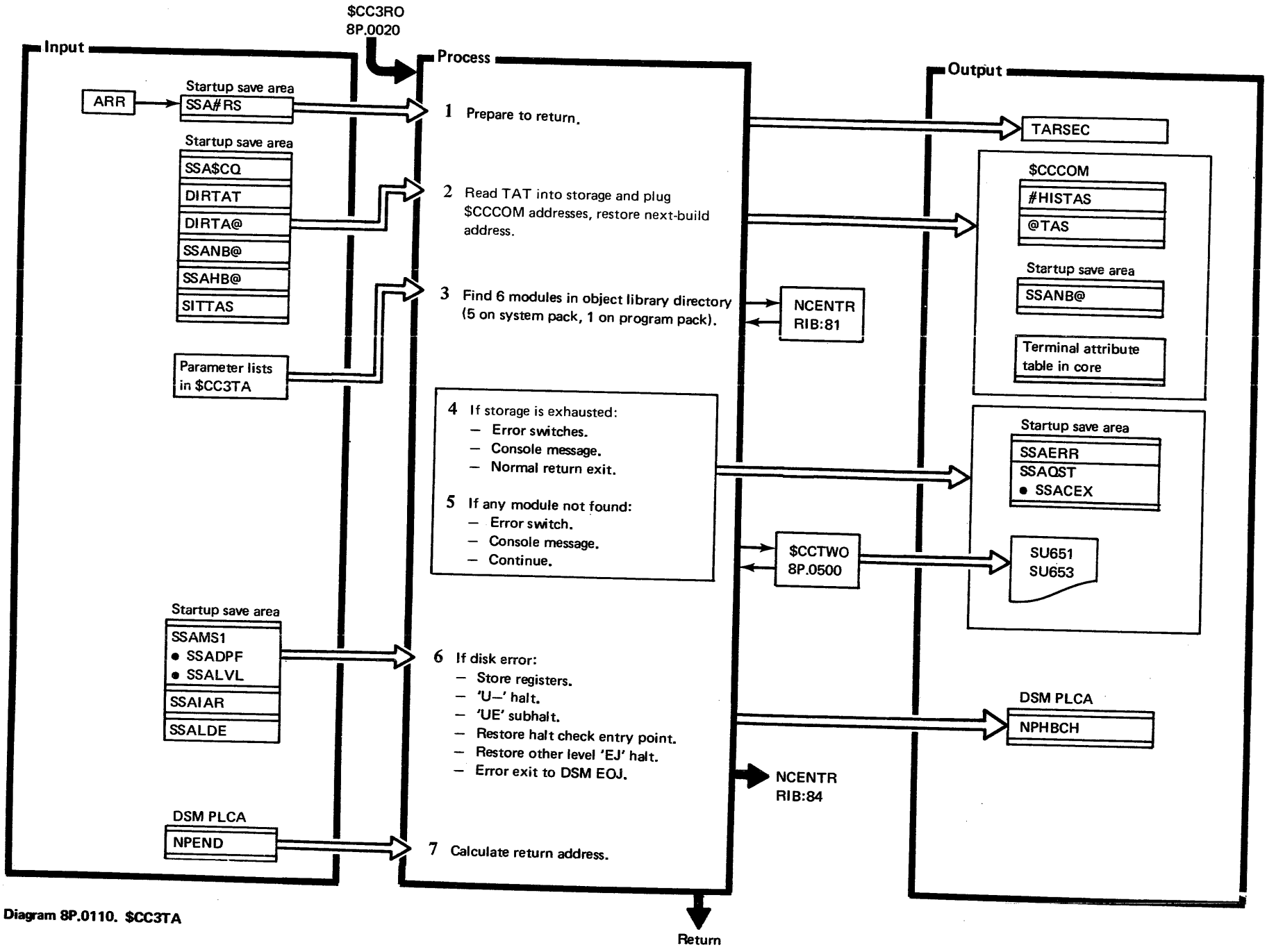


Diagram 8P.0110. $CC3TA

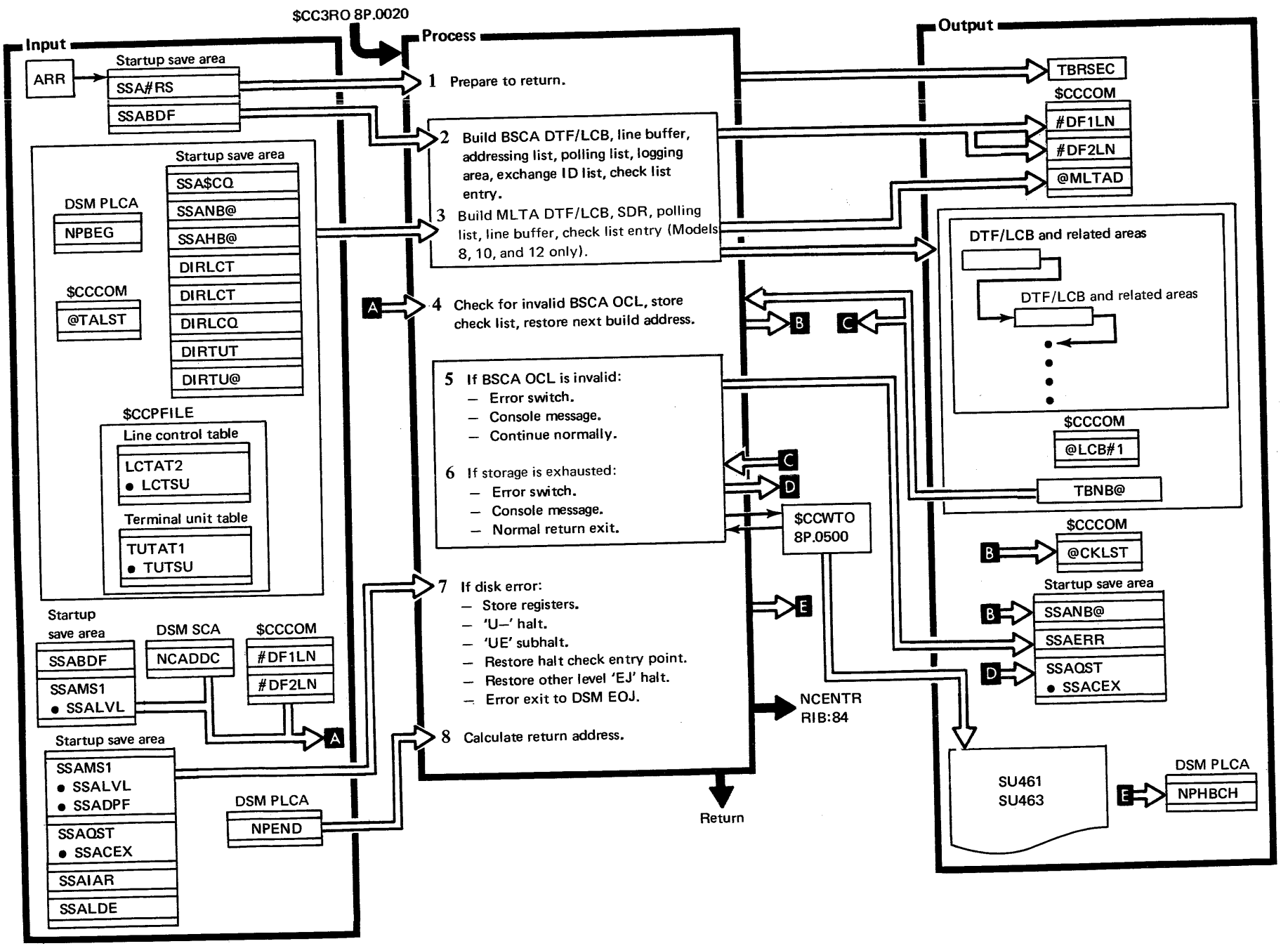


Diagram 8P.0120. $CC3TB

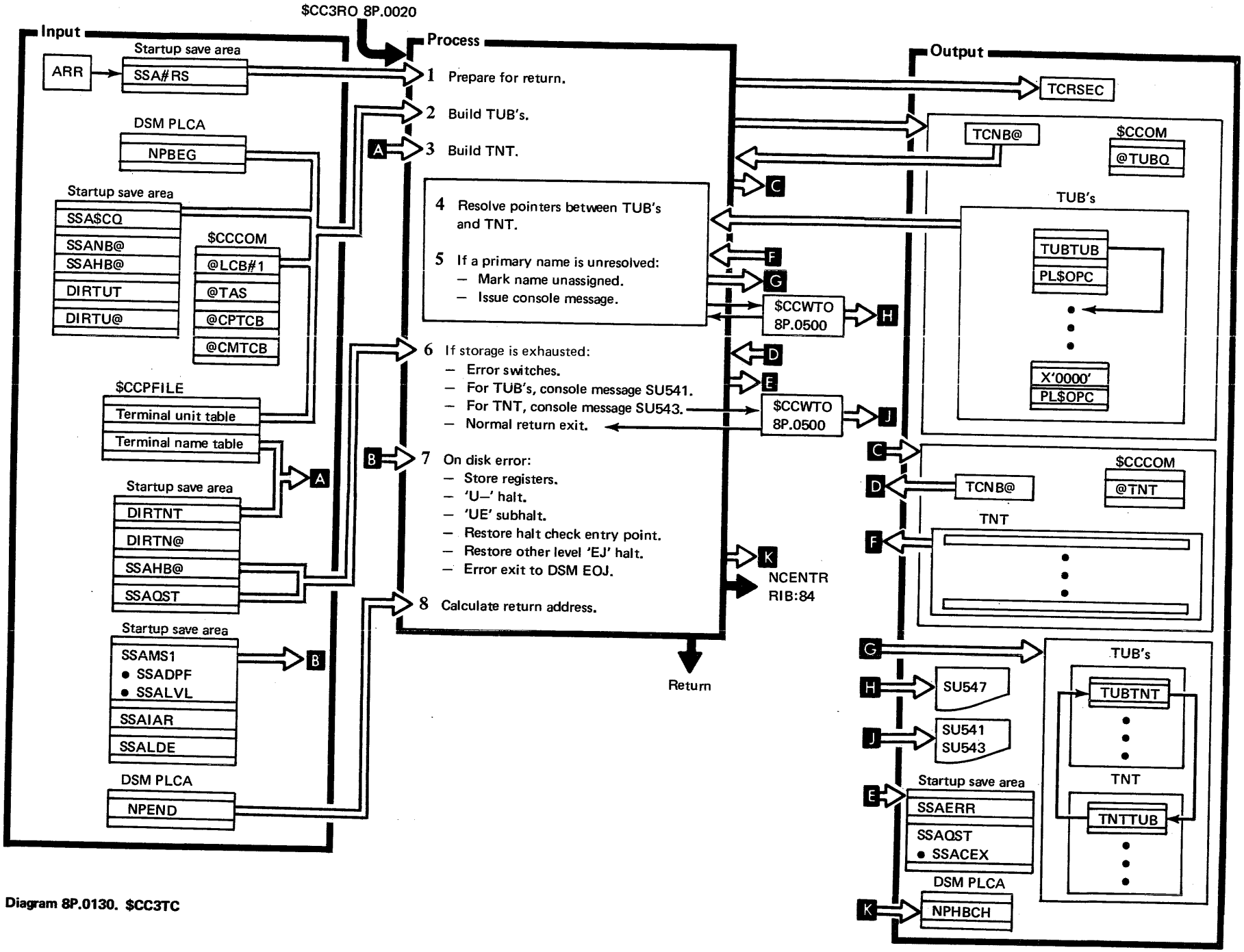


Diagram 8P.0130. $CC3TC

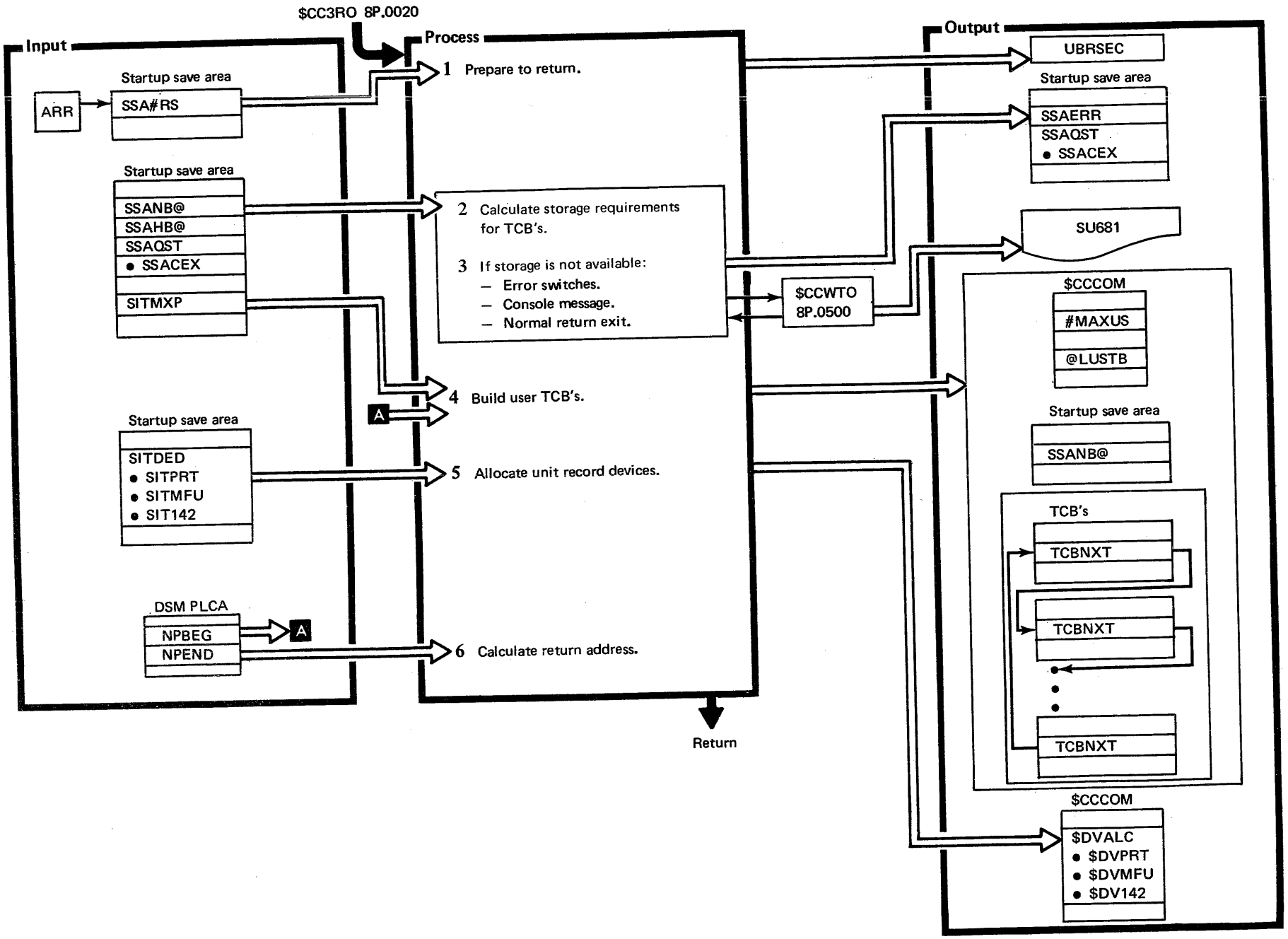


Diagram 8P.0140. $CC3UB

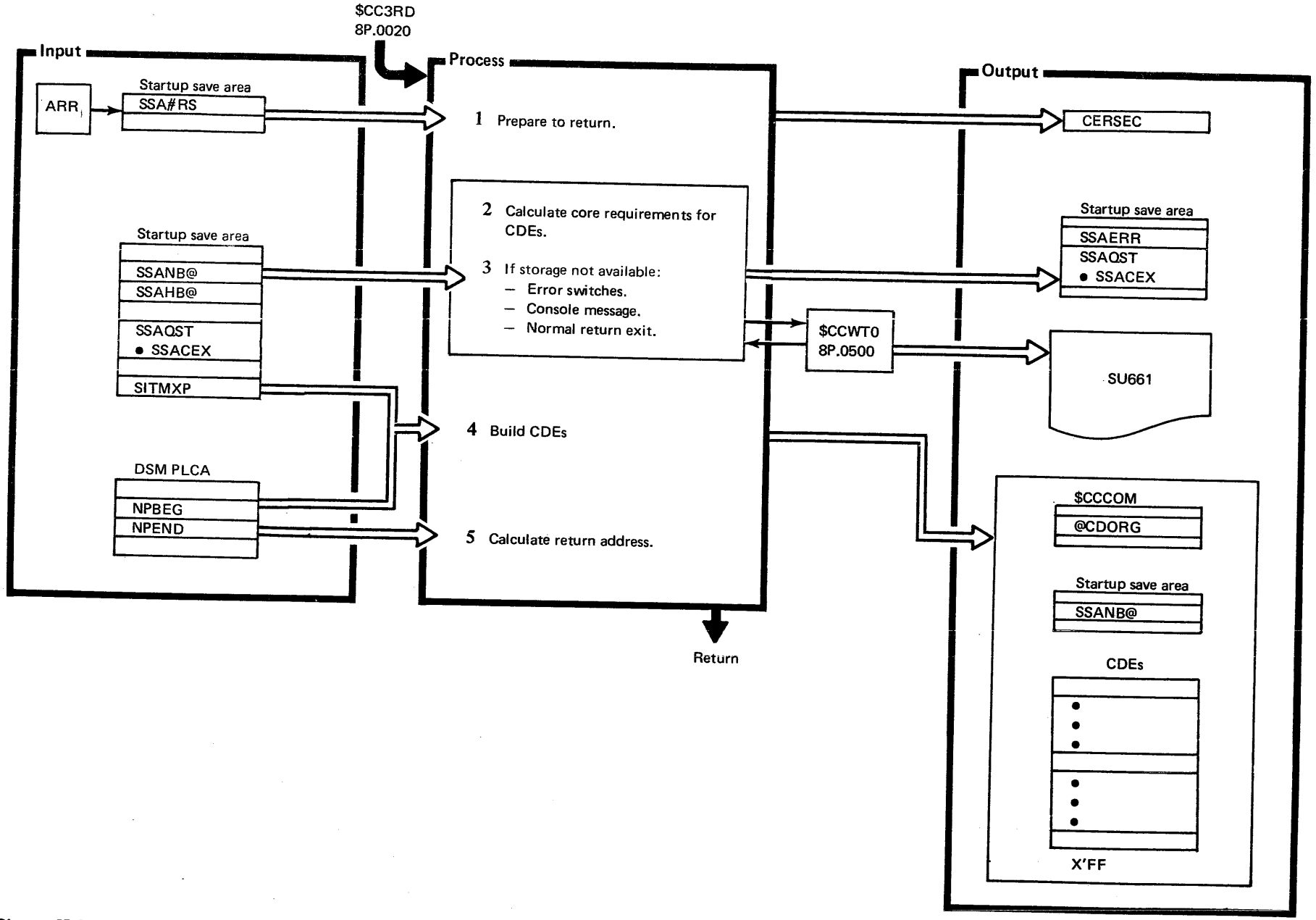


Diagram 8P.0150. $CC3CE

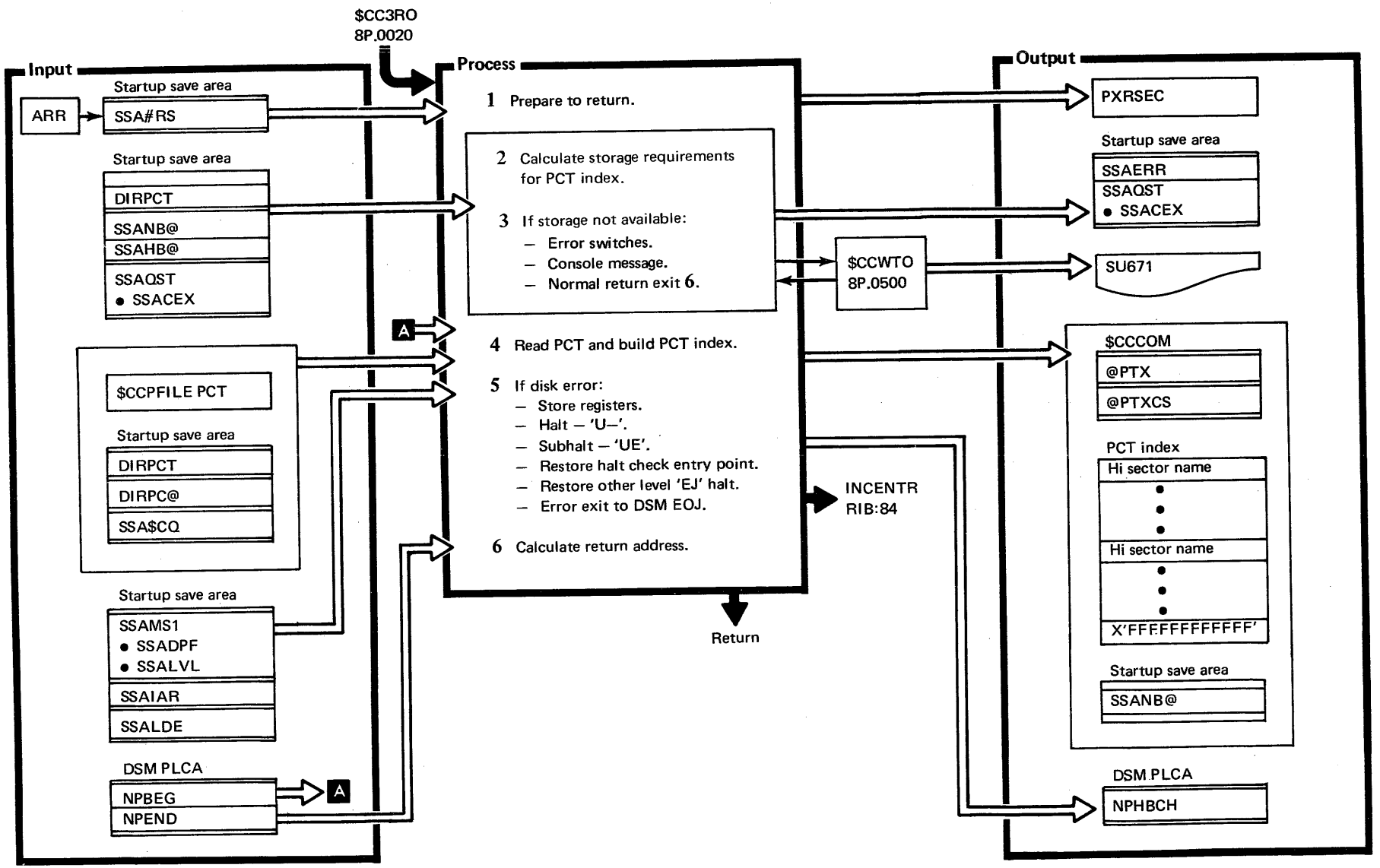


Diagram 8P.0160. $CC3PX

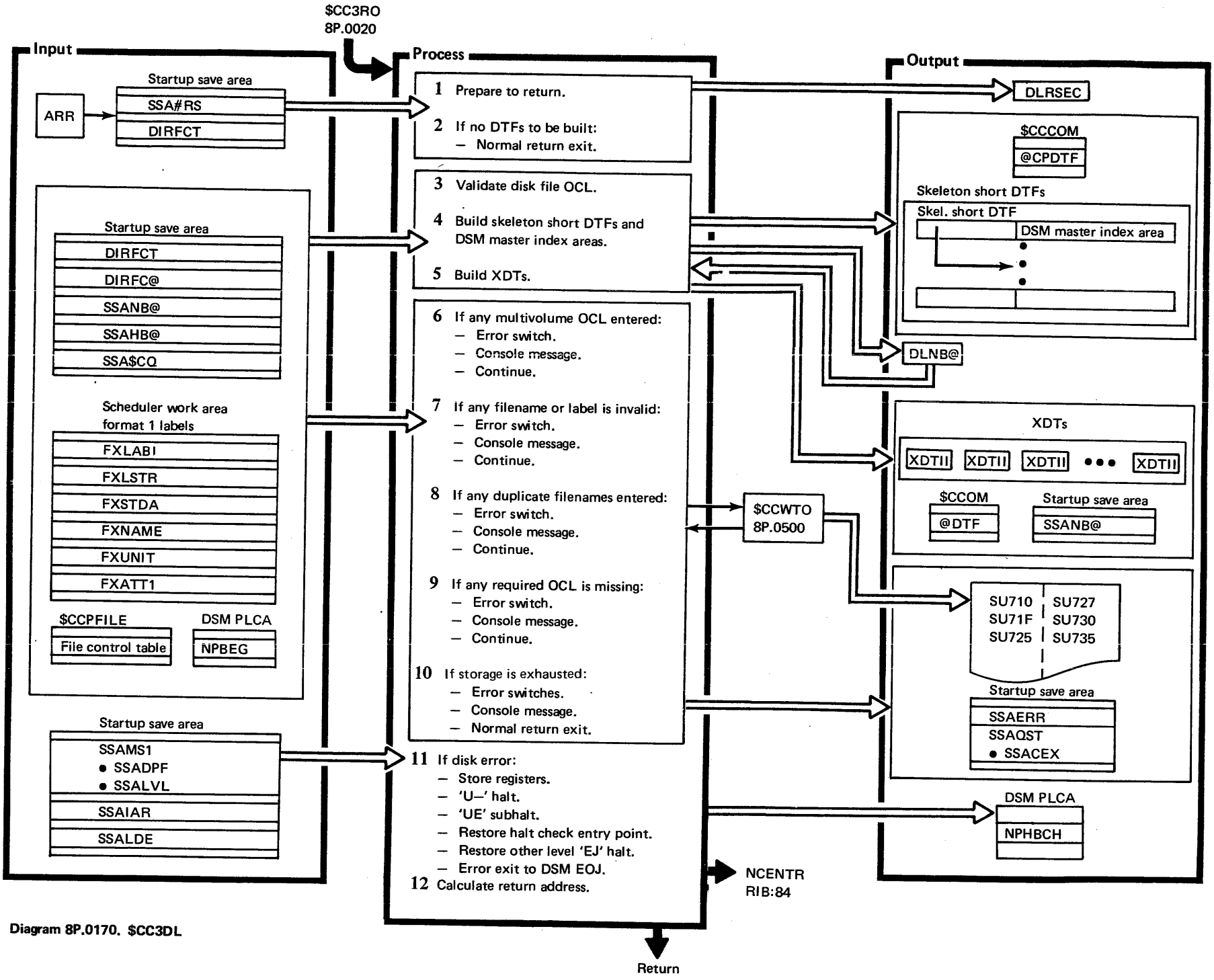


Diagram 8P.0170. $CC3DL

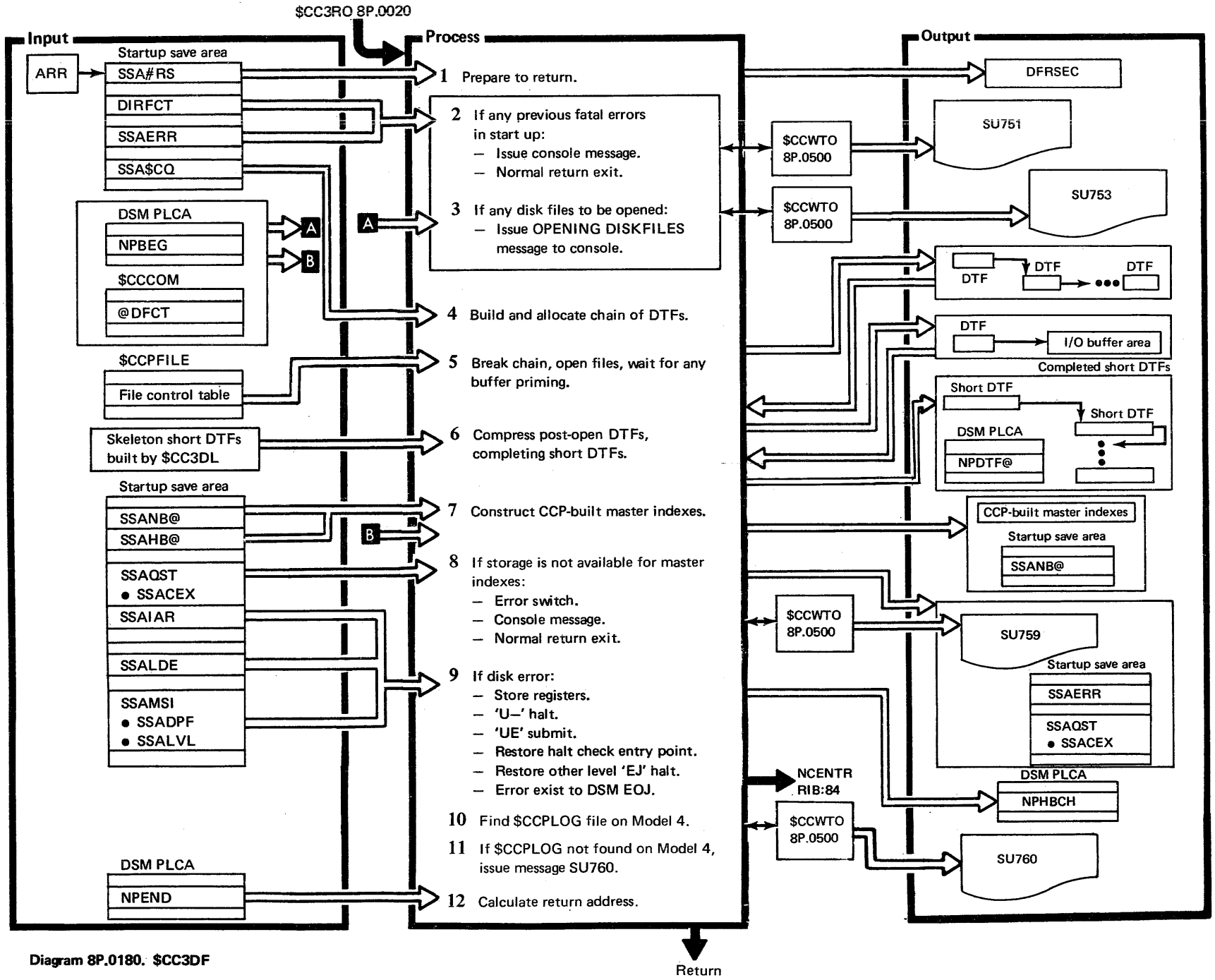


Diagram 8P.0180. $CC3DF

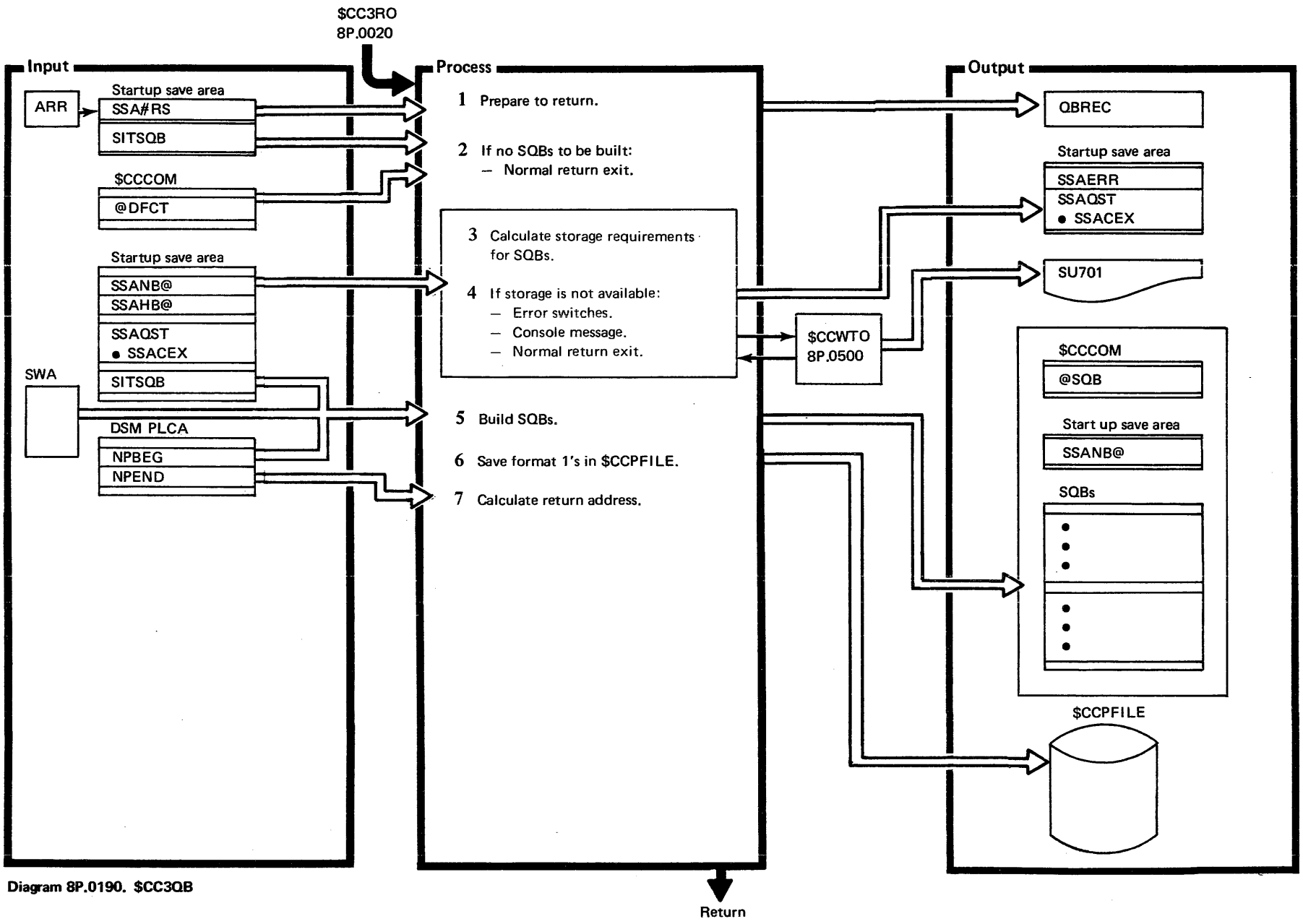


Diagram 8P.0190. $CC3QB

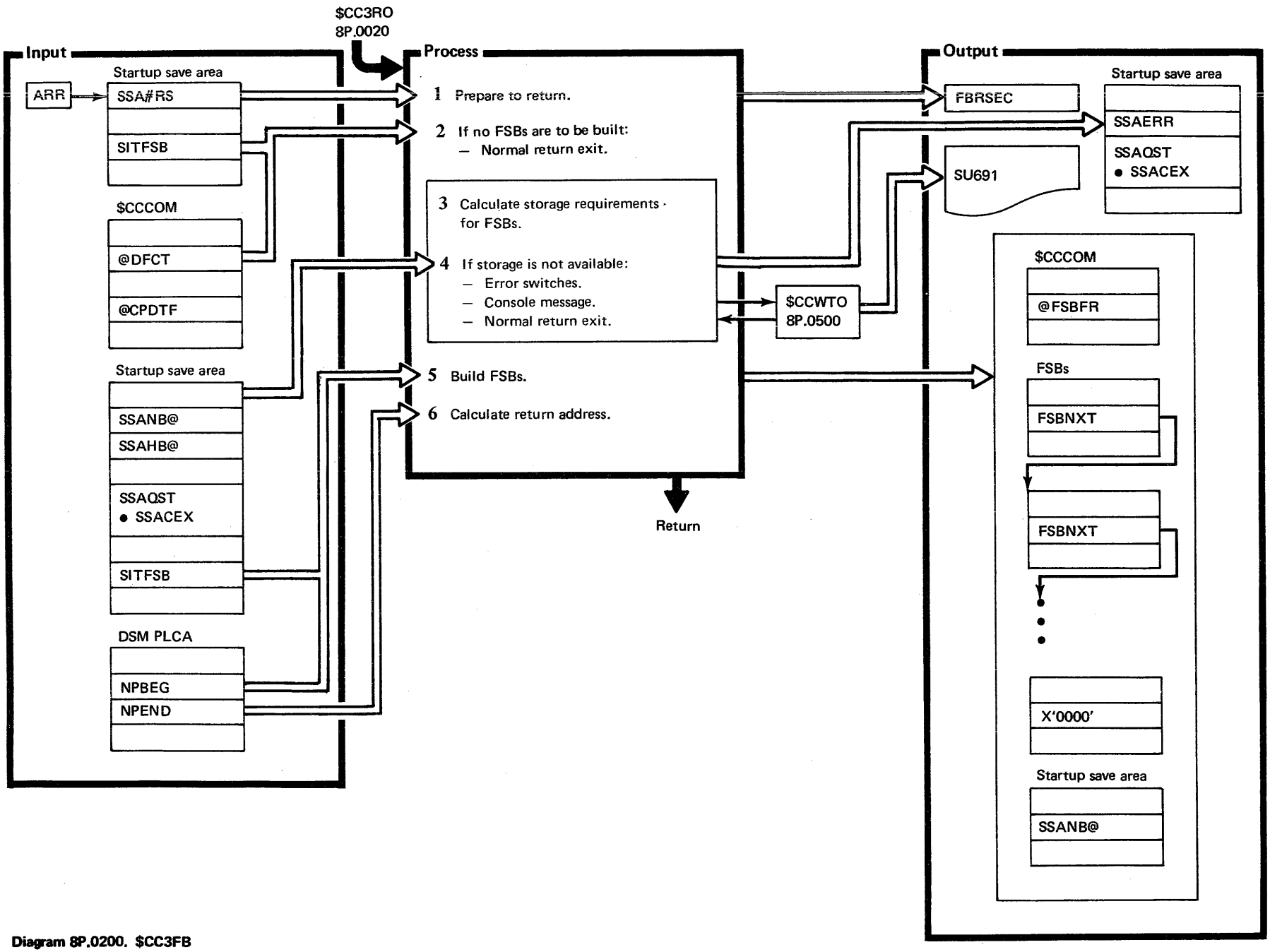


Diagram 8P.0200. $CC3FB

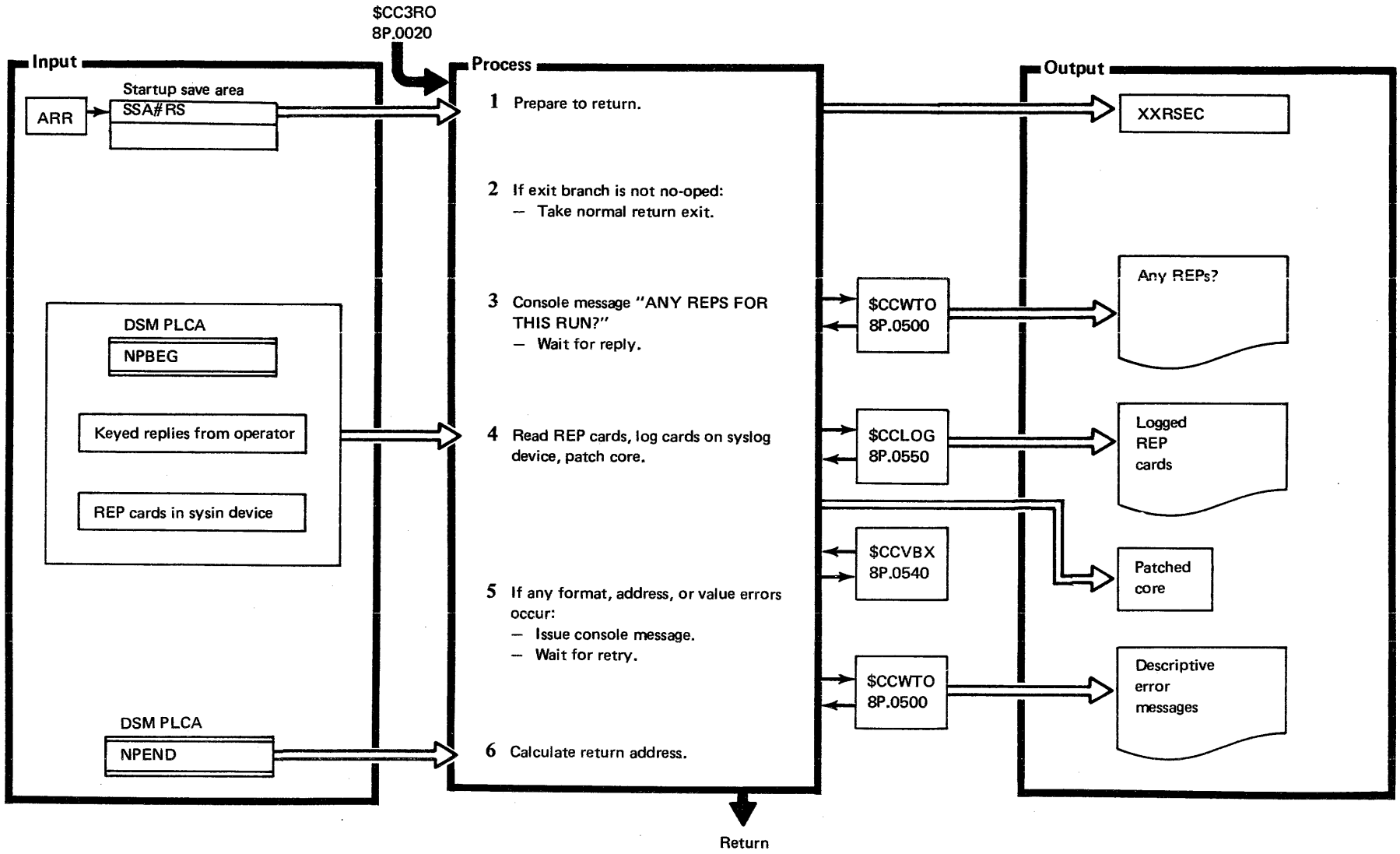
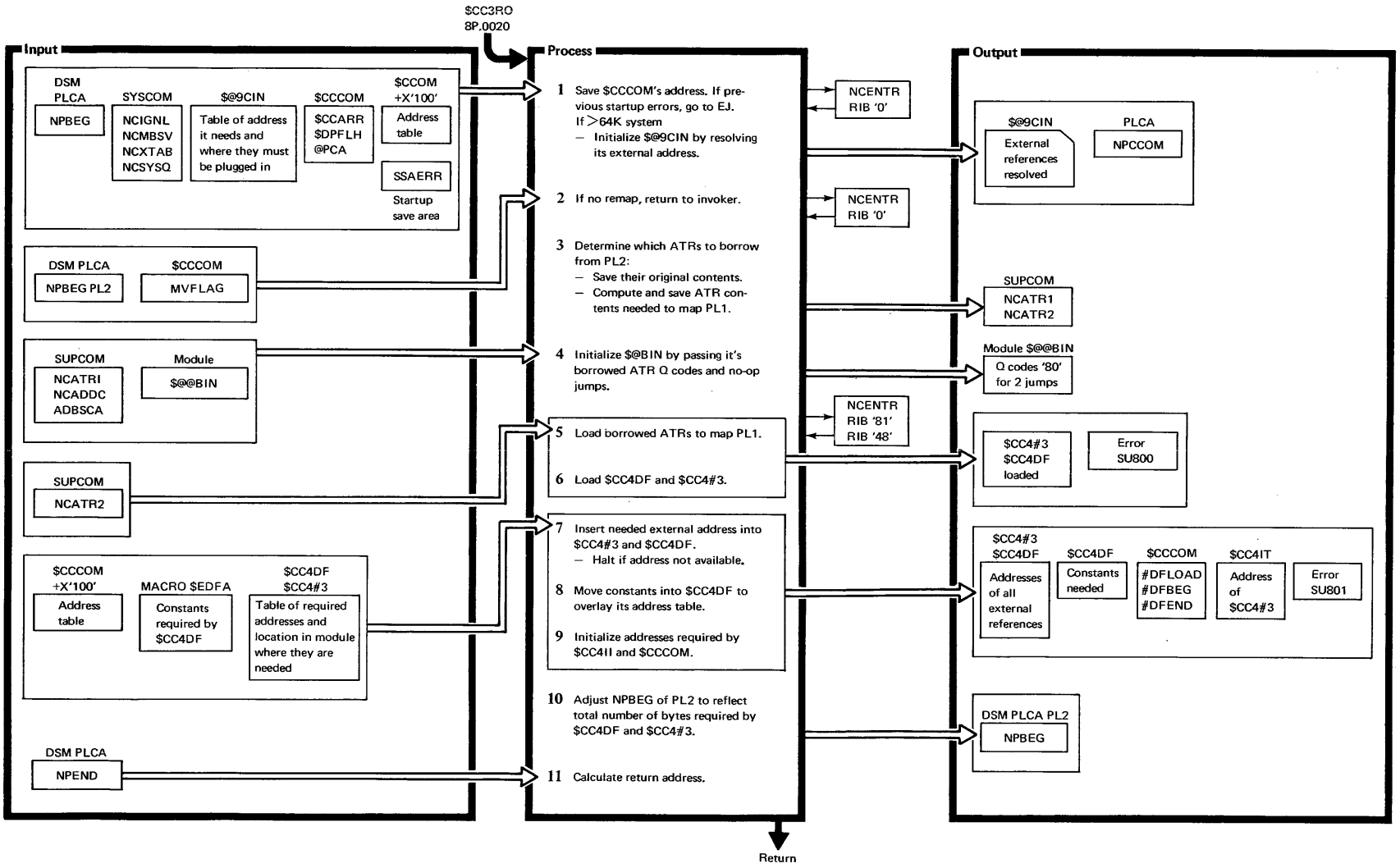


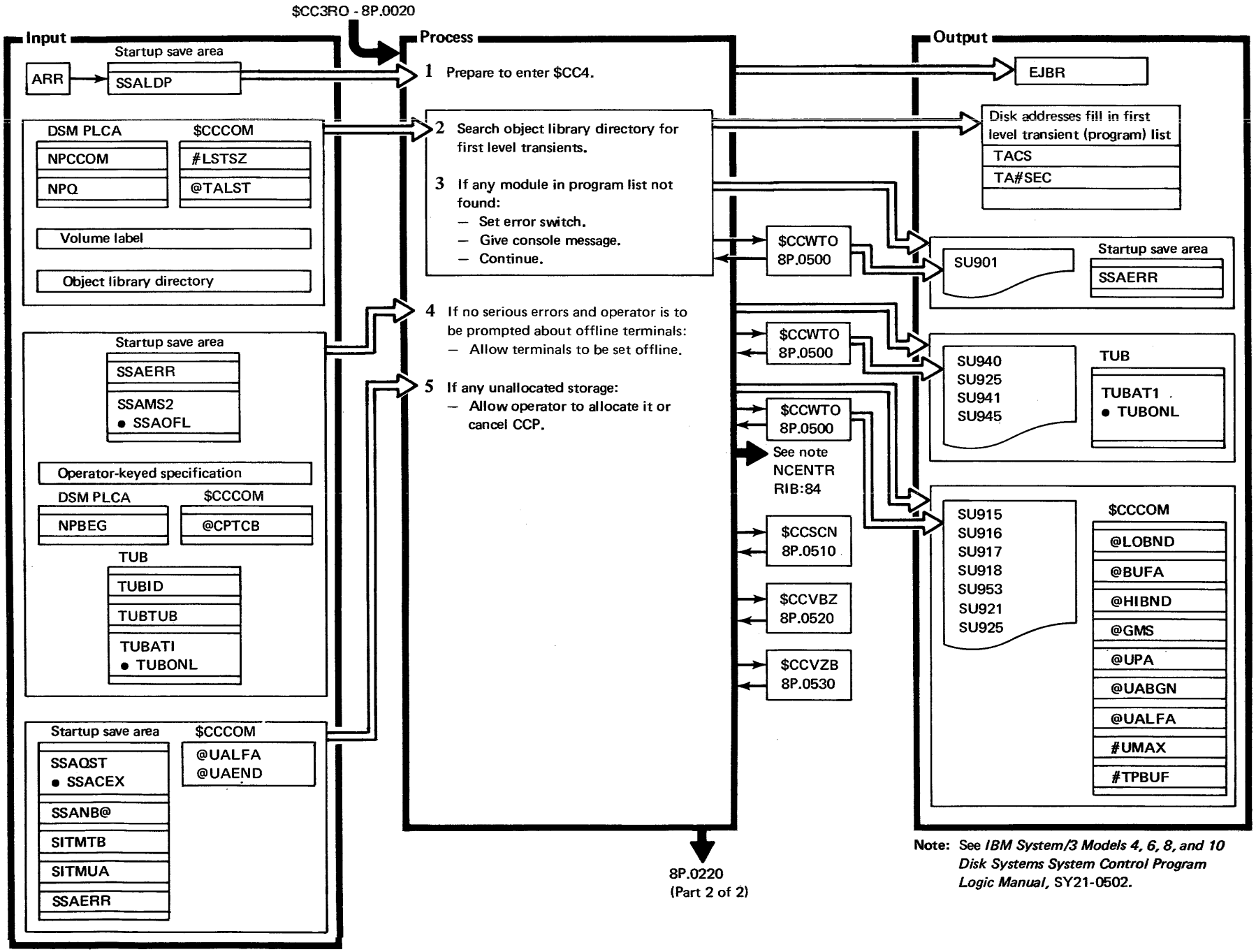
Diagram 8P.0210. $CC3CX



- **Diagram 8P.0215. $CC3MV**

Page of SY21-0531-2
Issued 24 June 1977
By TNL: SN21-5530

This page intentionally left blank.

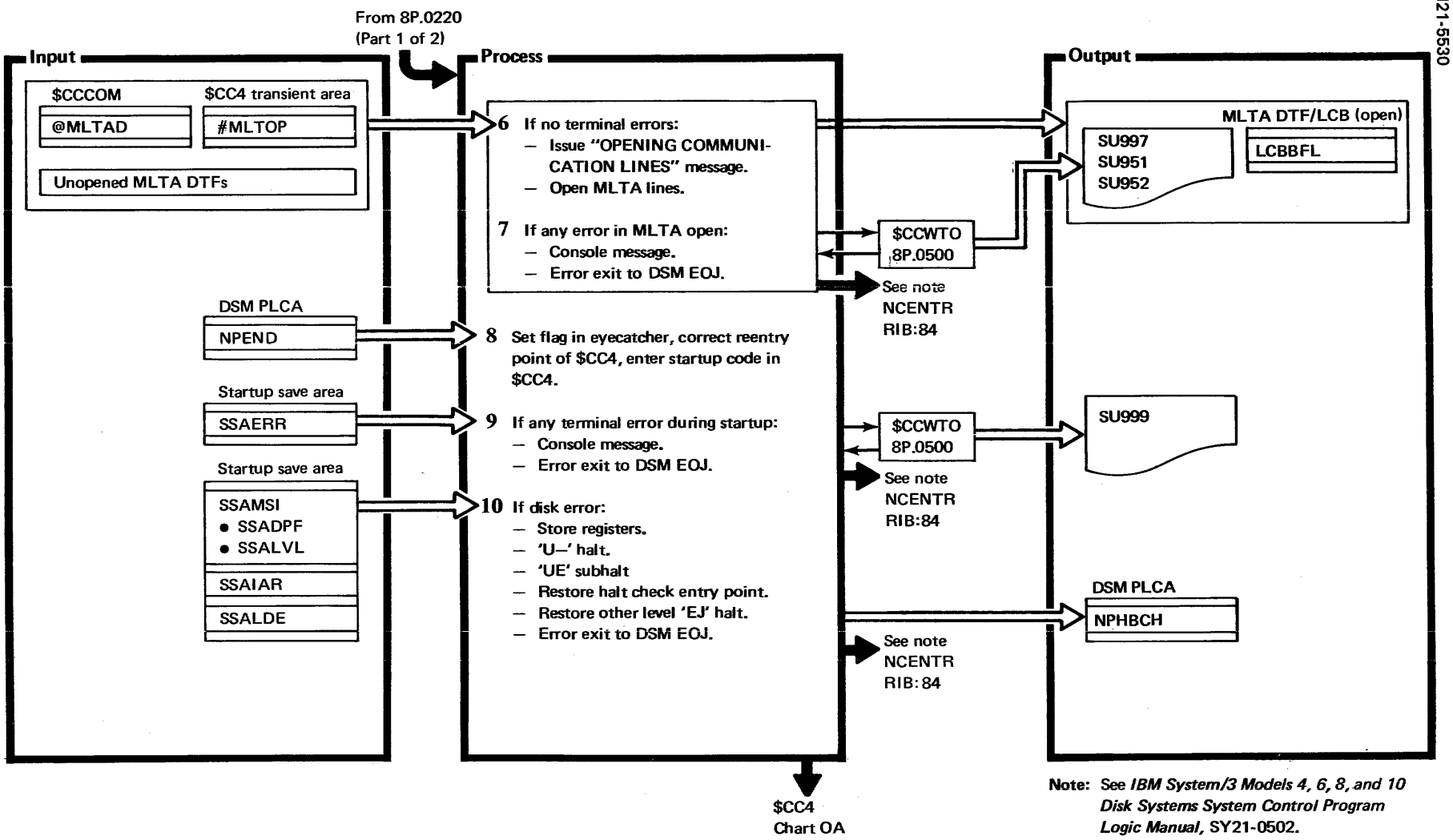


Note: See *IBM System/3 Models 4, 6, 8, and 10 Disk Systems System Control Program Logic Manual*, SY21-0502.

Diagram 8P.0220 (Part 1 of 2). $CC3EJ

Page of SY21-0531-2
Issued 24 June 1977
By TNL: SN21-5530

Page of SY21-0531-2
Issued 24 June 1977
By TNL: SN21-5530



Note: See *IBM System/3 Models 4, 6, 8, and 10 Disk Systems System Control Program Logic Manual*, SY21-0502.

Diagram 8P.0220 (Part 2 of 2). $CC3EJ

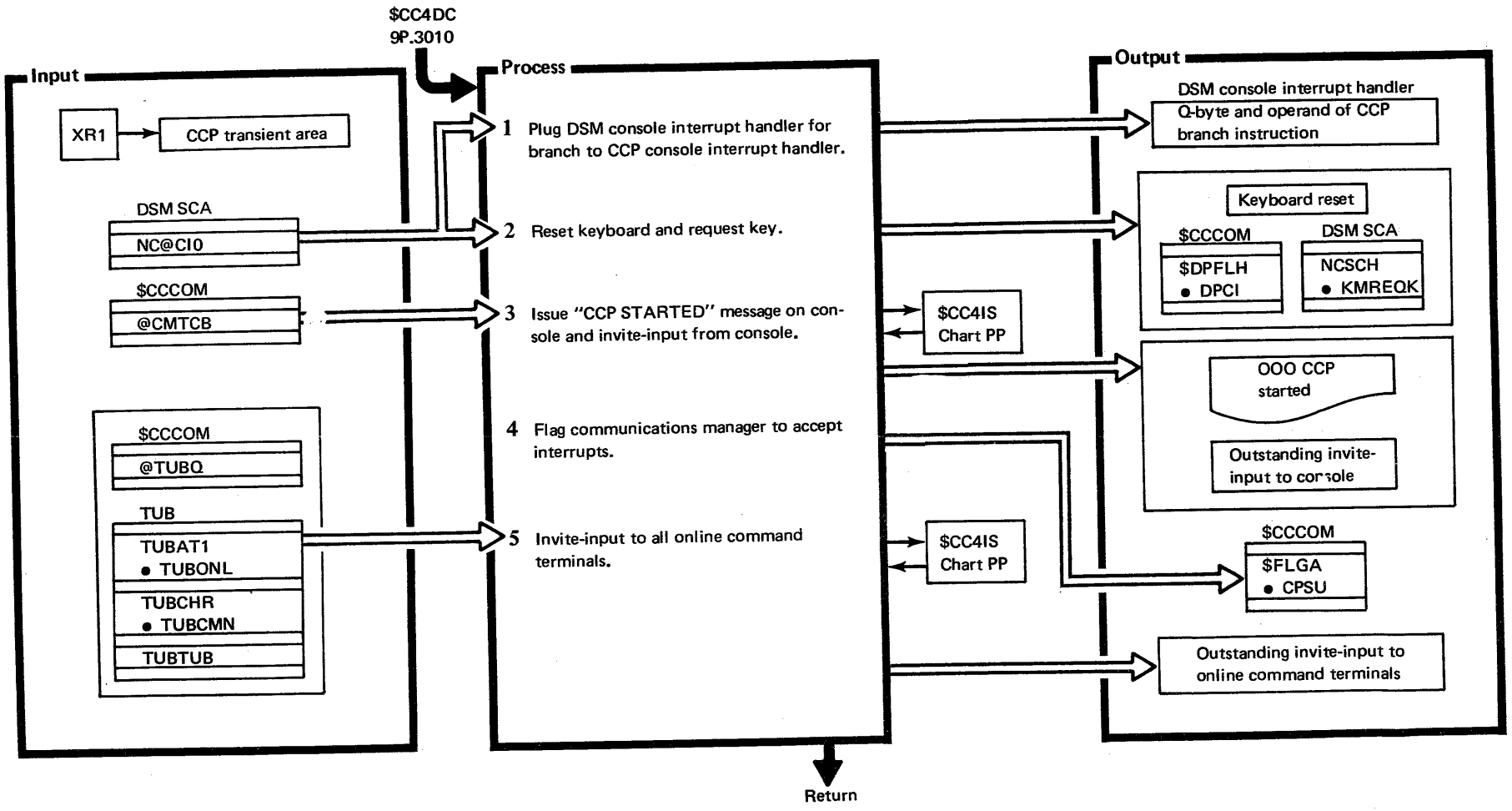
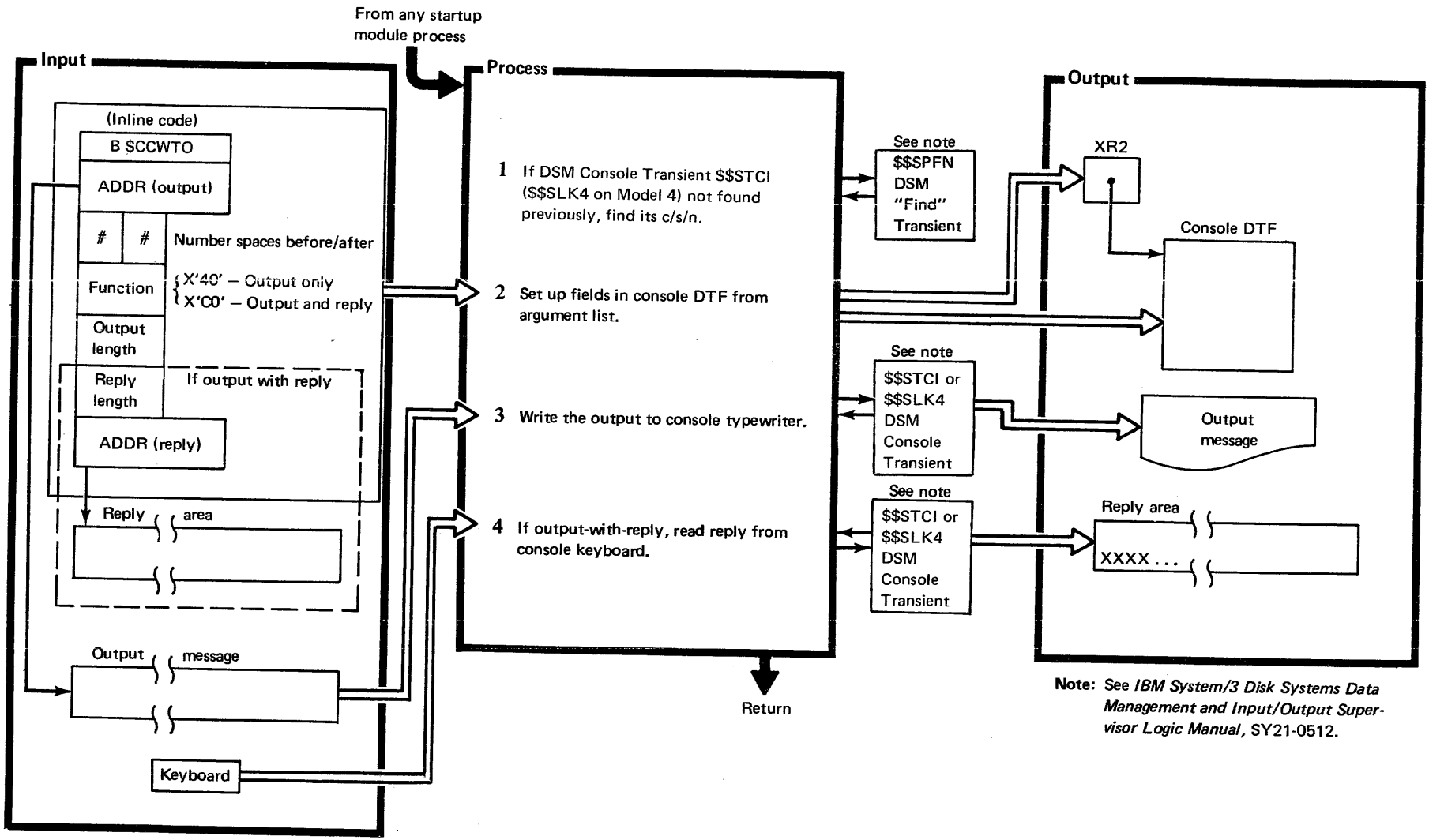


Diagram 8P.0230. $CC4SU



Note: See *IBM System/3 Disk Systems Data Management and Input/Output Supervisor Logic Manual*, SY21-0512.

Diagram 8P.0500. $CCWTO

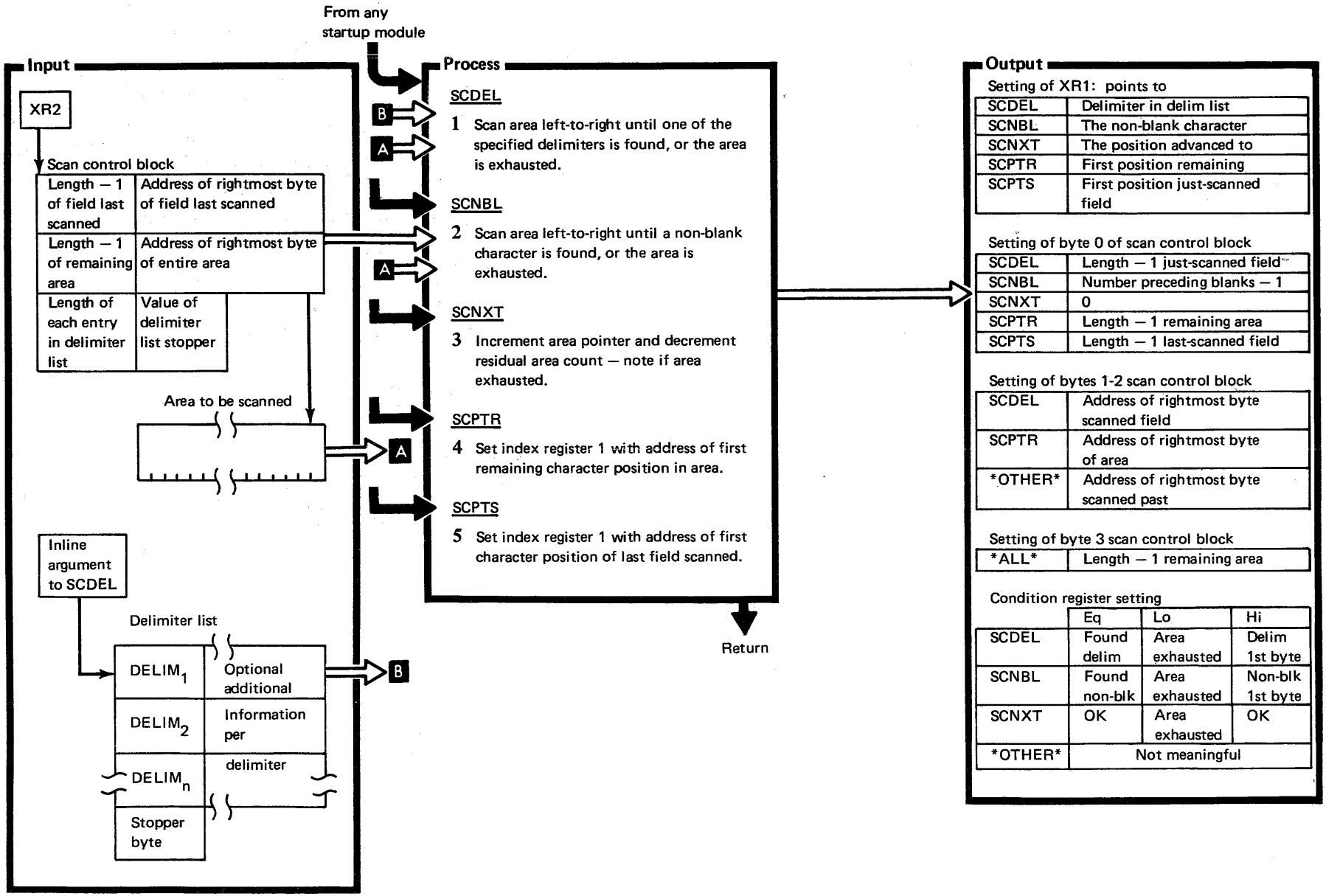


Diagram 8P.0510. $CCSCN

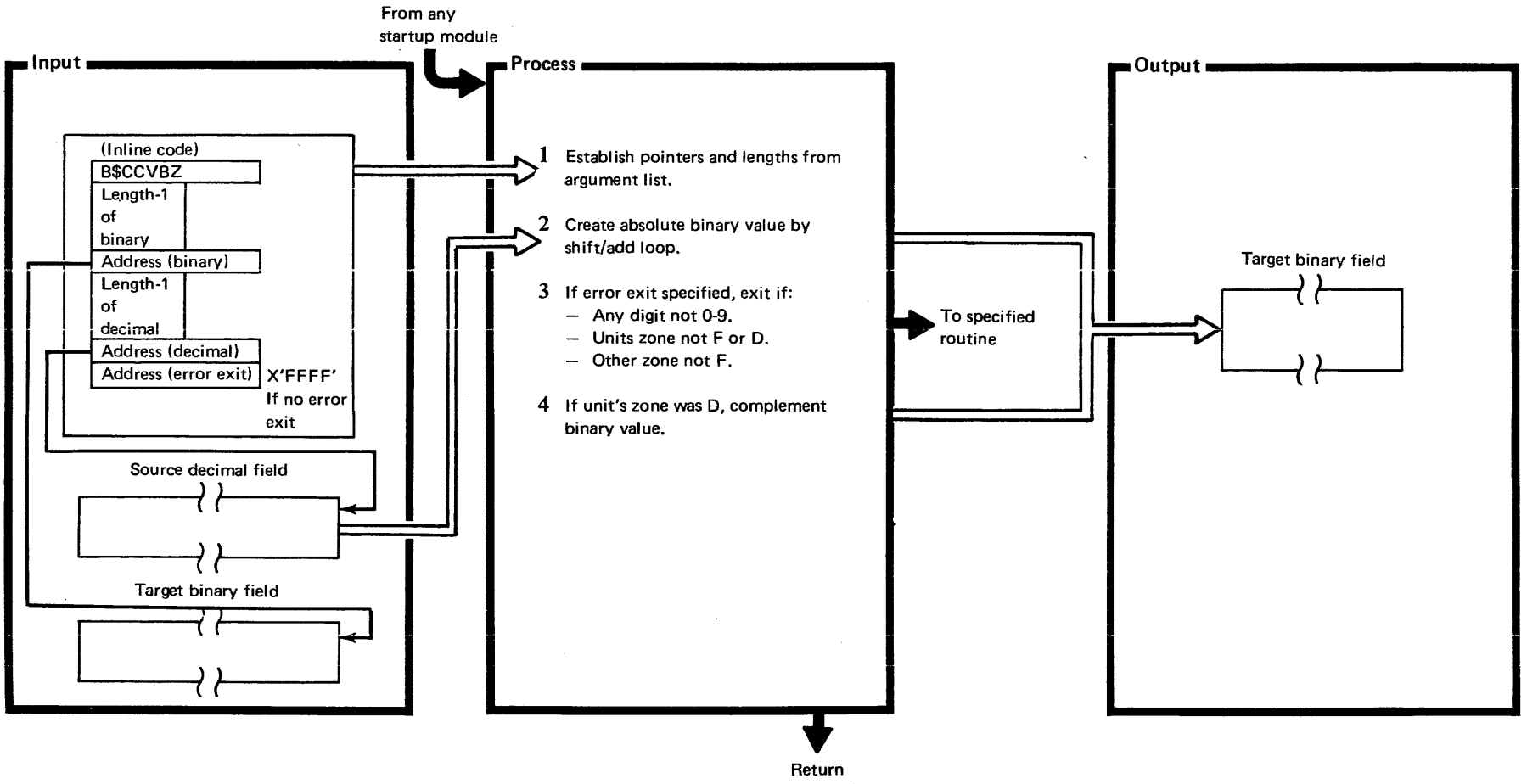


Diagram 8P.0520. $CCVBZ

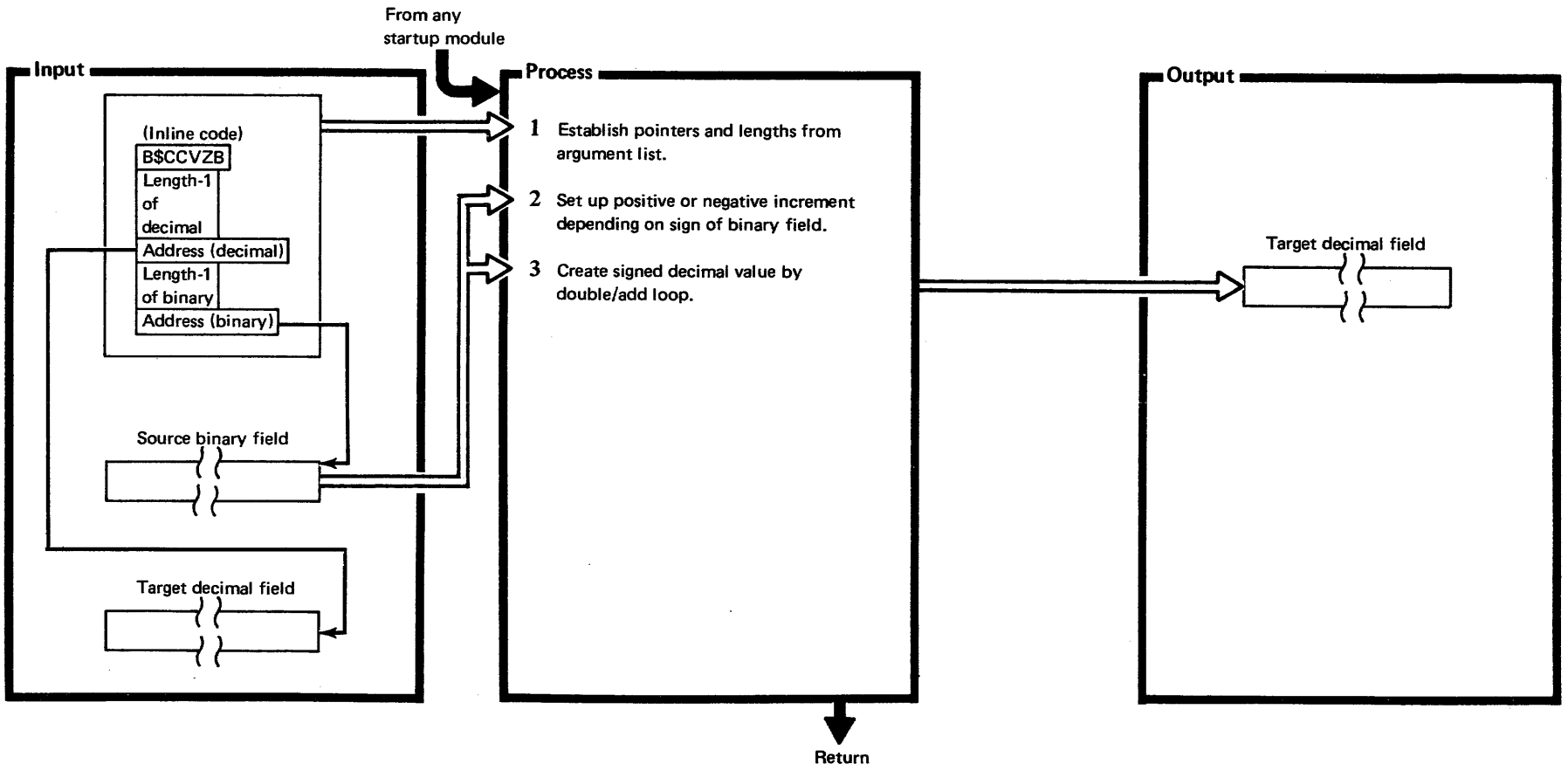


Diagram 8P.0530. $CCVZB

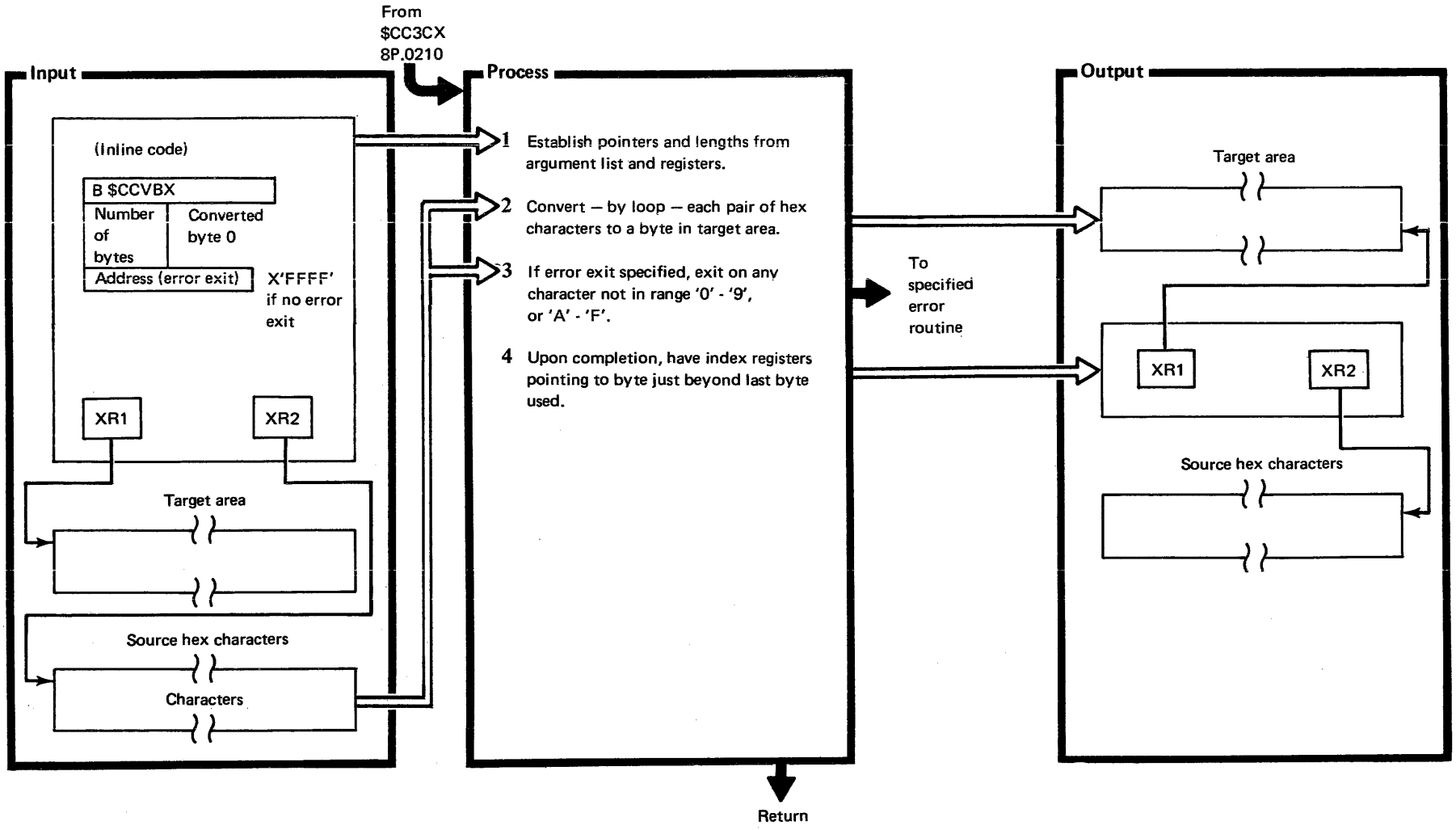


Diagram 8P.0540. $CCVBX

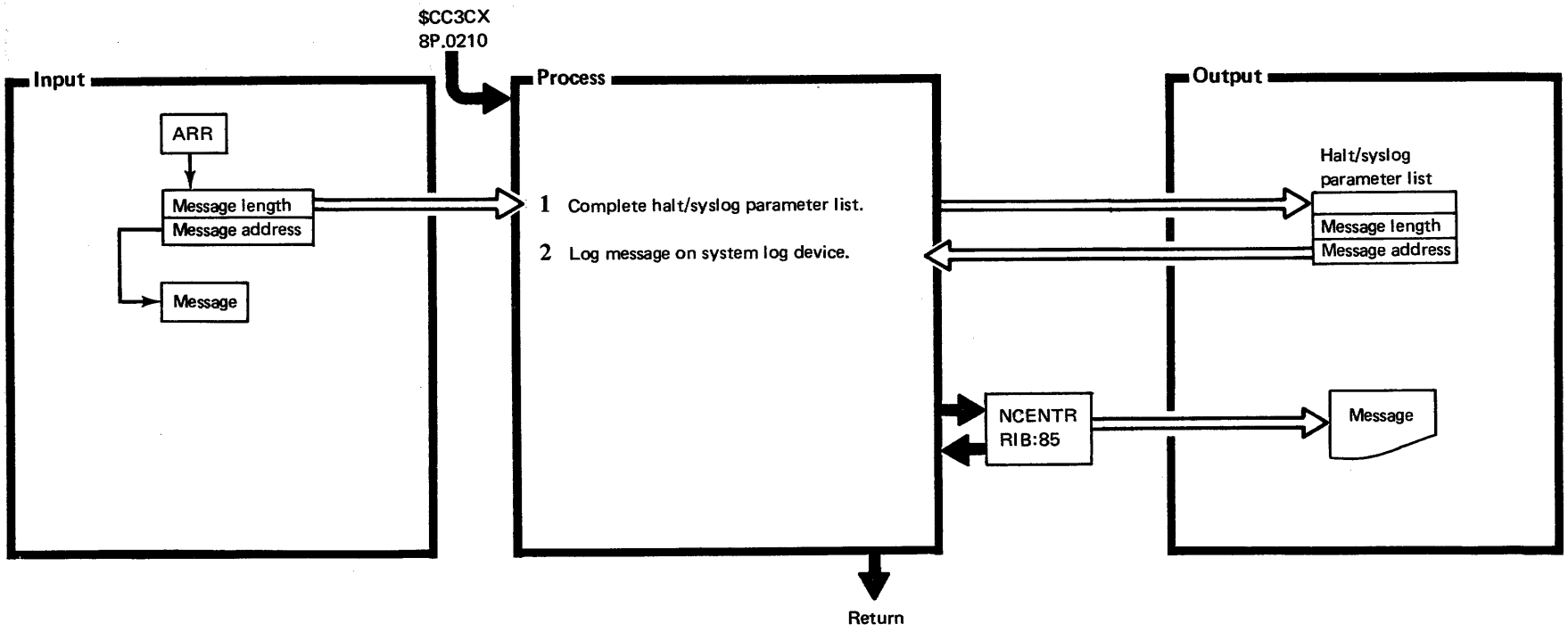


Diagram 8P.0550. $CCLOG

Chapter 9. CCP Execution

Introduction

CCP Execution includes all activities between Startup (Chapter 8) and Shutdown (Chapter 10). CCP Execution also includes the user program interface to teleprocessing I/O operations and the interface to terminal and system operator commands.

The system requirements for CCP Execution are the same as for Startup. That is:

- 5404 Processing Unit (64K of MOSFET memory)
  or
  5410 Model A15 Processing Unit (24K of main storage)--Model 10
  or
  5412 Model B17 Processing Unit (48K FET memory)--Model 12

- 5444 Model 2 Disk Storage Drive--Model 10
  or
  5447 Model 1 Disk Storage Drive--Model 4
  or
  3340 Model C2 Direct Access Storage Facility--Model 12

- 5471 Printer-Keyboard

- 5203 or 5213 or 1403 Printer

- Teleprocessing line (BSCA, MLTA, or local display adapter)

Method of Operation

Because of the broad range of functions included in CCP Execution, this section first presents a structural description of CCP Execution from an external viewpoint, then presents a series of component descriptions beginning with the most basic CCP Execution components. All component descriptions are a high level of detail. The last part of this section contains several typical situations represented in chronological flow of execution functions in order to summarize component relationships.

HIERARCHICAL STRUCTURE

One way to represent the total CCP system is with a pyramid of functions at the top of which is the user program. The highest level system interface for a user program concerned with teleprocessing is the I/O interface portion of CCP $CC4II. This is the function the user first encounters when he evokes teleprocessing I/O or I/O-related functions (Figure 9-1).

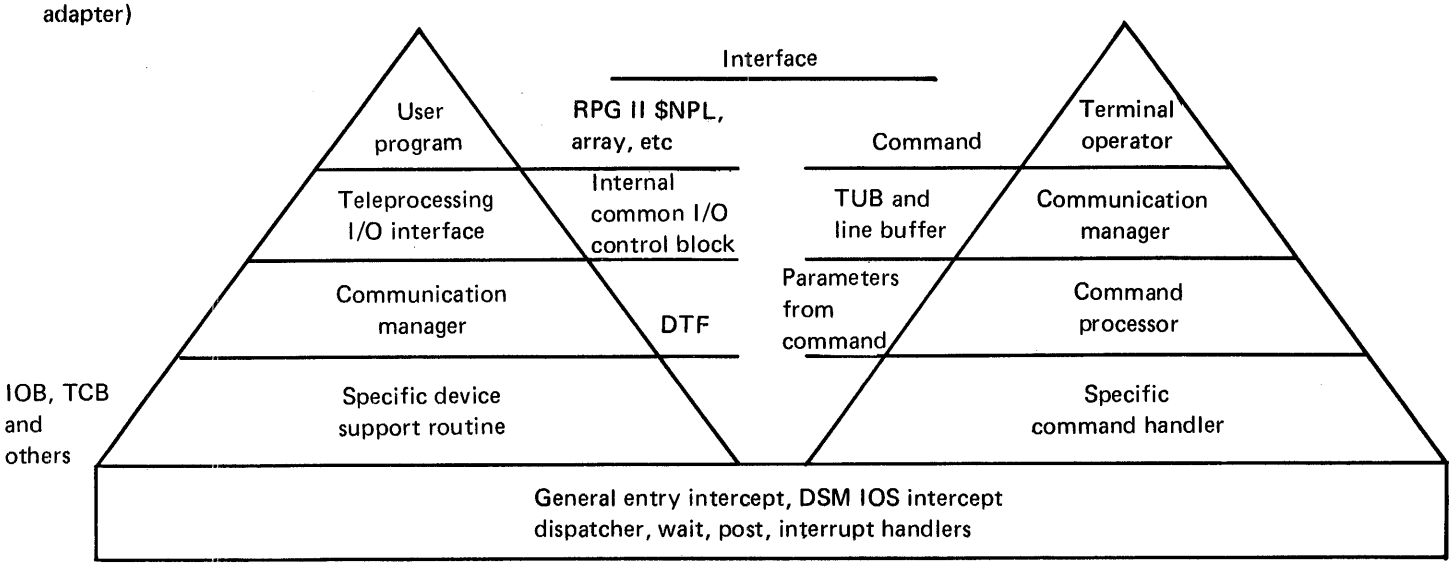


Figure 9-1. Hierarchical Structure

The assembler language user's side of this interface is the $NPL parameter building macro coupled with the $NCIO functional evocation statement. The RPG II user's is the SUBR9x modules, the EXIT/RLABL and/or SPECIAL, and the prescribed ARRAY manipulation. The COBOL user's is CCPCIO interface. The FORTRAN user's is the CCPFIO interface. All these interfaces are functionally similar, that is, they all exist at a structurally equivalent level.

One level further removed from the user is the CCP Communication Management interface, (most users will be unaware of this level). The interface medium is a modified parameter list. The mechanics of the interface are much more subtle, requiring indirect control flow through the synchronizing of independent work units or tasks that may be executing parallel in time. Sometimes the control does not appear to flow at all; the interfaces consist of inter-function communication via a synchronous setting of indicators, values, etc.

Below the Communication Management level of interface exists the teleprocessing data management; this level might be familiar to the users of MLMP or MLTA. The interface is via a DTF. At this same level exists the console data management, functionally similar to DSMs console data management from an interface viewpoint, but actually a function wholly contained in CCP.

At the next level is an integration of basic DSM and CCP control functions. Structurally, this is the lowest level of visibility; that is, there is no user interface below this. Here DSM provides disk I/O and loader services but CCP imposes special routing for certain DSM functions when called from a CCP task. The disk I/O request interface is where record integrity is enforced for sharing of update files by multiple CCP user tasks. New functions exist at this level for CCP use only (main storage allocation and CCP transient area handling are the most notable). Many various parameter types are used at this level: IOB, LOAD parameter lists, the ZERO-RIB, etc.

Beneath all this, is the task control routing function handled primarily by the CCP task dispatcher. The task synchronizing functions are handled by the wait and post functions augmented by an intercept of disk I/O requests. The TP and console interrupt handlers of CCP have additional functions to assist in this tasking process.

Below the lowest visibility level, functions are constant; functions change in the higher levels. The most typical function was outlined (user TP interface). Should there have been a terminal operator command, the levels of function would be different (Figure 9-1). At the first level, the terminal/system operator is unaware that the Communication Manager must first permit, as well as receive, the data from an operator when attempting to instruct the system to perform a given function. When conditions are proper, the Command Processing Task is task switched and the instruction data (a command) is passed in an internal parameter field to the Command Processing function. Once the command type is determined, control is passed to a yet lower level of function to perform the requested action. Beneath that are the same disk I/O, task switching and CCP transient area handling functions discussed earlier.

POST STARTUP RESIDENT CODE

The CCP Post Startup resident code consists of five main storage areas following the end of the DSM Supervisor. (Diagrams 9M.0010-9M.0021 illustrate CCP main storage.)

1. CCCOM (Common Communications Area). The Common Communications Area is a 256-byte area which consists mainly of tables and work areas and is used to provide control information between CCP tasks.

2. CCP Transient areas. The transient areas consist of two 512-byte areas: one for the Communication Task (transient area 2) and the other for non-communication (all other) tasks (transient area 1).

3. Resident CCP. Resident CCP consists of modules, work areas, data areas, control blocks, and other areas. Resident CCP is divided into fixed and variable areas. The following areas vary depending on user specifications at CCP generation:

   a. Modules generated by the user during CCP generation. These modules will vary in content based on the functions the user requires in his system:
      — $CC4DP (Task Dispatcher) will vary depending on multi-tasking, file sharing, dual programming, and minimum system.

- $CC4IH (Common Interrupt Handler) varies depending on MLTA, BSCA, Display Format Facility, and minimum system.
- $CC4WT (Wait) and $CC4PS (Post) vary depending on multi-tasking.
- $CC4IG (General Entry Intercept) varies depending on dual programming and RPG II support.
- $CC4II (User TP I/O Interface) varies depending on MLTA, BSCA, DFF, 1050, minimum system, and remap on Model 12.
- $CC4CM (Communication Task) varies depending on MLTA, BSCA, DFF, device types, minimum system, switched lines, station control, etc.

b. User Security Information Area. This area is generated only if SECURE-USER was specified on the $ESEC statement at CCP generation.

c. Additional modules if the user specifies at generation that he is using BSCA lines and terminals:
   - $CC4IB – BSCA Interrupt Intercept.
   - BSCA IOCS.
   - $CC4BT – BSCA Start Online Test.

d. Additional modules if the user specifies at generation that he is using MLTA lines and terminals:
   - $CC4IM – MLTA Interrupt Intercept.
   - MLTA IOCS.

e. $CC4MS -- Getmain/Freemain Routine. This routine is replaced by $CC4MM if the generated CCP system is a single task system without DFF.

4. A table area of trace routines and tables, control blocks, line buffers, and disk file master indexes. The table area varies according to what the user specifies at Assignment or Startup.

   Note: The master index area shown in Diagram 9M.0010 would result from the user specifying MSTRINDX–YES on the // DISKFILE statement at Assignment time. If, however, the user specified MIXSIZE and a value instead, or specified the // DISKFILE statement or indexed randomly accessed 5444 disk files, following each short CCP DTF (SDF) would be the index area for that particular file and there would be no master index area.

5. Dynamic Telecommunications Buffer area. The size of the Dynamic Telecommunications Buffer area depends on the size specified by the user at Assignment on the system statement MINTPBUF. This value may be changed at startup time.

6. The user program area follows the five CCP main storage areas. The size of the user program area depends either upon the Assignment // SYSTEM statement's MINUPA parameter or an overriding value given during startup.

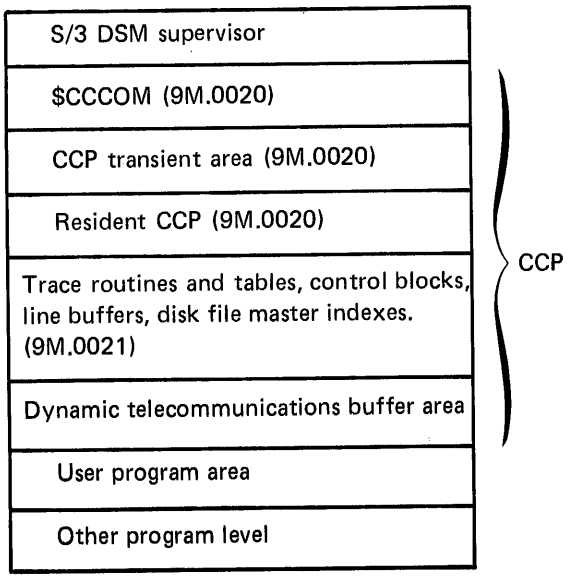


Diagram 9M.0010. Main Storage Layout

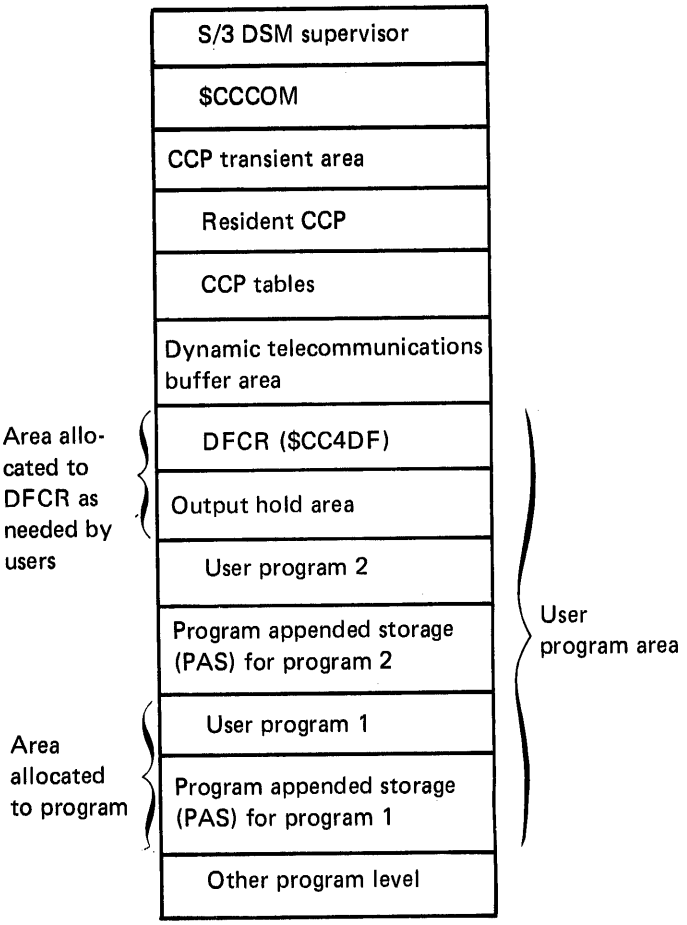


Diagram 9M.0015. Main Storage Layout, DFF

Diagram 9M.0016. Main Storage Layout, PAS

A program appended storage area (PAS) is allocated to each program which uses the services of DFCR.

PAS is composed of 3 parts.

1. Constants and work areas.
2. Terminal table (TT) and format table (FT) entries.
3. Field descriptor table (FDT) entries.

The actual size of a PAS is calculated by the assignment phase $CC2PG and inserted in the PCT in $CCPFILE.

The fact that this item (PCT DFF) is non-zero indicates that DFCR will be used by the program described by the PCT.

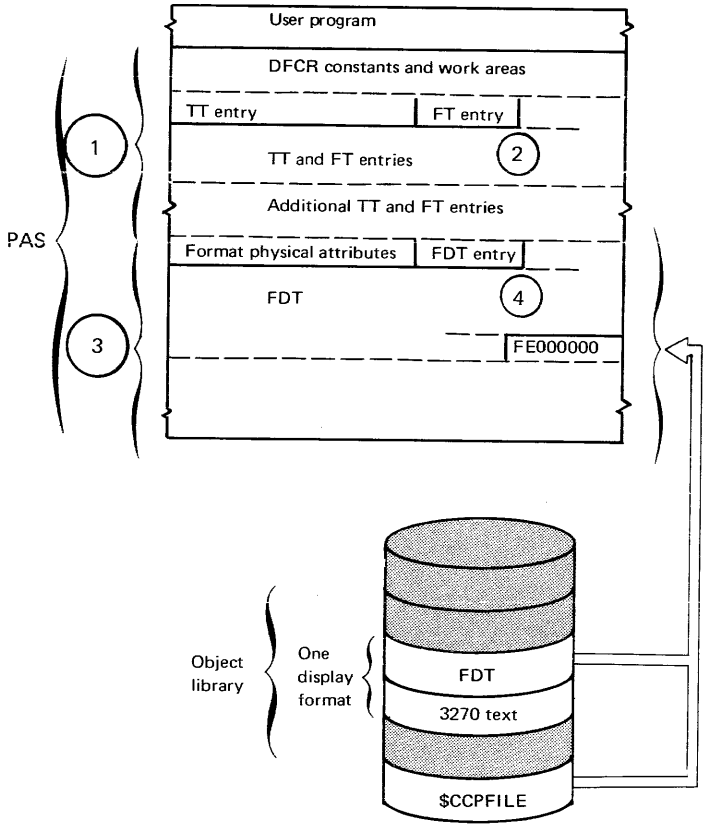
During program allocation, an area will be allocated for a PAS, the address of it will be placed at TCBDFF, and the length of the PAS and the length of the longest FDT is placed in CDEDFF and CDEFDT respectively.

PAS is used by DFCR to build 3270 text, to build up the user's input record properly, and to store all task dependent information, which allows DFCR to be reenterable.

PAS is always allocated on a X'0100' boundary, and is allocated in X'0100' segments.

During start-up, an index of display format names and disk addresses is built and stored in $CCPFILE. The starting location of this index is placed in @DFFIX in CCCOM.

Whenever the user program requests a display format, and an FT entry does not exist already in PAS, this index is read into the FDT area. If the format name appears in the index, the FDT will be read in from the object library into the FDT area (overlaying the index), and an FT entry will then be built.



1. The DFCR constants and work areas, TT and FT entries always requires at least 256 bytes. This allows enough space for 1 TT and 5 FT entries, 2 TT and 3 FT entries, or 3 TT and 1 FT entry. If more entries are required than what will fit in the first sector, additional segments of 256 bytes are allocated.

   The constants and work areas occupy the first part of this area. The TT and FT entries are built dynamically (as the program references them the first time) after the work areas. The constants and work areas are initialized by the allocate transient $CC4DA.

2. Immediately following the work areas will be a TT entry followed by a FT entry. These show which terminal a put message operation was performed to first in the program, and which format was used. Following these two entries may appear other TT and FT entries if the program communicated with more than the first terminal. The TT and FT entries are built as the program executes a put message or copy operation to new terminals with new formats.

3. The field descriptor table (FDT) area always starts on a X'0100' boundary, is a minimum of 256 bytes in length, and should be at least as large as the largest FDT used in the program. (This figure is printed on the format generation listing.)

4. Each FDT entry is 14 bytes long. The very first entry describes the physical attributes of the format. All other entries describe a field contained in the display format.

   The FDT is read into this area from the FDT as contained in the object library. There are 17 entries in the first sector and 18 entries in each succeeding sector. The last 4 bytes of each sector contain X'FE00 00 00'. The first byte after the very last entry contains X'FF'.

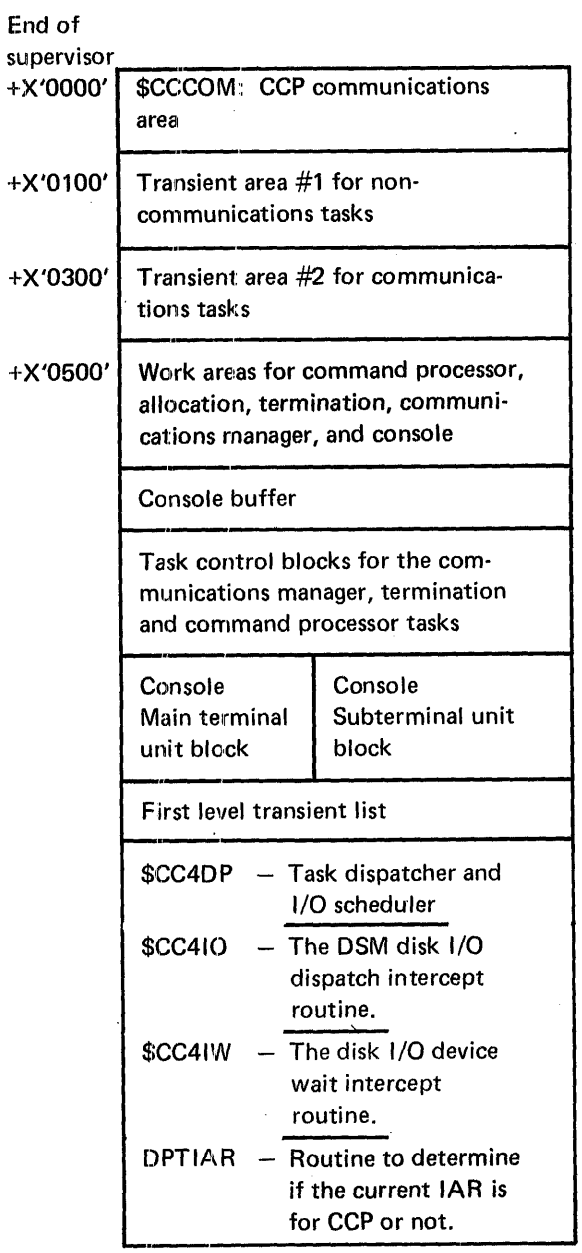


Diagram 9M.0020 (Part 1 of 3). CCP Resident Code

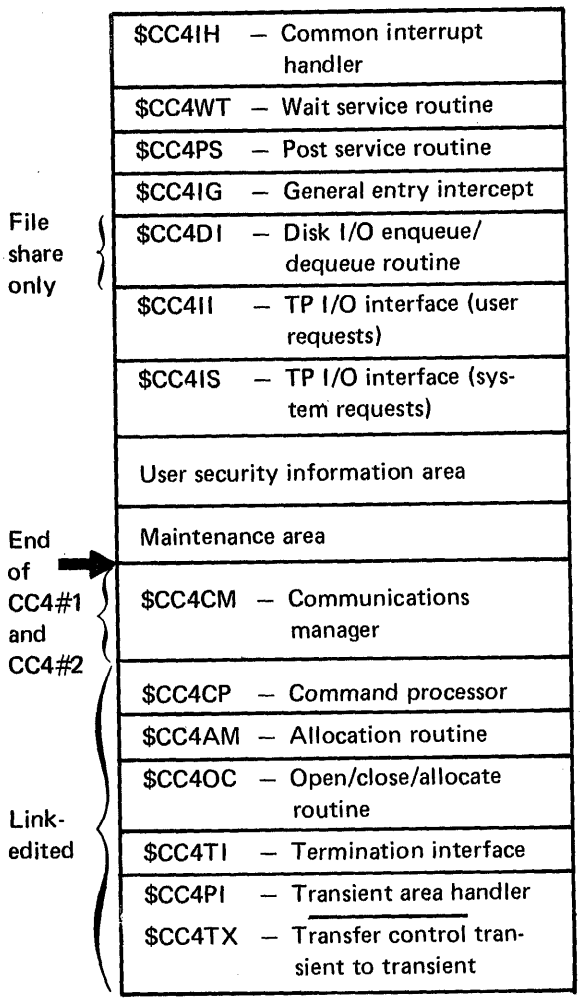


Diagram 9M.0020 (Part 2 of 3). CCP Resident Code

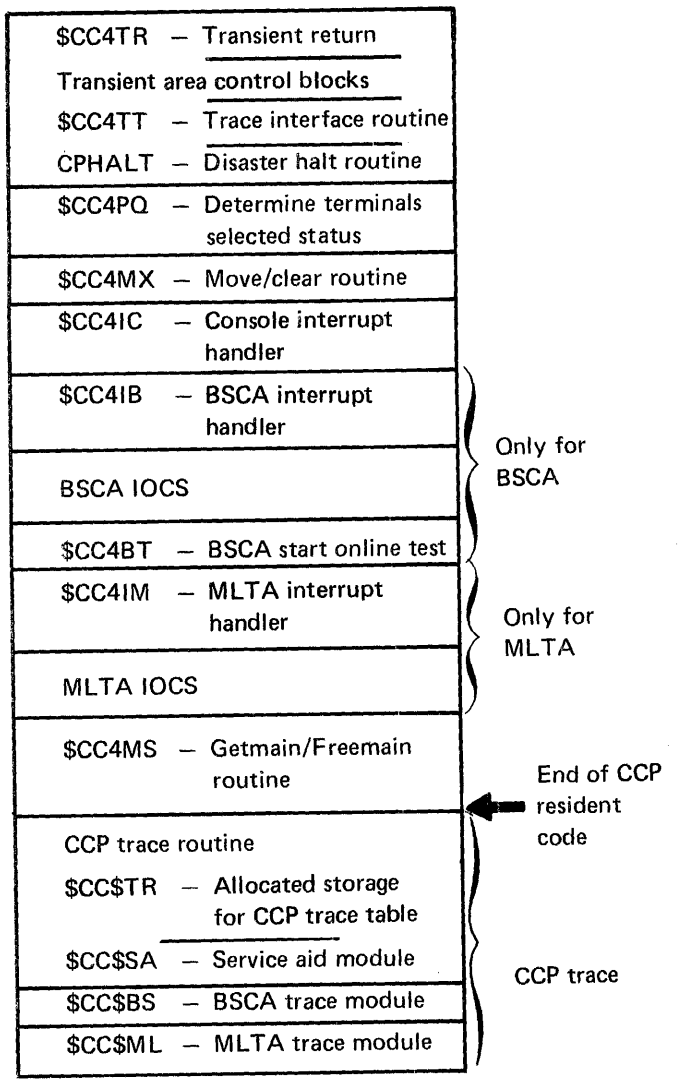


Diagram 9M.0020 (Part 3 of 3). CCP Resident Code

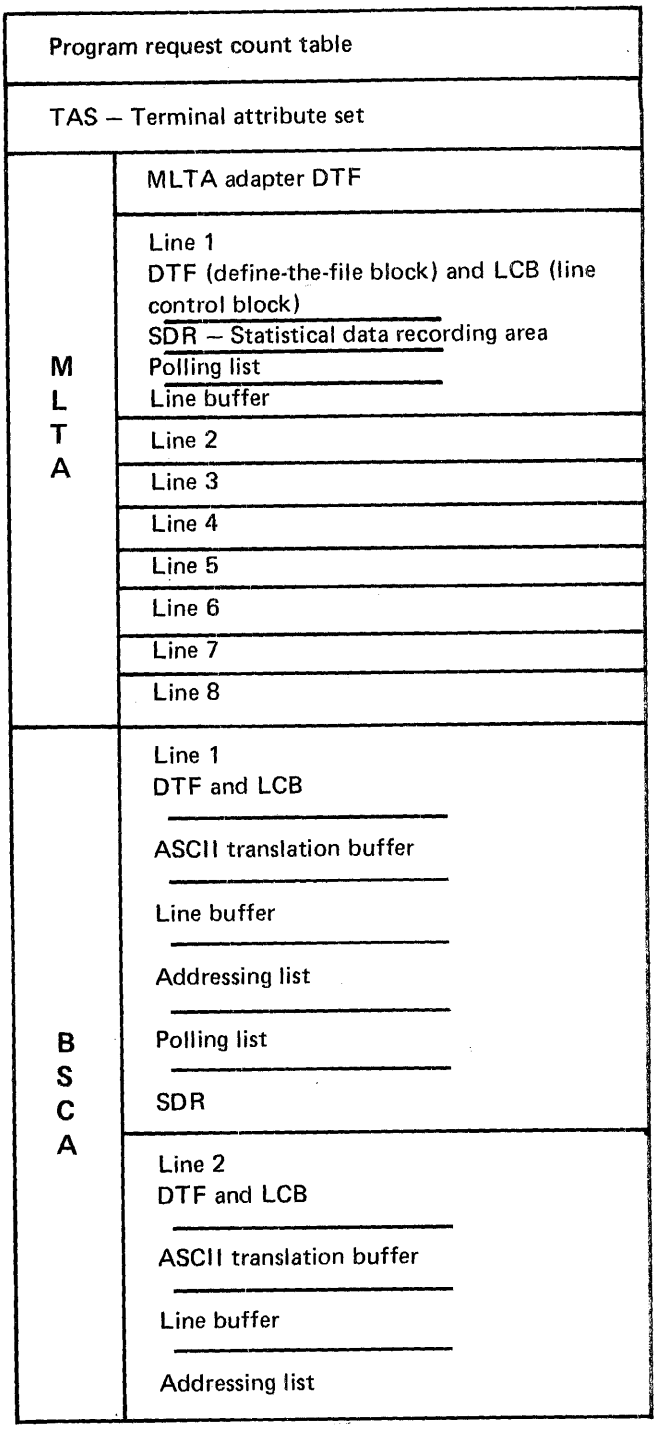


Diagram 9M.0021 (Part 1 of 2). Control Blocks and Data Areas

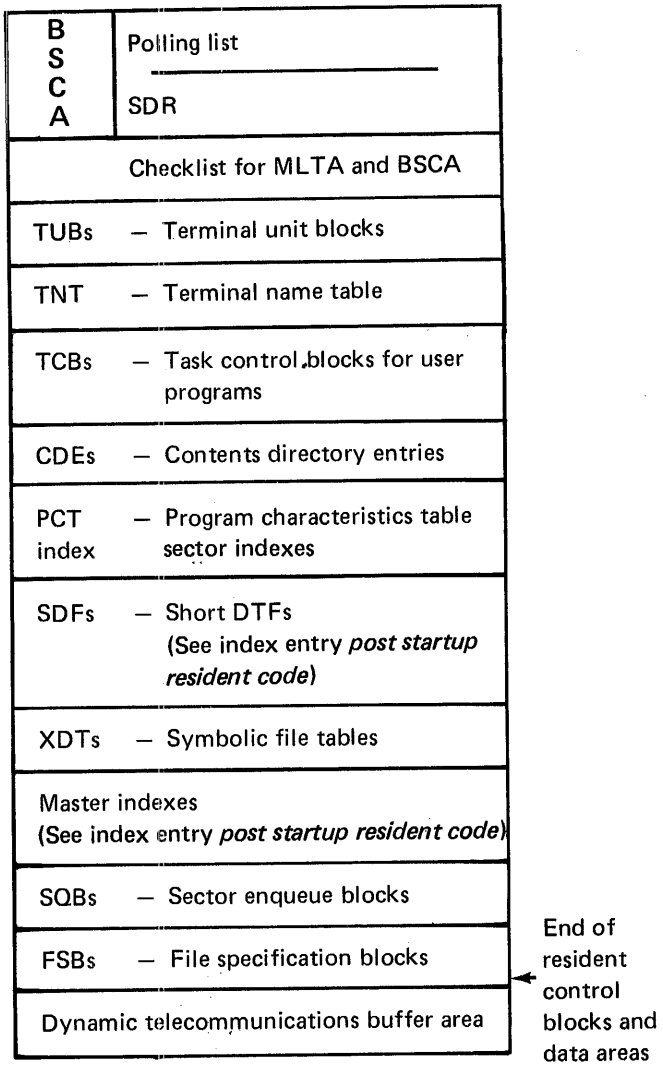


Diagram 9M.0021 (Part 2 of 2). Control Blocks and Data Areas

BASE SCP EXTENSION TO DSM

At the lowest level in the hierarchical structure of CCP (Figure 9-1) are SCP functions that CCP must perform beyond those provided by DSM. These are divided into 3 categories:

1. DSM request intercept (modules $CC4IG, $CC4IO, and $CC4IW). $CC4IG augments DSM General Entry to ensure that DSM functions such as open, close, end of job, and allocate are not invoked by CCP user tasks but that the appropriate CCP substitutes are called. There is also a General Entry RIB to permit a user program to link to CCP functions such as TP I/O without having the entry point address.

2. TP or console I/O interrupt handlers (modules $CC4IB, $CC4IM, and $CC4IC). The interrupt handlers process operation complete conditions when the DSM interrupt handlers (BSCA, MLTA, or console) have completed. This causes the CCP Dispatcher to invoke the communication task so that post-I/O operation logic can be executed.

3. CCP task control (modules $CC4IH, $CC4DP, $CC4IW, $CC4PS, $CC4WT, and $CC4DI – $CC4IG can also be placed in this functional category). Task control is a group of routines which provide a formal mechanism for directing control among user and CCP tasks. Some of the flow direction is indirect, as when $CC4IG and $CC4IW transform a DSM service request or wait into a CCP wait. All of these functions tie into $CC4DP, the Task Dispatcher. Diagram 9M.0030 depicts functional flow among the functions. Logic flow is found in the program organization section. The General Entry intercept logic flow (Diagram 9M.0040) is given to show the various paths the logic can take.

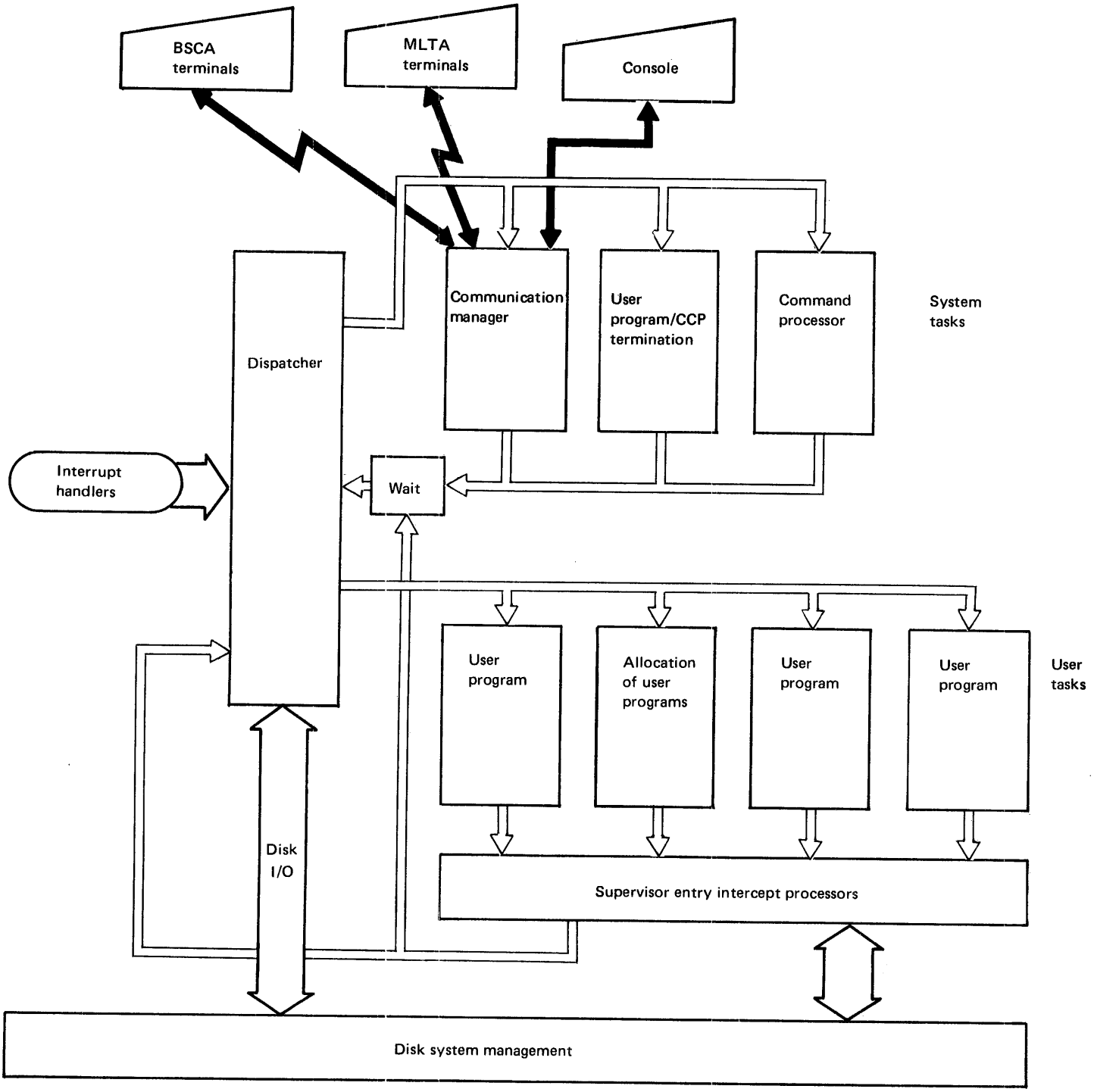


Diagram 9M.0030. Gross Flow of DSM Extensions (Models 8, 10, and 12 Only)

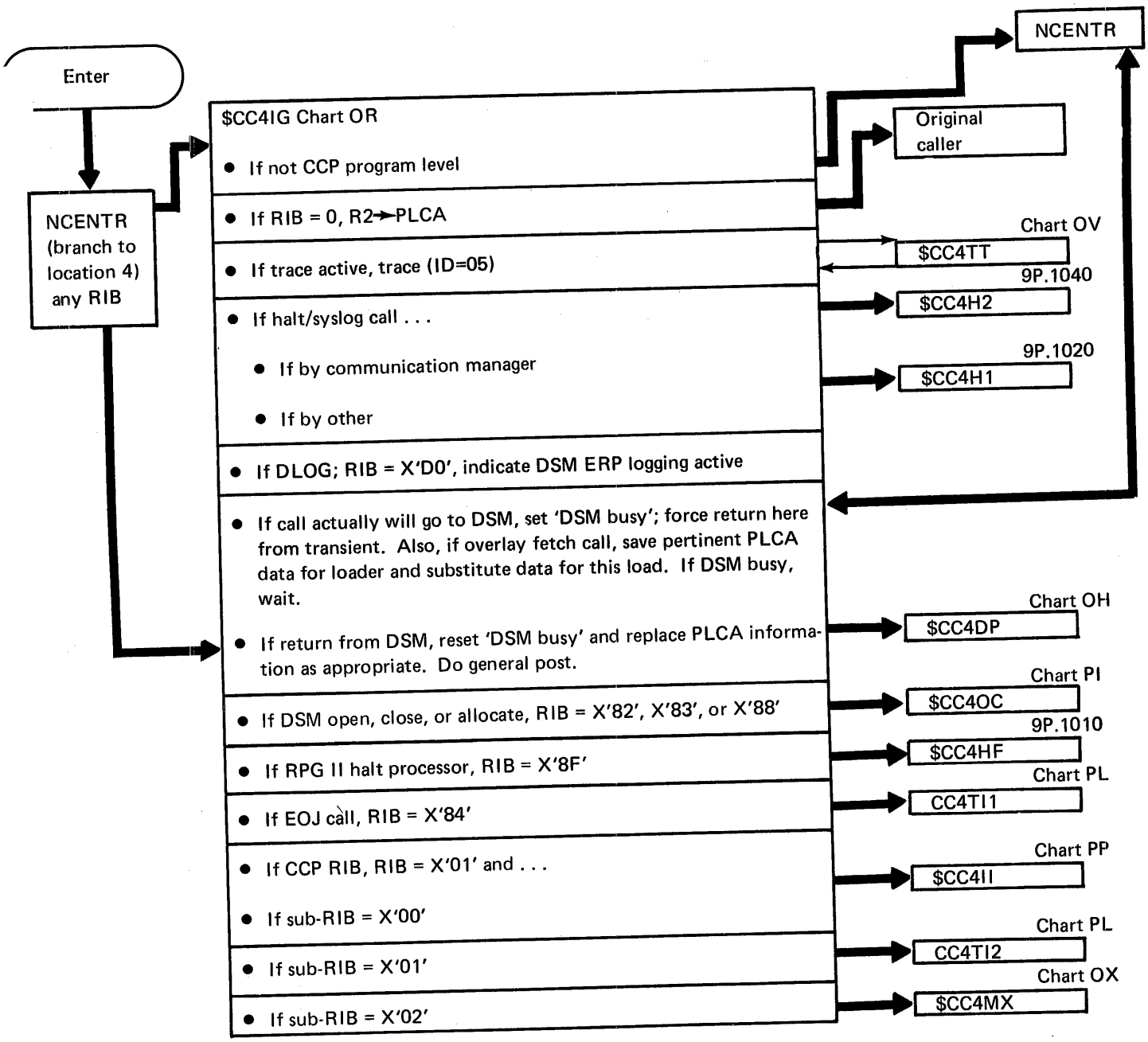


Diagram 9M.0040. Intermediate Flow of General Entry Intercept ($CC4IG) (Models 8, 10, and 12 Only)

Getmain/Freemain

CCP has need of the dynamic storage acquisition function, Getmain, and the necessary counterpart, Freemain (Diagrams 9M.0041-9M.0046). The first basic use is for physical TP I/O buffers which are allocated only on an as-required basis. The second use is for allocation of user program area. See index entry *Getmain by $CC4CM* to find the algorithm for determining the storage allocated in different situations.

User main storage requirements are obvious. The subtle aspect is the user of restrictive Getmain to attempt to place a program on a user area boundary, to force placement of a never-ending program, or to force placement of the DFCR ($CC4DF) at either extreme of the user program area. This capability prevents programs from inadvertently fragmenting execution storage. See detailed flow of the allocation manager for more specific information.

The dynamic TP buffer pool relates to the MINTPBUF parameter of the // SYSTEM statement in assignment. User main storage is derived from the MINUPA parameter also on the // SYSTEM statement.

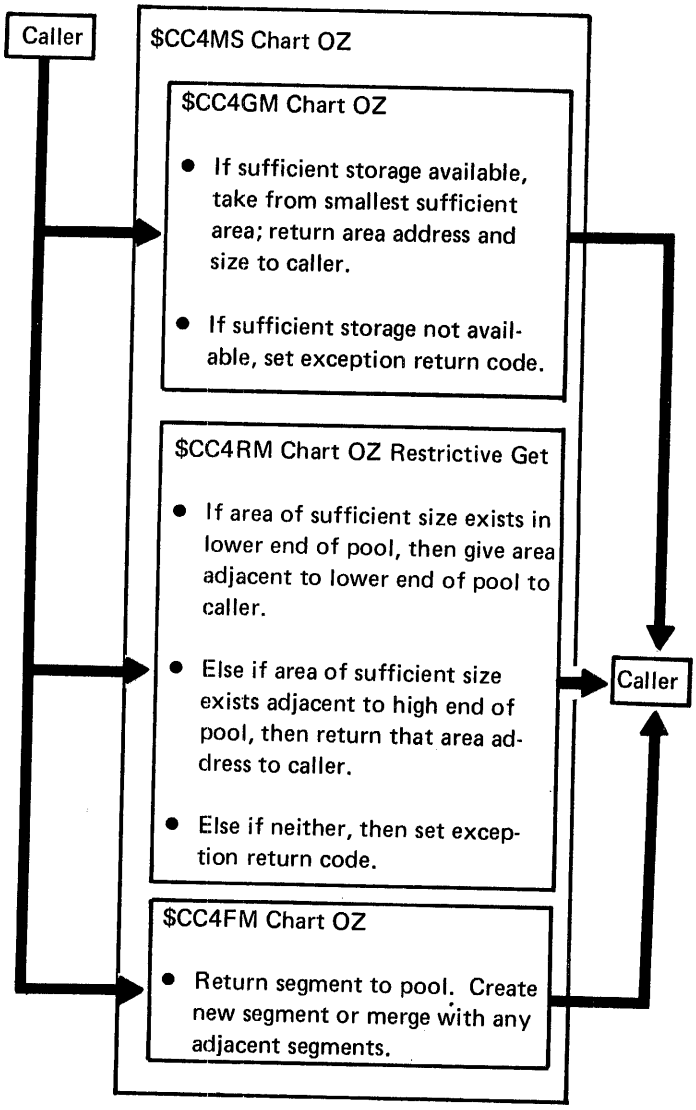


Diagram 9M.0041. High Level Flow of Getmain/Freemain

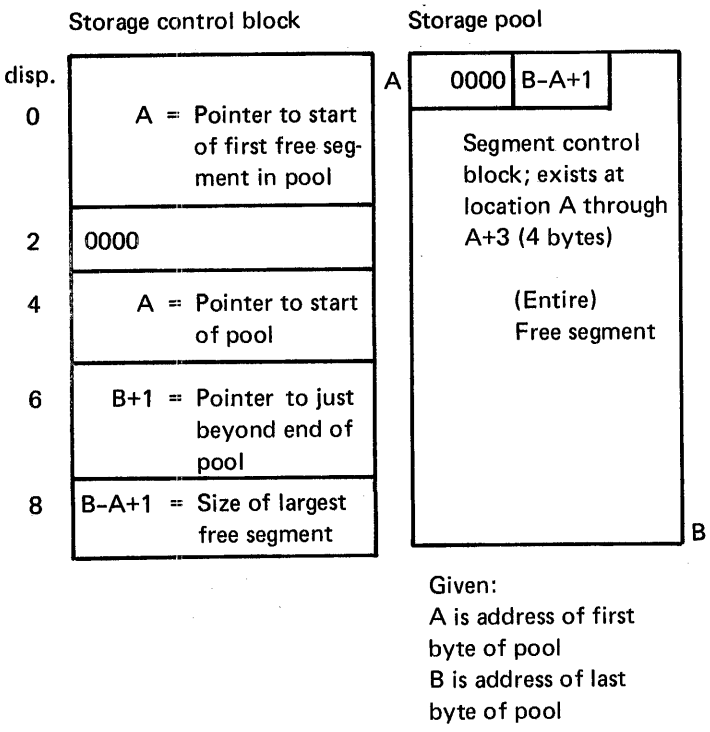
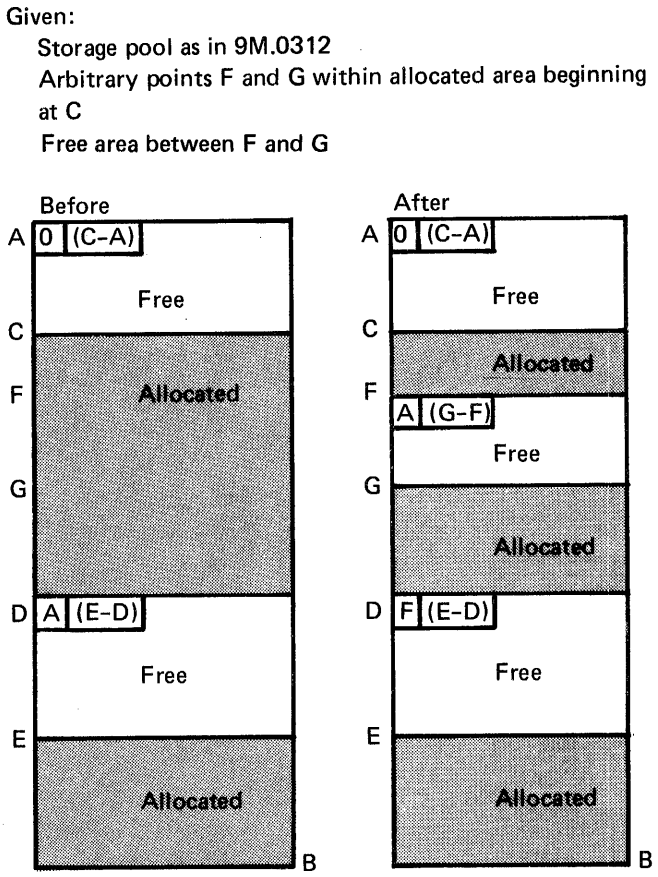
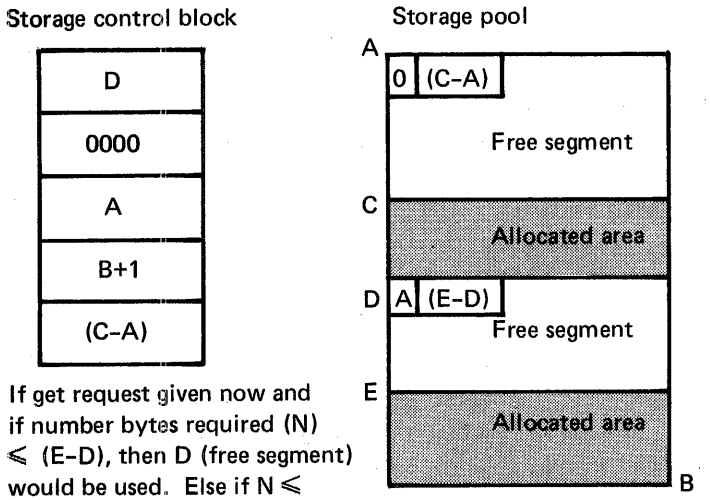


Diagram 9M.0042. Example of Storage Pool after Startup — MINTPBUF





If get request given now and if number bytes required (N) ⩽ (E-D), then D (free segment) would be used. Else if N ⩽ (C-A), then segment A would be used. A restrictive get of size N would work only if N ⩽ (C-A). Should (B-E) ⩾ N > (C-A), then segment E could be used.

Diagram 9M.0043. Example of Storage Pool after Storage Allocation

No change to storage control block unless (G-F) > (C-A)

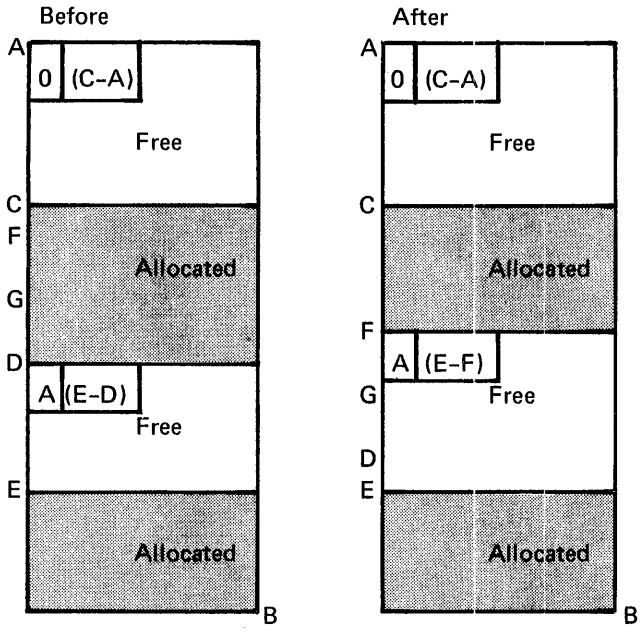
Diagram 9M.0044. Example 1 of Segment Changes after Freemain

Given:

Storage pool as in 9M.0312

Same points F and G as in diagram 9M.0044

Free area between F and D



The first field of the storage control block now contains F assuming (E-F) > B (C-A). The last storage control block field would change to reflect new largest free area.

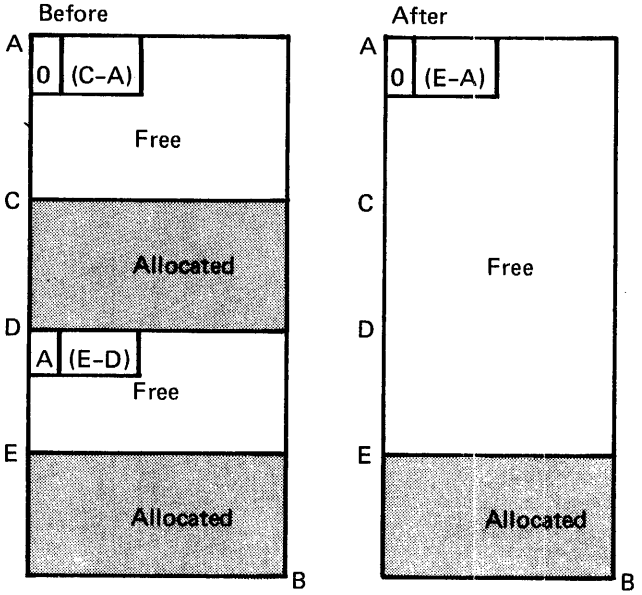
Diagram 9M.0045. Example 2 of Segment Changes after Freemain

Given:

Storage pool as in 9M.0312

Free area between C and D

Note: Segment control block at D is gone and no new one created at C.



The first field of the storage control block now contains A and the last field (largest free area) reflects the update size of segment A; (E-A).

Diagram 9M.0046. Example 3 of Segment Changes after Freemain

CCP Transient Area Handler

To conserve main storage in the CCP component, most of the executable code resides on secondary storage and is brought into main storage only as needed. The modules that reside on secondary storage during execution time are called transient routines. It is important to note that the DSM transient area is in no way involved with the CCP transients. This distinction is maintained because of the potential of interlocks due to conflicts that may occur in dual programming systems and within the CCP tasks themselves. In fact, CCP has 2 transient areas; one is dedicated to Communications Management (task); the other is shared among the remaining components (tasks).

When a CCP transient call is issued, the CCP Transient Area Handler (Diagram 9M.0048) indexes, via a parameter byte, into a Startup-initialized disk-address table (called the First Level Transient List) in much the same way as the DSM transient area handler used on the address table. Another similarity between the CCP and DSM transient area handlers is that when the desired transient area is in use, a wait is necessary.

Transients exit in one of two ways. They may return to the original caller or they may fetch another transient to overlay themselves by performing a transfer control (XCTL). The XCTL would relate to one DSM transient calling another without-refresh. There is no call-with-refresh function provided in CCP. All XCTLs are done via a disk address which has been set by Startup ($CC3CR) within the calling transient routine.

If the return is to the original caller, the formal post logic will be called so that any other non-communications tasks waiting for the transient area will again be able to contend for it.

Not only are disk addresses initialized in cross-reference tables by Startup, the addresses of all relocatable items are also resolved (by $CC3RT) for transients so that a 'core image' load of these transients is possible. Parameter passing between transients is done via a two-byte XCTL parameter and by resident data areas.

CCP transients are brought into main storage via a direct read of the transient using only the NCEIOS and NCEIOW routines of DSM.

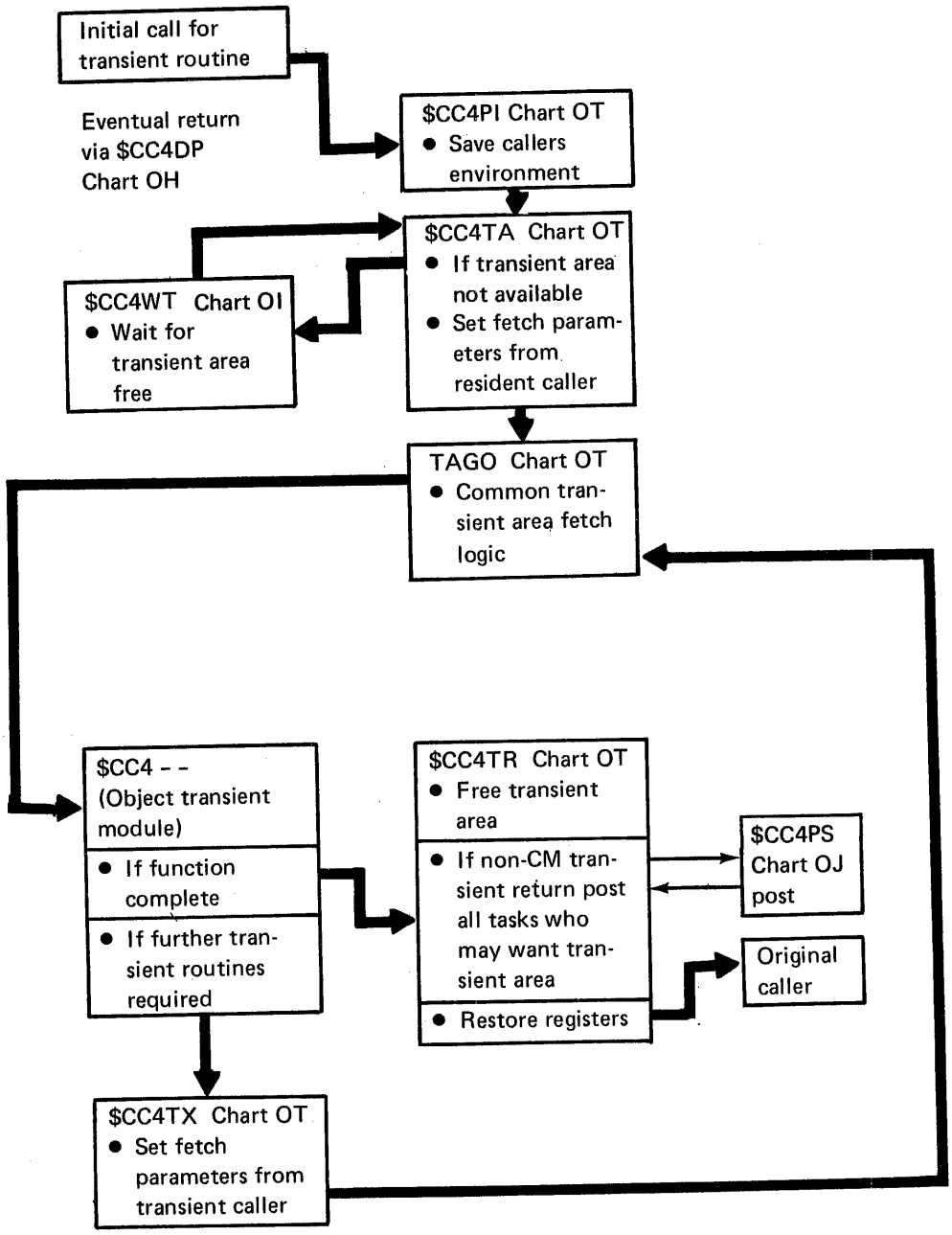


Diagram 9M.0048. CCP Transient Handler

CCP TELEPROCESSING I/O INTERFACE

The CCP Teleprocessing Input/Output Routine ($CC4II/ $CC4IS) handles all user and system task requests for TP I/O operations. This resident module serves as the interface (Diagrams 9M.0050 and 9M.0060) between a user or system task and the Communication Manager (CM).

System Task Requests

For system task TP I/O requests, $CC4IS determines whether the request was for the console or a terminal and posts ($CC4PS) CM that such a request is pending. $CC4IS then branches to $CC4WT to wait until the request has been completed. $CC4IS returns to the caller after the TP I/O request is processed.

User Task Requests

The CCP Teleprocessing Input/Output Routine validates all user task teleprocessing operation codes on a TP request prior to passing control to the module required to process the request.

The three types of user task teleprocessing operations handled by $CC4II are Accept Input, Display Format Facility, and status requests. Other user task TP requests are passed directly to CM. One other significant operation performed by $CC4II is that when a shutdown command has been issued, it sets the return code in the user task parameter list to indicate to the user that the shutdown was issued, thereby enabling the user programmer to recognize shutdown and take any desired action when a shutdown is received.

Accept Input

On an accept input from a user task, $CC4II verifies that an invite input has already been issued and calls the Diagnose and Process Transient $CC4AB to find a Terminal Unit Block (TUB) with input to be processed. If none is found, the Diagnose and Process Transient branches to wait $CC4WT. When a TUB is found, control is returned to $CC4II, which moves the data by calling the Data Move Routine $CC4MX to move the data from the invite input hold area to the user record area. When the move is completed, $CC4II frees the invite input hold area and returns control to the user task.

Display Format Facility (DFF) Request

For a DFF request $CC4II calls the DFF transients. The copy operation and put override operation are handled by special transients $CC4DC and $CC4DB, respectively. The Display Format Control Routine $CC4DF modifies all other DFF operation op codes and handles the data transfer. This data transfer is accomplished by a task switch to CM via the Dispatcher $CC4DP. CM then handles the transfer of every block of data except the last, which is handled by $CC4II. Put initial and put overrides are also handled in this manner. Control is returned to $CC4II when all the data has been transferred except for the last block and $CC4II completes the operation.

Status Request

There are three status requests; all are handled by $CC4II. The requests are acquire terminal, release terminal, and get attributes. These user task TP I/O requests are handled by transients called by $CC4II. Control is returned to the user task once the requested operation is completed.

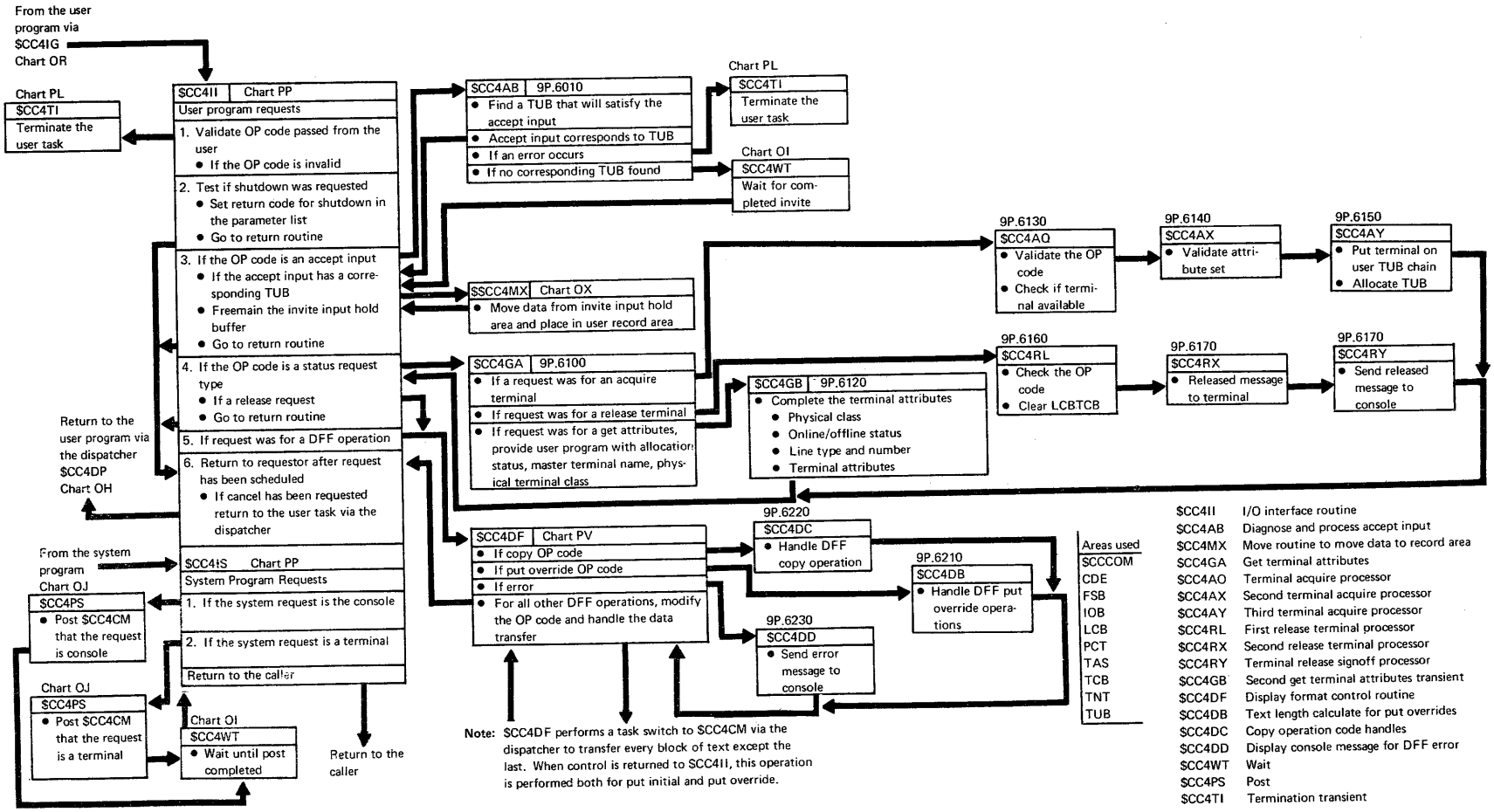
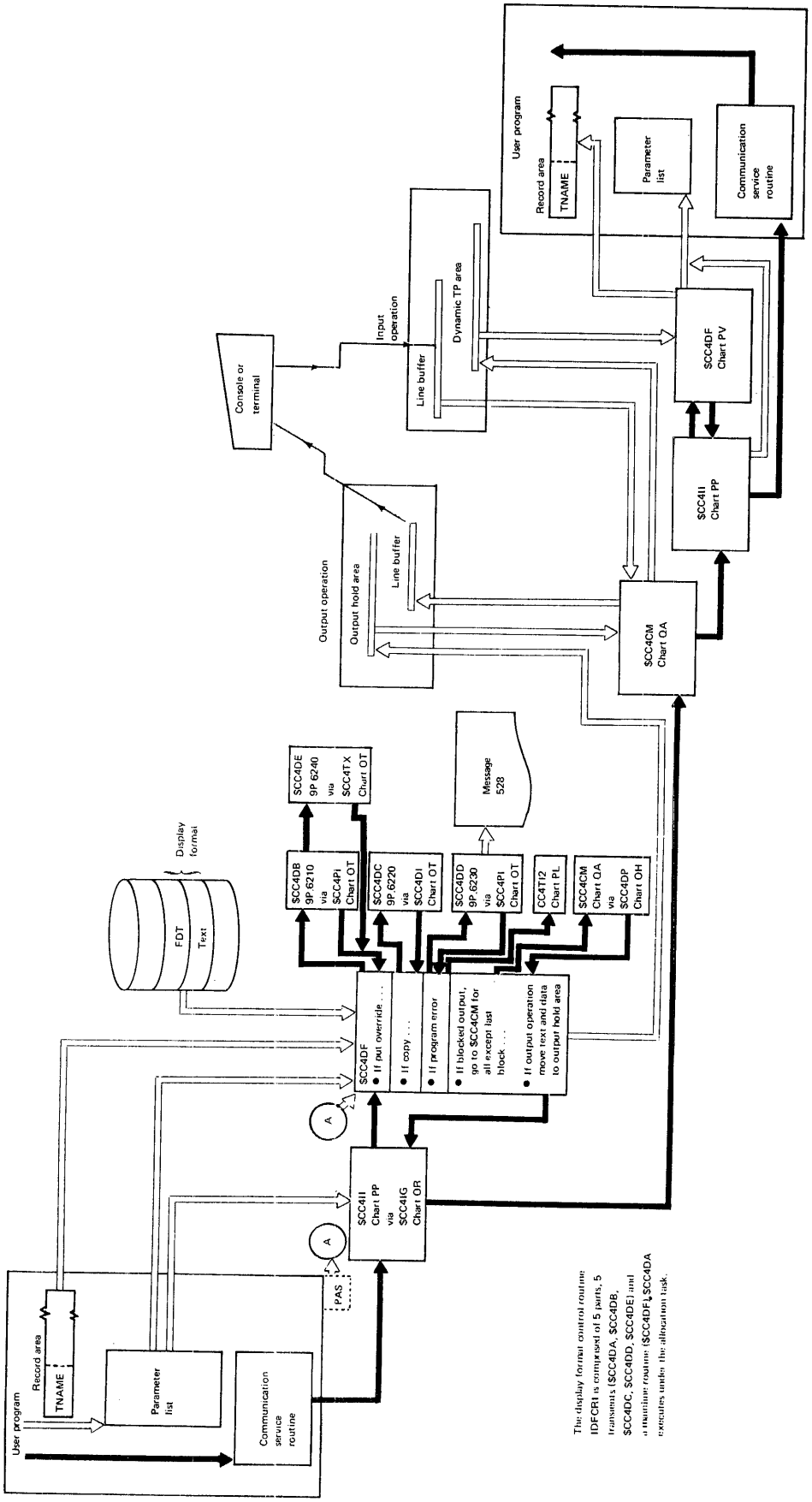


Diagram 9M.0050. TP I/O Interface Flow



The display format control routine IDFCRI is composed of 5 parts, 5 transients ($SCC4DA, $SCC4DB, $SCC4DC, $SCC4DD, $SCC4DE) and a maintainer routine ($SCC4DF). $SCC4DA executes under the allocation task.

Diagram 9M.0060. TP I/O Interface Flow with DFF Emphasized

CCP CONTROL TASKS

Four tasks control the CCP Execution System functions, communications, termination, command processing, and allocation (Diagram 9M.0070).

The Communication Task handles the logical aspect of terminal communications and total control of the 5471 operator's console. Though much of the Communication Task function is transient, a significant amount necessarily is resident code. When the Communication Task detects that data received is system control information (for example, data from a command terminal that is in command mode, initial mode, or command-interrupt mode), the Command Processor Task is given control.

The Command Processor Task sorts out the various commands and when the validity and propriety is confirmed, appropriate transients are called to take action with respect to that command. Almost the entire Command Processing Task is transient.

One of the most significant commands is the request for a program. Several transient routines perform verification of such a request to ensure that the requested program will have the required resources. When appropriate, the Allocation Task may be given control. Its function is to effectively give resources to the user program for whom execution has been requested. This control task has the peculiarity of running as a user task with a user's TCB. The allocation function is complete when all required resources are given, or bound, to the user task.

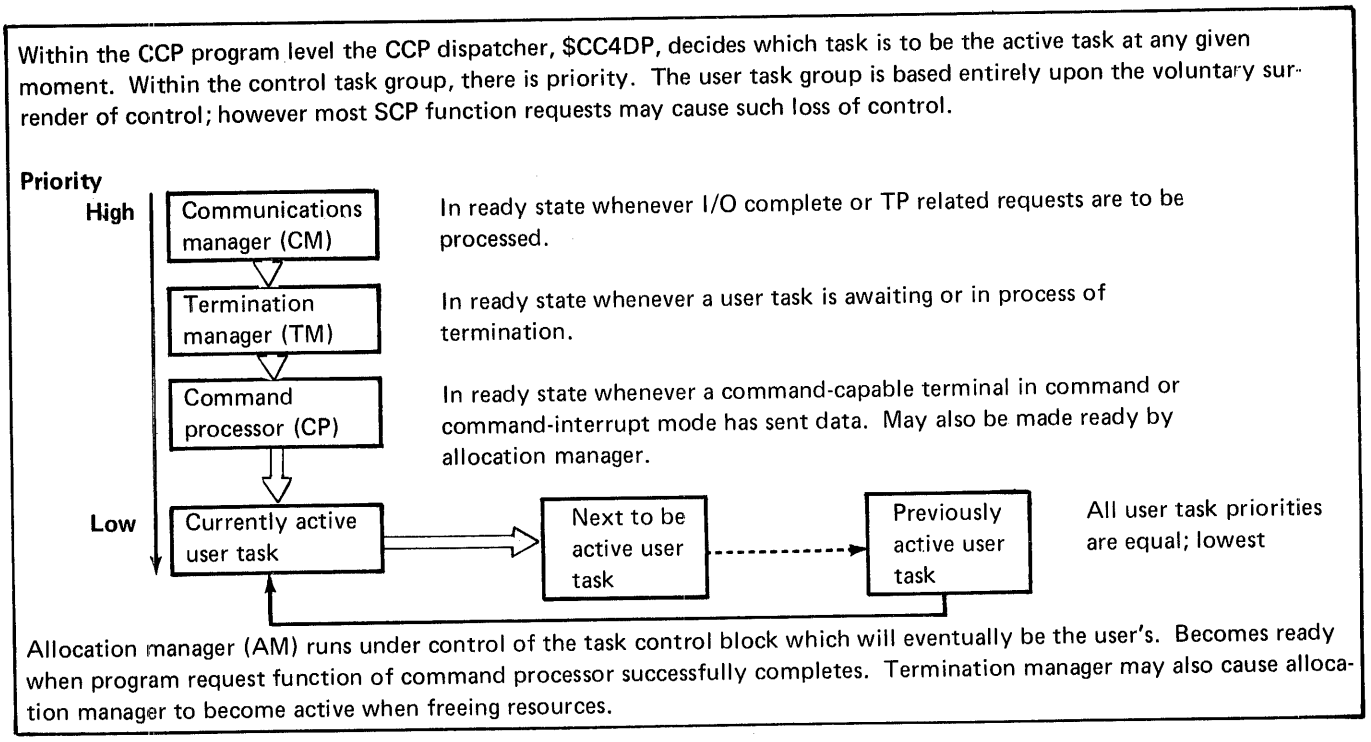


Diagram 9M.0070. Task Control

A user task will run until it decides it has reached end of job. Since the DSM end of job function would terminate CCP if it were called, CCP provides a Termination Task to handle orderly completion of user programs. It works closely with the other CCP control tasks since these tasks often depend on certain resources of whose availability only termination is aware.

The Termination Task is used to purge user programs (tasks) when their processing is completed. This occurs if the program went to end of job or has been terminated by CCP or the system operator.

Termination will free all resources bound by the task and inform Allocation, via Post, that required resources are available. In addition, CCP Termination (Cancel or Shutdown) is initiated by the Termination Task.

Communication Management Task

The Communications Manager routine ($CC4CM) performs all communications input/output functions for the console and for all the terminals defined in the CCP program level. Terminal I/O is performed by direct interface with the communications IOCS (MLMP/MLTA). All scheduling of line activity, data code translation, I/O error detection, and buffer acquisition are performed by $CC4CM.

$CC4CM functions as a system task (with its own Task Control Block) and has the highest priority of all tasks in the CCP system. It is a single task which handles both multipoint and point-to-point lines, both switched and nonswitched lines, both MLTA and BSCA, and both user requests and system requests.

$CC4CM is completely event driven. Only the occurrence of an event elsewhere in CCP which causes $CC4CM to be marked dispatchable will cause control to be passed to $CC4CM.

$CC4CM is marked dispatchable for the following reasons:

- User communications request accepted. This causes $CC4II to post $CC4CM that a communications request is to be scheduled.
- System communications request accepted. This causes $CC4IS to post $CC4CM that a communications request is to be scheduled.
- TP op end interrupt occurred. When a TP interrupt occurs, the IOCS interrupt level service routine gains control. It branches to a CCP interrupt intercept routine that analyzes the interrupt and determines if it was an op end which requires handling by $CC4CM. If so, the count of op end interrupts maintained in the CCP Communications Area is increased by one.
- Console op end interrupt occurred. On all console op end interrupts, the console interrupt routine ($@COIH) branches to the CCP console interrupt intercept routine ($CC4IC) which sets on a bit in the CCP Communications Area. The bit set is DPCI of byte $DPFLH.
- Freemain requested. Some of the functions performed by $CC4CM require the obtaining of dynamic main storage (Getmain). Occasionally the main storage will not be available and $CC4CM will not be able to proceed with that particular function but instead must perform other functions. In this case, $CC4CM waits until a retry of the Getmain can be attempted. Therefore, $CC4CM is posted by another task that a Freemain has occurred, and if $CC4CM is waiting for a Freemain, then $CC4CM will be marked dispatchable.

The Communications Management Task controls communications between both CCP user or system tasks and console or teleprocessing devices. CCP user and system I/O requests are passed to $CC4CM and it issues the necessary requests to the appropriate IOCS or CCP transient. For start-stop terminals, the MLTA IOCS is used. For binary synchronous communications, MLMP IOCS is used. CCP has its own set of communication transients for the console.

A user communication request in the form of a parameter list is initially received by the I/O Interface Routine, $CC4II (Diagram 9M.0080). After $CC4II has caused any Display Format Facility (DFF) reformatting of data and parameter list, $CC4II posts $CC4CM that a TP request is pending.

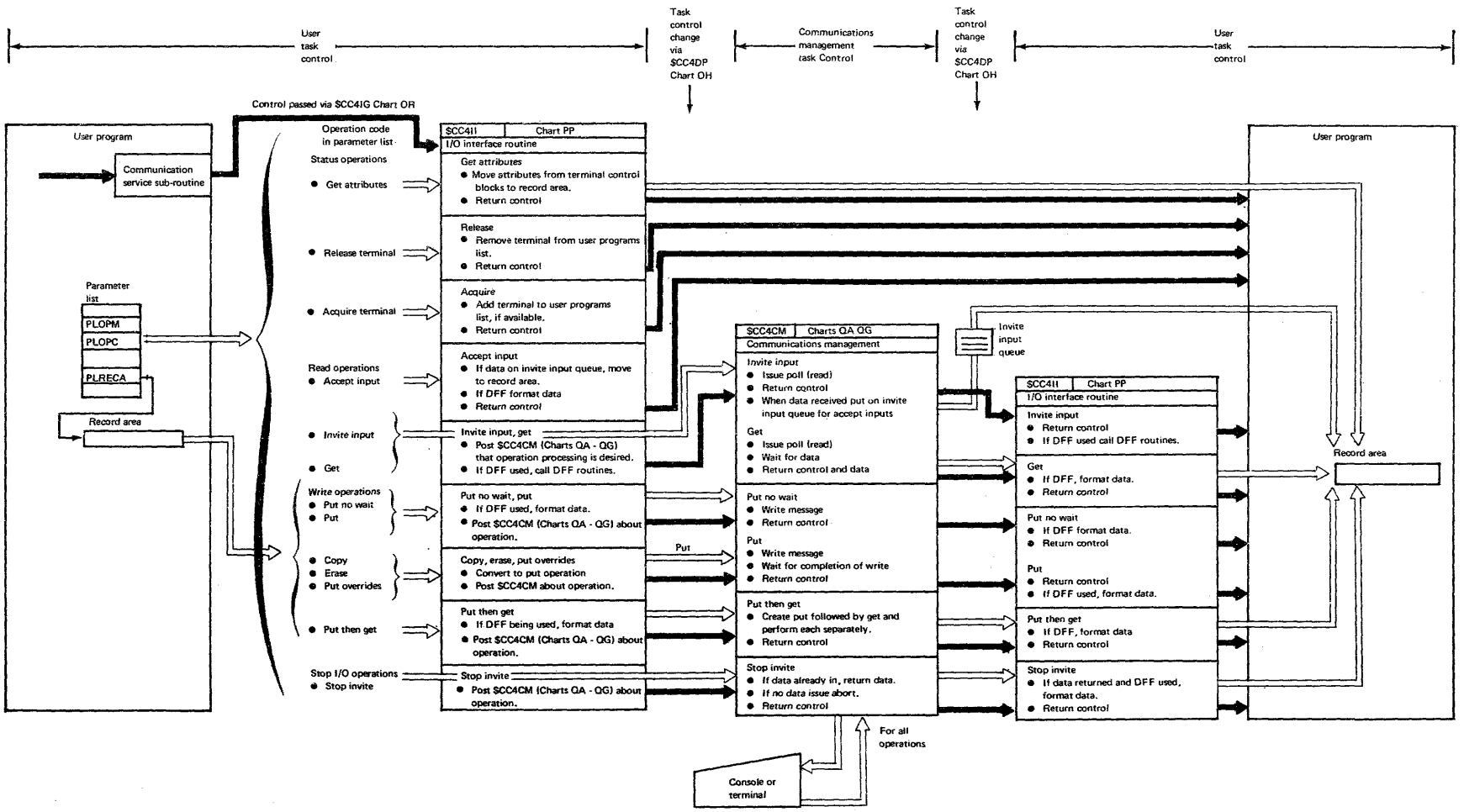


Diagram 9M.0080. TP Operation Handling Overview

Evocation and Input

The valid user communication requests that $CC4II passes to $CC4CM are:

- Get
- Invite input
- Put
- Put-no-wait
- Put-then-get
- Stop-invite-input

For a detailed description of each request, see *IBM System/3 Communications Control Program Programmer's Reference,* GC21-7579.

A CCP system communication request is passed to $CC4IS. $CC4IS passes the parameter list to $CC4CM by posting $CC4CM that a TP request is pending. The following system requests are handled by $CC4CM:

- System invite input, functionally the same as invite input. Input is always translated to upper case, and if data is from a 3270, the message is formatted for the Command Processor. Error Recovery procedures take into consideration that the request was issued by a system task.
- System put, functionally the same as put. If issued to 3270, special formatting is done. Error recovery procedures differ because they are issued by the system task.
- System put-no-wait, functionally the same as put-message-no-wait. If issued to 3270, special formatting is done. Error recovery procedures differ because they are issued by system task.
- System put-no-wait-then-invite, which is system put-no-wait followed by a system invite input.
- System stop invite, functionally the same as user stop invite.
- System purge, which is remove invite, get, or put that is active on the line.
- System put disconnect, which is disconnect switched line.
- System put disconnect-then-invite, which is system put disconnect followed by system invite input.
- MLTA online test, a request to set up MLTA online test DTF and request IOCS to perform test.
- BSCA online test, a request to create system put to satisfy online test request.

In addition to receiving control from the dispatcher due to posts by $CC4II and $CC4IS to process a new parameter list request, $CC4CM is given control by the dispatcher as a result of one of the following conditions:

- A previously started IOCS operation completed. The Communications Manager needs control to do further processing for a parameter list request and to initiate a new operation on the line.
- Main storage previously acquired by a CCP system task has been freed by $CC4FM. The Communications Manager needs control to start a communications request that has been waiting for the availability of more main storage for a buffer.

Processing

$CC4CM control flow is shown by Diagram 9M.0090. The handling of the console is given priority over the handling of all teleprocessing operations. Within the processing of TP requests, the clearing of completed operations and the rescheduling of the line with other previously received requests is given priority over the initial processing of new parameter list requests.

The sequence of processing is:

- Process new console parameter list (Diagram 9M.0100).
- Handle console operation-complete (Diagram 9M.0100).
- Start new output operation to the console (Diagram 9M.0100).
- Handle teleprocessing operation complete (Diagrams 9M.0110 and 9M.0120) and start a new operation for that line (Diagram 9M.0140).
- Search for a request that now can be started because of main storage being freed (Diagram 9M.0130) and start it (Diagram 9M.0140).
- Process new teleprocessing parameter list (Diagram 9M.0130) and start next operation to that line (Diagram 9M.0140).

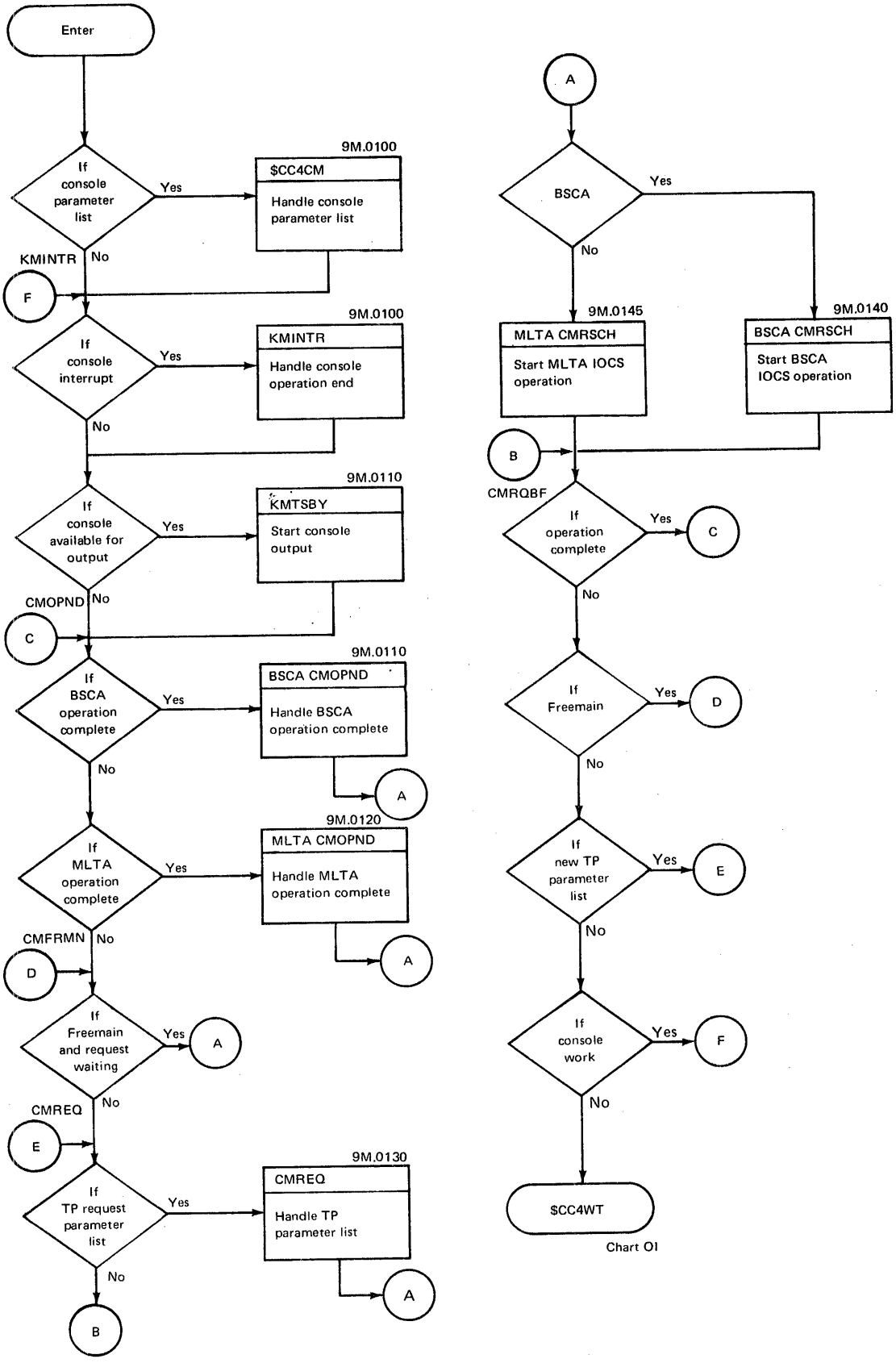


Diagram 9M.0090. Communications Management Overview (Models 8, 10, and 12 Only)

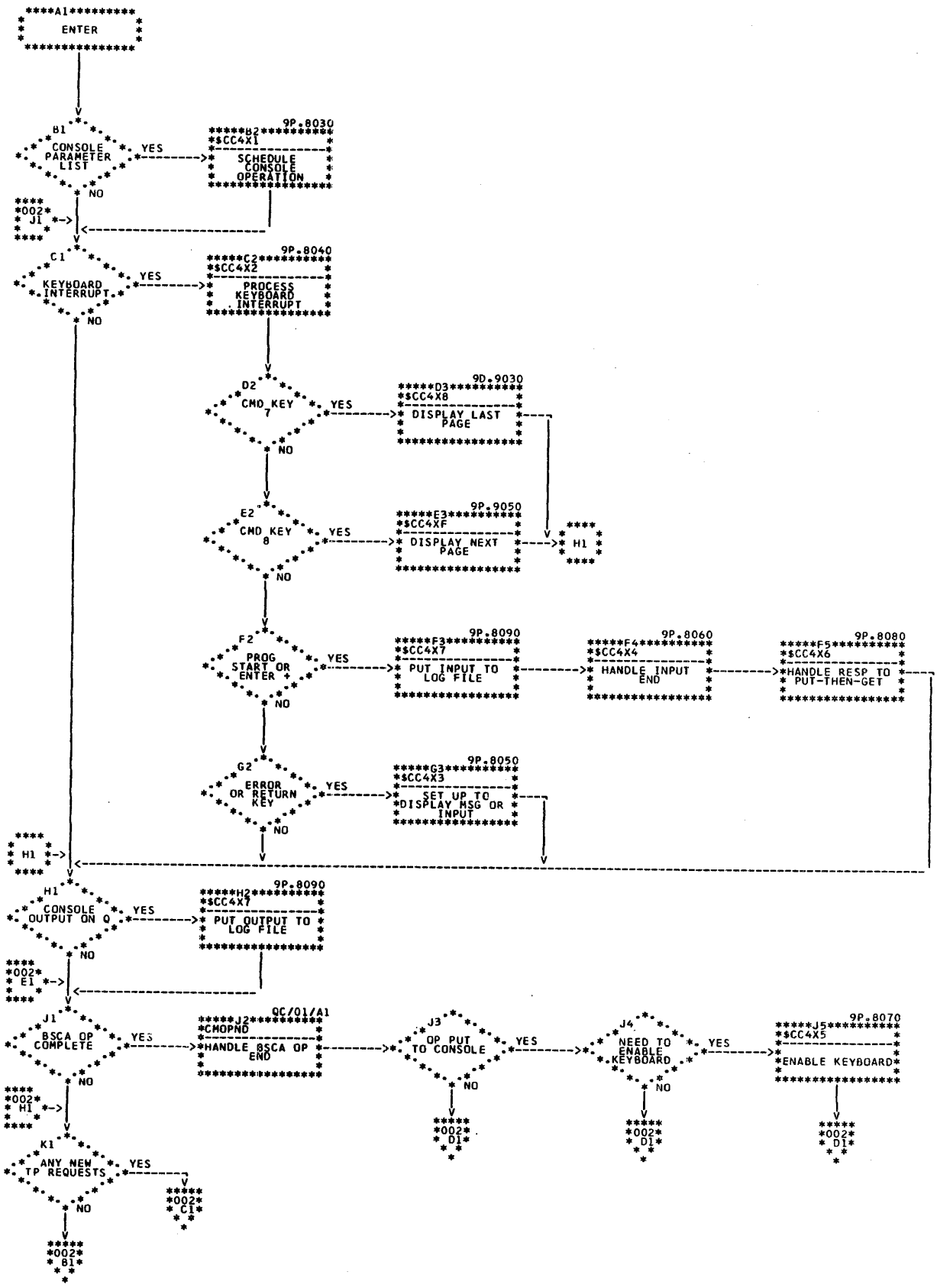


Diagram 9M.0095 (Part 1 of 2). Communication Management ($CC4CM) (Model 4)

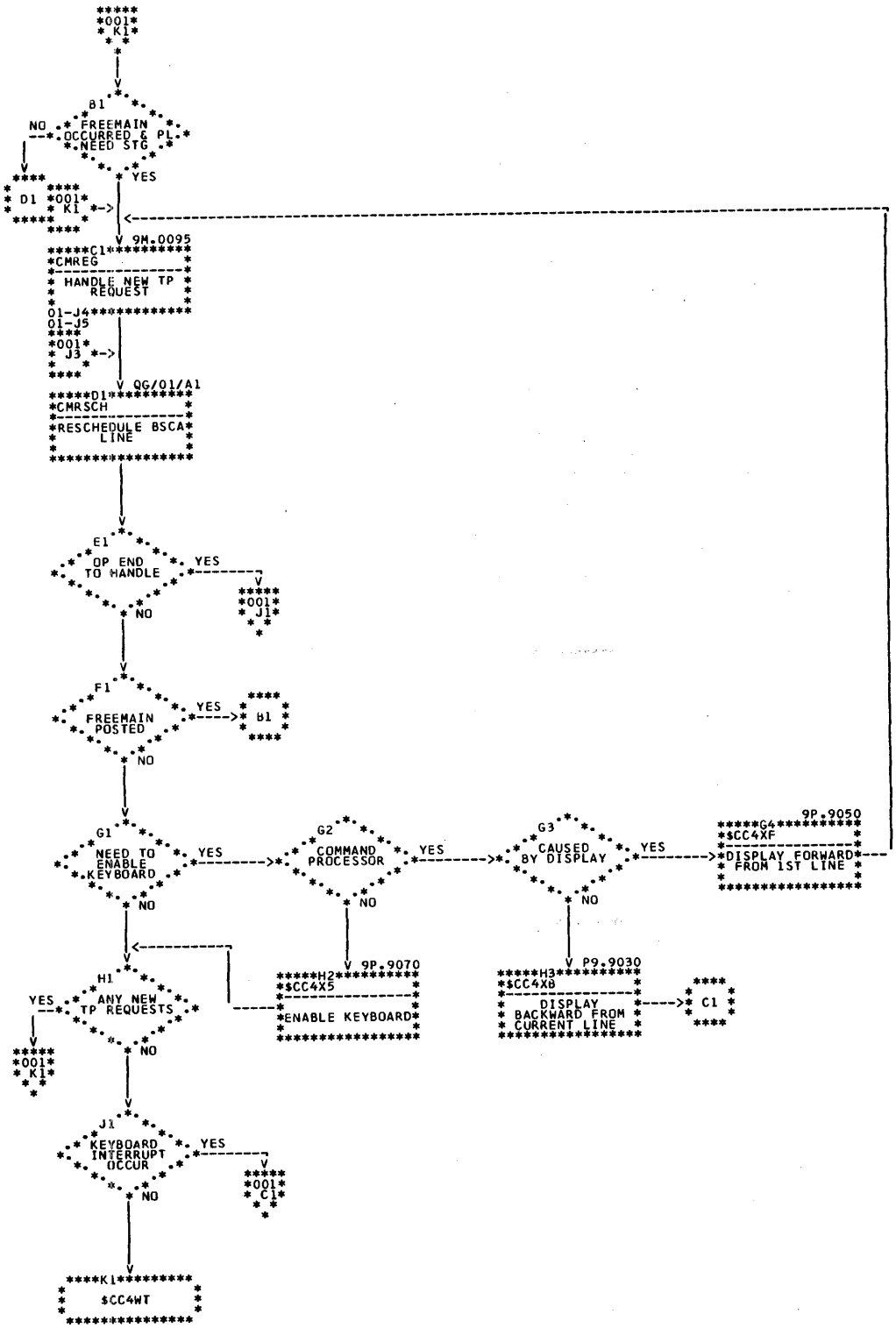


Diagram 9M.0095 (Part 2 of 2). Communication Management ($CC4CM) (Model 4)

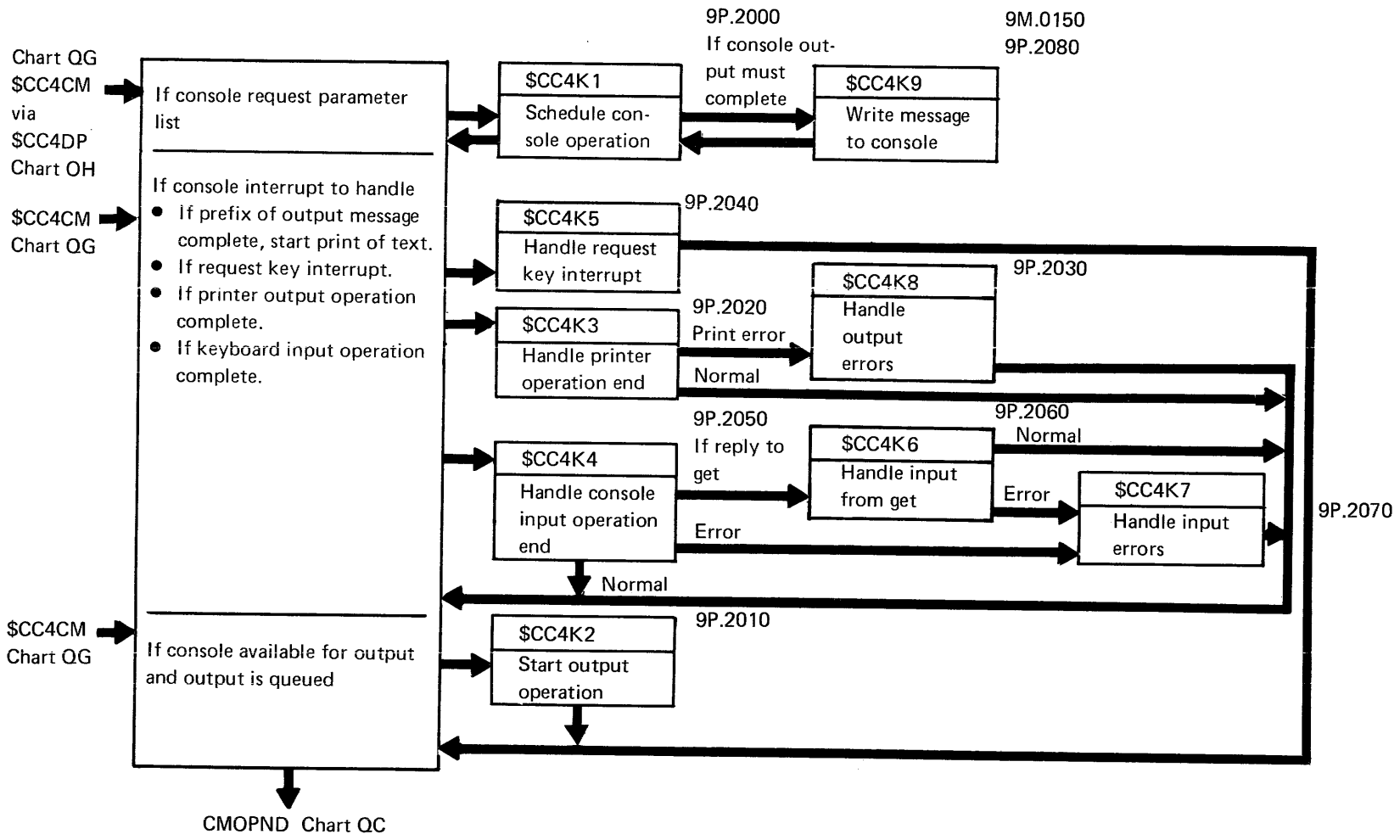
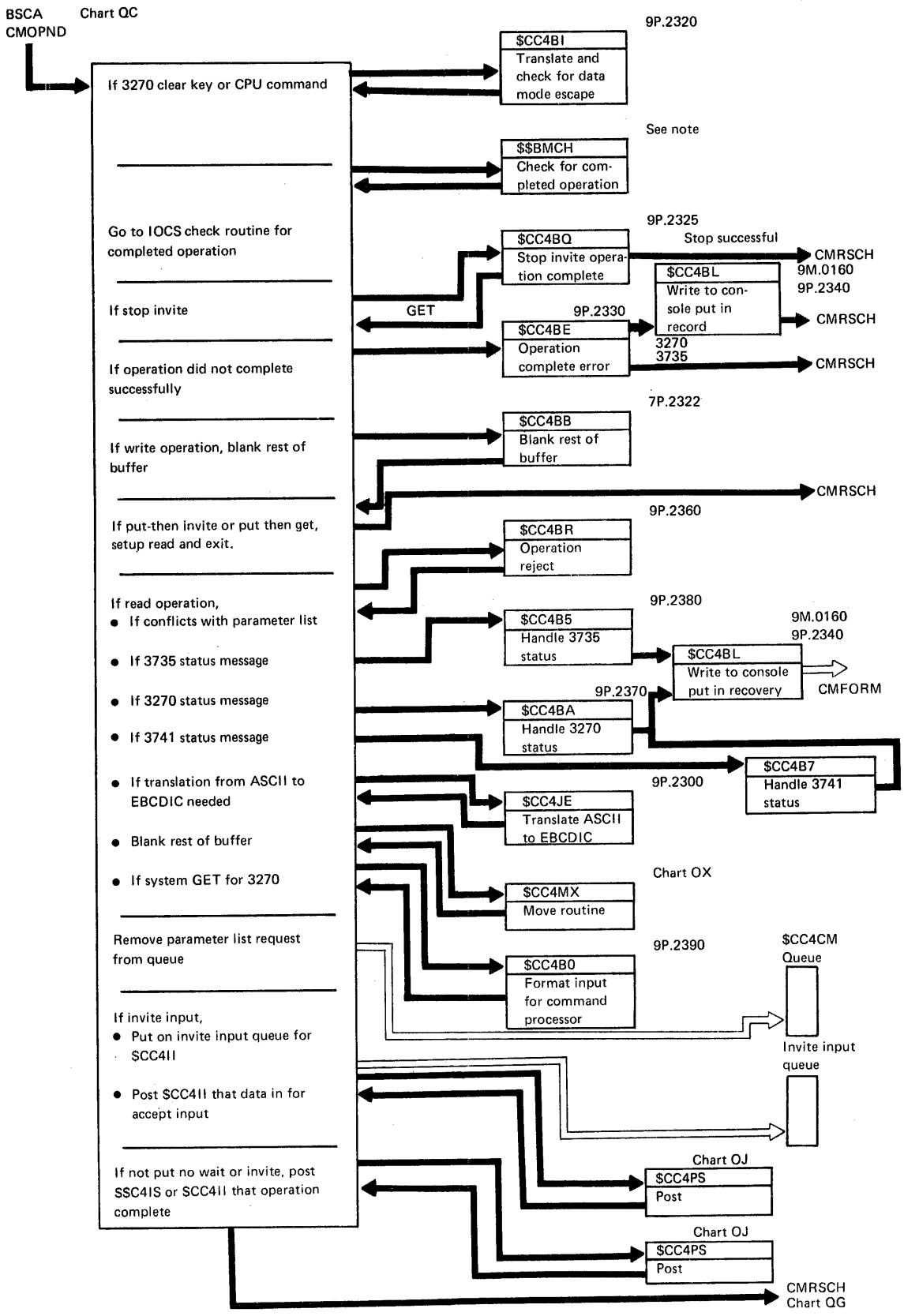


Diagram 9M.0100. CM Console Overview (Models 8, 10, and 12 Only)



Note: See *IBM System/3 Disk Systems Binary Synchronous Communications Programming Support Input/Output Control System Logic Manual*, SY21-0526.

Diagram 9M.0110. CM BSCA Operation Complete Handling

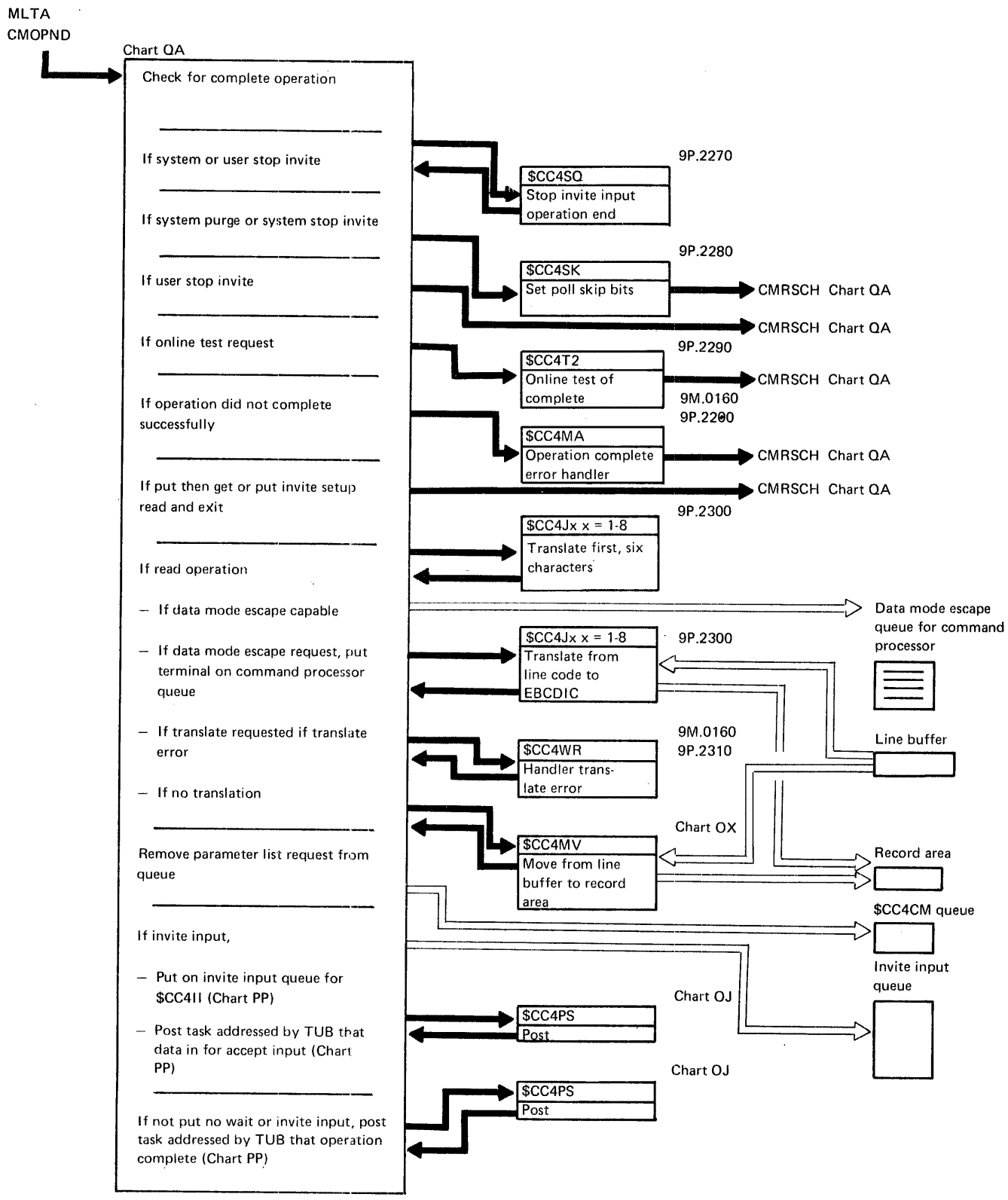
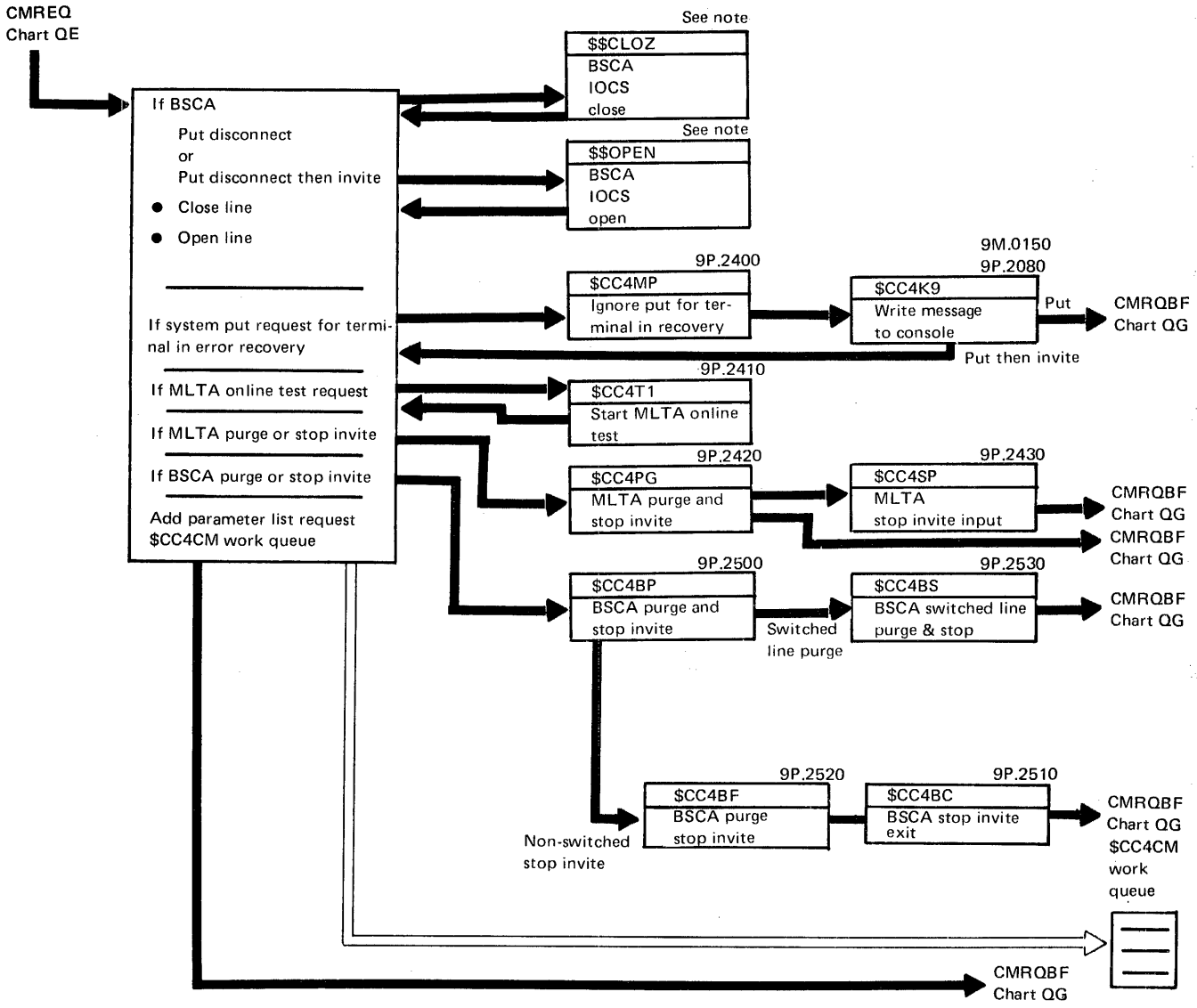


Diagram 9M.0120. CM MLTA Operation Complete Handling (Models 8, 10, and 12 Only)



Note: See *IBM System/3 Disk Systems Data Management and Input/Output Supervisor Logic Manual*, SY21-0512.

Diagram 9M.0130. CM MLTA/BSCA Request Scheduling (Models 8, 10, and 12 Only)

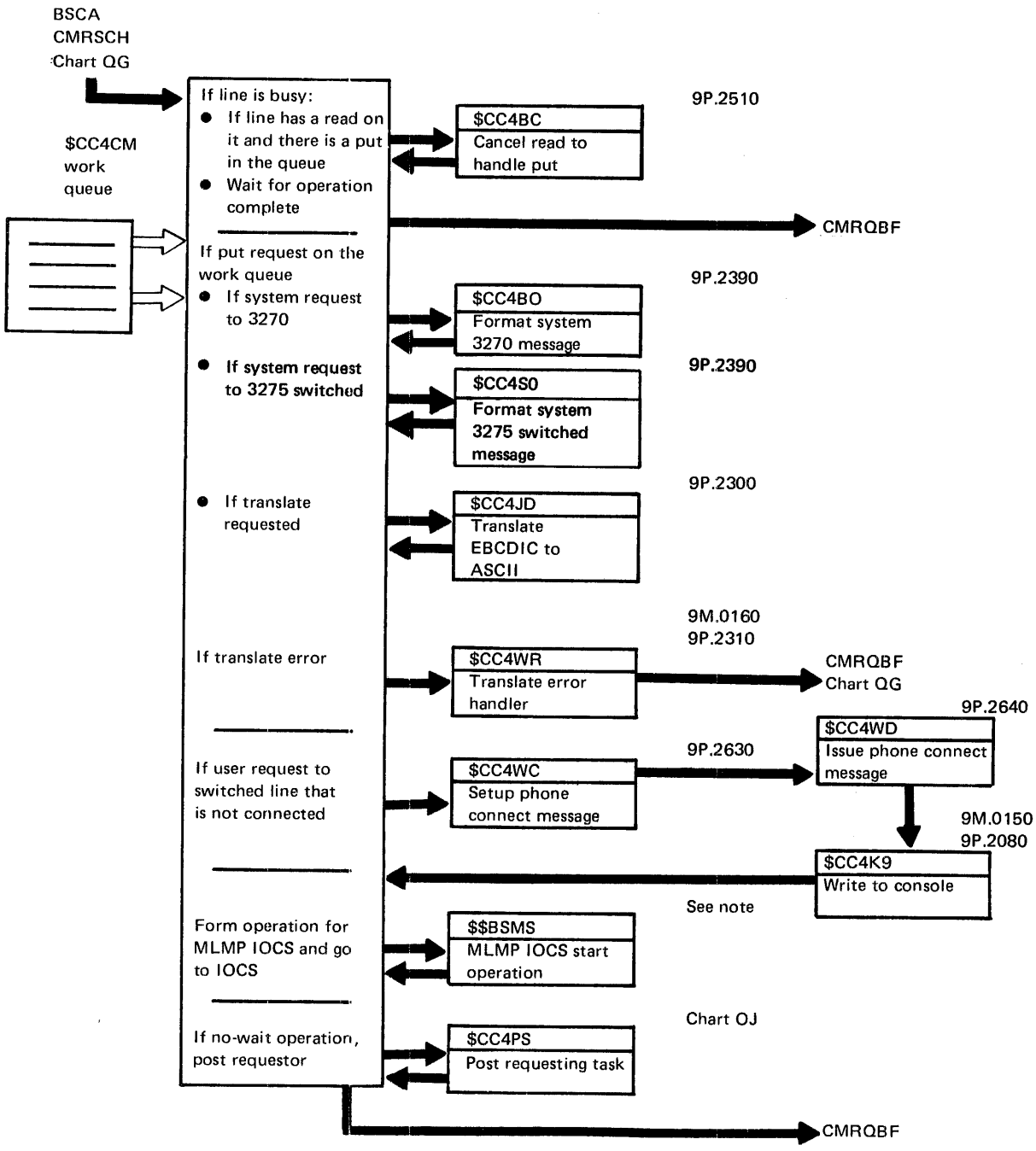
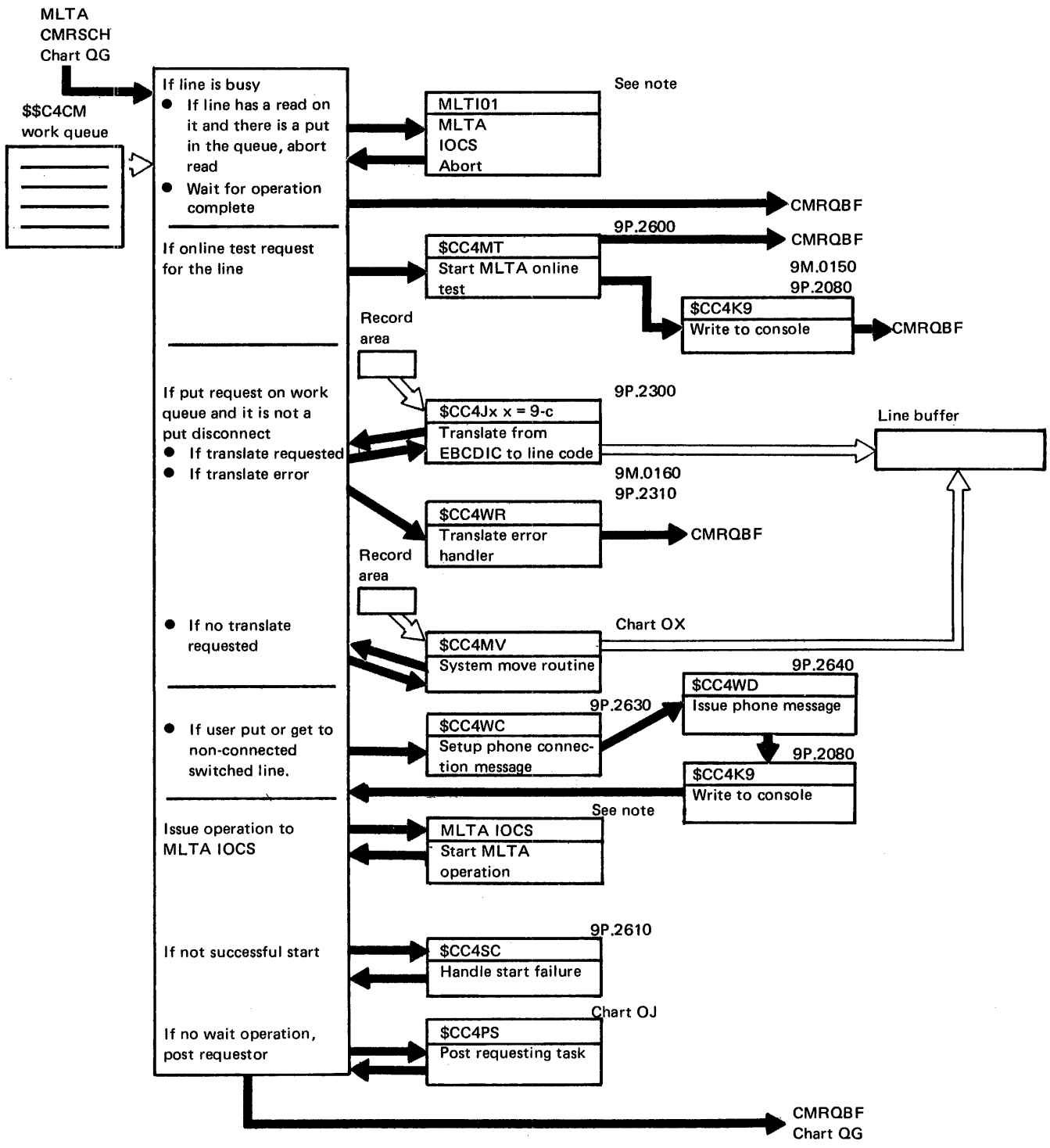


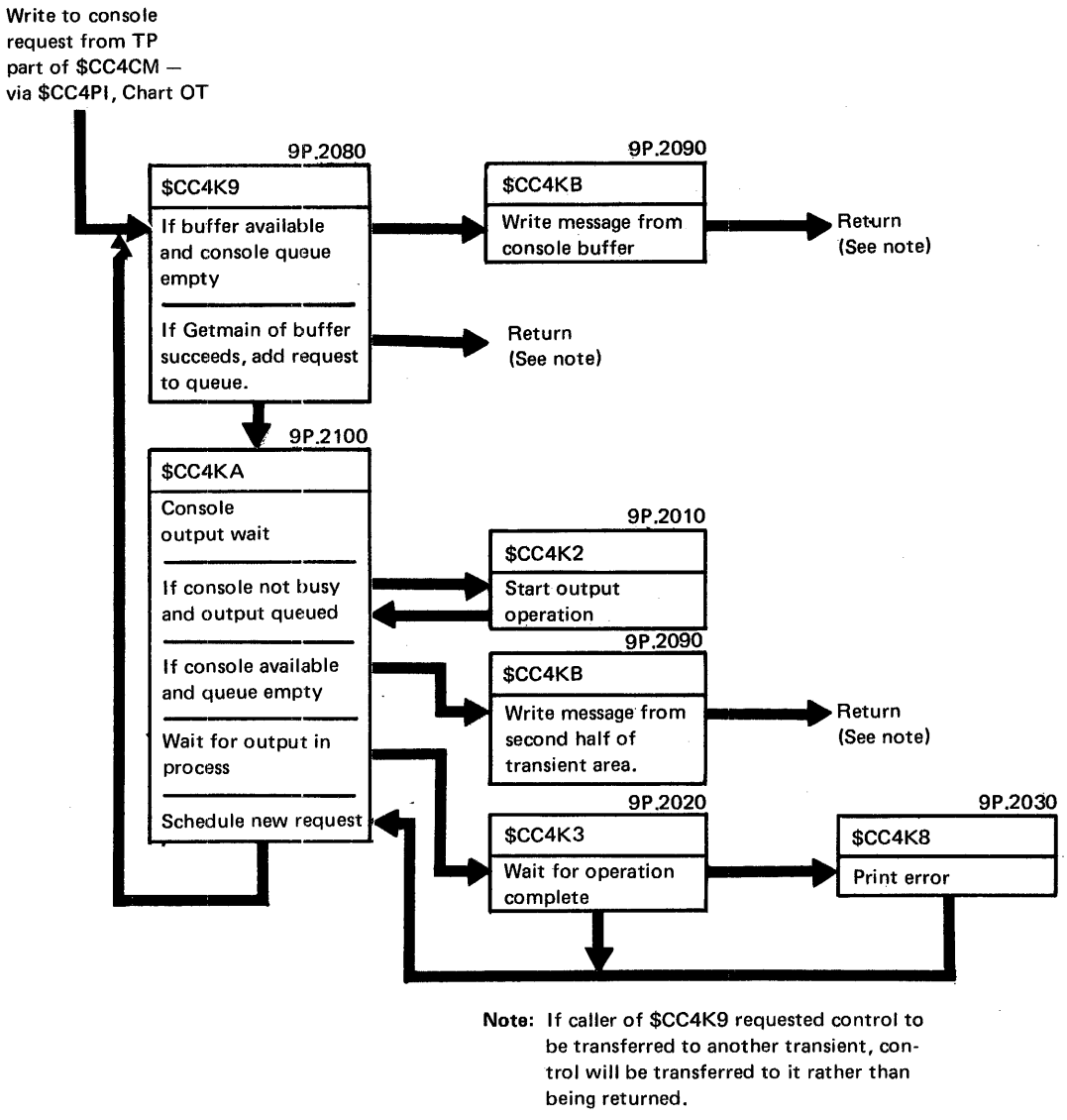
Diagram 9M.0140. CM BSCA Start Operation (Models 8, 10, and 12 Only)

Note: See *IBM System/3 Disk Systems Binary Synchronous Communications Programming Support Input/Output Control System Logic Manual*, SY21-0526.



Note: See *IBM System/3 Multiple Line Terminal Adapter RPQ Program Logic Manual*, SY21-0527.

Diagram 9M.0145. CM MLTA Start Operation (Models 8, 10, and 12 Only)



Note: If caller of $CC4K9 requested control to be transferred to another transient, control will be transferred to it rather than being returned.

Diagram 9M.0150. Console Output Transient Flow (Models 8, 10, and 12 Only)

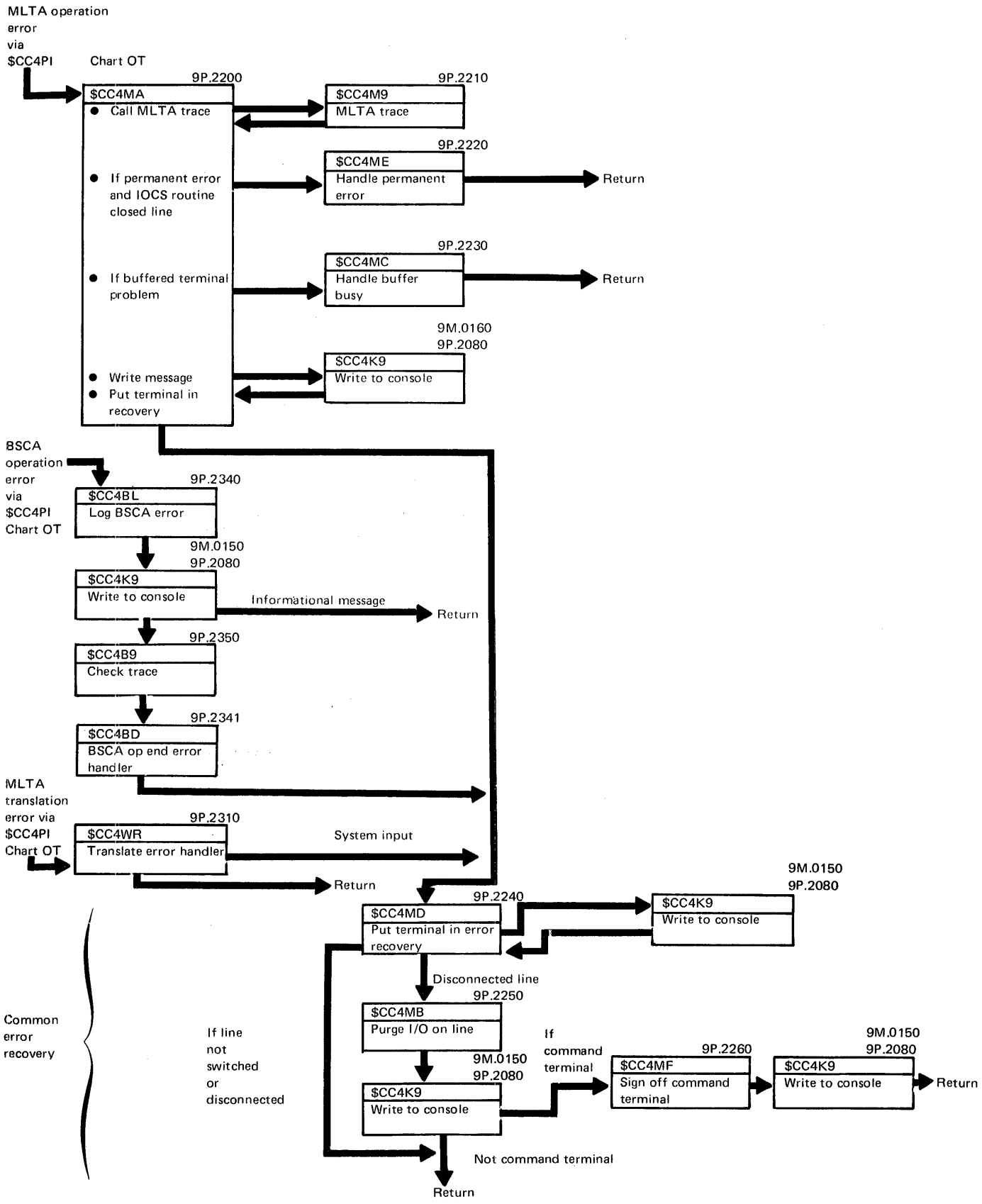


Diagram 9M.0160. MLRA/BSCA Error Handling Transient Flow

After a task posts $CC4CM, the task waits for $CC4CM to indicate that the request is sufficiently completed to return to the requester. In the case of wait operations like put, get, and put-then-get, this is done after the final operation-complete interrupt (CMOPND). For put-no-wait, invite input, input, system put-no-wait-invite, the task is posted to return as soon as the operation is scheduled on the line queue.

A completed invite input or system invite input operation causes a post to the requesting task when data is received and placed on the invite input queue. This post is done in the CMOPND section.

Note: Input is always queued to the command processor for system invite inputs, regardless of the requesting task.

Communication Management Functional Priorities

$CC4CM analyzes the work it has to do in terms of functional priorities. These priorities are based on the objective of keeping the console and all teleprocessing lines as active as possible. The order in which $CC4CM processes its work is:

1. Service console. Servicing of the console is performed by $CC4CM before analysis of teleprocessing service requirements.

2. Handle teleprocessing op ends/reschedule the TP line. $CC4CM checks the TP op end count maintained in the CCP Communications Area (#OPEND). If the TP op end count is greater than zero, $CC4CM handles one at a time the TP op ends that have occurred. Since the occurrence of a TP op end implies that the TP line is no longer active, the rescheduling of the TP line becomes part of the op end processing. After performing any rescheduling activity, $CC4CM re-analyzes the TP op end count. This cycle occurs until the TP op end count is reduced to zero.

3. Freemain analysis. If $CC4CM was posted that a Freemain occurred, $CC4CM attempts to obtain the main storage required to proceed with the processing of the pending function. For example, if an invite input cannot be scheduled because of the unavailability of a hold buffer, $CC4CM attempts to obtain the hold buffer with a Getmain each time a Freemain post occurs. Only a successful Getmain allows the invite input to be processed.

4. Schedule a new TP request. $CC4CM determines whether a new TP request has been received by checking the post mask in its TCB. If the post mask indicates there is a new TP request to be scheduled, $CC4CM gets the address of the parameter list that carries the TP request. The address is in the field @PRL in the CCP Communications Area.

Note: The one exception to the functional priority scheme described is that the scheduling of a new TP request which is a system put-no-wait will take priority over Freemain analysis. This is to reduce the chances that the system put-no-wait fails to get the necessary hold buffer space and is scheduled as a put (wait).

Line Scheduling Algorithm

All teleprocessing communications requests are issued via parameter lists. With one exception, the scheduling of these TP requests begins with the addition of the parameter list to a TP line queue. (The one exception is MLTA online test for which status information is placed in the TUB and the LCB). Basically, a scan of a TP line queue reveals all outstanding teleprocessing requests for a particular TP line.

The process of scheduling a TP line involves the scanning of the TP line queue and a selection of the parameter list to be honored. This selection is based on the following algorithm:

- For MLTA, a request for online test is honored first.

- For BSCA, a poll for status operation (3270, 3735) is honored first.

- The first output request found during the scan of the line queue will be selected; if none,

- All input operations will be honored (by enabling specific terminals on the line for input) with the following restriction: if the input operation is an invite input, the terminal will be enabled for input only if adequate hold buffer space can be obtained to hold the potential input.

The general philosophy of this algorithm is output over input. While the time to complete an output operation is finite, the completion of an input operation is dependent upon the activities of the terminal operator. Thus if a new TP request is received while the TP line is enabled for input, the input operation will be cancelled if no data is being received and the output operation started. At completion of output, input is restarted.

Getmain by $CC4CM

The action taken by $CC4CM on an unsuccessful Getmain varies according to the requested TP operation.

- Put-no-wait. An unsuccessful Getmain for a put-no-wait results in $CC4CM handling the request as a put-wait. Thus, the request will still be processed but the requester will not resume execution until the output has been transmitted to the terminal.

- Invite input. An unsuccessful Getmain for an invite input results in a suspension of the processing of the invite input. No terminal will be enabled for input unless there is either a record area or hold buffer into which the data can be moved. Thus an invite input cannot be started until adequate hold buffer space is obtained.

Disconnecting Switched Lines

- To disconnect a switched line a system put-disconnect operation (no wait) must be issued X'A006'.

- It is valid to issue a system put-disconnect/invite input X'A007'; in fact, for MLTA lines it is more efficient in terms of line time and CCP processing to issue the operations as one request.

- Only a system task can directly request the disconnection of a switched line. $CC4II will terminate any user program which requests an operation which involves the disconnect function.

- The disconnect operation must be issued only to switched lines.

- The disconnect operation cannot also transmit data to the terminal. Therefore the output length must be zero for any request involving a disconnect function.

Communications Manager Teleprocessing Control Blocks

Three major control blocks, whose contents are set and maintained by CCP, have significance specifically for communications management; these are:

The Line Control Block (LCB) – one LCB exists per each communications line in the system. The LCB is the CCP extention to the teleprocessing DTF. An LCB contains information relative to the nature and status of a line and consists of:

- The line DTF for the communications IOCS.

- The SDR tables for terminals on the line.

- The polling list, if any, for the line.

- Miscellaneous status and pointer data for the line.

The Terminal Unit Block (TUB) – one TUB exists for each uniquely identifiable terminal in the system. The TUB contains information defining the nature and status of the terminal with which it is associated. The information includes:

- The terminal's physical type and features.

- The current hardware condition of the terminal.

- The roles that terminal is permitted under CCP.

- The status of operations in progress on the terminal.

The Terminal Name Table (TNT) – one TNT entry exists for each symbolic terminal name permitted in the system. Each symbolic terminal name is assigned to a specific terminal and thus is associated with a certain TUB. During the running of the CCP system, the system operator may change the assignment of a terminal name, thereby associating the name with another TUB. Thus a Terminal Name Table entry consists of the symbolic name and a pointer to the currently assigned TUB. Any request for communications service by a user program references a symbolic terminal name as the identification of the terminal on which to perform the operation. On each such request, communications management must resolve that identity to a particular terminal (to a certain TUB) via the Terminal Name Table.

Console Management (Models 8, 10, and 12 Only)

The console management's support for the CCP program level resides in the Communications Manager ($CC4CM). While some of the support is resident, most of the routines that service the console are transient ($CC4K1, $CC4K2 through $CC4KB).

Console Management (Model 4 Only)

The console management's support for the CCP program level resides in the Communications Manager ($CC4CM). While some of the support is resident, most of the routines that service the console are transient ($CC4X1 through $CC4XG).

Note: The routine $CC4X9 is renamed $CC4K9 in Model 4 CCP installation.

The only standard DSM keyboard routine used to support the console is the Keyboard Interrupt Handler ($@COIH). The CCP console support is written to allow overlapping of console I/O with other system functions.

The Model 4 CCP console is a logical combination of the operator keyboard, a 3277 Model 1 screen, and a disk file. The name of the disk file is $CCPLOG. The console messages are written to this disk file. At the operator's discretion and after a command is entered, the information in the file is displayed on the 3277 screen. Five to 10 messages fit on the screen at one time.

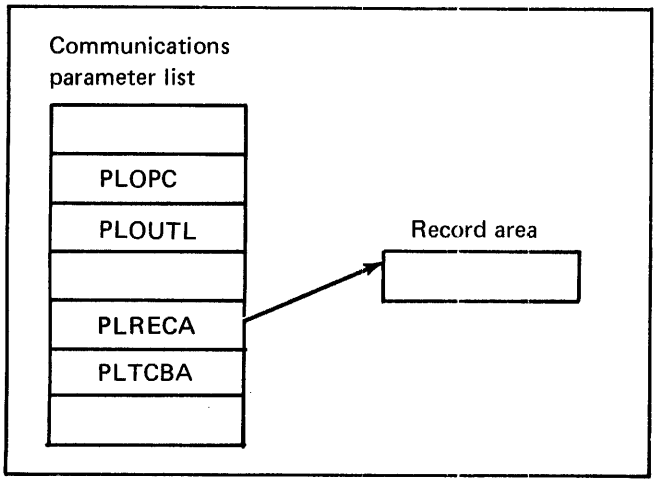
Console Output Request not from the Communications Manager: All console output requests from other than the Communications Manager are issued via a communications parameter list. The fields in the parameter list which are utilized are:

The operation code/modifier (PLOPC).

The output length (PLOUTL).

The record area address (PLRECA).

The TCB address of the requesting task (PLTCBA).



When $CC4CM receives an output request for the console from a task other than the Communications Manager, it will add the parameter list to the console output queue. Whenever it is determined by $CC4CM that the console is available for output and there is a parameter list in the console output queue, the transient $CC4X7 is called to write the output to the console. The message is moved to a 2-sector (512-byte) disk buffer. This buffer is written to the disk file ($CCPLOG) when a message overlaps the second sector. After the write is complete, the second sector is moved to the first sector in the buffer. This disk buffer resides in the console work area.

Console Output Request from the Communications Manager: This Communications Manager ($CC4CM) occassionally issues console output requests from its transient area. Since the routines to support the console execute in the $CC4CM transient area, a conflict exists which cannot be resolved with the console support used to handle requests from other than the Communications Manager.

Thus, a series of special transients are provided specifically for the support of console output requests from $CC4CM transients. Basically, the following considerations are involved in the handling of console output from the Communications Manager:

- The console output request is passed by using a message data area consisting of a 1-byte message length immediately followed by the message test.
- The message data must be at least 258 bytes into the transient issuing the request.
- The console transients which schedule output for the console ($CC4X9 and $CC4XA) must be no greater than 256 bytes long.
- Console output requests from $CC4CM transients are issued as follows:

  1. The address of the message data area is passed in the field #KMCPL in the console work area.
  2. Control is transferred to $CC4X9.

In addition, if it is desired to have $CC4X9 transfer control to another transient when the console request is scheduled, then:

1. The value of bit #KMXXE in byte #KMSWT in the console work area must be set to 1.

   Note: The console transients do not reset #KMXXE.

2. The disk cylinder/sector of the transient to which control is to be transferred must be passed in field #KMXCT.

Console Input: The keyboard is ready for input when the command processor has issued an input request to the console. The keyboard is ready most of the time. Keys that are not supported are bypassed.

All keyboard input is entered into the console buffer; the console buffer resides in the console work area. The console buffer is 80 bytes long; any attempt to enter more than 80 bytes of input results in an error message being displayed on the 3277 screen.

When program start or enter+ is pressed, signaling the end of input, the data is analyzed and classified one of three ways:

1. Program request: Period is first input character.

2. Reply to user task: Slash is second input character.

3. Command: Neither of the above.

Console Display on 3277: The 3277 screen has 12 lines of 40 characters each. These 12 lines are divided into lines 1-10 and lines 11-12. Lines 1-10 are used to display from 5 to 10 messages from the console file. Lines 11 and 12 are used to display some error messages and keyed input before it is entered into the system. Two logical functions display part of the file on the screen.

The first is display backward (from a given point in the file). This function is in routines $CC4XB and $CC4XC. Three sectors are read, ending with the given point in the file. Then a screen buffer is built from messages in these three sectors. This function is called by command key 7 or by CM to display backward from the newest message.

The second function is display forward (from a given point in the file). This function is in routines $CC4XF and $CC4XG. Again, three sectors are read, this time starting at the given point in the file. This function is called by command key 8 or by CM to display forward from a display command.

Keyed input is displayed on lines 11 and 12 of the screen when the return key is pressed. This function is handled in $CC4X3. Disk buffers and screen buffers are in the console work area.

Command Processor Task

The Command Processor Task ($CC4CP) controls the processing of all CCP input that is not specifically directed to a user program (Diagram 9M.0200). All teleprocessing devices that are not currently allocated to a user processing program are allocated to the Command Processors TCB. Therefore when the Communication Manager ($CC4CM) recognizes input from one of these devices, the terminal's TUB is queued onto the Command Processors TCBINQ chain by the Communications Task, the input is translated to upper case EBCDIC, and the Command Processor is posted that it has work to perform. The modules that perform all command processing consist of a small resident routine ($CC4CP) that enters a wait state when there is no input to be processed, and a number of transient routines that perform the command processing functions. These functions can be broken down into three parts:

1. Initial input processing, including basic syntax checking and control routing.

2. The processing of the input data (commands or program requests).

3. Final processing, including the issuing of I/O and Freemaining of input data buffers as directed by the processing routines.

The initial input processing occurs in both the resident Command Processor module ($CC4CP) and the first command processing transient ($CC4PC). Input is checked to ensure that it meets basic syntax requirements (the input must be at least two characters long), the Terminal Unit Block is checked to see if the input is the result of a failed purge or stop-polling operation, and the system status flags in $CCCOM are checked to see if terminal commands are still allowed (not suspended or a Shutdown has not been requested).

If the input is to be ignored, the module $CC4PF is invoked to issue a message to the terminal and, depending upon whether or not the terminal should be re-invited, the TP I/O interface routine ($CC4IS for system I/O) is invoked.

If the input data is to be accepted, $CC4PC determines whether or not another control routing determination module is to be called or whether it can handle the control routing itself. If it can handle routing itself, $CC4PC calls the appropriate command processing routine. $CC4PC can route program requests, /Q and /NOQ commands, and /FILE commands.

If the input requires further checking, control is passed to either the Terminal Command Routing Routine ($CC4PT) or to the Console Command Routing Routine ($CC4PK), depending on whether input is from a terminal or the console. These routines perform similar processing in that the input is checked to see if it is a valid command verb and if required input data exists. If all syntax criteria are met, control is passed to a specific command processing transient to handle the input. If the criteria are not met, an error message is issued by an error message transient and control is passed to the final command processing transient ($CC4PR) following the return to the resident routine ($CC4CP).

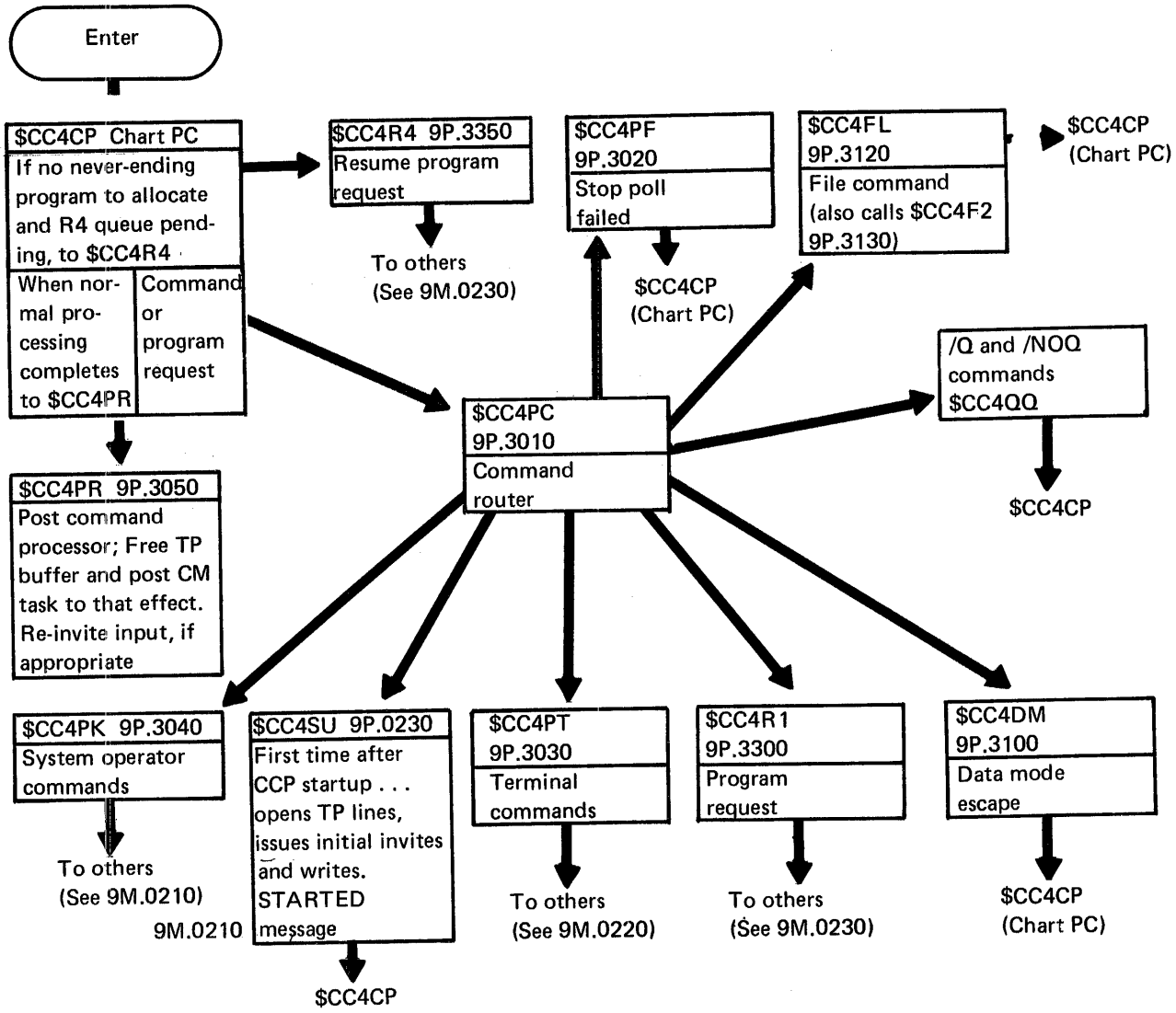


Diagram 9M.0200. Command Processor Task Overview

Command Processor Return Routine ($CC4PR)

The Command Processor Return Routine ($CC4PR) interrogates the bit settings in the Command Processor Work Area ($CPWK) and takes the actions to be performed:

1. Freemain the input buffer and post the communications task that the Freemain occurred.

2. Write a message to the terminal from the message area in the Command Processor Work Area ($CPMSG). The message length is the field $CPLMG.

3. Re-invite input from the terminal.

Once these actions are performed the return routine exits via $CC4TR, to the Command Processor Mainline Routine which will determine whether there is additional input to be processed. If no input remains to be processed, the wait routine is entered.

System Operator Commands (Diagram 9M.0210)

The commands that are valid only if issued by the system operator are:

- DISPLAY
- VARY
- CANCEL
- SUSPEND
- RESUME
- ASSIGN
- TEST
- SHUTDOWN
- TRACE
- ERP
- ALLOCATE (Models 8, 10, and 12 only)

A Message (MSG) command entered by the system operator causes a different transient module to be called by the command processor than a /MSG entered by a terminal operator.

DISPLAY is used to exhibit several different types of CCP status information and is depicted on a chart separate from the remaining commands.

All modules with module names beginning with $CC4E... (except $CC4EJ) are called to cause a specific output message to be sent to a terminal.

Terminal Operator Commands (Diagram 9M.0220)

The commands that are valid only if issued by a terminal operator are:

SIGN ON (The module named $CC4YA is a no op routine as shipped to the user. Its purpose is to allow the user to perform sign on diagnostics by simply replacing $CC4YA with another module.)

SIGN OFF

NAME

RELEASE

RUN

MSG

Data Mode Escape

Program Request (Diagram 9M.0230)

Program request processing is a two part process. The first part consists primarily of validating that the syntax of the request is correct. The second part is validating that the system can actually honor the program request.

The modules that are concerned with the first part of request processing are $CC4R1, $CC4R2, and $CC4R3. The first module, $CC4R1, tests to insure that the requesting terminal is in command mode and rejects the request if the terminal is not in a command mode. If the requesting terminal is the console, $CC4R1 insures that the console sub-TUB is currently available for a program request. If it is not available an error message is issued through the module $CC4EA and the request is rejected.

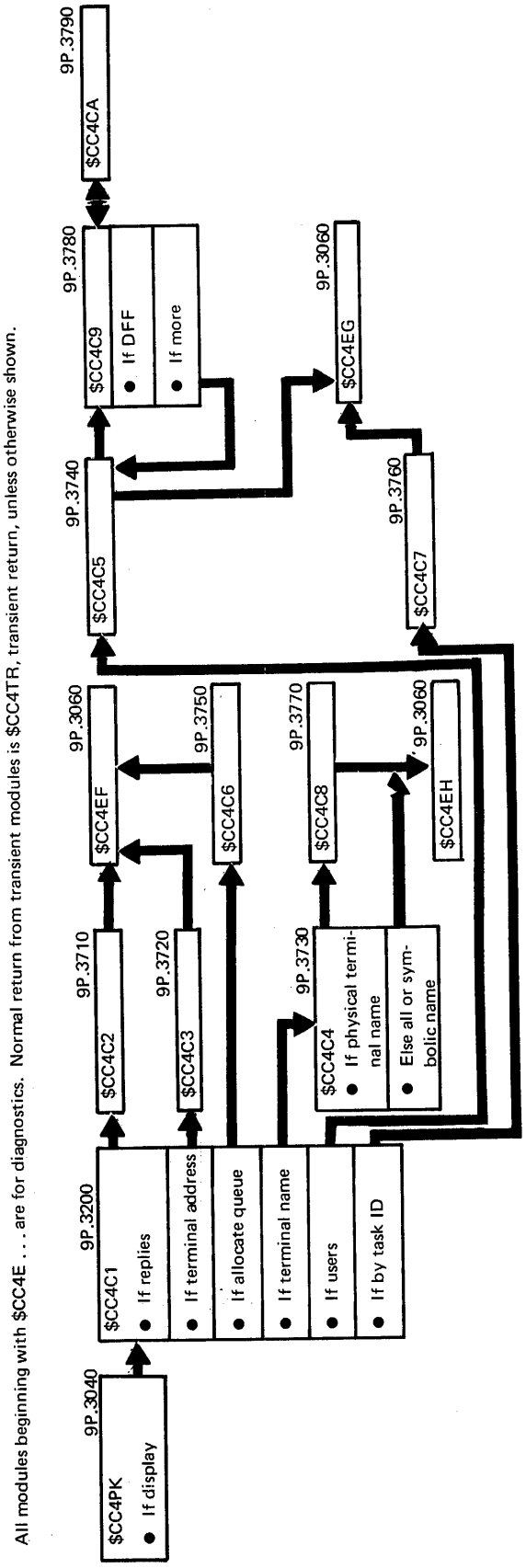
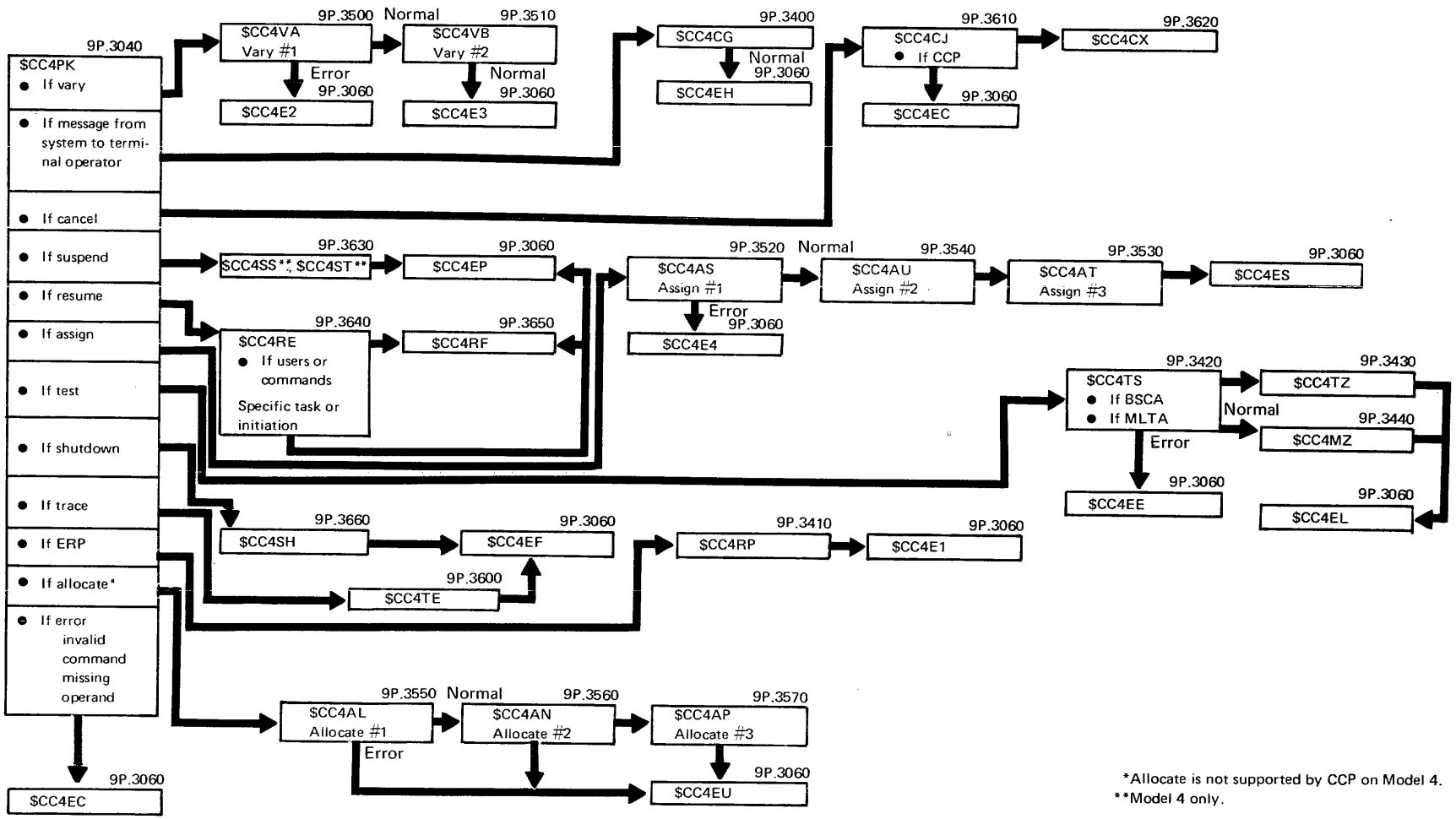


Diagram 9M.0210 (Part 1 of 2). Control Flow of System Operator Display Commands



*Allocate is not supported by CCP on Model 4.
**Model 4 only.

Diagram 9M.0210 (Part 2 of 2). Control Flow of System Operator Display Commands

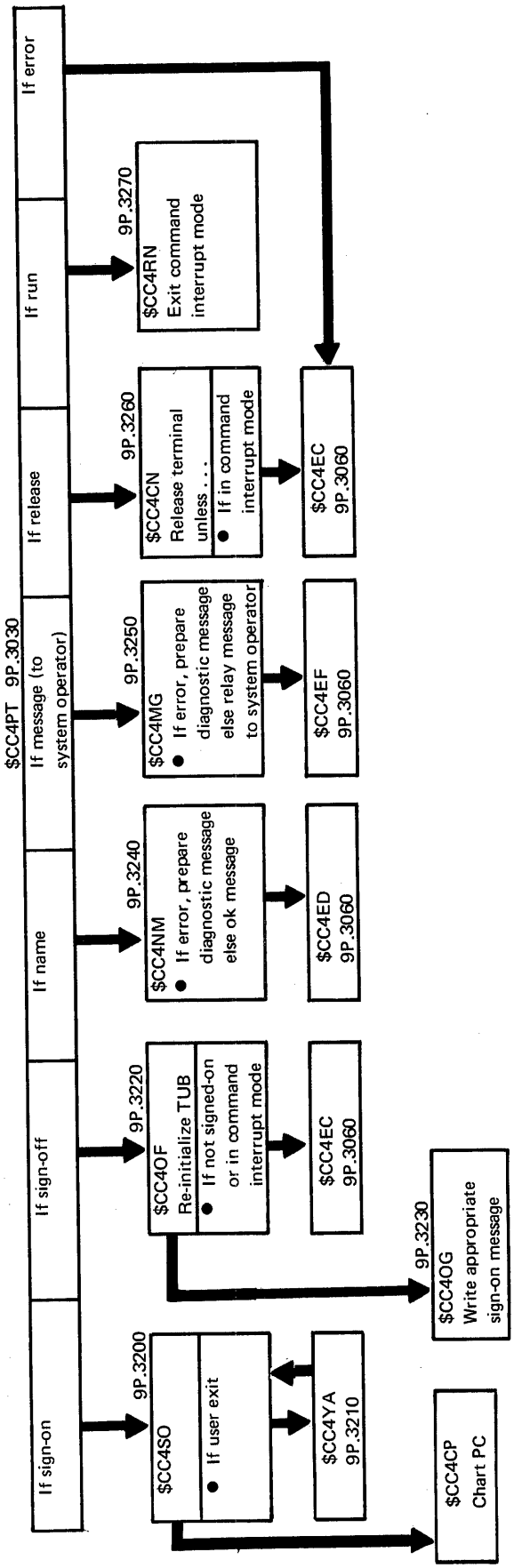
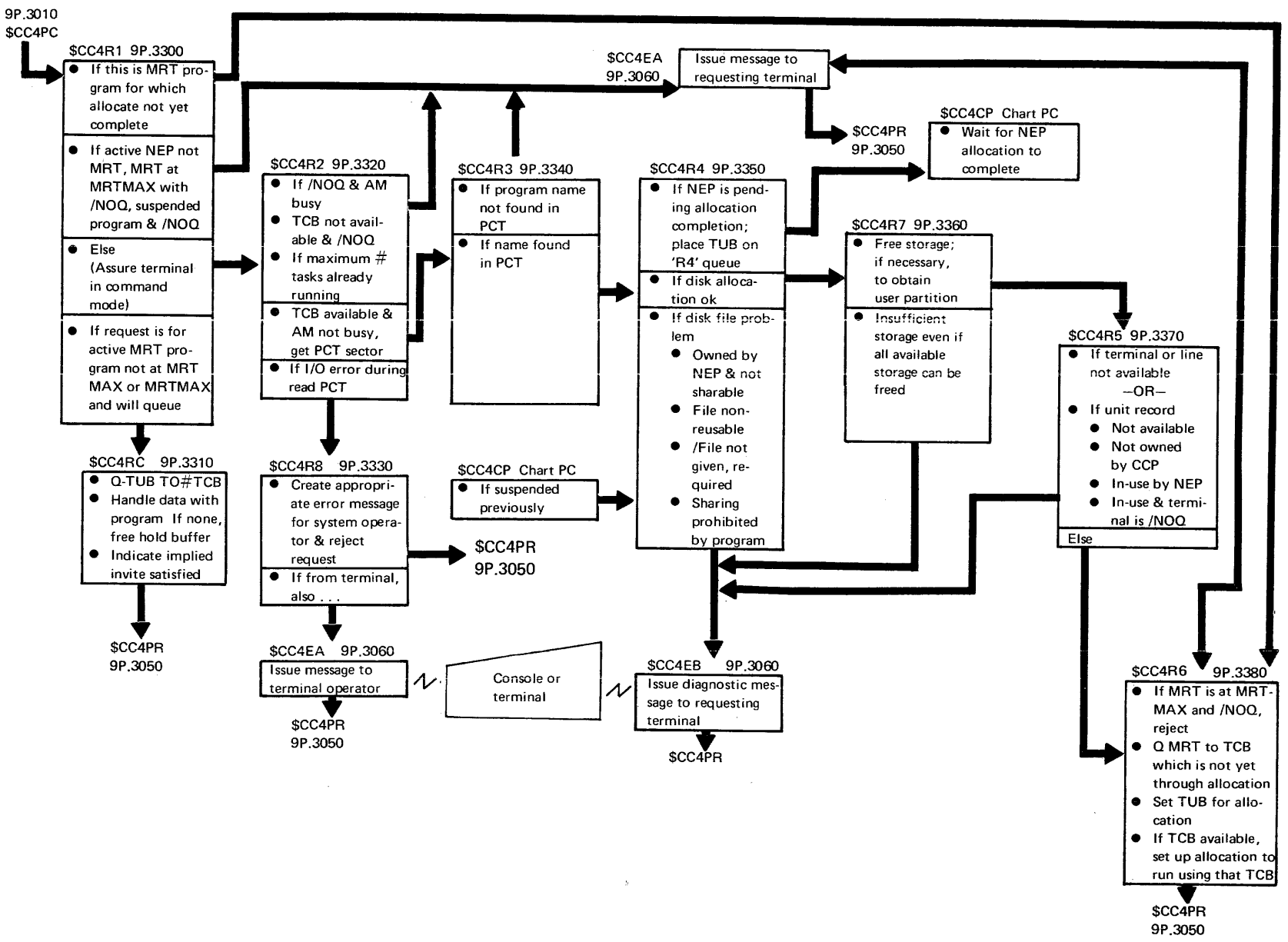


Diagram 9M.0220. Control Flow of Terminal Operator Commands

Diagram 9M.0230. Control Flow of Program Requests



Once it is determined that the terminal is in the correct mode, the length of the requested program's name is checked. If the name is longer than six characters, an error message is issued. (It has already been determined in the initial module of command processing, $CC4PC, that the input length is at least two characters long.)

The TAXPRM field to be passed to subsequent program request transients is set up and its content is affected by the following:

- The status of the requesting terminal — queue, or no-queue.
- The presence of input data.
- The length of the program name portion of the input buffer.

The routine then determines which module to call next. If the request is for an active, multiple requesting terminal support program, the routine $CC4RC is invoked to add the request to the program's input Terminal Unit Block queue. If the request is for a multiple requesting terminals support routine that is currently on the allocation input queue, control is routed to $CC4R6 to add the request to the queue for the already validated program request (the allocation input queue). If the program request is not one of the above, and is not for an active, never ending program that is non-MRTS, the second module of program request ($CC4R2) is called.

$CC4R2 ensures, if the requesting terminal is in NOQ (no queue) status, that the Allocation Task is not currently busy and that an available user TCB exists. If these conditions are not met the system cannot honor the NOQ status of the terminal (an immediate entry into Allocation cannot be guaranteed) and the system must reject the request. If these conditions are met, the disk address of the Program Characteristics Table sector in which the requested program's entry exists is calculated from the PCT index in main storage, a disk read IOB is constructed, and the PCT sector is read into the second half of $CC4R2. (If a disk read error occurs during the input of the PCT in $CC4R2, $CC4R8 is invoked by $CC4R2 to write an error message to the system operator and to the terminal operator to inform both that a system error occurred.) The module then transfers control, specifying a one sector transient read, to $CC4R3 to complete the syntax check.

$CC4R3 scans the PCT sector for the PCT entry which matches the requested program's name. If an entry is not found, an error message is issued through $CC4EA and the request is rejected. If a PCT entry is found, the PCT data for that program is moved into the buffer addressed by the TUBs TPRECA filed. The PCT data is moved to immediately follow the input data (whose length is in TPEFFL). This move completes the syntax check of program request. The next module ($CC4R4) begins the determination as to whether or not the system can honor the program request.

$CC4R4 ensures that all disk files specified by the PCT entry for the requested program will be available to the program. If the file will not be available, an error message is sent to the requesting terminal and the request is rejected. If all disk file requirements are met, control is transferred to the main storage requirements validation module, $CC4R7. $CC4R7 determines that enough main storage will be available for execution of the requested program. Storage requirements include the following:

- If the program uses DFF, main storage must be available for the requested program, $CC4DF, and the output hold areas, unless they are already in storage. ($CC4DF will always be in storage with remap.)
- If the program is a never ending program it must be determined that main storage will be available on a boundary of the user program area. In addition, if the program will also use DFF, main storage must be available on the other boundary of the user program area for the DFF module and the output hold area. (Output hold areas only if remap is specified.)

It may be necessary in this module, if the requesting terminal is unwilling to queue its request, to purge modules from the user program area to ensure that main storage will be available. When this is done the programs that are currently in main storage but not in use (as shown by inactive CDEs) are purged by Freemaining their storage and marking the CDEs as available.

It may also be necessary to purge the DFF module and the output hold areas from main storage if they are not currently in use to determine if sufficient space will be available at execution time. (The DFF module will never be purged for remap.) When this is done, the boundaries of the DFF module in $CCCOM (#DFBEG and #DFEND) are reset to their null values, the DFF load address (#DFLOD) is cleared to indicate that the module is not in main storage, and the DFF load parameter list in $CCCOM is re-initialized.

Once the main storage availability tests are complete, the next module of program request ($CC4R5) is invoked to determine the availability of any unit record and teleprocessing devices required by the program. The unit record and terminal requirements are checked to see if the needed devices are currently available and not encumbered by a never ending program (or in the case of a unit record device on a DPF system, that the device is allocated to the CCP program level).

Each required terminal that is currently online and in initial mode (capable of commanding the CCP and being polled) will have a stop polling operation issued to it. If the stop is successful, the terminal is available for Allocation. If the stop is unsuccessful, all terminals that were previously stopped for this request validation are restarted, and the request is rejected with an appropriate error message issued by $CC4EB. The request is also rejected if any required terminal is on a connected switched line.

The next module, $CC4R6, is the final program request module. It is invoked directly from $CC4R1 to add a MRTS request to one already queued for Allocation or from $CC4R5 if all aspects of the request have been validated. $CC4R6 queues the request onto the input queue to Allocation (the field @TUPST in $CCCOM) and, if the Allocation Task can be started (a free user task control block exists and allocation is not currently busy), sets up a TCB to address the resident Allocation Control Routine ($CC4AM) and makes the TCB dispatchable.

Allocation Task

Allocation binds all resources required for the execution of a requested program to a user Task Control Block, loads the user program (and the DFF module if needed), and gives control to the user program (Diagram 9M.0240).

The first module of Allocation is the Resident Mainline Routine $CC4AM. One function of this routine is to wait to be dispatched to begin the allocation process or to wait for resources that are currently in use by another user task. The other function of the resident routine is to exit to the Dispatcher to cause control to be given to the user program once the allocation process has completed. All other Allocation processes are performed by transient modules, the first of which is called by the resident routine.

The first transient is $CC4A1. Its function is to determine, the first time it is entered to begin allocation for a task, what resources are required. This is done by inspecting the programs PCT entry in the input hold buffer addressed by the TPRECA field of the Terminal Unit Block addressed by the $CCCOM field @TUPST. After resource requirements are determined, $CC4A1 sets control bits in the allocation work area to indicate the routing to specific allocation routines to bind the required resources.

Allocation is a two-pass process. If a required resource is not available during the first pass, the resource is marked as needed and the specific Allocation transient passes control to the next module of the process. Once the last module has finished, it determines whether or not all resources were allocated and, if they were not, exits (via $CC4TR) to the Allocation Mainline Routine to wait for Termination or the release routines to post Allocation that all required resources are available.

The second time the Allocation control transient is invoked it inspects the Allocation post mask to determine where control should first be routed. If no Allocation resources were bound during the first pass, the control flow for the second pass will be the same as it was for the first pass.

The first resource binding modules to get control are those which allocate the main storage, load the program into main storage, build the CDE and TCB from information contained in the PCT (relative to the attributes of the program) and, if needed, load and initialize the DFF module $CC4DF.

The first main storage allocation transient is $CC4CR. This module allocates the storage needed for DFF and/or the output hold areas. This module loads $CC4DF for non-remap operations. It finds an available CDE to use for the user program and passes control to the second main storage allocation module, $CC4CS. $CC4CS allocates the user program main storage area and passes control to $CC4CT. $CC4CT first loads the user program into main storage using the standard interface to DSM load (a RIB of hex 48 or hex 68, depending upon whether the program is on the system pack or the program pack), then finishes the initialization of the TCB and CDE and, if the DFF module was loaded with this request, initializes it. If DFF is used by the user program the next routine to get control is the DFF Initialization Module ($CC4DA) for the user program's appended storage.

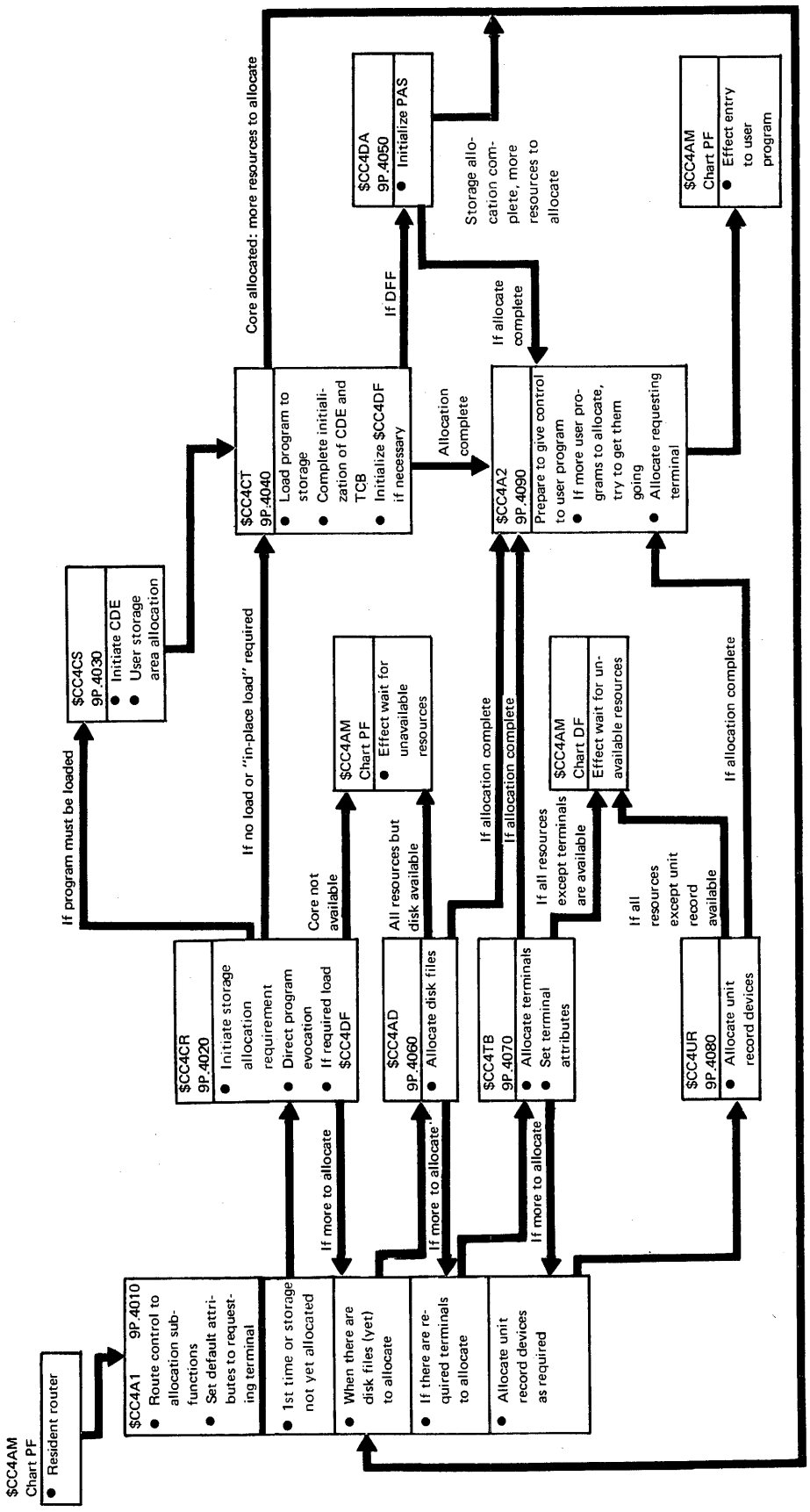


Diagram 9M.0240. Control Flow of Allocation Task

The next processing done during the Allocation process is disk file allocation. The module $CC4AD allocates all required disk files, setting allocation and usage indications in the CCP short DTFs and turning on the disk file's allocated bit in the TCB.

$CC4TB allocates all required terminals and implicitly required switched lines to the TCB. Each line and terminal used by the task has the address of the TCB put into it. In addition, the terminal attributes will be set for each terminal including, if necessary, the attributes of the requesting terminal if it is also one of the program selected (required) terminals.

The final module for allocation of a specific resource is $CC4UR, which allocates the printer, MFCU, 3741, and/or the 1442 to the user task. This process includes the setting of bits in the TCB indicating that the device is allocated and that the device is unavailable to any other task or, in a DPF system, to the other program level.

Following successful allocation of all resources the final Allocation transient, $CC4A2, is invoked by whichever module was the last one to run. $CC4A2 finishes the allocation process, including the allocation of the requesting terminal to the TCB and putting that terminal on the tasks input queue. If there is more work for Allocation (the requesting terminal's TUBPST field is non-zero, indicating additional terminals are queued for allocation), $CC4A2 scans the user Task Control Blocks to see if there is another TCB available for use. If there is, the TCB is set up to run (in a manner similar to that of the final module of Program Request) and the TCB is made dispatchable. This causes Allocation to run again for the next request as soon as all active user tasks, including the one just allocated, are waiting. $CC4A2 then returns, via $CC4TR, to the Allocation Mainline module to give control to the user program.

Termination

User program termination is a two step process. The first step is that of communicating to the Termination Task routines that a user task is to be terminated. The second step is the termination of that task, which includes the freeing of all encumbered resources, clearing all indications that the task existed, possibly getting the allocation task started, or terminating CCP (Diagram 9M.0250).

Step One of Termination

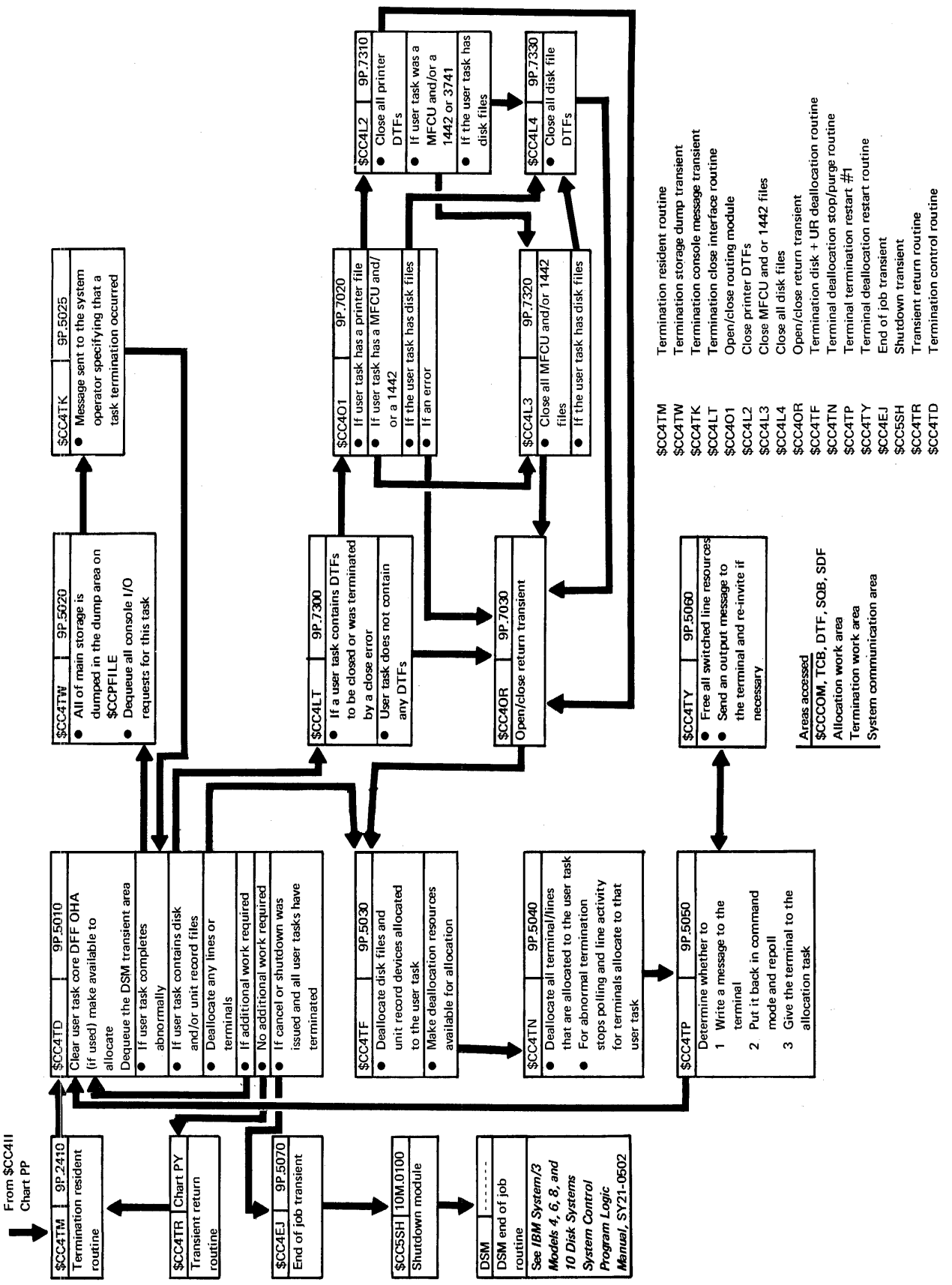
The first step is done in the Termination Interface Routine ($CC4TI). This module has three entry points to allow flexibility to various parts of the system. One entry point allows the user program to terminate itself by issuing an EOJ RIB. The RIB call is intercepted by the General Entry Intercept Routine ($CC4IG) and, once the hex 84 value is recognized by it, control is routed to the termination interface. The interface marks the task for normal termination and makes the Termination Task dispatchable (if the Termination Task is not already busy – if it is busy, $CC4TD finds the TCB and terminates it without further interface requirements).

Another entry point of the termination interface allows a system routine operating under the control of the user TCB to terminate the task. This might be done by the TP I/O Interface Routine ($CC4II) when the user makes an error in requesting an I/O operation. $CC4II calls the termination interface, passing a completion code to it to be put into the TCB prior to the interface routine relinquishing control.

The third entry point is used when some task other than the user task is running and has determined that the user task must be terminated. This is done by the Communication Task when a user related error occurs and the task can no longer be allowed to execute. The interface is entered from the Communication Task and marks the TCB (of the user) for termination, readies the Termination Task, and returns control to the Communication Task to allow it to continue processing on its current entry.

Step Two of Termination

The second step of termination is that of the Termination Task itself. Termination consists of a small resident portion and a series of processing transients. The Resident Termination Routine ($CC4TM) is used to wait for work and, when the Termination Task is made dispatchable, invokes the first Termination transient ($CC4TD).



- **Diagram 9M.0250. Control Flow of Termination Task**

Upon initial entry, the first Termination transient scans the user Task Control Blocks to find one that has been marked by the termination interface to be terminated. Once a TCB is found, the processing begins. If the task is terminating abnormally, the Termination Dump Routine ($CC4TW) is invoked. $CC4TW dumps all of main storage to disk (if space is available in the dump area in $CCPFILE) and also dequeues any non-system console I/O that is currently queued for the terminating task. After the main storage dump to disk, $CC4TW calls $CC4TK, which will write a message to the system operator telling him the task identification, the program name, and the circumstances of the termination (the task completion code). Control is then returned to the Termination control transient to continue the termination processes.

The control routine marks the task's CDE that it is no longer in use by an active TCB and, if the Allocation Task is waiting for main storage, posts the Allocation Task that main storage is available. Following this the control routine invokes the File De-allocation Routine ($CC4LT) if there are any disk or unit record files allocated to the task. If there are not, the Terminal/Line De-allocation Routine ($CC4TN) is invoked. Once these routines return, the control routine frees any DFF output hold area encumbered by the task and posts its availability to any task that may be waiting for it. Following this, the TCB is re-initialized and the control routine returns to its initial entry logic to see if there are any more TCBs to be terminated. If there are none, the control routine exits in one of two ways.

If there is a Cancel or Shutdown of CCP pending as indicated by flag bits in $CCCOM, control is passed (via $CC4TX) to the CCP final Termination Routine ($CC4EJ). If there is no Shutdown or Cancel pending, or there is a Shutdown pending but there are more programs to be run, control is returned to the Termination main routine (via $CC4TR) to wait for more work for the Termination Task.

The file termination routines function in two parts. The first part, initiated by $CC4LT, closes the disk and unit record files and writes out any buffers pending I/O. The second part de-allocates the files from the task, posting the Allocation Task if any of the files are needed by it. The file de-allocation routine passes control to the Terminal/Line De-allocation Routine ($CC4TN) when the file processing is completed.

The Terminal/Line De-allocation Routines are:

- $CC4TN – Line De-allocation and Terminal Stop Polling/Purge Routine.
- $CC4TP – First Terminal Restart Routine.
- $CC4TY – Second Terminal Restart and Terminal De-allocation Routine.

$CC4TN de-allocates all switched lines allocated to the terminating task that have no terminals currently allocated to them. This includes the resetting of the LCBTCB address field to zero, disconnecting the line and posting allocation if the line is needed. The routine then, for each terminal allocated to the terminating task, ensures that all TP I/O is quiesced on the device, frees any TP buffer associated with each terminal, and posts the Communications Task that the Freemain has occurred. Following the stop polling/purge for each terminal, the next terminal termination routine ($CC4TP) is invoked.

$CC4TP examines the status and mode of each terminal being de-allocated from the terminating task and sets up parameter bit settings in TAXPRM to be interrogated by $CC4TY.

Each terminal to be de-allocated requires processing by both restart routines. Depending upon the TAXPRM parameter bit settings, $CC4TY performs the following:

- If the terminal is on a switched line and disconnected during user processing (and did not reconnect), $CC4TY signs off the terminal and marks it offline. This includes, if the terminal was a requesting terminal, clearing any file specifications, resetting the terminal name in the Terminal Name Table, and informing the system operator that the terminal is now offline.
- If the terminal is still online and was a requesting terminal (or a program selected terminal that is command capable), $CC4TY re-invites input from it to allow it to command CCP.
- If the terminal was a requesting terminal and the program wanted an ended message sent to it, $CC4TY issues the message.
- $CC4TY de-allocates the terminal from the terminating tasks TCB and allocates it to the Command Processor TCB, making it available for subsequent allocation.

After the two terminal restart modules have completed processing, control is returned, via $CC4TX, to the Termination Control Routine ($CC4TD). If the control routine determines that the system is to be shut down, control is passed to $CC4EJ after all tasks are terminated. This routine quiesces all pending terminal I/O and then loads the Shutdown module, $CC5SH. After the Shutdown module is loaded into the user program area, it is relocated relative to certain locations within the resident CCP module ($CC4). Control is then given to it to complete the Shutdown processing and to return to DSM via an end of job RIB.

BASE SCP REPLACEMENTS

Pseudo Open and Pseudo Allocate

Pseudo open is the handling of the CCP user program requests to allocate and open the disk and/or the unit record files that have already been opened and allocated at startup by the standard DSM allocate and open facilities.

When a user task under CCP requests that a file be allocated or opened, the pseudo open routines perform the allocation and/or pseudo open (Diagrams 9M.0260 and 9M.0270).

Pseudo open consists of an open/close/allocate resident module ($CC4OC) and a group of transients. The main transient is the Open/Close/Allocate Initial Transient Loader ($CC4OP). This transient actually performs the file or device allocation if allocation has been requested. If open is requested this transient passes control to other transients which format the DTFs and build the IOBs for the file being opened. It then validates them against the DTFs that were opened at startup.

Special transients are called when the user task is a FORTRAN program. FORTRAN files use the pseudo tape access method even though disk files are being accessed. Two special transients are called: $CC4OH on a pseudo FORTRAN open and $CC4OT on a pseudo tape access method open. In both of these cases, control is returned to the resident pseudo open module ($CC4OC) when the open has been completed.

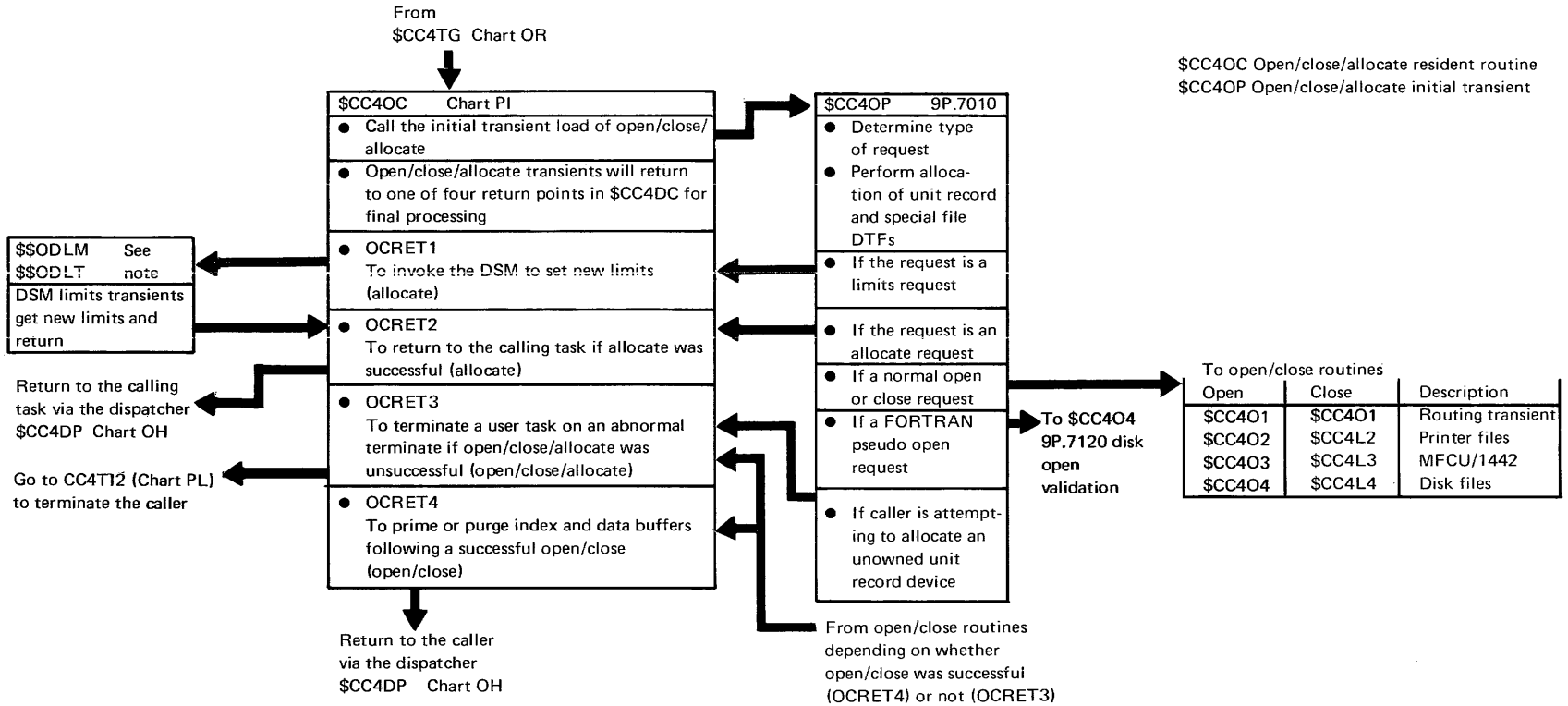
On successful completion of allocation or open $CC4OC returns control to the calling task via the Dispatcher ($CC4DP). If the allocate or open was unsuccessful, the resident module passes control to the Termination Interface Routine ($CC4TI), which causes termination of the user task.

Note: Unlike DSM, CCP open (and close) requires only one pass of all modules to process all DTFs to be opened (or closed).

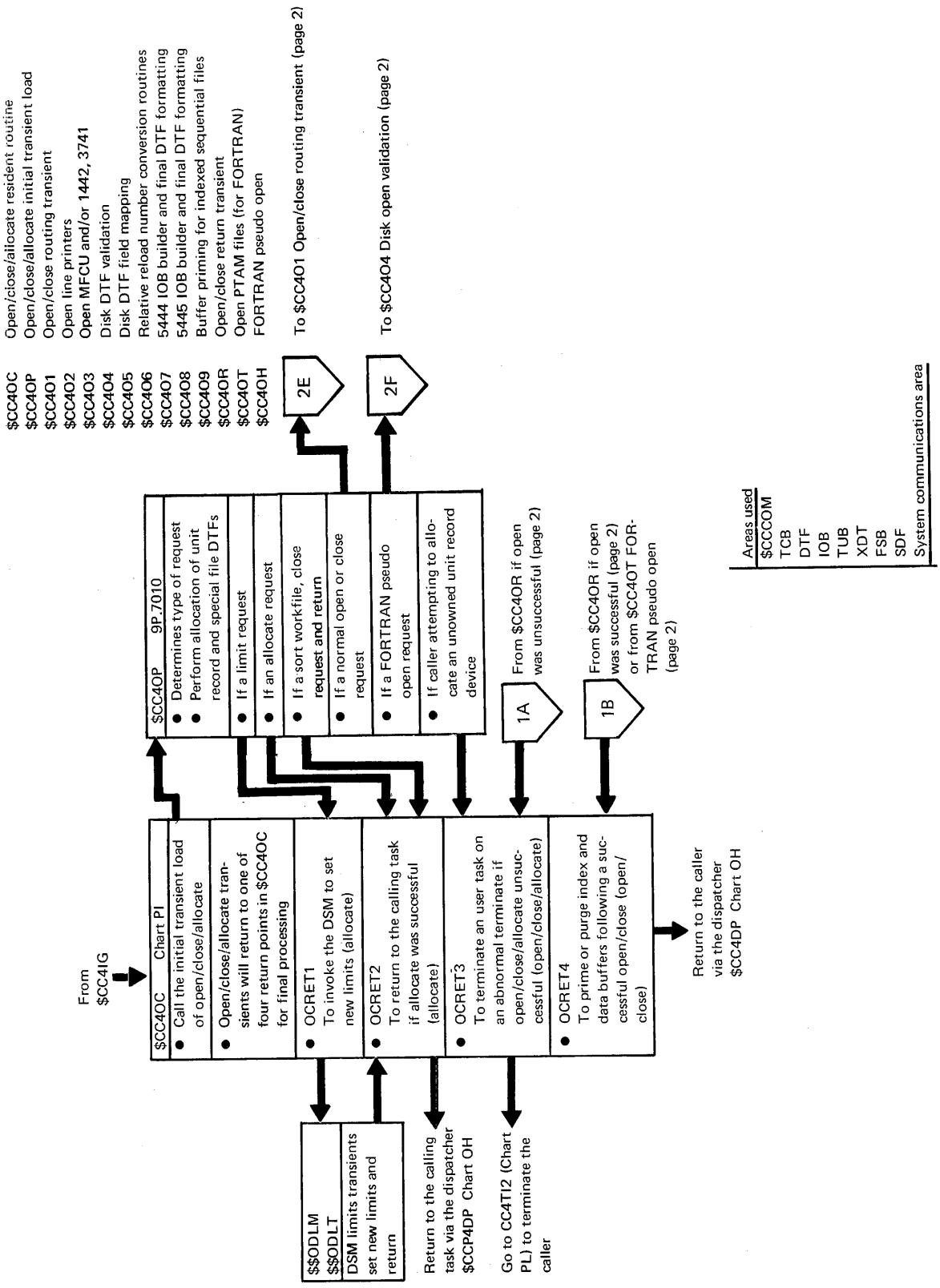
If an open request is received for a sort work file, a multivolume file extent table is created from the DTF and control is returned to resident open module ($CC4OC).

When an allocate request is received from a sort program and symbolic files are being used, the symbolic files are resolved at this time instead of at open time.

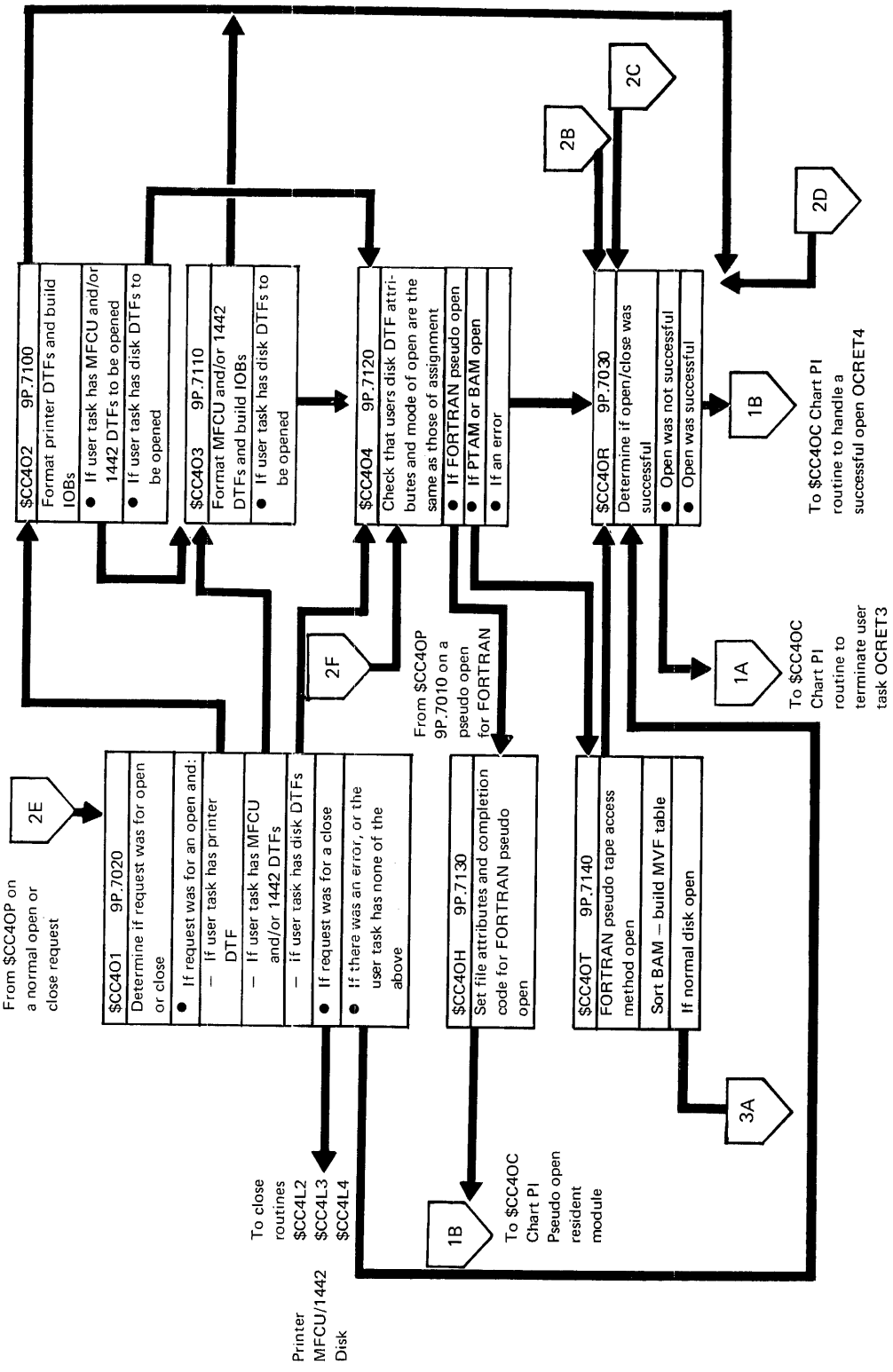
Diagram 9M.0260. Control Flow of Pseudo Allocate



Note: See *IBM System/3 Disk Systems Data Management and Input/Output Supervisor Logic Manual*, SY21-0512.



● **Diagram 9M.0270 (Part 1 of 3). Control Flow of Pseudo Open**



- **Diagram 9M.0270 (Part 2 of 3). Control Flow of Pseudo Open**

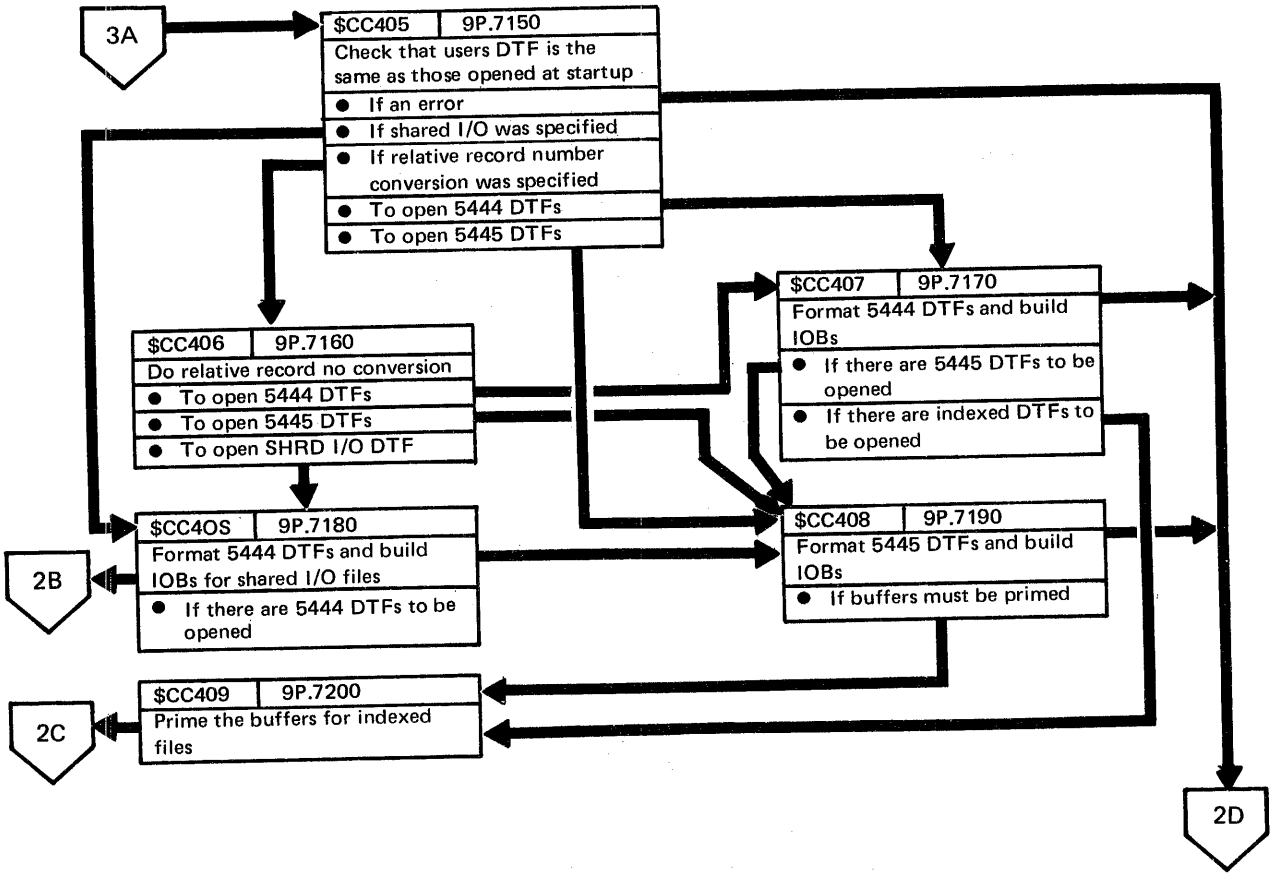


Diagram 9M.0270 (Part 3 of 3). Control Flow of Pseudo Open

Pseudo Close

Pseudo close (Diagram 9M.0280) handles CCP user task requests to close any disk and unit record files.

As no file is DSM-closed by CCP until Shutdown, a pseudo close is necessary to complete an individual program's processing of the files that it has used.

When CCP user tasks request a close, pseudo close handles the user's request and indicates that the files are no longer in use. It is, however, only at termination of CCP that the files are actually de-allocated from the program level.

Pseudo close consists of a resident module ($CC4OC, the same module used by open and allocate) and a group of transients. The primary purpose of $CC4OC is to load the transient loader ($CC4OP, the same module used for open and allocate). This transient passes control to other pseudo close transients to close the files, according to the types of files the user task is closing.

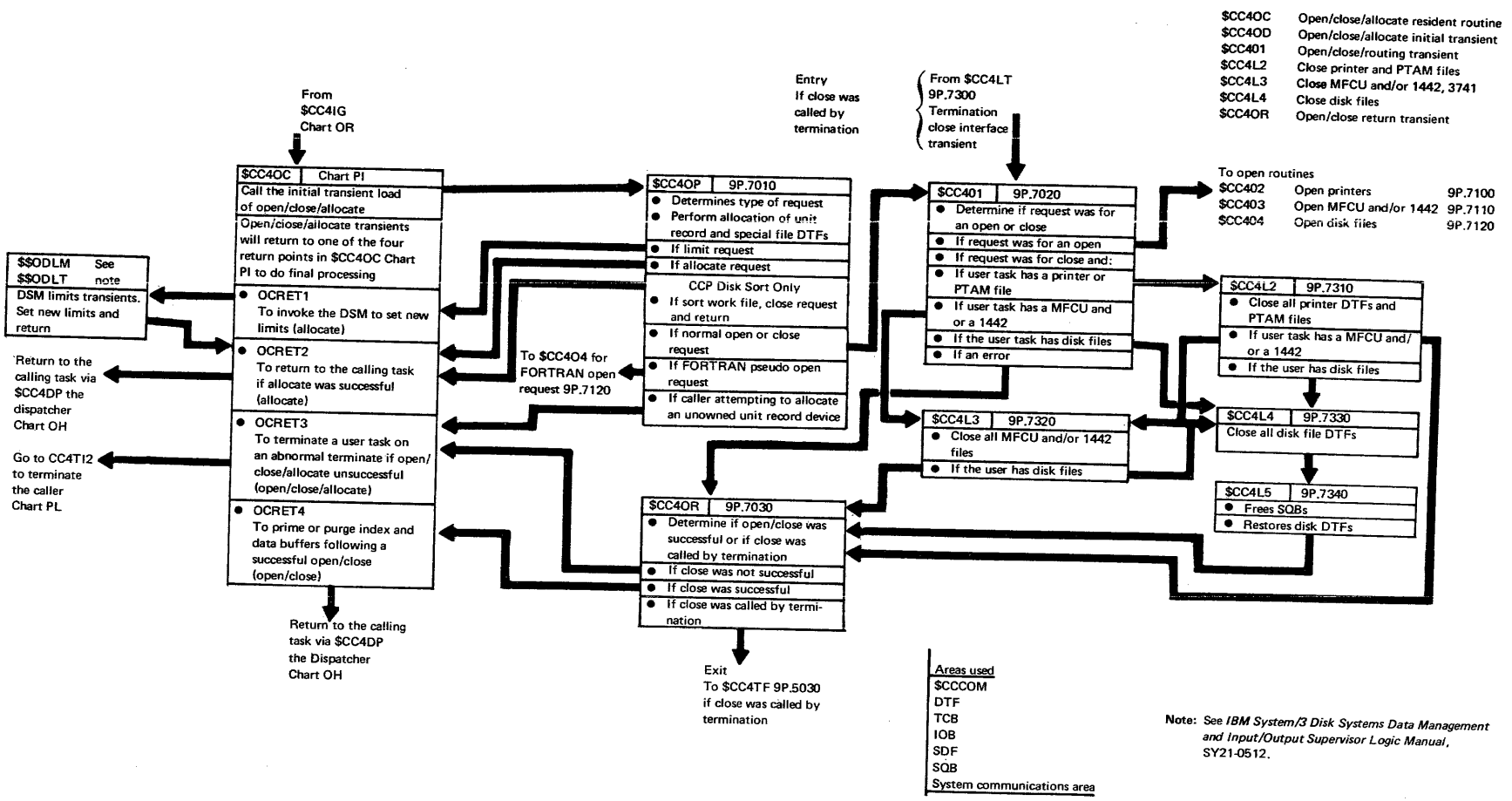
When a sort program is closing the sort work f le, control is returned immediately to the resident module since this is a special case DTF.

No special transients are called for FORTRAN user tasks as in pseudo close, however, FORTRAN pseudo tape access method close routines are in the Close Printer File Transient ($CC4L2).

On a successful close the resident module ($CC4OC) returns control to the user task via the Dispatcher ($CC4DP). If close was unsuccessful, the resident module ($CC4OC) passes control to the Termination Interface Routine to terminate the user task.

Some of the pseudo close transients are used by the Termination Task when a user task goes to end of job or has an abnormal termination.

Diagram 9M.0280. Control Flow of Pseudo Close



CCP End of Job

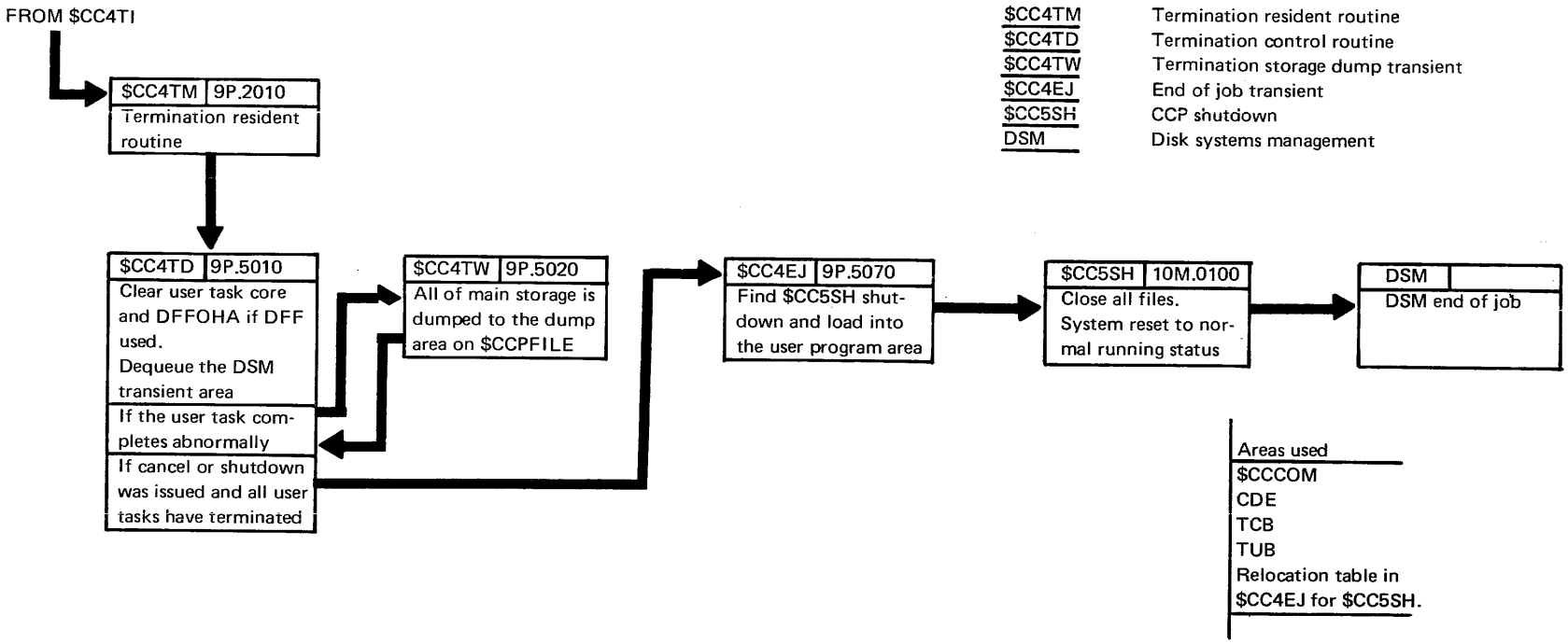
CCP end of job (Diagram 9M.0290) is a group of modules that control the termination of CCP and bring it to end of job.

When a Shutdown command is issued, the user tasks that are in operation at the time the command was issued are permitted to complete their operations. However, when CCP accepts the command, it prevents any further program requests or terminal type commands except those issued in a command interrupt mode.

When a Cancel CCP command is issued, all user tasks are immediately stopped and CCP is terminated via the normal Shutdown processing.

When either the Shutdown or Cancel CCP command is issued, the Termination Task is dispatched and the Terminator Resident Routine ($CC4TM) calls the Termination Control Transient ($CC4TD). The Termination Control Transient calls the End of Job Transient ($CC4EJ). This transient finds the Shutdown module $CC5SH and loads it into the user program area and then transfers control to it. $CC5SH closes all files, resets the system to normal running status, and calls DSM end of job.

Diagram 9M.0290. CCP End of Job (Shutdown)



Halt/Syslog

One of the functions of General Entry Intercept ($CC4IG) is to analyze all halt/syslog requests issued during the Operational Stage of CCP. Depending on the results of the analysis, one of the following actions will be taken:

- If the halt/syslog request came from the non-CCP program level, the request is passed to DSM for normal handling.
- If the halt/syslog request came from the CCP level, the proper CCP halt/syslog transient is called to handle the request.
  - If the RIB is X'8F' (RPG II halt), the transient $CC4HF is called.
  - If the RIB is X'85' or X'C5' (halt/syslog) and the request is not from the Communications Manager, the transient CC4H1 is called.
  - If the RIB is X'85' or X'C5' (halt/syslog) and the request is from the Communications Manager, the transient $CC4H2 is called.

CCP controls all halt/syslog requests from the CCP program level; DSM controls all halt/syslog requests from the non-CCP program level. All the CCP halt/syslog transients consider the log device for the CCP program level at all times to be the console.

The various classes of CCP halt/syslog requests and the action taken by CCP on each are described in Diagram 9M.0295. Diagram 9M.0300 is an overview of CCP halt/syslog logic flow.

Requestor	Type of halt	CCP action
User task	None	Log the request on the console.
User task	Unit record hardware error	Log the request on the console and display the halt.
User task	Non-unit record hardware error	Log the request on the console and cancel the user program.
System task	None	Log the request on the console.
System task	MLMP IOCS halt (informational)	Take automatic option to the halt and return
System task	Non-MLMP IOCS halt (informational)	Cancel CCP-Issue U– halt exit to CPHALT via branch to 0000

Diagram 9M.0295. CCP Halt/Syslog Actions

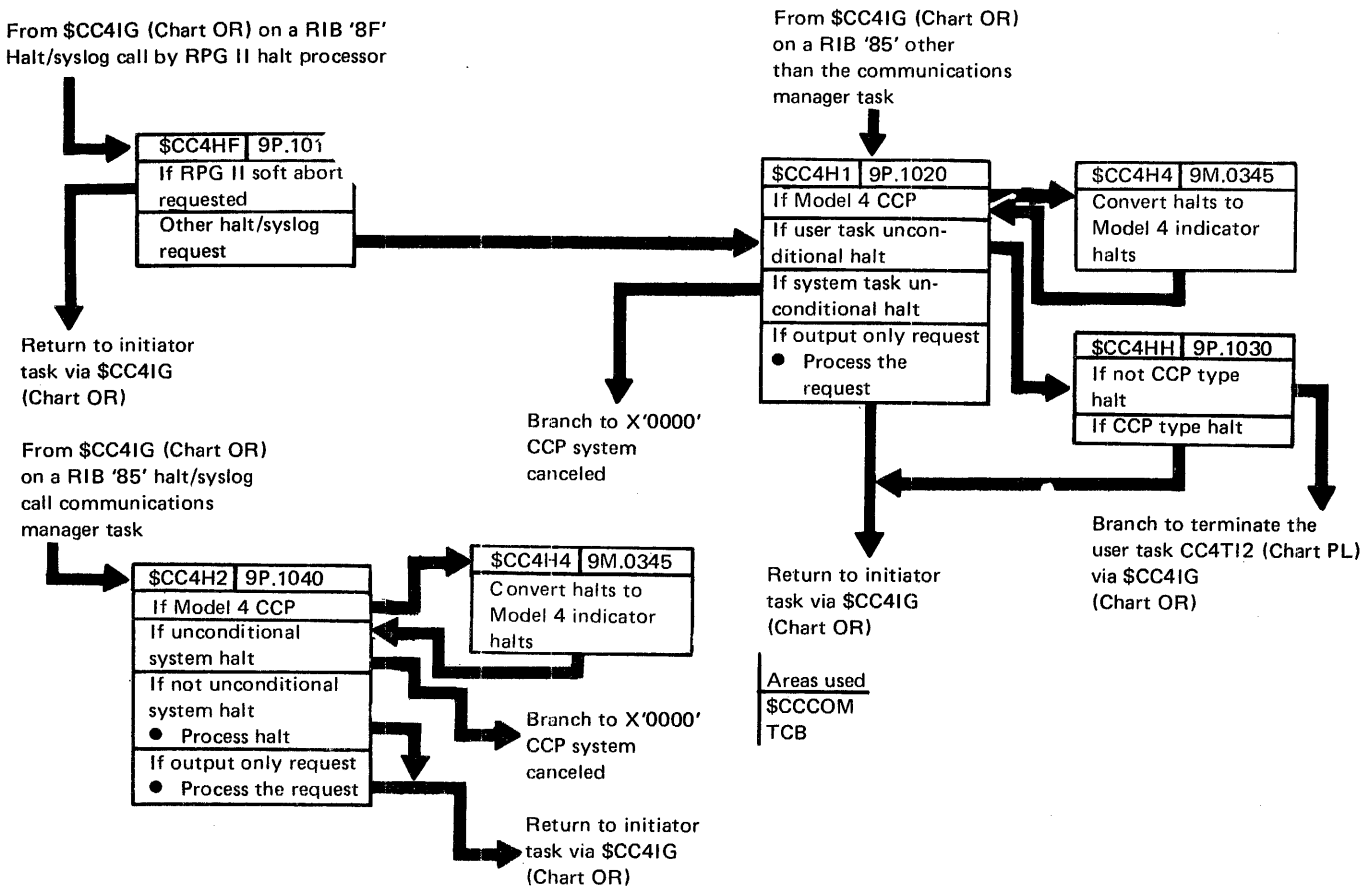


Diagram 9M.0300. Halt Syslog Flow

USER INTERFACE TO CCP

Basic Assembler Language Program Interface

The Basic Assembler programmer is responsible for understanding the user requirements and the CCP system actions (Diagram 9M.0340).

The Assembler programmer must define one or more parameter lists in the program to specify communications operations to be performed, and must also define one or more record areas in which a symbolic terminal name and input/output data is to be contained.

After ensuring that the required information has been specified for a communications operation and having set the address of the parameter list in index register 2, the Assembler programmer must branch to the DSM supervisor's General Entry with an in-line RIB and sub-RIB. Upon return of control to the instruction immediately following his branch, the programmer must examine information returned from the operation.

Because the Assembler language can perform all the user functions of the Standard Communications Interface, such as RPG II SUBR93, no Communications Service Subroutine is supplied with CCP to perform any of those functions. All user operations are performed in the source program module itself.

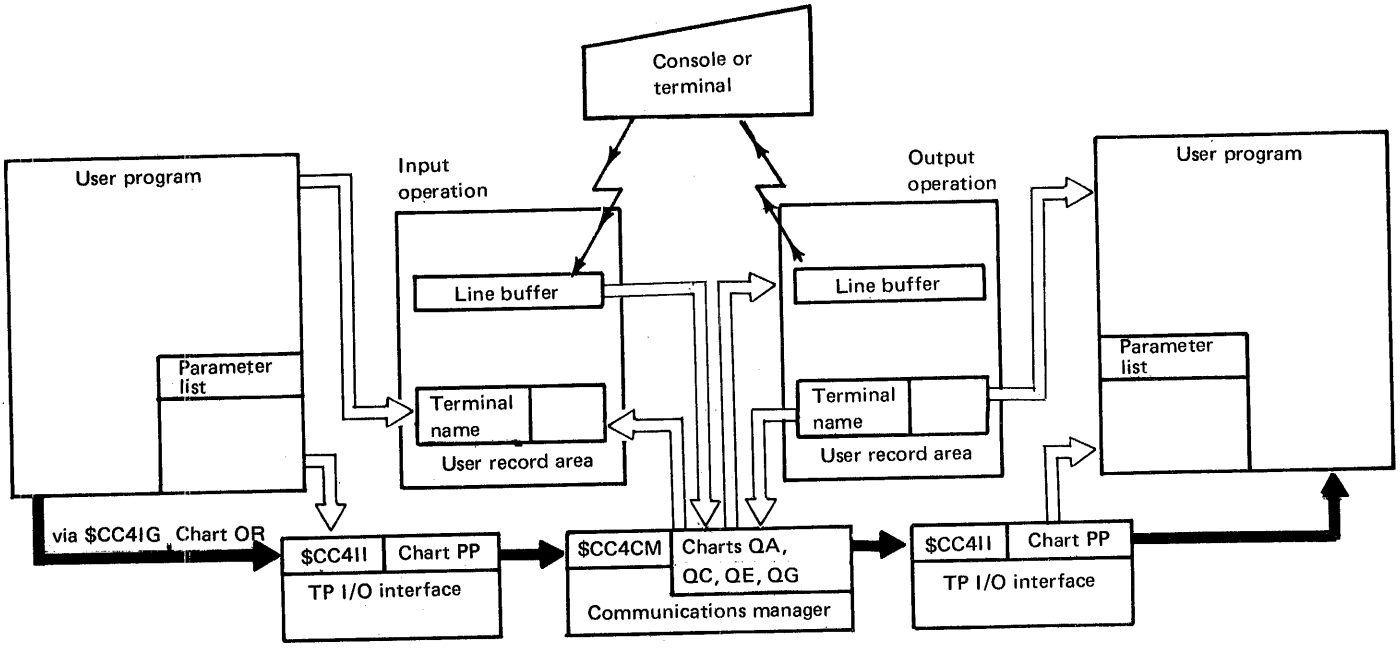


Diagram 9M.0340. Assembler TP I/O Operation Flow (without DFF)

However, as an aid to the Assembler programmer, several macros are supplied by CCP for the generation of the appropriate code in the source program. Each of the supplied macro-definitions is distributed as a source library member to be incorporated in the source library of the user's program-preparation pack.

These macros constitute the only special facilities provided for the Assembler user.

Six macros are provided for the Assembler user's programming. Four of these macros generate equates for values of significance to communications programming under CCP:

1. $NCOM – Generates common equates for registers and certain DSM supervisor values.

2. $NPLO – Generates equates for the offset of each field in a communications parameter list.

3. $NOPV – Generates equates for the value of each operation code and operation modifier.

4. $NRTV – Generates equates for the value of each return code that is issued by CCP.

A fifth macro, $NPL, is provided for the generation of a parameter list with the initial value of each field specifiable by the programmer (9M.0350).

The sixth macro, $NCIO, is provided for the request of a communications operation (9M.0360). This macro permits the programmer to specify, in a macro instruction:

- The address of his parameter list, to be set in index register 2 if not already there.

- The operation code and modifiers to be set for the operation (if required) or the location at which they may be found.

- The output length or attributes identifier to be set for the operation (if required) or the location at which they may be found.

- The maximum input length to be set for the operation (if required) or the location at which it may be found.

- The address of the record area to be set for the operation (if required) or the location at which it may be found.

- The symbolic terminal name to be set for the operation (if required) or the location in the machine at which it may be found.

- Whether to set the fields specified and then branch to General Entry to perform the operation, or simply to set the specified fields without branching to General Entry (without requesting the operation be performed at this time).

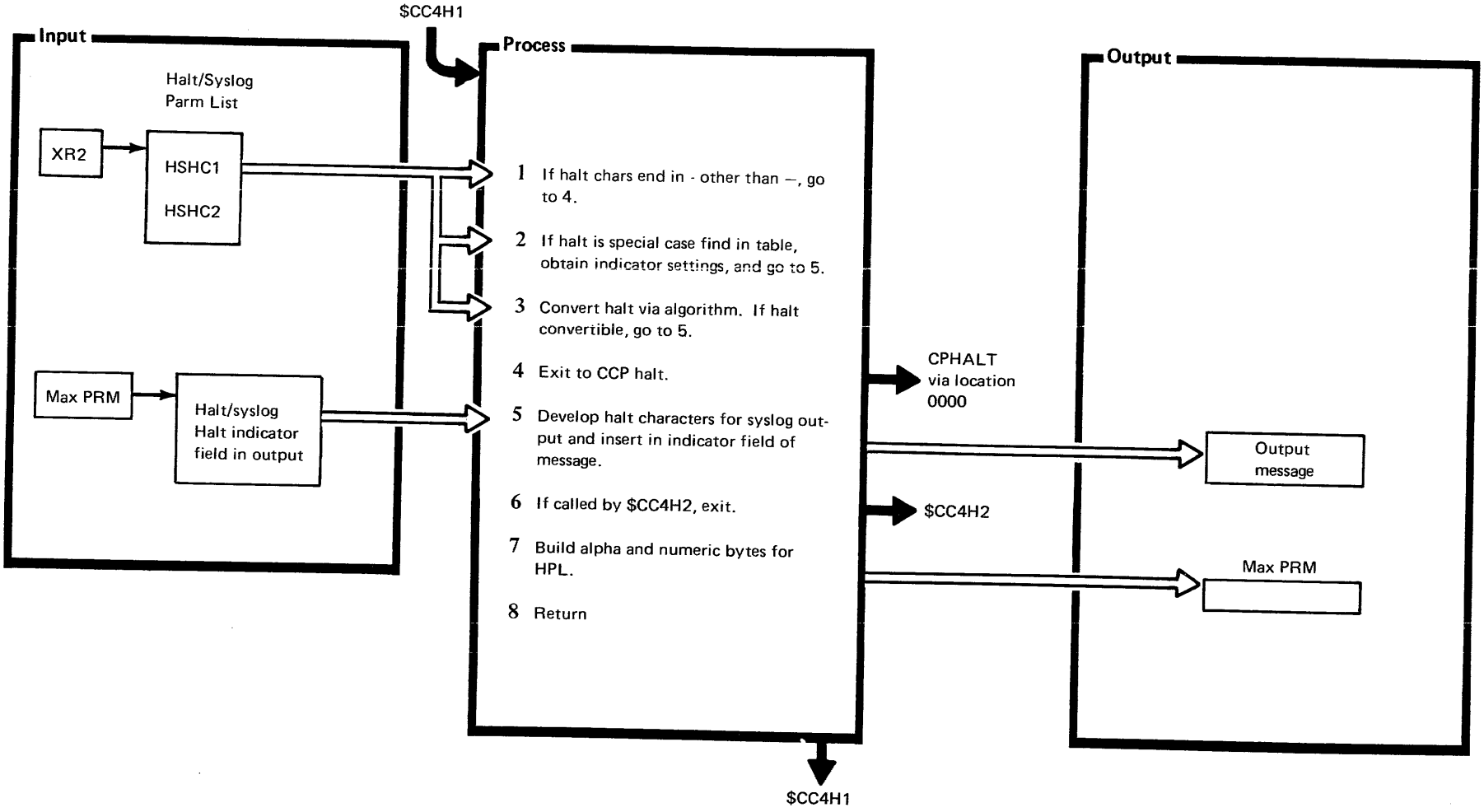
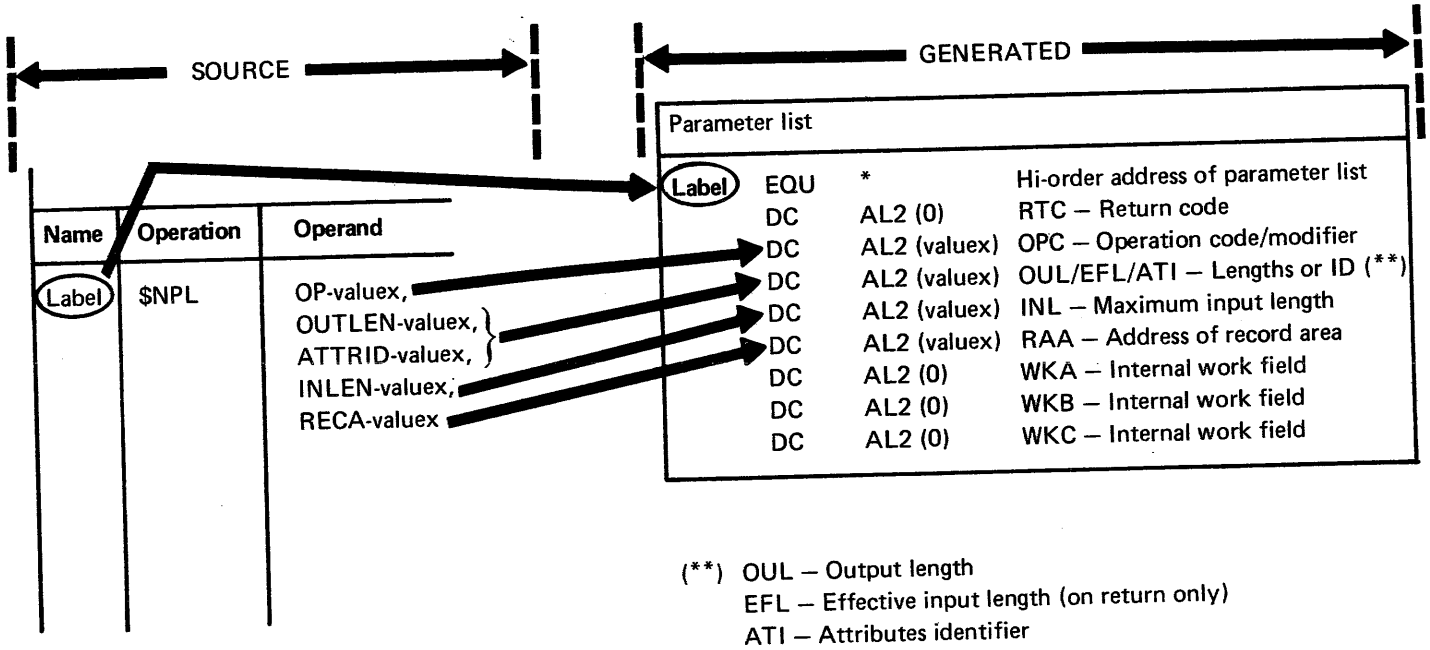


Diagram 9M.0345. $CC4H4 (Model 4 only) Halt Conversion and Syslog Message Transient



(**) OUL – Output length
EFL – Effective input length (on return only)
ATI – Attributes identifier

Diagram 9M.0350. $NPL Macro Flow

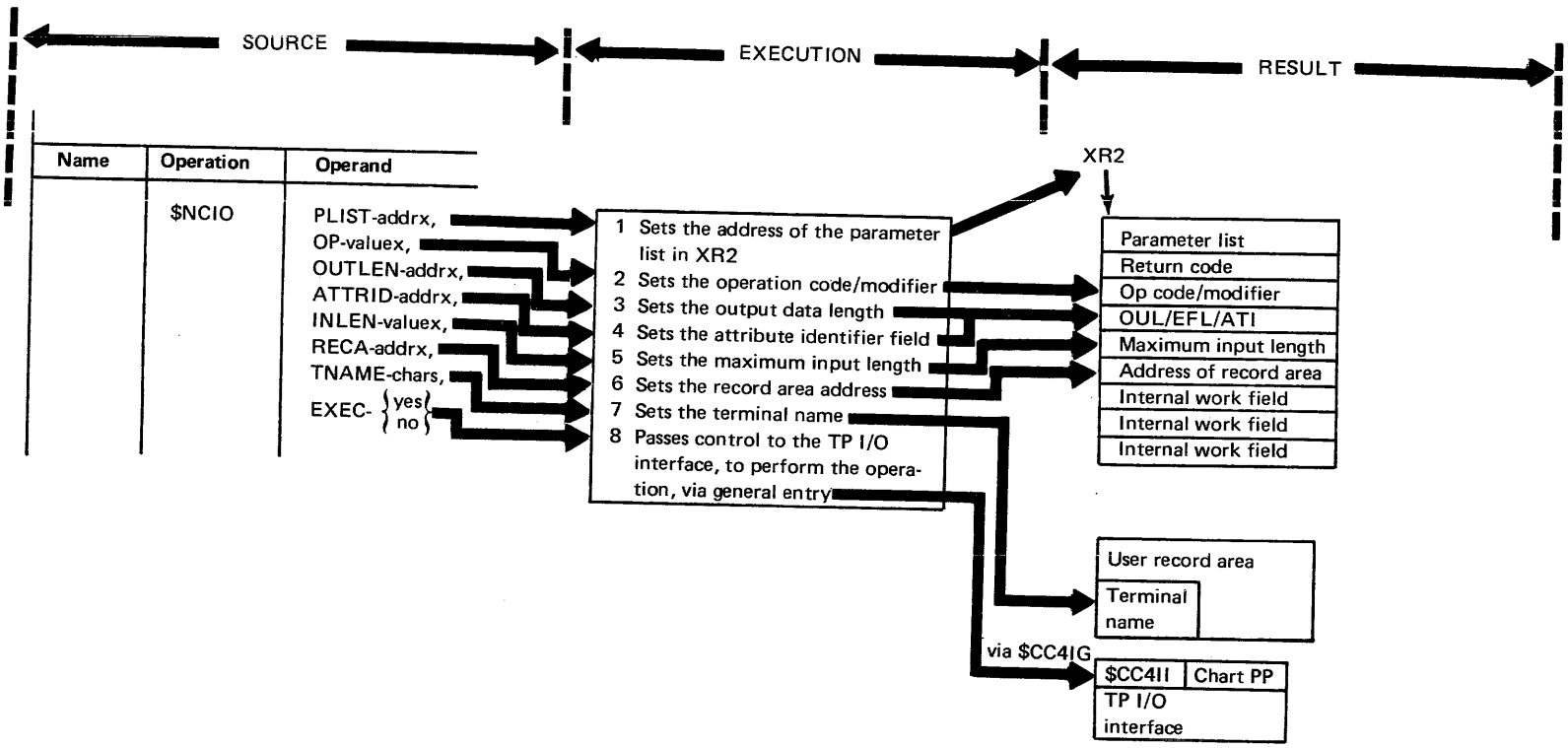


Diagram 9M.0360. $NCIO Macro Flow

RPG II Program Interface

Of the four RPG II subroutines provided by CCP, two of them perform I/O (SUBR92 and SUBR91), and the other two provide service functions to the RPG II program (SUBR90 and SUBR93).

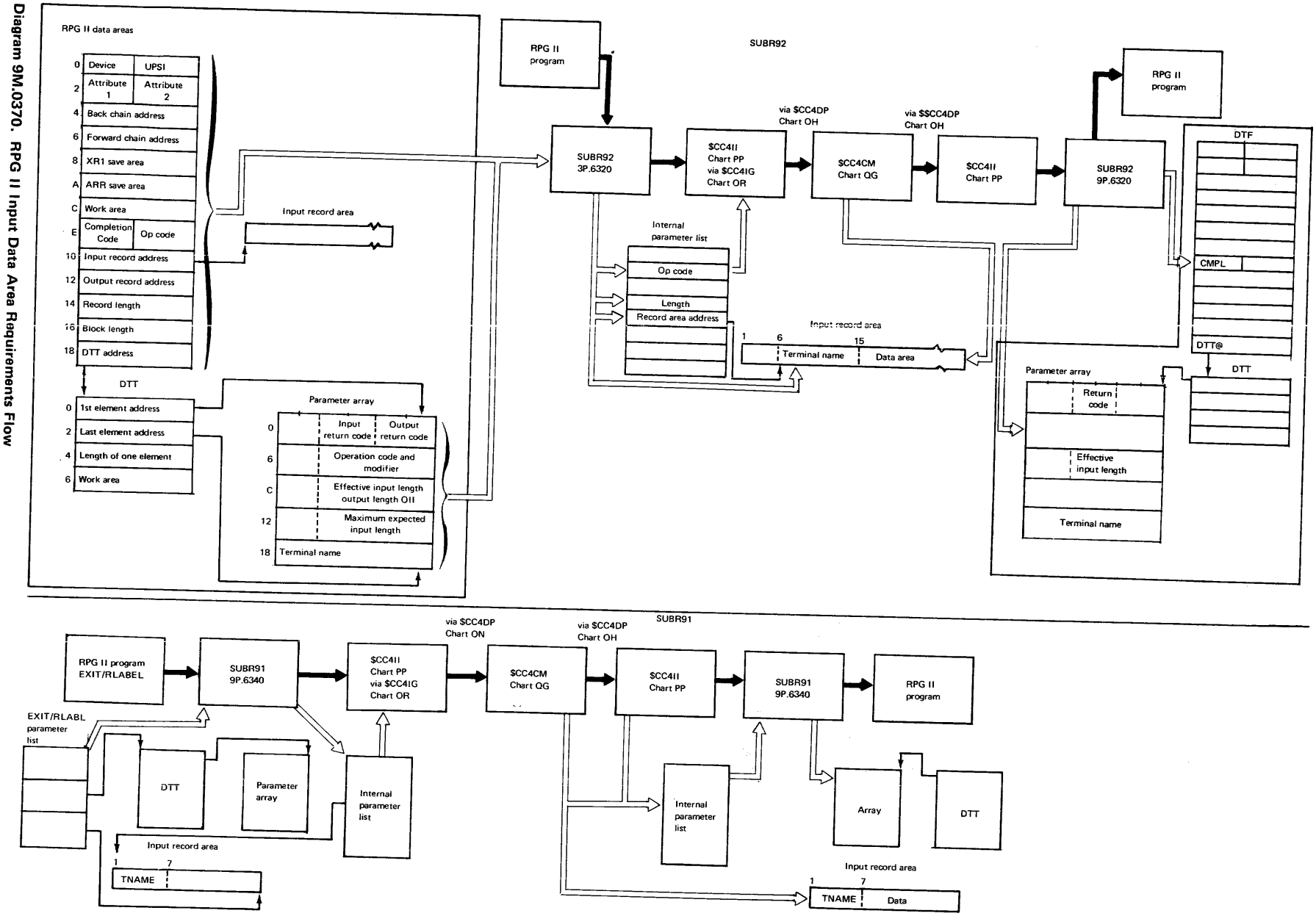
SUBR93 (Diagram 9P.6330) is used with a SPECIAL DTF. It requires no parameters. The second time the subroutine is called (either with the same file or a second file) it will be marked as being at end of file. The record size may be any length. A File Description Extension Specification is not required or used by this routine.

SUBR90 (9P.6350) is used to move fields into a large field or to move data from a large field or array to smaller fields. This subroutine is used with EXIT/RLABL. It should be used to build up a record area for CCP, or to break down a record area into individual fields. Fields are moved continuously from left to right (high order to low order) to (or from) the record area from (or to) the individual fields. A diagnostic is provided (termination code 20) in case the record area is too small to contain all the fields to be moved into it. However, unpredictable results could occur if the length of all the fields is too small to contain all the data in a record area.

SUBR91 is used with EXIT/RLABL and requires only two RLABLs, one to designate the parameter array and the other for the record area. The record area used with this routine has the same structure as that defined for the other language interfaces: that is, terminal name in the first six (leftmost) positions followed by the data. A parameter array is used with this routine. The array has four 6-position elements. The structure of the array is given in the RPG II Parameter Array in *Appendix B.* The information in the array is converted to the form expected by CCP and placed in a parameter list internal to this routine. Diagrams 9M.0370 and 9M.0380 show the logic and data flow for the routine. The functional diagram for SUBR91 is given in 9P.6340.

SUBR92 (9P.6320) is used only with RPG II SPECIAL files to perform TP I/O via CCP. The subroutine is link-edited into the user RPG II program as part of the RPG II compilation. This subroutine uses a parameter array, as described in the RPG II Parameter Array in *Appendix B.* The array is associated with the SPECIAL file by means of the RPG II File Description Extension Specification. For all input operations, information about the operation is derived from the parameter array. For an output file operation, the operation information is derived from the output record area. The information for an operation is decoded to internal form and placed in a parameter list contained within SUBR92.

Diagram 9M.0370. RPG II Input Data Area Requirements Flow



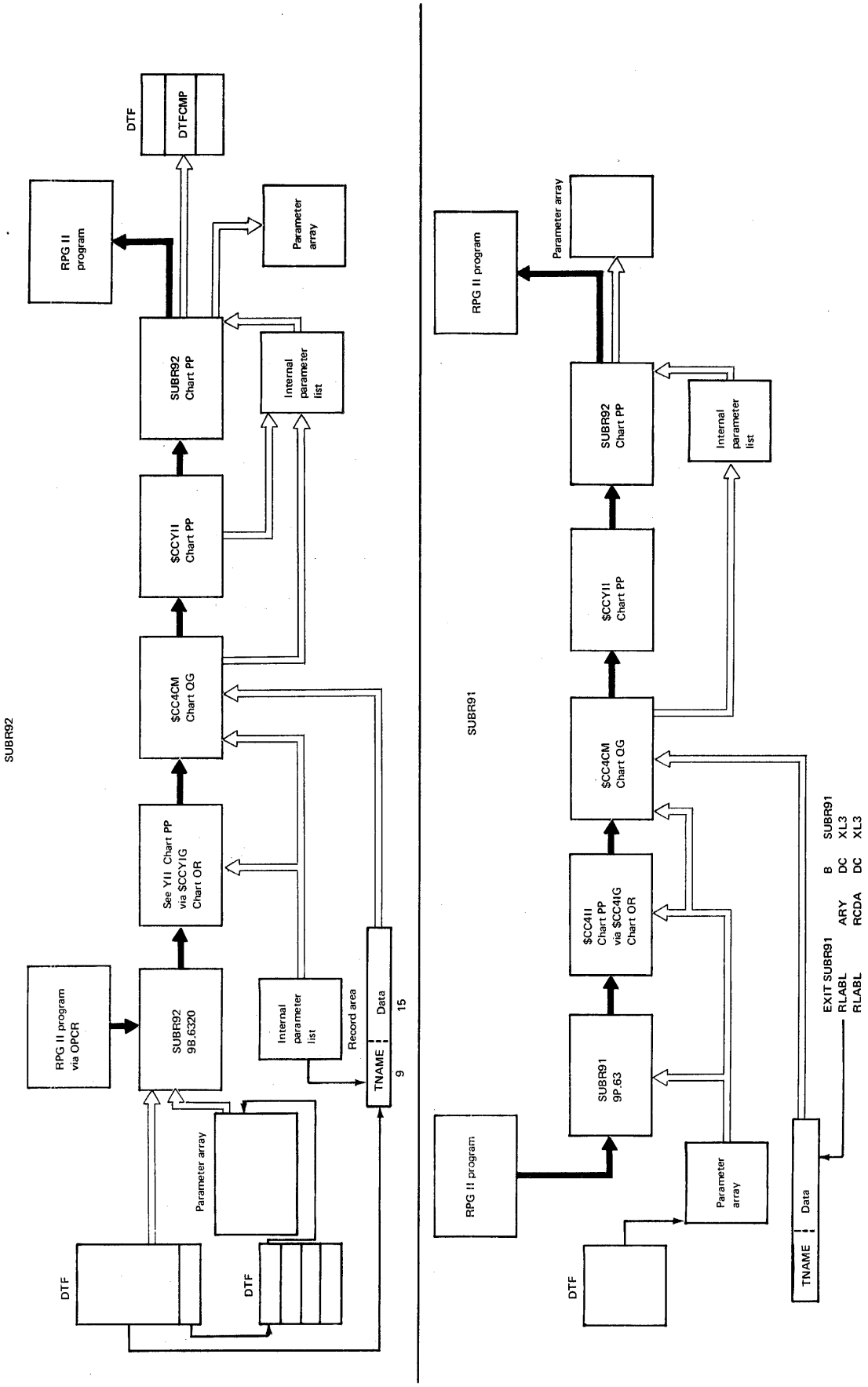


Diagram 9M.0380. RPG II Output Data Area Requirements Flow

The one exception to the placement of all information for an output operation in the record area is the invite input, in which case the expected input length is taken from the parameter array. In this case, positions 5 through 8 are not used in the output record area.

For the put part of a put-then-get operation the output record area is saved in the physical input area, the output data length of the operation is saved in the block length area in the DTF, and bit 0 of the second attribute byte in the DTF is set on to indicate this situation has occurred. The very next input operation, regardless of the op code in the parameter array, will be forced to the get part of the put-then-get operation. The symbolic terminal name from the put part is used; the name in the parameter array is ignored. The put-then-get is actually performed on the get part.

An input operation for SUBR92 is one which comes through RPG IPCR (READ op code, primary, secondary file). An output operation is one which comes to SUBR92 through RPG OPCR (EXCPT op code, 1P, detail, total output).

For an output operation with blanks provided by the user for a terminal name, the symbolic terminal name of the requesting terminal is placed in the parameter array after the operation. For all input operations except the get part of a put-then-get operation, the symbolic terminal name of the terminal for which the operation just completed is placed in the parameter array.

Indicator 91 is set on for any error return code, and indicator 92 is set on for any exception return code. They are never set off by SUBR92.

For operation codes S, H S, W, H W, two separate operations are sent through CCP.

The data area definition of the RPG II SPECIAL DTF is given in *IBM System/3 Disk Systems RPG II Logic Manual*, LY21-0501. The second attribute byte of the SPECIAL DTF after RPG II open of files has the following meaning:

Bit	Value	Meaning
0	1	Put pending for put-then-get
1	1	Allocated file
7	1	File opened

For the get attributes operation through both SUBR92 and SUBR91, the information which is normally returned in the parameter list is not available to the RPG II program.

COBOL Program Interface (Models 8, 10, and 12 Only)

The COBOL user program requests a communication operation by a CALL to the communications service subroutine CCPCIO, which is link-edited with the user program (Diagram 9M.0420). As arguments of the CALL, the user program passes to CCPCIO the address of a communications parameter list, defined in the COBOL working-storage section, and the address of a record area, also defined in the working-storage section. The parameter list must already contain the operation code and modifier for the operation to be performed, and whichever data length value is required by that operation. The record area must already contain a symbolic terminal name, if the operation requires one, a display format name if the operation is a DFF put, and any output data if required.

The subroutine CCPCIO first completes the user's parameter list by setting the user-passed record area address into the fifth element; then, after setting the address of the user's parameter list in index register 2, it performs a branch to General Entry, with a RIB of X'01' followed by a sub-RIB of X'01', signifying a communications operation request.

The branch is intercepted by CCP's General Entry Intercept ($CC4IG), which recognizes the RIB as specifying a CCP function request, and the sub-RIB as indicating that a user program communications operation is being requested.

Control is passed by $CC4IG to $CC4II, CCP's communications I/O interface routine, to carry out the user program's request. $CC4II continues to run under the TCB of the user program's task. $CC4II invokes the Communications Management Task where appropriate (as described earlier) and the user task waits for completion of the requested event.

When the operation requested is complete (note that invite input and put-no-wait operations are complete as soon as the request has been accepted by the Communications Management Task), the user task, which had waited within $CC4II, is again made dispatchable. $CC4II returns control to the subroutine CCPCIO. At this point the return code has already been set in the user's parameter list and, depending on the operation, the third element of the parameter list (effective input length on an input operation, count of outstanding invites on a release operation) has been set as well. Further, if the symbolic terminal name in the record area is to be set by the current operation, it too has already been set.

The subroutine CCPCIO has nothing further to do at this time than return control to the COBOL mainline program at the point from which the subroutine was invoked.

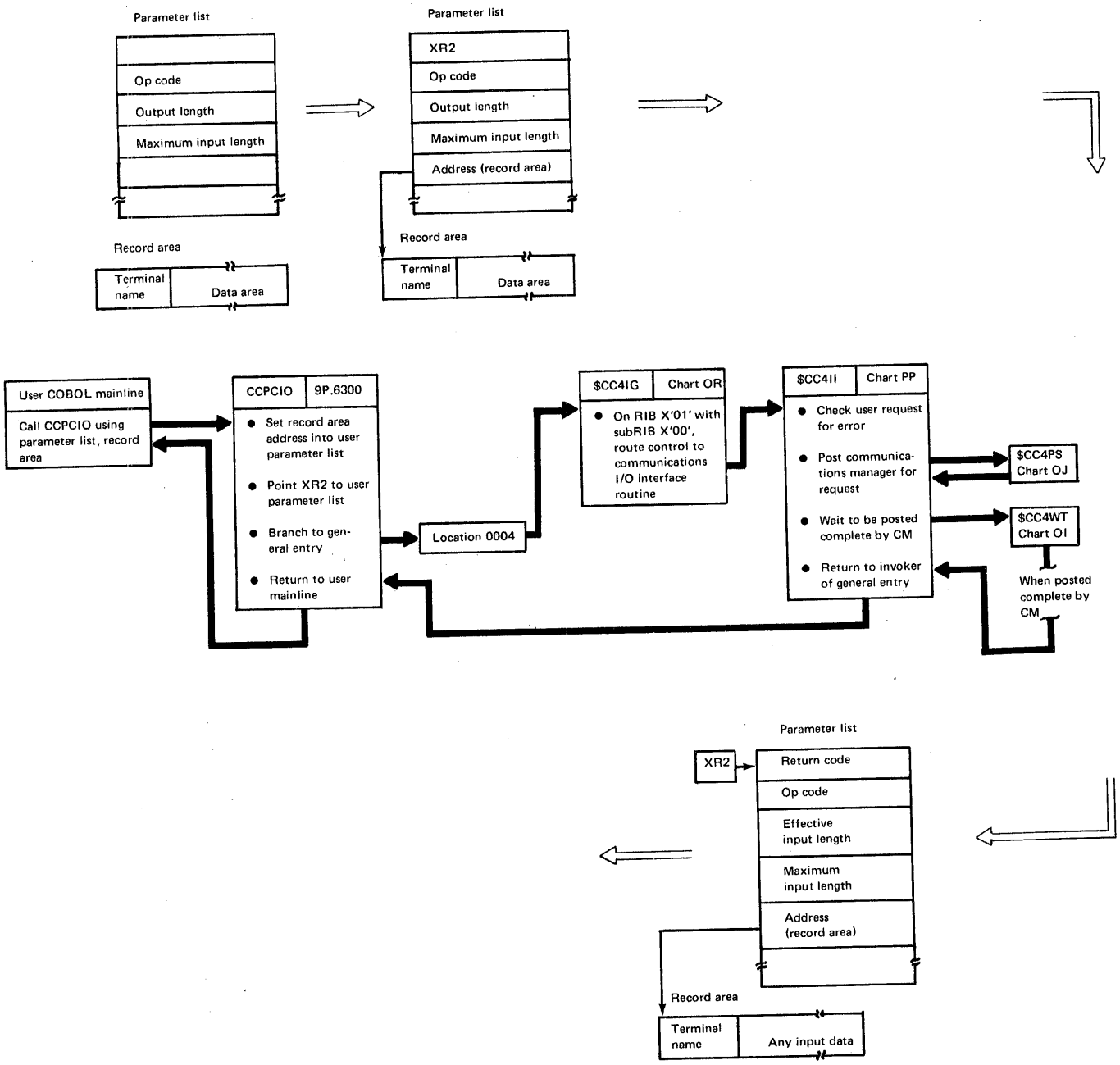


Diagram 9M.0420. COBOL User Interface

FORTRAN Program Interface (Models 8, 10, and 12 Only)

The FORTRAN user program requests a communications operation by a CALL to the communications service subroutine CCPFIO, which is link-edited with the user program (Diagram 9M.0430). As arguments of the CALL the user program passes to CCPFIO the address of an INTEGER*2 array which is the communications parameter list, and the address of another INTEGER*2 array which is the record area.

The parameter list must already contain the operation code and modifier for the operation to be performed, the output data length if the operation is a put, and must contain in the fourth element (for any operation) the number of elements in the record area array.

Because the CCP communications I/O interface does not use READ/WRITE and FORMAT statements, data in the record area array must be, for output, already edited into the form in which it will be sent to the terminal, and will, upon the completion of input operations, contain the raw data sent from the terminal. To facilitate the editing of data via the FORTRAN-supplied Commercial Subroutines, the data in the record area is always treated by CCPFIO as being carried one character per array element — that is, one character per two bytes.

Because the CCP communications routines do not deal with data in this format, the subroutine CCPFIO, before passing on the user's request to the CCP control program, packs the data in the record area to one character per byte (standard) format. CCPFIO also sets the address of the record area array in the fifth element of the user's parameter list, and sets the address of the user's parameter list in index register 2.

CCPFIO then branches to General Entry with a RIB of X'01' followed by a sub-RIB of X'00', signifying a communications operation request.

The branch is intercepted by CCP's General Entry Intercept ($CC4IG), which recognizes the RIB as specifying a CCP function request, and the sub-RIB as indicating that a user program communications operation is being requested.

Control is passed by $CC4IG to $CC4II, CCP's communications I/O interface routine, to carry out the user program's request. $CC4II continues to run under the TCB of the user program's task. $CC4II invokes the Communications Management Task where appropriate (as described earlier) and the user task waits for completion of the required event.

When the operation requested is complete (note that invite input and put-no-wait operations are complete as soon as the request has been accepted by the Communications Management Task), the user task, which had waited within $CC4II, is again made dispatchable. $CC4II returns control to the subroutine CCPFIO. At this point the return code has already been set in the user's parameter list and, depending on the operation, the third element of the parameter list (effective input length on an input operation, count of outstanding invites on a Release operation) has been set as well. Further, if the symbolic terminal name is to be set by the current operation, it too has already been set.

When the subroutine CCPFIO receives the return of control from $CC4II, the user's record area array is still packed. CCPFIO unpacks the record area (reverting the form to one character per two bytes), then returns control to the FORTRAN mainline program at the point from which the subroutine was invoked.

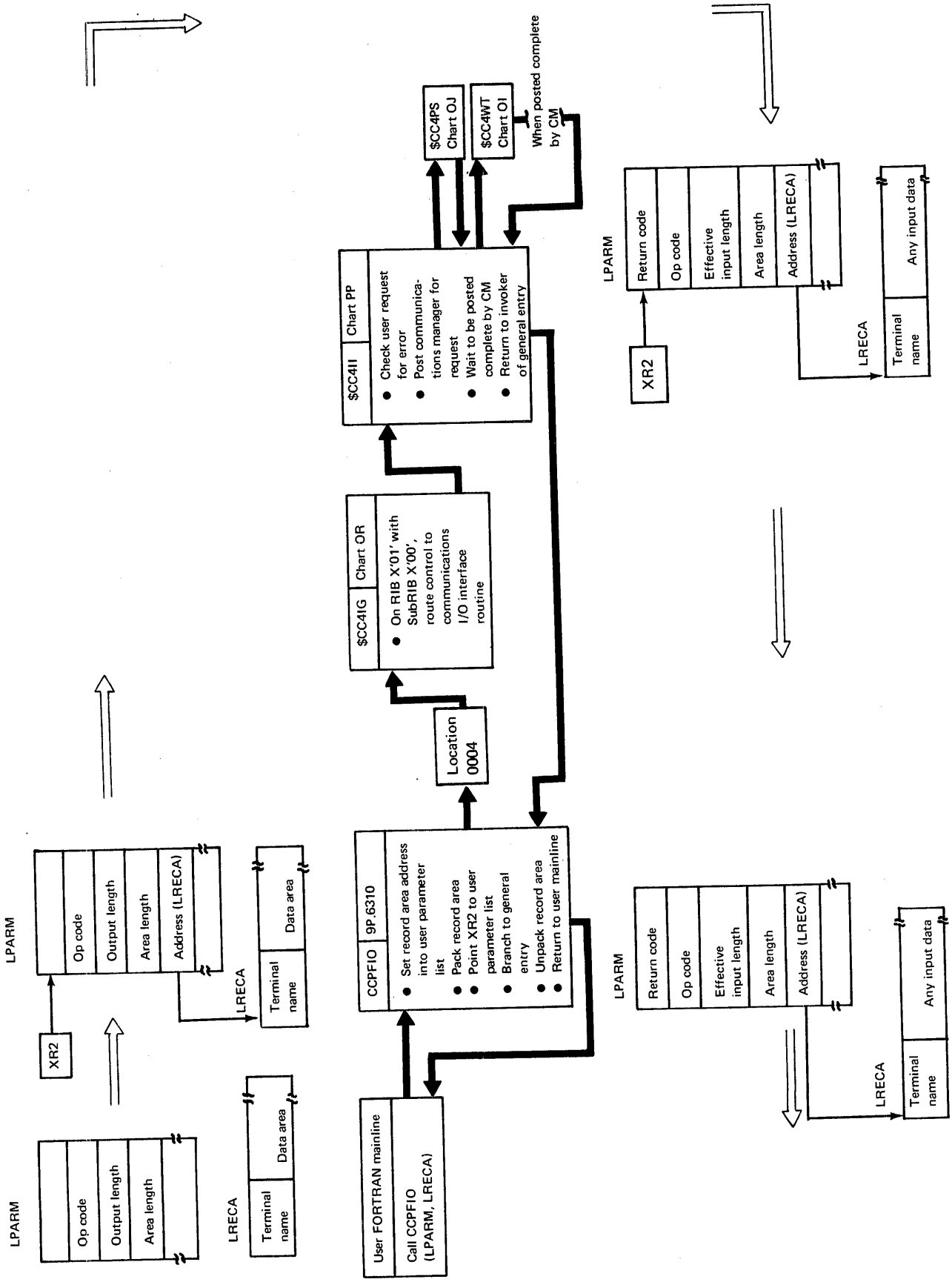


Diagram 9M.0430. FORTRAN User Interface

CCP EXECUTION TIMELINES

Accept Timeline

Diagram 9M.0440 shows the flow of control during an accept input operation when data is ready to satisfy the request.

The flow begins with the user program issuing the accept operation. This causes a branch to real location 4 that causes $CC4IG to gain control. RIB 01 with sub-RIB 00 indicates that the request should be routed to the user I/O interface module, $CC4II.

$CC4II determines that there is data to satisfy the accept and that there is no need to wait or ask for further functional assistance. Therefore, all that remains is to get control back to the caller with the data record.

Since $CC4II does not know where the caller is, as the caller was similarly unaware of $CC4II's location, return must be via the dispatcher, $CC4DP. $CC4IG and $CC4DP must save necessary user information. The caller's environment is thus restored and he continues processing knowing only that the data he expected was received.

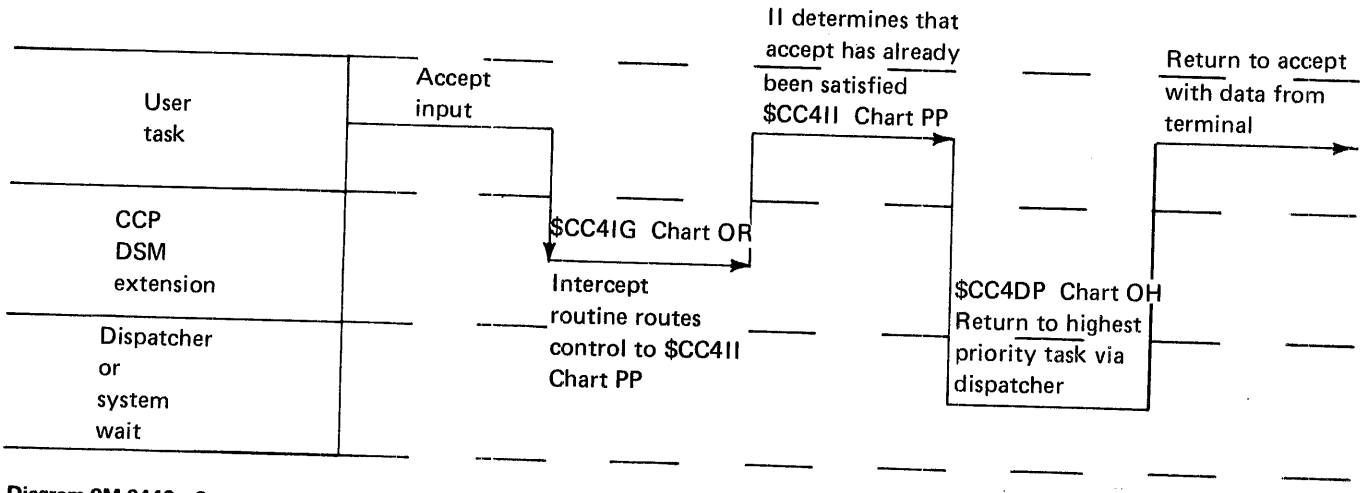


Diagram 9M.0440. Control Flow of Accept Input Operation

Get Timeline

Accept and get have both similarities and differences as they appear to a user. There are also internal similarities and differences. The similarity the user sees is that following the get statement the user knows that he has data. The basic internal difference is associated with the multiple-requesting-terminal (MRT) program in that the accept can cause a get to actually receive data prior to the issuance of the request.

Diagram 9M.0450 shows the flow of control through a typical get operation. Assume that the user program wants to obtain a transaction from the operator at the multi-point terminal.

Again, the sequence begins with a branch to location 4, RIB of 01, sub-RIB of 00, intercepted and routed by $CC4IG to $CC4II (II). Here the internal similarity between accept and get ends. II needs to acquire the data for the user. II informs the Communication Manager ($CC4CM) of this need via a post operation. By calling $CC4PS (PS), $CC4CM (CM) can be made active. Parameter passing is via internal conventions.

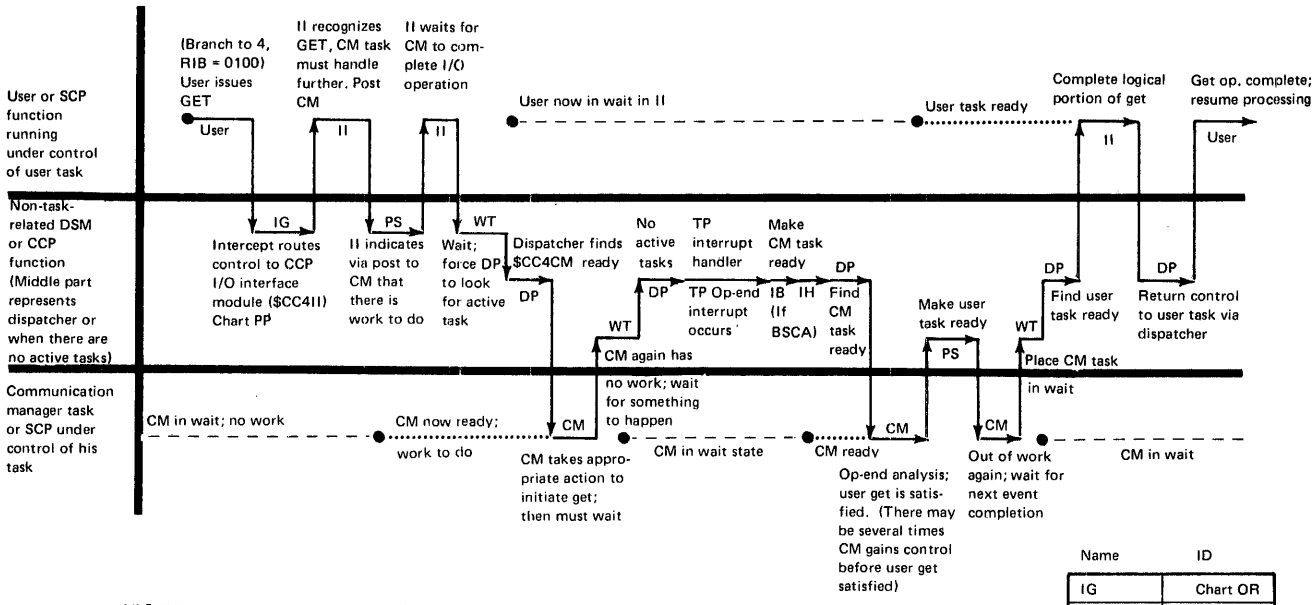
PS does not cause a higher priority task to immediately gain control; it returns to its caller. After II receives control back from PS, it has nothing more to do until CM can obtain the data requested. II must now relinquish control to CM. This is accomplished by calling wait ($CC4WT).

Wait gives control to the dispatcher (DP) who will switch control to CM. DP ensures that the environment of the ceding task is preserved and that the newly-activated task's environment is properly reinstated. II, running as a subroutine of this user task, has placed that user task in a wait state and CM is now the active task.

CM is now aware of the get request and takes the proper actions to remember what to do with the data when it subsequently arrives. Important too is that polling for that terminal must be resumed (it was terminated after the preceding operation). Having done all it can for the moment, CM will wait for it to resume polling. By going to wait, then, assuming no other tasks are active, the dispatcher will enter its system wait loop.

Finally, the terminal causes an interrupt which is the positive response to the poll with receipt of the awaited data. The appropriate I/O interrupt handler is the first to gain control (not shown in Diagram 9M.0450). The DSM interrupt handlers pass control to one of the three possible intercept routines: $CC4IB, $CC4IM, or $CC4IC. Assuming a 3270 terminal, it would be $CC4IB (IB). IB's function is primarily that of interfacing with BSC IOCS's trace on the one hand and evoking $CC4IH (IH) on the other. CM is posted, not by calling PS, but through a special (dispatcher) flag setting; this peculiarity of not using a formal post, (PS) occurs because it is difficult for PS to ensure that he is not the one who happened to already be in control when the interrupt occurred.

When the TP interrupt handler concludes processing, the reset interrupt level passes control to the Dispatcher. DP finds CM in a ready state and directs control to him. CM's op end analysis would find the appropriate user task and call PS to make that user task ready. Having nothing else to do, CM would now exit via WT which would cause the Dispatcher to look for active tasks. Eventually, if not immediately, the user task enabled by CM would regain control. The resume point is in II where the wait for the get was done. II completes the get by going to DP, who finally resumes the user at a point just beyond his get statement.



Name	ID
IG	Chart OR
II	Chart PP
PS	Chart OJ
WT	Chart OI
DP	Chart OH
IB	Chart OD
IH	Chart OF

All 2-digit names are abbreviations of the $CC4__ module or entry point names (Example: DP is $CC4DP.)

– – – – – – means task in wait state

..... means task ready, but not currently active

Diagram 9M.0450. Control Flow of Get Operation

Command Timeline (/Q Command)

The next situation involves a /Q command entered by a terminal operator (Diagram 9M.0460).

The $CC4IB (IB) module, evoked by BSC interrupt handler, calls $CC4IH (IH) which indicates a pending interrupt to the dispatcher, $CC4DP (DP) following the BSC interrupt handler reset interrupt level. The $CC4IB module then finds the Communication Manager (CM) task is in a ready state. (See get timeline flow for more detail.) CM recognizes that this input data is from a command processor terminal and evokes $CC4PS (PS) to post Command Processor, $CC4CP (CP), informing CP that there is work for him. That work is conveyed via a TUB queue containing the pertinent information. Upon return from PS, CM finds no more work so he evokes $CC4WT (WT).

DP gets control from WT and passes control to CP. CP immediately calls for its command routing transient, $CC4PC (PC). To do this, CP calls $CC4PI (PI) to load the appropriate CCP transient area. PI uses the parameter provided by CP to look up the disk address and proceeds to call DSM disk IOS. Disk IOS is temporarily intercepted by $CC4IO (IO), which performs an IOB trace, but then IOS can start the operation. For simplicity, assume that the disk arm is located at the proper cylinder and that no other disk I/O is active. PI must wait, calling DSM disk wait which is intercepted by $CC4IW (IW). IW forces the current task, CP, into a wait state via WT. WT exits to DP which handles checking for operation completions.

When DP finds the disk IOB complete, the CP task is reactivated via PS and resumes in the transient area where PC now resides. PC examines the command data and determines that module $CC4QQ (QQ) must further handle the command. To effect a transfer control (XCTL) to QQ from PC, $CC4TX (TX) is called. TX is actually a special entry point in PI because other than the environment preservation and a small difference in passing the parameter designating the transferred transient, it is the same module. The remaining logic is identical as in the load of PC. Finally, when DP finds the operation complete, QQ is now in the transient area and DP gives him control.

The normal processing of the /Q command is conducted by QQ which concludes the CP task's operation. First, however, CP must re-enter the formal wait. To do this, QQ calls the transient return entry point of PI, $CC4TR. Other than minor housekeeping, this function merely returns control to the original point at which the preceding series of transient modules was evoked. Note that the Dispatcher is bypassed.

CP will invoke $CC4PR (PR), the Command Processor Return Routine to:

1. Free any encumbered TP buffer and post CM.
2. Send an appropriate message (set up by QQ) to the requesting terminals.
3. Re-invite the terminal.

$CC4PR will then return to $CC4CP via $CC4TR.

CP now is in control again and when he finds no more work pending he will wait. WT will place him in a wait state and exit via the Dispatcher, at which time the system would again be in a wait state or some other ready task could gain control.

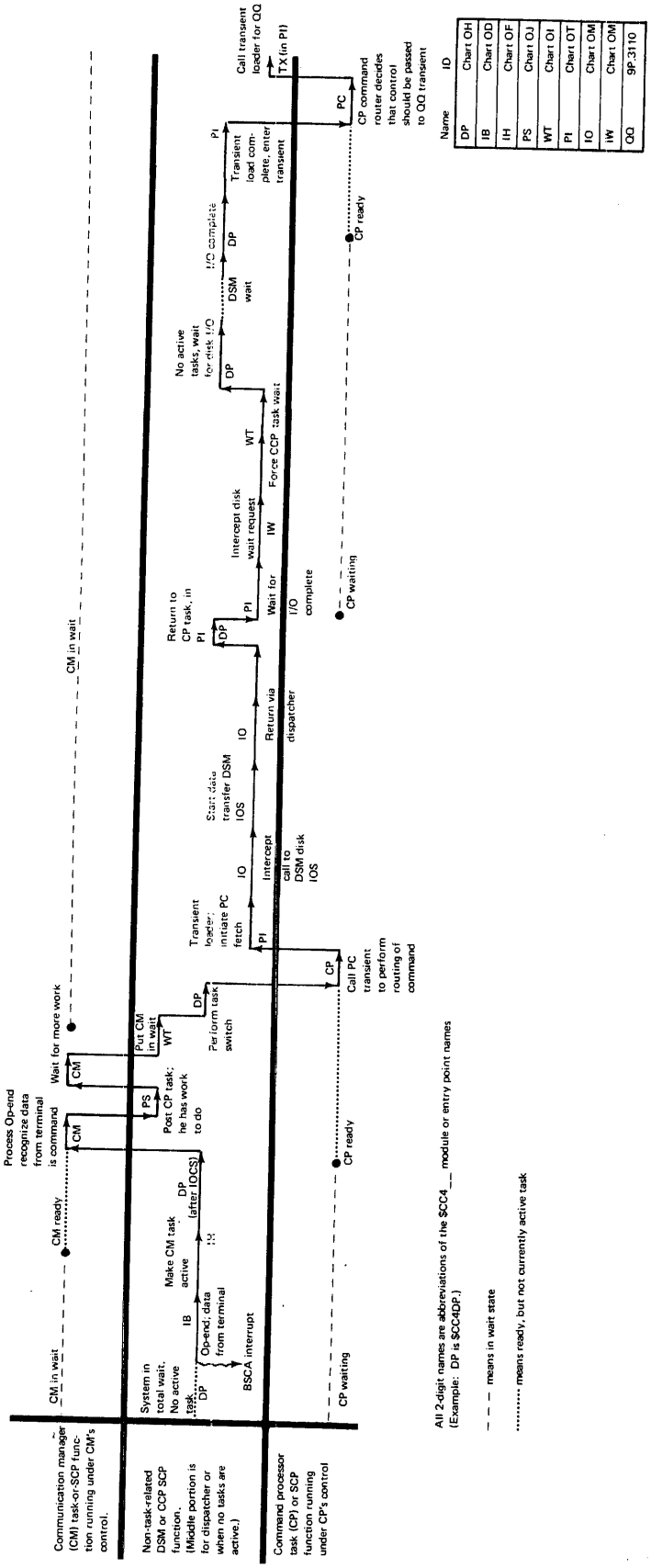
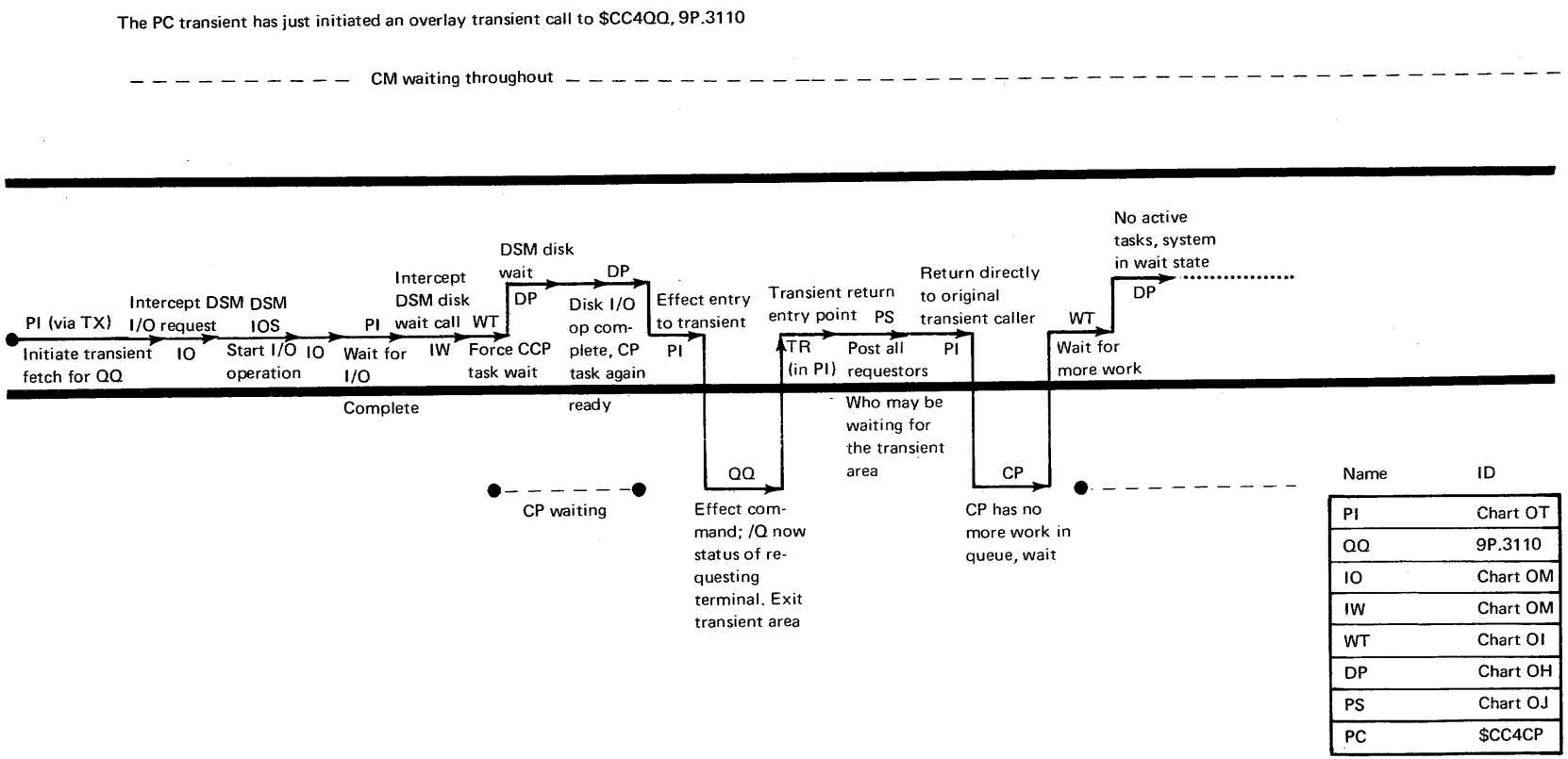


Diagram 9M.0460 (Part 1 of 2). /Q Command Timeline Diagram


The PC transient has just initiated an overlay transient call to $CC4QQ, 9P.3110
CM waiting throughout
PI (via TX)
Initiate transient fetch for QQ
Intercept I/O request
IO
DSM IOS
Start I/O operation
IO
PI
Wait for I/O
Complete
Intercept DSM disk wait call
IW
WT
Force CCP task wait
DSM disk wait
DP
DP
Disk I/O op complete, CP task again ready
Effect entry to transient
PI
QQ
Effect command; /Q now status of requesting terminal. Exit transient area
CP waiting
Transient return entry point
TR (in PI)
PS
Post all requestors
Who may be waiting for the transient area
PI
Return directly to original transient caller
CP
CP has no more work in queue, wait
WT
Wait for more work
No active tasks, system in wait state
DP
Name
ID
PI
Chart OT
QQ
9P.3110
IO
Chart OM
IW
Chart OM
WT
Chart OI
DP
Chart OH
PS
Chart OJ
PC
$CC4CP

Concurrent File Sharing Timeline

This timeline (Diagram 9M.0470) emphasizes the area of disk update file sharing integrity assurance. The TP portions are treated lightly as they are incidental yet maintain timeline continuity. In the example there are two user tasks which will be called T1 and T2 for simplicity. The Communication Manager task (CM) will also be occasionally involved.

Normally disk I/O done between two tasks will overlap nicely. However, CCP has provided the capability of updating the same file from two or more concurrently open DTFs. This means that access conflicts are possible and the following will illustrate how the situation is handled. Whenever an input is done, that physical portion of the file is inhibited for further reference by all update files until the next input operation (which will also establish a new set of locked sectors) or an output operation is done to a random DTF.

It will be assumed that $CC4II (II) has just given T1 a transaction. T1 takes that transaction and uses a portion of it to access a random file via a DSM disk access method (ACSMD). The ACSMD calls disk IOS which causes $CC4IO (IO) to gain control. When IO detects the (potential) shared update file he calls $CC4DI (DI). DI has the function of establishing exclusive access to the contiguous group of sectors defined by the physical disk IO buffer. DI also assures that the given request does not overlap an area already being accessed. It will be assumed that the T1 request is not in conflict.

Once DI has taken the necessary steps, DI returns to IO who calls IOS. Disk IOS may start the operation returning to IO. With the file sharing option, IO also forces a wait by going to $CC4WT (WT). WT causes the user task to enter the wait state. When the IO operation is complete, the user task will effectively resume in DP who will call PS, then return to the ACSMD. (Without the file sharing option the wait is done when the ACSMD calls DSM wait and $CC4IW (IW) intercepts an incomplete IOB.) When the ACSMD calls DSM wait, IW will see an already complete IOB and would normally return directly via DP.

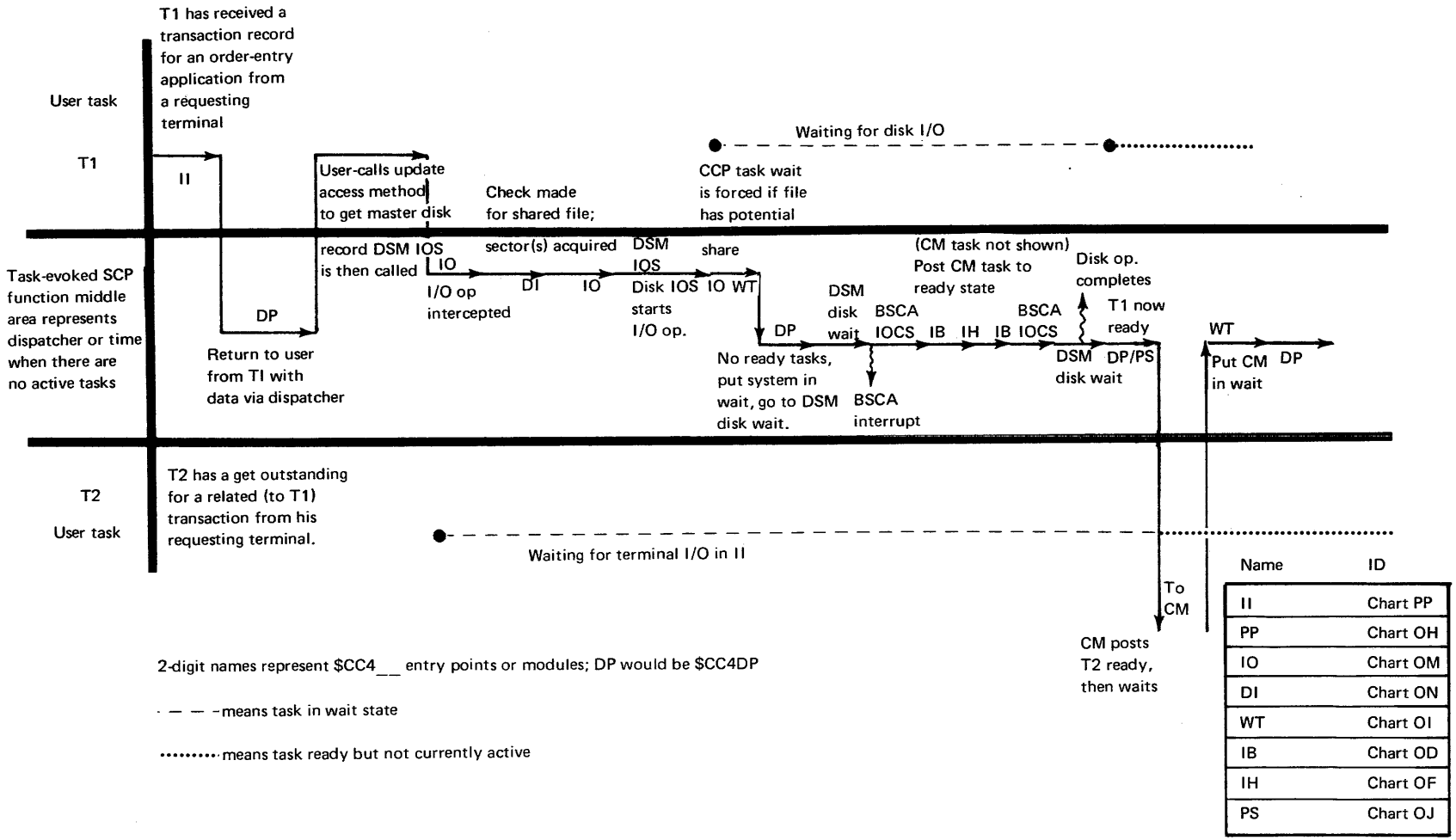
Backing up slightly in time, note that as the I/O operation was just begun DP would look for another ready task. Assume that while T1's disk I/O operation is in process T2 becomes ready and receives an input transaction from II. Similar to T1, T2 uses this transaction to access a record in the same file as T1 is presently accessing. T2's ACSMD evokes DSM disk IOS, which is intercepted by IO. If that record on disk resides wholly or in part within even one of the sectors to which T1 has been assigned exclusive access, T2 must wait before going on. This is done by DI who calls $CC4WT (WT), the CCP wait routine. DI will have effected an enqueue on that disk address.

WT exits via DP who now finds no active tasks. Eventually, the I/O operation will complete and DP will make T1 ready again. T1 will call disk wait and that will effectively be NO OPed, as previously mentioned.

Now T1's ACSMD will be called by the user to update the retrieved record.

As in the read operation, the flow is from ACSMD to disk IOS, intercepted by IO, to DI, back to IO, to DSM IOS, and finally DP via WT. When the write finally completes (being a random access method) not only is T1 made active but DP calls DI at a special entry point which releases the queued status of the sector group owned by T1 and that enables $CC4PS (PS) to be called in order to make T2 ready.

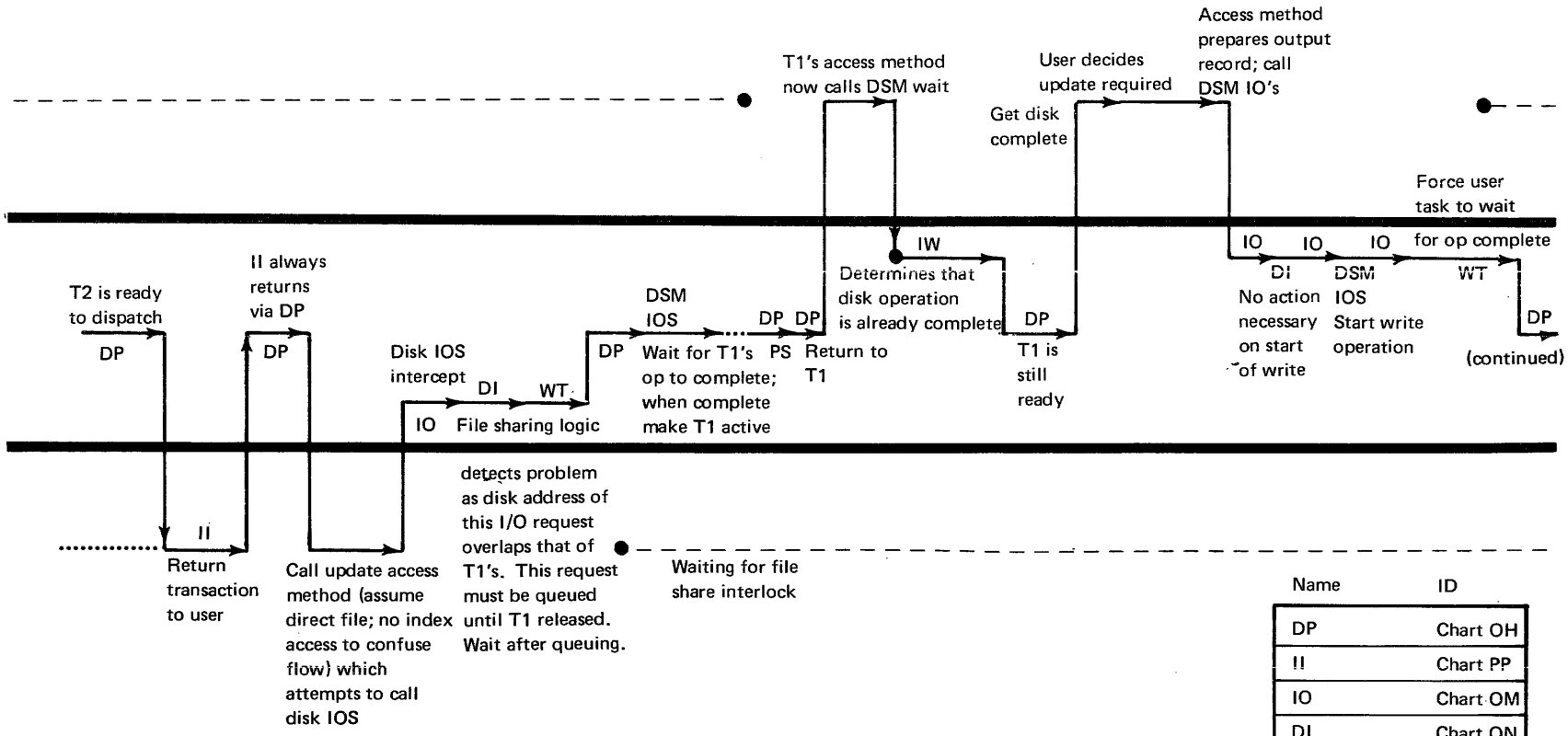
DP would gain control again, now with both T1 and T2 active. T2 would resume in DI and proceed to establish ownership of his set of physical sectors. DI returns to IO who calls DSM IOS before placing T2 in a wait state via WT. DP would now find T1 ready. T1 would wait for the PUT which would immediately cause control return. Finally, T1 waits for something else which enables T2 to continue.



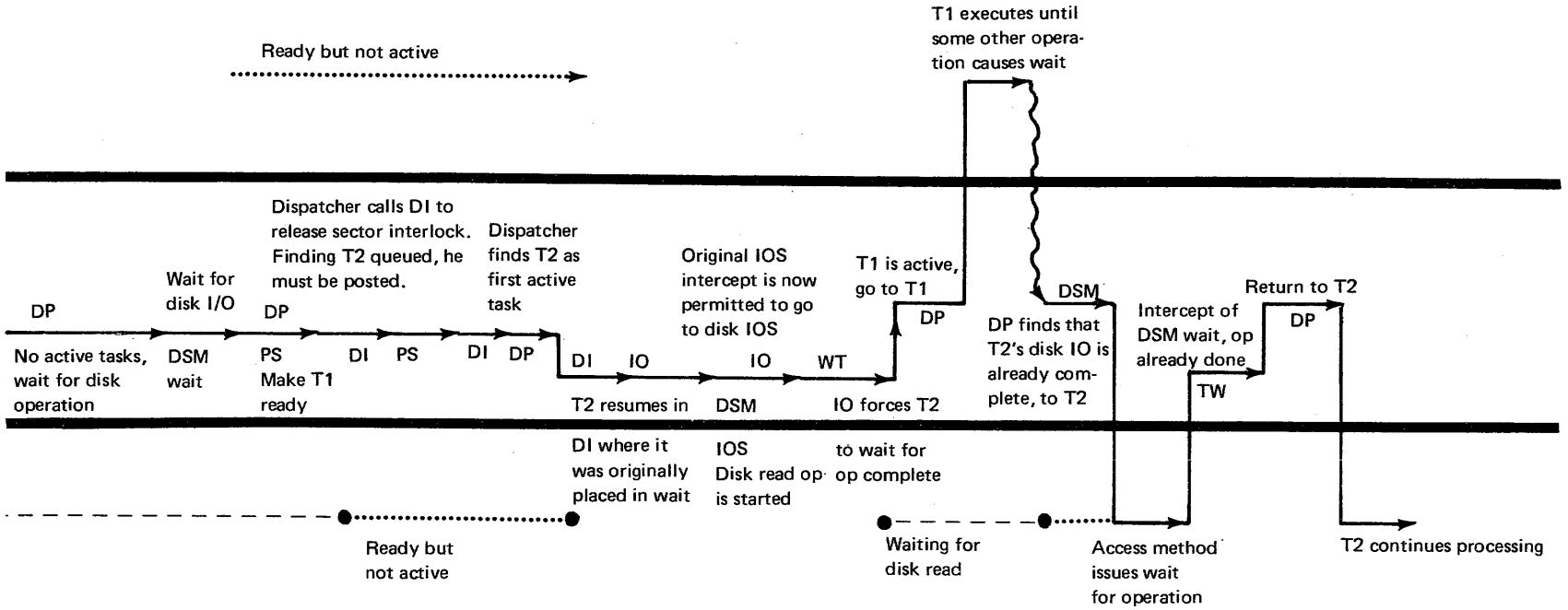
Name	ID
II	Chart PP
PP	Chart OH
IO	Chart OM
DI	Chart ON
WT	Chart OI
IB	Chart OD
IH	Chart OF
PS	Chart OJ

Diagram 9M.0470 (Part 1 of 3). Concurrent File Sharing

Diagram 9M.0470 (Part 2 of 3). Concurrent File Sharing



Name	ID
DP	Chart OH
II	Chart PP
IO	Chart OM
DI	Chart ON
WT	Chart OI
PS	Chart OJ
IW	Chart OM



Name	ID
DP	Chart OH
PS	Chart OJ
DI	Chart ON
IO	Chart OM
WT	Chart OI
IW	Chart OM

Program Organization

Resident Startup Initialization Routine (CPINIT)

ENTRY POINTS:
- CPINIT (initial entry from Startup)
- SU@BCH plus two bytes (re-entry point after non-resident Startup completes)

CHART: OA

FUNCTIONS:
- Performs the Startup initialization of CCP which must be link-edited with $CC4.
- Invokes the final Startup routine, $CC4SU.

OPERATION:
- At initial entry from Startup communicates to the non-resident CCP Startup code the address of the resident CCP halt intercept.
- After all Startup phases have executed, allocates and opens the BSCA DTFs.
- Sets the address of the CCP BSCA interrupt handler into the resident DSM BSCA interrupt handler.
- Sets the IAR Q-byte value of the CCP program level into the resident CCP Execution code.
- Modifies the DSM low main storage entry point address of dump, General Entry, disk IOS, and disk I/O wait to cause entry into CCP when these functions are invoked.
- Sets the address of the flag byte in DSM (NCMBSV in SUPCM) which contains the disk ERP indication for testing by the CCP disk IOS and disk I/O wait intercept routines.
- Sets up the CCP transient area handlers IOB Q-bytes for the program pack.
- Restores the DSM resident halt entry point address from that set during CCP startup.
- Initializes the CCP General Entry Intercept Routine ($CC4IG) to contain the Q-bytes of the system pack and the program pack.
- Exits through the Command Processor to the transient Startup routine $CC4SU to complete all Startup functions.

INPUT: None

OUTPUT: All areas discussed under operation are set for Execution.

EXIT: To Command Processor transient $CC4SU via $CC4CP.

EXTERNAL REFERENCES:
- Address of other level IAR hold area in Startup.
- Address of the BSCA DTFs built by Startup.
- Addresses of IAR Q-bytes in $CC4DP, $CC4IH, $CC4PI.
- Addresses of resident disk I/O and I/O wait intercept routines $CC4IO and $CC4IW.
- Address of General Entry Intercept ($CC4IG).
- Address of dump intercept routine (CPHALT).
- System Communication Area (NCSYS@).
- CCP Command Processor.

Console Interrupt Intercept ($CC4IC)

ENTRY POINT: $CC4IC

CHART: OC

FUNCTIONS:
- Sets flag bit (IHC) in flag byte (IHFLAG) of Common Interrupt Handler ($CC4IH). This bit signifies that $CC4IC has been entered so that control is not lost if $CC4IH is repeatedly entered due to nested interrupts.
- After return from $CC4IH, sets dispatchability bit (DPCI) in the Dispatcher flag byte ($DPFLH) of $CCCOM, indicating that an operation-end (or REQ) interrupt has made the Communications Task dispatchable.

INPUT: XR1 points to the resident console IOB. SKDIDS in System Communications Area contains X'1X' if REQ has been pressed.

OUTPUT:
- Flag bit (IHC) in flag byte (IHFLAG) of $CC4IH set on before entry to that routine, and off after return from it.
- Flag bit (DPCI) in Dispatcher flag byte ($DPFLH) of $CCCOM set on after return from $CC4IH.
- #TT#IC stepped by one if DPCI is set on.

EXIT: Always to $@COIH, via ARR, to complete interrupt processing.

BSCA Interrupt Intercept ($CC4IB)

ENTRY POINT: $CC4IB

CHART: OD

FUNCTION: Intercepts each BSCA op end and determines whether the Communications Management Task has work to perform as a result of the interrupt.

INPUT: XR2 contains the address of the BSCA Work Area for the line.

OUTPUT:
- XR2 is not restored for the return.
- OP end count in the LCB and CCOECT are incremented to reflect a completed operation.
- #TT#IC stepped by one.

EXITS: Return to BSCA interrupt handler at NSI.

EXTERNAL REFERENCES: Common intercept routine ($CC4IH).

Common Interrupt Handler ($CC4IH)

ENTRY POINT: $CC4IH
CHART: OF
FUNCTIONS:
- Determines if the CCP program level is non-interruptable and, if not, changes the CCP program level IAR to cause the CCP level to resume, following interrupt level processing, in the CCP Dispatcher.
- Saves the current task's Dispatcher control byte $DPFLG and sets both Dispatcher control bytes ($DPFLG + $DPFLH) to cause the Dispatcher to save the interrupted task's status, process the interrupt, and resume the interrupted task.
- Prevents repeated entry during a nested interrupt to insure data and instruction address register integrity.
- Posts CCP to handle interrupts by setting flag bits in the work area ($CCCOM).

INPUT: The CCP program level's IAR.
OUTPUT:
- CCP program level IAR reset to the entry address of the CCP Dispatcher ($CC4DP).
- CCP Dispatcher flag bytes $DPFLG + $DPFLH set to cause interrupt processing by the appropriate CCP routine.

EXITS:
- To invoker via ARR.
- To invoker of nested lower level interrupts by resetting the lower interrupt level's IAR to the address of the instruction following the branch to $CC4IH.

EXTERNAL REFERENCES: CCP common work area, $CCCOM, current TCB, the invoking interrupt level routines next sequential instruction via an EXTRN and address constant.

MLTA Interrupt Intercept ($CC4IM)

ENTRY POINTS: Three entry points, each corresponding to an entry point in the active MLTA trace routine: $$MLDD, MLTDD1, and MLTDD2
CHART: OE
FUNCTION/OPERATION: All MLTA interrupts are routed through this routine. The interrupt is analyzed to determine if it is for an op end. If an op end interrupt has occurred, the count of op ends (#OPEND) in $CCCOM is incremented by one. This op end count is used by the Dispatcher to determine when $CC4CM is dispatchable. A check is also made to determine whether the MLTA IOCS trace routine has been loaded into main storage. If so, then the MLTA trace is linked to so that active trace entries are taken.

INPUT: XR1 contains address of MLTA common area. XR2 contains address of MLTA line DTF for which interrupt occurred.
OUTPUT: Add one to op end count for op end interrupt (to #TT#IM in $CCCOM).
EXITS: Return at ARR or, if MLTA trace active, link to the appropriate MLTA trace.

CCP Task Dispatcher ($CC4DP)

ENTRY POINT: $CC4DP
CHART: OH
FUNCTIONS:
- Dispatches any ready system or user task.
- Allows a user task to exit the must-complete state.
- Routes control following a teleprocessing or console interrupt to the Communications Task to process that interrupt.
- Gives control back to an interrupted user task following interrupt processing.
- Determines which disk IOB should be scheduled into the disk I/O wait routine of Disk System Management.
- Causes CCP to wait when no work can be performed.

INPUT: Variable fields and flag bytes in $CCCOM to indicate what actions are to be taken by the Dispatcher.
- The address of the returning (or current task) currently running under the CCP in the $CCCOM field @CURTB.
- $CCCOM field $DPFLH, which indicates the type of entry being made to the Dispatcher. This can be either a normal exit from a task (no indicators are set), or it can be an entry from one of the interrupt levels caused by an interrupt level routine resetting the CCP instruction address register to arrest control from the current task and forcing control to resume in the CCP Dispatcher.
- An indication in the $CCCOM field $DPFLG as to whether the exiting task is relinquishing control as it must wait for some event or it is only exiting the must-complete state.
- When a task is exiting the must-complete state, the CCP Dispatcher will allow any of the system tasks (Communication, Termination, or Command Processor) to run and then give control immediately back to the task that last exited the must-complete state (if that task is still dispatchable).

OUTPUT: None

EXITS:

- To the instruction addressed by the TCB ARR field of the to be dispatched TCB.
- If no ready tasks exist but disk IOBs are in process, then $CC4DP will branch to DSM NCEIOW entry.
- If neither of the above are applicable, the Dispatcher will APL (or halt in a MINRES-NO system) allowing, in a DPF system, the other program level to gain control of the CPU.

EXTERNAL REFERENCES: DSM NCEIOS + NCEIOW entry points, $CCCOM, CCP Register Save and Restore Routine, CCP wait and post routines, console management work area.

CCP Wait Routine ($CC4WT)

ENTRY POINT: $CC4WT
CHART: OI
FUNCTION: Provides the interface required to allow a task to wait for one or more asynchronous events.
OPERATION:

- Validates that the expected event count in the input parameter list is non-zero. If it is not, the CCP halt routine will be entered with a subhalt code of PS.
- Moves the wait parameter list wait mask and expected event count to the current TCBs wait mask and count fields, sets the residual count equal to the event count, and zeros the TCB post mask.
- ORs the wait parameter list flag byte into $DPFLG in $CCCOM, restores all registers, and invokes the CCP Dispatcher via load IAR instruction.

INPUT: The linkage used to invoke CCP wait is:

```
B       $CC4WT
DC      AL2 nnnn        TCB wait mask
DC      IL1'n'          Expected event count
DC      AL1 nnnn        $DPFLG control byte
                        OR bits
```

OUTPUT: The wait routine will set the TCB wait mask, expected event count, the residual event count, and the TCB post mask.
EXIT, NORMAL: To the CCP Dispatcher. The task will resume at the instruction following the wait parameter list.
EXIT, ERROR: To the CCP halt routine if the expected event count is zero.

Task Post Routine ($CC4PS)

ENTRY POINT: $CC4PS
CHART: OJ
FUNCTIONS:

- Indicates to a specific waiting task that an expected event has occurred.
- Allows a task that has recognized the occurrence of an event which is by definition significant in the CCP system, to signal that event to all other CCP tasks that may be waiting for that event.

OPERATION:

- Validates the input parameter list. If the post mask is all zero or the address of the TCB to be posted is all zero (both bytes), the CCP halt routine will be entered with a subhalt code of PS.
- Following successful validation, the CCP trace routine is invoked with an ID of seven to record the post entry parameter data.
- If the post parameter list TCB address has a non-zero high byte, a specific TCB is to be posted. If the high byte is zero, all TCBs in the system are to be tested to see if they are awaiting this specific event.
- Each TCB or the specified TCB is inspected prior to posting. First, it is determined if the task has already been posted for the event and, if not, an additional test is made to see if the task wait mask indicates that the task expects the event. If these conditions are met, the appropriate post mask bit is turned on to indicate which event the post is for, the residual event count is decremented by one and, if the result is zero the task's dispatchability bit is turned on.
- After the TCB(s) are checked, post restores the caller's registers and returns via branch to the instruction following the post parameter list.

INPUT: The following calling sequence is used for post:

```
EXTRN $CC4PS
B       $CC4PS
DC      XL2' post-mask'
DC      AL2 @ of TCB to be posted
    or
DC      XL2'00nn'       Indicating general post
                        (nn is non-zero)
```

OUTPUT: See the discussion of operation for an explanation of what fields are set.
EXIT, NORMAL: To the instruction following the input parameter list.
EXIT, ERROR: CCP halt routine with a PS subhalt.

CCP Disk I/O Intercept ($CC4IO)

ENTRY POINT: $CC4IO
CHART: OM
FUNCTION: Intercepts all entries to location eight within DSM and routes them to disk or printer IOS and back to the invoker through the CCP Dispatcher.

OPERATIONS:
- If disk ERP is in process, allows the disk IOS entry to continue with no action taken by this routine.
- Traces the disk IOS entry via CCP Trace (ID=02).
- Determines if the entry is from the non-CCP program level and takes no action if it is.
- Saves the invoker's registers in the current TCB.
- If the system is a multi-tasking system and the input IOB has an associated DTF, goes to the file sharing routine.
- Allows the disk IOS request to be scheduled by invoking NCEIOS but forcing it to return to this routine.
- Forces a disk wait if there is an associated DTF.
- Exits through the Dispatcher back to the invoker.

INPUT: The disk IOB address in register one.
OUTPUT: The appropriate bits set in $CCCOM $DPFLG to indicate what action the Dispatcher is to take with the task.

CCP Disk I/O Wait Intercept ($CC4IW)

ENTRY POINT: $CC4IW
CHART: OM
FUNCTIONS: Intercepts all entries to location twelve within DSM and routes them to disk or printer I/O wait and puts the task into a wait.
OPERATION:
- If disk ERP is in process allows the wait entry to continue with no action taken by this routine.
- If the IOB is marked complete, doesn't trace the entry.
- Traces the disk wait entry via CCP Trace (ID=03).
- Determines if the entry is from the non-CCP program level and takes no action if it is.
- Saves the invoker's registers in the current TCB.
- If the IOB is marked complete, doesn't cause the task to wait, else indicates to the Dispatcher that the task is to wait for disk I/O complete.
- Exits to the CCP Dispatcher via Wait ($CC4WT) or directly.

INPUT: The disk IOB address in register one.
OUTPUT: The appropriate bits set in $CCCOM $DPFLG to indicate what action the Dispatcher is to take with the task.
EXITS:
- DSM Disk I/O Wait Routine (NCEIOW).
- CCP Dispatcher ($CC4DP).
- To Disk I/O Wait (NCEIOW) if the current IAR is not the CCP IAR.
- CCP Dispatcher to cause the task to wait/exit.

EXTERNAL REFERENCES:
- CCP Trace Routine ($CC4TT).
- IAR Determination Routine (DPTIAR).
- CCP Register Save Routine (CCPSAV).
- Current TCB address (@CURTB in $CCCOM).
- Current TCB TCBTSK byte.

CCP File Sharing Enqueue–Dequeue Routines ($CC4DI)

ENTRY POINTS:
- $CC4DI – IOB enqueue routine
- $CC4RQ – SQB release routine
- RQDPFR – Disk Wait Intercept entry to SQB release

CHART: ON
FUNCTIONS:
- Allows file sharing in update mode among the user programs while maintaining file integrity.
- Allows several programs to access added records in indexed files concurrently while ensuring that only one DTF is actually adding at any time.

OPERATION:
- $CC4DI comprises three routines:
  - $CC4DI – IOB enqueue–dequeue routine
  - $CC4RQ – SQB release routine
  - $CC4AO – Adder validity check routine
- At entry to $CC4DI, intercepted disk or printer I/O requests are examined. If the request is a read and the DTF is opened for update, the IOB is added at the end of a queue of IOBs awaiting enqueued data sectors, the SQB owned by the IOB (if any) is located and the SQB Release Routine is entered. If the request is not to read, or the DTF is not opened for update, the Adder Validity Check Routine is entered.
- The SQB Release Routine will first free the input SQB, then examine the IOB queue for I/O requests which may now be honored. The disk address and N-byte of the queued IOB is compared to the disk addresses in each active SQB for conflict. If there is no conflict, the IOB may be freed, and an SQB is built to reflect the data extent of the I/O request. The task which owned the freed IOB is posted out of the wait. All IOBs on the queue are thus examined. If a queued IOB which cannot be freed belongs to the current task, the CCP Wait Routine ($CC4WT) will be invoked to cause the current task to wait.
- The Adder Validity Check Routine tests the I/O request to determine if it is an add. If so, the SDF for the file is located and tested to see if the task is allowed to add. If so, and no other DTF is currently adding, the DTF address is stored in the SDF and becomes the current adder. Otherwise, the Resident Termination Routine is invoked to terminate the task.

INPUT:
- XR1 containing IOB address – input to $CC4DI.
- XR2 containing SQB address – input to SQB release.

OUTPUT:
- IOB fields IOBCHN and IOBTCB modified for queued IOB.
- SQB built or freed.

EXITS, NORMAL:
- Return to caller via CCPRET.
- Return to Disk Wait Intercept via ARR.

EXITS, ERROR: CC4TI2 – To terminate an invalid adder or a task if an available SQB cannot be found.

General Entry Intercept ($CC4IG)

ENTRY POINT: $CC4IG
CHART: OR
FUNCTION/OPERATION:

- Intercepts *all* invocations of DSM General Entry (branches to location 4). The primary purpose of the intercept is to assure the serial use, by the CCP program level, of the non-reentrant DSM supervisor.
- If the invocation is from the other program level, passes it on directly to DSM General Entry.
- For RIB of X'00' from CCP program level, provides in index register 2 the address of the CCP Program Level Communication Area.
- If the invocation is a halt/syslog request, handles it via the CCP substitute halt/syslog processor.
- If the invocation is from within the DSM supervisor's own main storage area:
  - Determines if disk error log request; if so, sets on the error-log-in-progress bit in the System Communication Area.
  - In any case, passes the invocation on to DSM General Entry.
- For RIB indicating program load/fetch, modifies those fields in the PLCA which control relocation, and passes on the invocation to DSM General Entry.
- For a RIB requesting transient by C/S/N, passes the invocation on to DSM General Entry.
- For RIB indicating request for specific transient, passes the invocation directly on to general entry unless:
  - Open/close/allocate – Passes invocation to CCP Data Management Routine $CC4OC.
  - EOJ – Passes invocation to CCP Termination Interface Routine CC4TI1.

EXIT, NORMAL: To DSM general entry or to CCP open, close, allocate, or halt routines, depending on request.
EXIT, ERROR (Model 4 Only): To CCP halt routine when a RIB is intercepted for certain types of fetches and finds not supported by CCP.

CCP Transient Area Handler ($CC4PI, $CC4TX, $CC4TR)

ENTRY POINTS: $CC4PI, $CC4TX, $CC4TR
CHART: OT
FUNCTIONS:

- Provides the logic necessary for the CCP system transient routines to be invoked and for them to invoke other routines.
- Processes requests for transient modules into one of two transient areas within CCP. One area is for the exclusive use of the Communications Task, the other is shared by all other tasks operating under CCP. Simultaneous fetching of transients into both areas can occur.

Each section of the Transient Area Handler has interfaces with the other section. These are discussed in greater detail for each entry point.

$CC4PI

ENTRY POINT: $CC4PI
CHART: OT
FUNCTIONS: Provides the control logic necessary for multiple tasks requesting transient services and fetches the required modules.
OPERATION: Validates that the identifier of the requested transient does not exceed the number of entries in the first level program list in CCP.

- At entry checks for transient area availability. If this fails, the current task is put into a wait for a transient area by invoking $CC4WT (CCP Wait).
- When a transient area is available, the requested routine is read into the transient area via branch to DSMs disk I/O routines.
- If the transient is being XCTLed to, a two-byte parameter from the XCTLer is moved into the transient.
- Register one is set to the entry point address of the transient.
- Register two is restored from the current save area of the current TCB. (Note – The save area address is not changed.)

INPUT:

- $CCCOM (#LSTSZ, @TALST)
- The ARR points to the numeric value of the program to be invoked. This is the relative entry number of the disk address of the transient in the first level program list.

OUTPUT: The transient is loaded and control given to it.
EXIT, NORMAL: To the invoked routine.
EXIT, ERROR:

- To CCP halt routine if a request is made with a reference that exceeds the boundary of the table.
- CCP is in error and will halt with a code of U– and subhalt of 12.

NOTES:

- All CCP transient routines must use the $EBEG (transient begin) macro to set up their initial instructions to be invoked by $CC4PI.
- The CCP transient areas are contained in the base CCP module.
- If the requested transient area is unavailable, $CC4PI queues the task and exits via Wait until an area is available for this task.

$CC4TX

ENTRY POINT: $CC4TX
CHART: OT
FUNCTION: Allows a CCP transient (XCTLer) to XCTL (transfer control) to another CCP transient (XCTLee).
OPERATIONS:
- Computes the address of the XCTLee's NCS values in the XCTLer.
- Enters the $CC4PI logic at the point where preparation for disk I/O is done.

INPUT:
- Within the XCTLer, is the field TAXTID containing the displacement from the beginning of the transient area of the disk address of the routine being invoked (the XCTLee).
- Also within the XCTLer is the two byte XCTL parameter field (TAXPRM) which is passed to the invoked routine's TAXPRM field.

OUTPUT: Control is passed to the XCTLee after it is read into the transient area.
EXIT: To invoked module.
NOTES:
- The XCTL tables are five bytes per entry length, beginning at the symbolic displacement TAXCID within the module. The first two characters are the suffix of the XCTLee and the next three are the disk address, length, and CCP control information. The end of the entries is signified by a dollar sign ($) delimiter.
- The two-character suffix of the XCTLee is moved into the transient areas IOB IOBCHN field for the disk I/O trace.

$CC4TR

ENTRY POINT: $CC4TR
CHART: OT
FUNCTIONS: Provides the interface for a CCP transient routine to:
- Return to the invoking routine.
- Make the currently in use transient area available.
- Post all tasks waiting for the transient area that is available (non-communications tasks only).

OPERATION:
- Makes the transient area available.
- If the current task is not Communications, posts transient area availability.

INPUT: The address of the current task in $CCCOM to determine which transient area to free.
OUTPUT:
- The transient area will be marked available.
- All tasks awaiting transient area one will be posted.
- The invoker's registers will be restored and control returned to it.

EXIT: To invoker of first level transient via CCPRET routine.

Program Request Queue Test ($CC4PQ)

ENTRY POINT: $CC4PQ
CHART: OU
FUNCTION: Determines if a pending program request exists for a terminal or a switched, multi-terminal (BSCA only) line.
OPERATION: Scans the input queue to program allocation (the pending request queue) to see if any request requires the terminal (or switched line LCB that the terminal is on) whose address was input to the routine.
INPUT: TUB address in register two.
OUTPUT: Program status register (PSR)
- False bit off – Terminal is needed.
- False bit on – Terminal is not needed.
- Binary overflow indicator on – Line is needed (BSCA only).

EXIT: To invoker's next instruction via CCPRET subroutine.

CCP Register Save and Restore Routine (CCPSAV)

ENTRY POINTS:
- CCPSAV and CCPSV2 – Register Save
- CCPRST – Register Restore
- CCPRET – Register Restore and Return (to higher level)

CHART: OV
FUNCTIONS:
- Saves the contents of XR1, XR2, and $CCCOM $CCARR fields in the current Task Control Block.
- Restores the registers from the current Task Control Block.

OPERATION:
- Register Save Routine – Increments the TCB register save area address field in the current TCB and saves the input XR1, XR2, and ARR in the current save area in the TCB.
- Register Restore – Decrements the save area address field in the current TCB and restores the XR1, XR2, and ARR from there. Returns to invoker using the ARR value at entry to this routine.
- Register Restore and Return – Same as Register Restore but return using the restored ARR rather than the ARR value at entry.

INPUT:
- CCPSAV and CCPSV2 – Input XR1 and XR2, $CCARR in $CCCOM, current TCB.
- CCPRST – Current TCB address in $CCCOM (@CURTB).
- CCPRET – Current TCB address in $CCCOM (@CURTB).

OUTPUT:
- CCPSAV and CCPSV2 – Registers saved in current TCB and save area address in the TCB incremented to the next save area.
- CCPSV2 only – XR1 = Address of current TCB in $CCCOM. XR2 = Address of $CCCOM.
- CCPRET and CCPRST – Registers restored and current TCB save area address decremented to next lower level.

EXIT, NORMAL:
- CCPSAV, CCPSV2 and CCPRST – To invoker.
- CCPRET – To invoker of invoker (next higher level of control).

EXIT, ERROR: If the number of save areas in the Task Control Block is exceeded, a branch to the CCP Halt Routine (CPHALT) to give a U– halt (or a 1234 halt on Model 4) and a SL (save level) subhalt.

EXTERNAL REFERENCES: Current TCB @ and saved ARR value in $CCCOM.

CCP Trace Interface Routine ($CC4TT)

ENTRY POINT: $CC4TT
CHART: OV
FUNCTION: Provides an interface to the dynamically loaded real CCP trace module and the invokers of that module.
OPERATION:
- If Trace is loaded by Startup, the instructions in this module are replaced by a load IAR instruction which will cause control to continue in the real trace routine CCTR.
- If Trace is not loaded, the routine will step the ARR over the one byte Trace argument and return to the invoker.

INPUT: The ARR addresses the one-byte trace id value to be used for Trace.
OUTPUT: If Trace is invoked, a trace entry is built in the CCP trace table. If Trace is not invoked, no output occurs.
EXIT: To the second byte following the branch to this module.
EXTERNAL REFERENCES: CCTR if real Trace is loaded.

CCP Stand Alone Halt Routine (CPHALT)

ENTRY POINT: CPHALT
CHART: OV
FUNCTIONS:
- Allows CCP to halt in case of an internal error or to intercept a branch to location zero by a processing program.
- Allows, in a DPF system, the non-CCP program level to terminate normally before dumping the CCP to determine the cause of the halt.

OPERATIONS:
- Saves the contents of the current ARR, XR1, and XR2 in a location preceded by an eyecatcher constant of HALT.
- Restores the DSM low main storage entry address of Dump, General Entry, disk IOS, and disk I/O wait.
- Ensures that re-entry causes a dump by moving a load IAR instruction into the first instruction of the routine.
- Determines if the invoker is running under the CCP's IAR. If not, immediately stop the system with a processor check instead of halting. This is necessary as the invoker may be running in an interrupt level and, if so, an HPL instruction will not work as it is treated as a no-operation by the system hardware.
- HALT with a U– (or a 1234 halt on Model 4) in the first (or primary) halt.
- If the system operator presses HALT RESET/ START (non-DPF), HALT with a second subhalt moved from the two bytes immediately following the value in the ARR at the time of entry to the routine.
- Go to DSM Dump entry when the system operator resets the halt.

INPUT: The two bytes addressed by the input ARR.
EXIT: To DSM Dump routine via load of program level one's IAR. (The DSM Dump is dependent upon running under program level one IAR.)

Generalized Move–Service Routine

ENTRY POINTS:
- $CC4MX – Moves data item of length 1 – 65535 with possibly different target and source lengths:
  - If lengths are equal, performs simple move.
  - If target shorter than source, moves only first n bytes of source to target (up to target length).
  - If target longer than source, moves source to first n bytes of target, and fills remaining right end of target with character blanks.
- $CC4MV – Performs simple move, given only a single length.

CHART: OX
FUNCTION: Moves item of length 1 – 65535.

INPUT:
- XR2 must point to the leftmost byte of the user's move parameter list, which must contain:
  - In bytes 0 – 1, the length of the target area (this is the *only* length specified to $CC4MV).
  - In bytes 2 – 3, the leftmost address of the target area.
  - In bytes 4 – 5, the leftmost address of the source area.
  - In bytes 6 – 7, (these bytes are required for $CC4MX *only*) the length of the source data.
- The source data, of any form whatever.

OUTPUT:
- The target area, set to the contents of the source data.
- For $CC4MX *only*, the shorter of the two lengths specified returned in bytes 0 – 1 of the user's parameter list.
- For $CC4MX *only*, the condition register set as follows:
  - Equal if the target was equal to or longer than the source.
  - Low if the target field was shorter than the source (that is, when truncation of the source data occurred.

EXIT: Always to invoker at address contained in ARR.
EXTERNAL REFERENCES: None
NOTES:
- This routine must not be passed a length of 0. If a length of 0 is specified, it will be treated as a length of 256.
- The move is performed in a strict right-to-left fashion.
- XR1 is saved, used, and restored.
- XR2 is used as pointer to parameter list and is not modified.

Getmain/Freemain Service Routine ($CC4MS)

ENTRY POINTS:
- $CC4GM – Getmain entry (normal).
- $CC4RM – Getmain entry (get segment at edge of storage pool).
- $CC4FM – Freemain entry.

CHART: OZ
FUNCTIONS:
- Allocates a free segment or a part of a free segment from a predefined main-storage pool to the requester (Getmain).
- Return to the main-storage pool a previously allocated storage segment (Freemain).

INPUT:
- XR1 must point to a main-storage control block of 10 bytes, which must be *outside* the allocatable storage pool, and which must contain the following fields:

Name	*Disp*	*Len*	*Contents*
SCBCHN	+1	2	Address of first free segment (00XX if none).
	+3	2	Always X'0000'.
SCBLO	+5	2	Address of low-bound of storage pool.
SCBHI	+7	2	Address of high-bound of storage pool +1.
SCBMAX	+9	2	Size of largest free segment in storage pool (initially, size of entire pool).

- XR2 must point to the user's request parameter list, which must contain the fields:

Name	*Disp*	*Len*	*Contents*
GMADDR	+1	2	Getmain – Content does not matter. Freemain – Hi-order address of segment to free.
GMSIZE	+3	2	Getmain – Size of storage segment required. Freemain – Actual size of segment to free.

- Initially, the storage pool area must have been established with the following content in the first four bytes thereof:

Name	*Disp*	*Len*	*Contents*
SEGCHN	+1	2	X'0000'
SEGLEN	+1	2	Size of entire storage pool (including these four bytes themselves).

OUTPUT:
- For a normal Getmain ($CC4GM):
  - A condition–code setting, indicating the success or failure of the attempt to allocate a segment: Logical overflow *off* if successful, logical overflow *on* if unsuccessful.
  - If (and only if) successful, the parameter list field GMADDR is set to the address of the leftmost byte of the storage segment allocated to the user.
  - If (and only if) successful, the storage segment, whose content has not been set to any determinate value, allocated on the basis that it is the smallest segment available that will satisfy the size requested by the user.
- For a restrictive Getmain ($CC4RM), output identical to that of $CC4GM, except that the segment is not allocated unless it can be allocated on the high- or low-bound of the total storage pool.
- For Freemain ($CC4FM), the segment specified in the user's parameter list is returned to the storage pool.
- In all cases, the index registers upon return contain the same addresses as they contained upon entry.

EXITS, NORMAL: Always to user at his ARR address.
EXITS, ERROR: On erroneous Freemain (Freemain outside bounds of storage pool), a branch is made to the CCP abort routine, CPHALT, yielding a halt of U-- and a subhalt of FE, abortively terminating the CCP run.

Main-Storage Supervisor (Getmain/Freemain Service) ($CC4MM)

ENTRY POINTS:
- $CC4GM – Getmain entry (normal).
- $CC4RM – Getmain entry (effectively identical to $CC4GM).
- $CC4FM – Freemain entry.

CHART: OZ
FUNCTIONS:
- Allocates a free segment or a part of a free segment from a predefined main-storage pool to the requester (Getmain).
- Return to the main-storage pool a previously allocated storage segment (Freemain).

INPUT:
- XR1 must point to a main-storage control block of 10 bytes, which must be *outside* the allocatable storage pool, and which must contain the following fields:

Name	*Disp*	*Len*	*Contents*
SCBCHN	+1	2	Address of first free segment (00XX if none).
	+3	2	Always X'0000'.
SCBLO	+5	2	Address of low-bound of storage pool.
SCBHI	+7	2	Address of high-bound of storage pool +1.
SCBMAX	+9	2	Size of largest free segment in storage pool (initially, size of entire pool).

- XR2 must point to the user's request parameter list, which must contain the fields:

Name	*Disp*	*Len*	*Contents*
GMADDR	+1	2	Getmain – Content does not matter. Freemain – Hi-order address of segment to free.
GMSIZE	+3	2	Getmain – Size of storage segment required. Freemain – Actual size of segment to free.

- Initially, the storage pool area must have been established with the following content in the first four bytes thereof:

Name	*Disp*	*Len*	*Contents*
SEGCHN	+1	2	X'0000'
SEGLEN	+3	2	Size of entire storage pool (including these four bytes themselves).

OUTPUT:
- For a Getmain ($CC4GM or $CC4RM):
  - A condition-code setting, indicating the success or failure of the attempt to allocate a segment: Logical overflow *off* if successful, logical overflow *on* if unsuccessful.
  - If (and only if) successful, the parameter list field GMADDR set to the address of the leftmost byte of the storage segment allocated to the user.
  - If (and only if) successful, the storage segment, whose content has not been set to any determinate value, allocated on the basis that it is the smallest segment available that will satisfy the size requested by the user.
- For Freemain ($CC4FM), the segment specified in the user's parameter list returned to the storage pool.
- In all cases, the index registers upon return contain the same addresses as they contained upon entry.

EXITS, NORMAL: Always to user at his ARR address.
EXITS, ERROR: On erroneous Freemain (Freemain outside bounds of storage pool), a branch is made to the CCP abort routine, CPHALT, yielding a halt of U– and a subhalt of FE, abortively terminating the CCP run.
NOTES: This routine (shorter than $CC4MS) is used in place of that routine in a single-user-task system without DFF.

Command Processor Mainline ($CC4CP)

ENTRY POINT: $CC4CP
CHART: PC
FUNCTION: Routes terminal and console commands to the appropriate command processing routine.
OPERATIONS:
- Accepts input from command or the console terminals.
- Routes control to the appropriate command processing subroutine.

INPUT: The address of the input device's terminal unit block in command processor's TCB field TCBINQ.
EXITS:
- To $CC4R4, $CC4PC, or $CC4PR for processing.
- To $CC4WT when no work is available.

Allocation Task Resident Control Routine ($CC4AM)

ENTRY POINT: $CC4AM
CHART: PF
FUNCTION: Invokes the first allocation control routine when there is work for the allocation task.
OPERATIONS:
- Allocation is entered when program request validation has found an available TCB for Allocation or Termination has found the allocate post pending bit on in the Command Processor work area (set by program request processing).
- This routine will invoke the first Allocation transient, $CC4A1.
- When control is returned to this routine, the allocation work area wait mask is tested to see if the allocation is complete.
- If allocation is complete this routine exits to the CCP Dispatcher indicating to exit the must complete mode and to give control to the user program.
- If allocation is not yet complete, this routine will wait for all required resources to become available. When this occurs the task is posted and made dispatchable. At this time it will re-invoke the first allocation transient, $CC4A1.

INPUT: None
OUTPUT: None
EXITS:
- $CC4DP when user task allocation has completed.
- $CC4WT – To wait for resources to become available for allocation.

NOTES: Allocation is accomplished in two passes:
- The first pass allocates whatever resources are available and sets indications for all unavailable resources that it needs.
- The second pass runs when all resources that were not available during the first pass become available.

Open/Close/Allocate Mainline ($CC4OC)

ENTRY POINT: $CC4OC
CHART: PI
FUNCTIONS:
- Calls the initial transient load of Open/Close/Allocate.
- Invokes DSM to process requests to set new limits.
- Primes and purges buffers for open/close requests.
- Invokes Termination of Open/Close/Allocate was unsuccessful.

OPERATION: Open/Close/Allocate transients will return to one of four return points in $CC4OC to perform final processing:
- OCRET1 – Invokes DSM to set new limits. The returning transient will have placed the C/S of the appropriate DSM limits transient into the parameter list.
- OCRET2 – Returns to the calling program.
- OCRET3 – Terminates the calling program. The returning transient will have set the task completion code into the parameter byte.
- OCRET4 – Purges or primes buffers following successful open/close.

INPUT: Address of the DTF chain in XR2.
OUTPUT: $CCCOM and current TCB flags set to restore control to the calling program.
EXIT, NORMAL: Return to the calling program via the Dispatcher.
EXIT, ERROR: CC4TI2 to terminate the caller.

Termination Interface Routine ($CC4TI)

ENTRY POINTS:
- $CC4TI – Entry from a CCP system task to terminate another task (user).
- CC4TI1 – From General Entry Intercept to terminate a user task that has issued an EOJ RIB (X'84') in the normal course of its processing.
- CC4TI2 – From a CCP service routine that has detected an error condition attributable to the user program and wants to terminate the task.

CHART: PL
FUNCTIONS: Provides an interface for CCP system functions or CCP system tasks to cause the termination of a user program.
OPERATIONS:
- Finds the user task control block to be terminated.
- Moves the task completion code value into the TCB completion code field.
- Marks the TCB that it is to be terminated.
- Makes the CCP Termination Task dispatchable if it is not currently busy.
- Exits to either the CCP Dispatcher or to the invoker of Termination Interface.

INPUT:
- CC4TI – ARR addresses the task completion code. Register two contains the address of the TCB to be terminated.
- CC4TI1 – ARR points to a DSM RIB of X'84'. TCB to be used is the current TCB address (@CURTB) in $CCCOM.
- CC4TI2 – ARR addresses the completion code to be used. Address of the task to terminate is in current TCB address field in $CCCOM (@CURTB).

EXIT, NORMAL:
- To the invoker via the input ARR plus one.
- To the CCP Dispatcher ($CC4DP).

EXIT, ERROR:
- To CPHALT with a subhalt of OS if the task to be terminated is a CCP system task.

Termination Task Mainline Routine ($CC4TM)

ENTRY POINT: $CC4TM
CHART: PL
FUNCTION: Provides the resident interface required to the Termination Task transient routines.
OPERATION: At initial entry, goes to the CCP Transient Area Handler to invoke the first Termination transient routine, $CC4TD.
- After the Termination transients have completed processing, waits for work to do.

INPUT: None
OUTPUT: None
NOTE: The termination task is not posted using the normal CCP post routine if a task is to be terminated and Termination is not currently busy (as indicated by the bit $TMBSY in the Termination Work Area ($TMFLG in $TMWK)). Instead the TCB of Termination is found and the dispatchable bit is turned on by the invoker of Termination (the termination interface or the Cancel Command Processor).

I/O Interface Mainline ($CC4II/$CC4IS)

ENTRY POINTS:
- $CC4II – Entry for user requests.
- $CC4IS – Entry point for system requests.

CHART: PP
FUNCTION: Processes both user and system I/O requests. Extensive diagnostics are performed on each request prior to scheduling the TP operation.
INPUT: Input to $CC4II is a parameter list pointed to by XR2.
OUTPUT: Output from $CC4II will be a scheduled put operation, data on a get operation, or one of the following termination codes in case of error:
- 01 – Invalid op code/modifiers.
- 02 – Invalid operation for console.
- 03 – Undefined symbolic terminal name.
- 04 – Terminal not allocated to program.
- 05 – Terminal referenced by other than allocated name.
- 06 – Blank terminal name for MRTS program.
- 07 – Blank STN and requester released.
- 08 – Blank STN invalid for this operation.
- 09 – Invalid use of sub-terminal name.
- 0A – STN not assigned to terminal.
- 0B – Terminal attribute set invalid for this operation.
- 0C – Terminal I/O capability does not match operation.
- 0D – Invalid output length.
- 0E – Invalid input length.
- 0F – Input length greater than TP buffer size.
- 10 – Invalid op with data from prog. req. outstanding.
- 11 – Invalid op with invite input outstanding.
- 14 – Invalid op with no invites outstanding.
- 15 – Input area not large enough for BSCA get block.
- 16 – Copy requested to terminal without mapping.
- 17 – Copy to terminal name not found.
- 18 – Copy invalid with 3275.
- 19 – Erase requested to terminal without mapping.
- 1A – Put override to terminal without mapping.
- 1B – Invalid put with mapping – No EOT.
- 1C – Record area too small for BSCA get with ITB.
- 1D – Accept request impossible to satisfy.
- 1E – Ourput length to console greater than 80.
- 1F – Output length is greater than TP buffer length.
- 35 – DFF terminal referenced by a non-DFF task.

EXIT, NORMAL: A normal exit to CCP Dispatcher occurs if all goes well.
EXIT, ERROR: An error exit to Termination with the appropriate termination code occurs in case of error.

Display Format Control Routine ($CC4DF)

ENTRY POINT: There is only one entry point from $CC4II – DFA000. However, a task under which this routine is executing can be interrupted (waiting for availability of OHA, put blocking, etc). This routine will cause the task to wait and be posted later at DFB850 and DFH070.

CHART: PV

FUNCTIONS:

- Provides logical data management support for components of the 3270 system. This mainline routine will control a series of transients which operate in the general transient area in CCP. The prime function is to screen out the 3270 line and data control characters between the user and the 3270 system.
- This routine is field-orientated, table-driven, segmentally reusable. It is capable of servicing more than one task at a time.
- The functions provided for the various op codes are as follows:
  - Get message and stop-invite-input – The user parameter list will be modified to tell $CC4CM how much of a hold area to Getmain for the 3270 text. After the op end occurs the data will be moved from the dynamic TP buffer to the user's record area. The address of the TP area which contains the 3270 text will be placed in the user's parameter list by $CC4CM. The parameter list will be restored to the original condition before returning to $CC4II. This routine will Freemain the TP area before returning to $CC4II, except if an error or exception condition exists, in which case the area will be freed by $CC4CM.
  - Invite input – The PLDATA and PLINL as set by the user is ignored by this routine. $CC4CM will Getmain an area for the 3270 text, and PLINL is filled in by this routine from TTINLH.
  - Accept input – This operation is handled for the most part like get message, except the data area address of the 3270 text and PLEFFL is obtained from the TUB. There is no Freemain of the dynamic TP area by this routine since $CC4II will do that. This routine will not handle the data which enters the data with the program request. For this particular operation (the implied invite-input-accept), the AID will not be the first character in the data area. That operation will be handled entirely by $CC4CM.
  - Put message – PLOUTL in the user's parameter list is modified by this routine to reflect the actual amount of 3270 text being sent by this routine. The text is read from the display format in the object library into the output hold area (OHA). After the user's field data has been merged into the text, the text will be moved from the OHA to the line buffer by $CC4CM. Therefore, PLDATA is also modified by this routine. If blocking is required, all but the last block of text will be sent by posting $CC4CM in this routine. For the last block, or if no blocking is required, control will pass through $CC4II before $CC4CM is posted.
  - Put override – The transient $CC4DB will be evoked to diagnose the use of the operation and to calculate the length of text to generate, also determining if blocking is required. The 3270 text will be generated dynamically by this routine in the OHA. If blocking is required, the same procedure is used as that for put message.
  - Copy – The transient $CC4DC will be evoked to diagnose the use of the operation, and also to build tne text. The 3270 text will be built in PAS, and the user's parameter list will be modified to reflect this fact. The parameter list will be restored to its original condition before returning to $CC4II the final time. This operation will go through $CC4CM as a put message.
  - EAU – Erase all unprotected – The text for this operation is contained in PAS. The user's parameter list will be modified to reflect this fact. This operation is sent to $CC4CM as a put message operation. The user's parameter list will be restored before returning through $CC4II the final time.
  - Release terminal – This operation involves blanking out the TT entry for the respective terminal. No repercussions are taken out on the user for releasing terminal which had not previously been used with this routine.

INPUT:

- XR2 must contain the high order address of the user CCP parameter list. A user program appended storage area (PAS) must be available for storing data specific to this task.
- Display formats must have been already generated and must exist in the object library. The format must be in two parts. The first part is the Field Descriptor Table (FDT), and the second is the text stream for an initial put to a 3270 terminal.
- For all operations, PLRECA must point to the high-order byte of the terminal name for which the operation is intended (except for accept input, in which case the pointer is to the high order byte of the data area). Immediately following the 6 character terminal name must be the beginning of the data area.

For output operations, the following information is also required:

- Put msg – Display format name in first 6 positions of the data area, followed by any data for fields which were defined as begin defined at execution time.
- Put override – The WCC must be the first position in the data area. This is the minimum entry for this operation. Optionally, 9 positions should be given for each field which is to be overriden: 6 for field name 1 each for type, cursor, and modify.

- For all output operations, the text is either read into the output hold area (OHA) or built directly into it. Only one output operation may be performed per task and per line in an OHA.

OUTPUT:

- Formatted data in users record area for input operation, or proper text stream sequence to 3270 for output operation.
- Updated areas in PAS to reflect current status of operation for this task.
- If a task will be terminated, the transient $CC4DD will be evoked to determine if the console message 528 should be issued.

DIAGNOSTICS:

- The following diagnostics are performed and issued by this routine. (Others are given by $CC4DB and $CC4DC.)
  - Termination codes:
    FF – 255
    FE – 254
    FD – 253
    FC – 252
    FB – 251
    F9 – 249
    F8 – 248
    F6 – 246
    F3 – 243
    F2 – 242
  - Error return code: RCRIDR – FFE3
  - Exception return code: RCXDTR – 0001
- Called by – $CC4II – There will be two calls to this routine for the following operation codes, once before evoking $CC4CM and once after evoking $CC4CM:
  - Put message.
  - Put overrides.
  - Get message.
  - Invite input.
  - Copy.
  - Erase all unprotected.
  - Stop invite input.

The following operation codes will cause $CC4II to call this routine once:

- Accept input.
- Release terminal.

This routine will also be posted by the CCP Dispatcher.

EXITS:

- $CC4CM – Communication management
- $CC4PS – Post
- $CC4WT – Wait
- $CC4MX – Move
- $CC4DB – DFCR transient to diagnose put overrides.
- $CC4DC – DFCR transient to build copy parm list.
- $CC4DD – DFCR transient to issue message 528.
- $CC4PI – Transient initiator
- $CC4FM – Freemain
- CC4TI2 – Terminate a task
- NCEIOS – Disk IOS
- NCEIOW – Disk wait

NOTES:

- This routine is loaded TO CORE in main storage in the user program area when a user program is loaded and it requires the services of this routine. This routine is loaded by $CC4CR.
- There are references in this routine to routines within the resident CCP. These addresses are resolved by $CC4CT when this routine is loaded. The pointers to these references are located in the first few bytes of this routine, and are overlayed, after the references have been resolved, by constants which this routine uses.
- All task dependent information is kept in the PAS attached to the user program.
- Control will not be taken away from this routine after a hardware interrupt.
- $CC4DA is a transient which is also part of DFCR, and it executes under the allocate function of CCP. Its responsibility is to initialize the PAS.

CCP Trace (CCTR)

ENTRY POINT: CCTR
CHART: PY
FUNCTION: Provides a chronological log of significant CCP system events.
OPERATIONS:

- Trace records, in a circular in-core table, pertinent information regarding the flow of control through CCP.
- Optionally, when the table overflows, it will be written to $CCPFILE for subsequent printing.

INPUT: The current TCB address field in $CCCOM, the invoker's ARR, XR1 and XR2, and the ARR, XR1, and XR2 in $CCCOM saved by some of the routines which invoke Trace.

OUTPUT: The input data into a circular trace table and optionally, the trace table written to disk when the in-core table overflows.
EXIT, NORMAL: To invoker via the ARR.
EXIT, ERROR: None – If Trace I/O error occurs, the Trace I/O will be turned off by Trace I/O routine itself.
EXTERNAL REFERENCES: See input.

Trace Halt Service Aid (CCSA)

ENTRY POINT: CCSA
CHART: PZ
FUNCTION: Facilitates halting CCP when specific events are traced.
OPERATION: CCSA is entered whenever CCP Trace is invoked to trace an event within the CCP program level. The CPU address switches are sensed to determine the action taken. Two halts may appear:
- A small d, capital P indicates that trace was invoked to trace the specified event.
- A small c, small u indicates that X'FFFF' was sensed. The desired trace ID may now be dialed in the address switches.

Note: On Model 4, these halts are:
dP = ABC2345
CU = ABC12

INPUT:
- Address of $CCCOM in XR1.
- Address of the Trace Control Block in XR2
- CPU address switches

OUTPUT: None
EXIT: Return to CCTR via ARR
NOTES:
- When loaded by Startup, CCSA is in initial status and ignores all trace entries until X'FFFF' is sensed in the CPU address switches. When X'FFFF' is sensed, a 'cu' halt appears. The desired trace entry ID may now be dialed in the righthand address switches and HALT RESET pressed. When CCSA is entered with a matching ID in the current trace entry, a 'dP' halt appears. If HALT RESET is pressed, and X'FFFF' is not dialed in the address switches, CCSA returns control to CCP. If X'FFFF' is dialed in, the 'cu' halt appears to allow the specification of a new trace ID. A response of '0000' to the 'cu' halt returns CCSA to initial status. If an invalid trace ID (that is, not 'XX00'–'XX1E') causes re-issuance of the 'cu' halt, a trace ID of '00' causes a halt on every trace entry.
- CCSA never halts if '0000' is dialed in the address switches.

Reschedule the TP Line (CMRSCH)

ENTRY POINTS:
- CMRSCH – Reschedules work on the line.
- CMPAII – Posts requester if TP scheduled.
- CMTSRQ – Checks for more work to be performed.

CHARTS: QA and QG
FUNCTIONS: Schedules work on lines that are not busy, or stops polling on a line and schedules put operation if put operations are pending in the line queue.
OPERATIONS:
- If the line is busy and no put operation is in the queue, posts requester if TP was scheduled. Checks for more work that can be done.
- If line is busy polling and put is in the queue, cancels the get and handles the put.
- Clears any BSCA op ends left in the LCB, and adjusts the op end total count (CCOECT).
- If MLTA OLT is in process, calls in transient to continue the OLT operation.
- If queue for line is empty, posts the requester. If TP was just scheduled, checks for other function that can be performed by CM.
- If a BSCA poll for status operation in the queue, handles it before any other operation.
- If a put operation can be started. performs it next.
- If neither of above two, then tries to schedule an input operation. If no input to schedule, posts the requester if TP was just scheduled. Then checks for other CM function that can be performed now.
- If a write operation can be started, sets up the output data buffer for the line. Translates the data as required or specified. Adds device dependent control characters.
- If new operation on a BSCA, carves up the IOB(s) and line buffer(s) as necessary to perform the operation. Sets up the line dependent section of the DTF. (Polling/addressing characters, switch ID verification IDs, switch line call/answer options.)
- If BSCA operation, sets up to do the get or put-normal, put-block, put-end of file, or put-EOT-to-WACK operation, as appropriate.
- If MLTA operation, sets up to do the read, write with the appropriate disconnect, continue or conversational option as required by the terminal and/or current line circumstance.
- Issue of IOCS call after the DTF is setup. For BSCA, calls $$BSMS; for MLTA, calls MLTIO1.
- After issuing the IOCS call, traces the results through $CC4TT.

- If MLTA start operation code indicates a start failure, calls in a transient to determine the problem. Otherwise, posts the requester if TP was just scheduled. Checks for more work for CM to perform at this time.
- If BSCA operation is completed without an op end interrupt, fakes an op end interrupt to keep the function going til completion. Posts the requester if TP was just scheduled. Checks for more work to do at this time.
- If checking for more work to be done, then:
  - Checks for MLTA OLT start failure. If yes, goes back and tries to reschedule the operation.
  - If BSCA parm list was just queued which is the continuation of a previously started line operation, then handles the next operation it causes.
  - If there are any op ends to be handled, calls the op end handler.
  - If there is a Freemain posted or a TP parameter list to be scheduled, calls the TP scheduler.
  - If there is a line to be rescheduled because of ERP, goes back to the start of the reschedule function.
  - If there is a console function pending, gives control to the KM console management function.
  - Otherwise, if none of above, waits for more work.
- If posted with work to do, gives control to the KM function.

INPUT:
- CMSLCB – Address of DTF to be scheduled.
- CMSPL – Address of TP parameter list to be scheduled.
- CC@PRL – Address of TP parameter list last scheduled.
- CCOECT – Op end pending count.
- TCBMPK – Post mask of work to do.
- #CMERP – Address of DTF for rescheduling because of ERP.
- @KMQTB – Address of console parameter list awaiting operation.

OUTPUT:
- CMSLCB – Address of DTF for line scheduled.
- CMSPL – Address of TP request scheduled.
- DTF (LCB), TUB – Set up for the operation performed.

EXIT, NORMAL:
- To CMFRMN if Freemain posted or TP request to be handled.
- To CMRSCH if another line to reschedule.
- To CMOPND if an op end to be handled.
- To KMINTR if a console operation to be handled.

EXTERNAL REFERENCES:
- $CC4BC – Stops polling to handle put on BSCA line.
- $CC4MT – Handles MLTA OLT continuation.
- $CC4BO – Formats 3270 command output.
- $CC4WR – Handles translate errors in output.
- $CC4JX – X is the appropriate line translate transient.
- $CC4SC – MLTA start code failure transient.
- $CC4TT – CCP Trace Routine.
- $CC4WC – Switch line call/answer log transient.
- $CC4WT – CCP wait/post routine.
- CMIVGM – Invite input/get operation schedule routine.
- CMRQBF – Posts requester of TP scheduled.
- CMMCT – Sets up multi-component terminal addresses.
- CMGINL – Sets up input record length.
- CMPSCH – Finds switch ID entry in switch ID list.
- CMASCH – Finds addressing entry in addressing list.
- MLTIO1 – MLTA IOCS.
- $$BSMS – BSCA IOCS.

Op End Analysis (CMOPND)

ENTRY POINT: CMOPND
CHARTS: QA and QC
FUNCTIONS:
- Analyzes each op end for a TP line and determines what if anything must be done to complete the current TP operation.
- Routes completed operation back to the user.
- Reschedules work on the line if no more op ends to be handled.

OPERATIONS:
- If no op ends, gives control to TP scheduler to check for Freemain posted.
- Sets up the check list to only look at BSCA DTFs that have operation completed. Looks at all MLTA DTFs.
- For BSCA DTFs that are set up for check, sets up the DTF and TP parameter list fields as appropriate for the operation. Sets up the TUB status, record length and line ownership fields. Checks for Data Mode Escape request on input operations.
- Calls the common TP check routine with the address of check list in XR2.
- For BSCA, if 56 completion code, adjusts op end count in appropriate BSCA DTF. Traces the operation. *Note:* The trace entry will show the current completion code of the DTF, not necessarily the 56.
- If abort operation, handles the abort until it is complete. Then reschedules the line.
- If stop invite request, calls the appropriate transient to handle the request. (BSCA–$CC4BQ, MLTA–$CC4SQ.)

- If MLTA operation and swallow the data operation, then ignores the incoming data.
- If MLTA OLT op end, calls transient to handle the operation. Then goes to schedule the next operation.
- Finds the op ended parameter list and sets on the poll skip bit.
- Traces the op end after calling check through $CC4TT.
- If an error occurred, calls the appropriate ERP transient. (BSCA–$CC4BE, MLTA–$CC4MA.) On return, either posts the results to the user, reschedules the line, or handles the data in the line buffer.
- If write op end, then:
  - If BSCA record was shorted then defined at assignment time, calls transient to fill out the record to correct length.
  - Sets up the return code for the results of the operation.
  - If BSCA is not complete, calls reschedule function to complete the operation:
    - EOT must be sent.
    - Operation not complete because CLEAR key is being handled.
  - If BSCA operation is complete, then:
    - If DFF PUT, frees the DFF hold buffer.
    - If op end of a user PUT, sets off PRUF indicator (TUBSCS) in the TUB.
    - If op end of a PUT-PRUF, sets on PRUF indicator (TUBSCS) in the tub.
  - If put-then-get, sets up the get operation in the internal op code, then reschedules the line.
  - If operation is completed, sets up to post the requester of the results.
- If read op end, then:
  - If BSCA receive initial was last op, checks for any conflicting operation in the line queue, and rejects them if any found.
  - If BSCA message mode input, sets up to consolidate all blocks into one message. Sets up to reschedule line and get to EOT before returning to the user.
  - If Data Mode Escape recognized in the input data stream, sets up post of the Command Processor.
  - If valid input data, moves, translates, and truncates the data as required.
  - If 3270 BSCA input, checks for CLEAR key hit and handles it.
- If completed data operation, then:
  - Removes the TP request from the line queue.
  - Frees up put-no-wait hold buffers.
  - Posts the requester of the TP operation that it is complete.
  - Queues invite input parameter list onto the TCB.
  - Checks for more op ends and handles if there are some, otherwise goes to reschedule line.

INPUT:
- CCOECT – Op end pending count.
- CCLCB1 – Address of list of TP lines in CCP.
- CCKLST – Address of CCP check list.

OUTPUT:
- CCOECT – Adjusted for op ends handled.
- CMSLCB – Address of DTF handled for last op end.
- CMSPL – Address to TP parameter list for last op end.
- TCBINQ – Queue of invites that have completed.

EXIT, NORMAL:
- To CMRSCH to reschedule work on the line.
- To CMOPND if more op ends to be handled.
- To CMFRMN if no op ends to handle.

EXTERNAL REFERENCES:
- $CC4BI – BSCA DME Transient.
- $CC4BQ – BSCA Stop II Queue Analysis Transient.
- $CC4SQ – MLTA Stop II Queue Analysis Transient.
- $CC4SK – MLTA Skip Bit Transient.
- $CC4T2 – MLTA OLT Op End Transient.
- $CC4TT – CCP Trace Routine.
- $CC4BE – BSCA ERP Transient.
- $CC4MA – MLTA ERP Transient.
- $CC4BB – BSCA Record Blank Transient.
- $CC4PS – CCP Post Routine.
- $CC4BR – BSCA Reject Transient.
- $CC4BA – 3270 Sense/Status Transient.
- $CC4B5 – 3735 Sense/Status Transient.
- $CC4B7 – 3741 Sense/Status Transient.
- $CC4JX – X is appropriate translate transient.
- $CC4WR – Translate Error Transient.
- $CC4BO – 3270 Input Format Transient.
- CMBSKP – BSCA Skip Bit On/Off Routine.
- CMONSK/CMOFSK – MLTA Skip Bit On/Off Routine.
- CMGINL – Input Record Length Routine.
- CPHALT – CCP Halt Routine.
- CMFMRT – Freemain Routine.
- CMGMRT – Getmain Routine.
- CMPSRQ – Post TP Scheduled Routine.
- $$BMCH – Common TOP Check Routine.

Accept TP Requests (CMFRMN)

ENTRY POINTS:
- CMFRMN – Checks for Freemain posted.
- CMREQ – Handles new TP request parameter list.

CHARTS: QA and QE

FUNCTIONS:
- Accepts TP parameters from the system and/or user.
- Performs the function requested if it can be handled immediately, otherwise, places the parameter into the LCB line queue of work to be done.

OPERATIONS:

- Before handling any new TP parameter list, unless the parameter list is a put-no-wait, handles any line rescheduling resulting from a Freemain post, which indicates the availability of more main storage core. If a put-no-wait is received, schedules it before checking for Freemain posted.
- Sets up the internal operation indicators for the TP request just received.
- If this is a disconnect request, insures that the line is not connected to the specific terminal. If no invite request is involved, posts TP completed to the system. If an invite is indicated, treats like an invite only request.
- If a put operation to a terminal in CCP ERP, ignores the put operation. Posts the user TP complete if no input operation indicated. If a put-then-get, treats like a get/invite only operation.
- If DFF get request, makes internal op code an invite input and treats it like an invite input through the rest of CM.
- If MLTA online test (OLT), sets up the execution of the MLTA OLT request.
- If a stop invite or purge I/O, calls in the MLTA ($CC4PG) or BSCA ($CC4BP) series of transients to handle these operations. On return from these transients, does windup operations of scheduling TP request. If the BSCA transient indicates an abort of the line is required, sets up the abort operation.
- If a regular TP data operation to a BSCA line, checks to insure that it does not conflict with what is currently being performed on the line.
- If invite input request, moves the TP parameter list into the TUB parameter list field and uses the TUB parameter list for any further reference to this operation.
- If put-no-wait to MLTA or put-no-wait message to BSCA and space is available for the parameter list and data, moves the parameter list and data to a hold buffer and use the hold buffer for any further reference to this operation. Otherwise, treats the request like a put-wait operation.
  Note: A put-no-wait message to a BSCA switched line is also treated like a put-wait.
- If a TP request that requires data transfer, queues the request onto the line queue for the appropriate LCB.
- After queueing up a new TP request, checks the status of line to:
  - Include a new get/invite if polling is already going on.
  - Stop a polling sequence if a put request was just received.
  - Continue operation if the request just received is the continuation of a previously started BSCA line operation.
- Exits to the reschedule logic to:
  - Post completion of the TP scheduled operation.
  - Perform the next priority action on the line.
  - Check for other work within CM that can be done at this time.

INPUT:

- CC@PRL – Address of TP request parameter list.
- CMLLCB – Address of last LCB handled for Freemain posted check.

OUTPUT:

- CMSPL – Address of the TP request last placed into the line queue.
- CMSDTF/CMSLCB – Address of the line DTF for the last TP request scheduled.

EXIT: To the line rescheduling function for the reasons specified in the last point under operations.

EXTERNAL REFERENCES:

- $$CLOZ – Closes BSCA switched line for disconnect operation.
- $$OPEN – Re-opens the BSCA switched line after a disconnect operation.
- $CC4MP – Transient to ignore a put to a terminal in CCP ERP.
- $CC4T1 – Transient to handle MLTA OLT request.
- $CC4BP – Transient to handle BSCA stop II/purge I/O.
- $CC4PG – Transient to handle MLTA stop II/purge I/O.
- $CC4BR – Transient to reject a TP request that conflicts with a current line operation.
- CMIVGM – Handles a new invite input request.
- CMSTOR – Determines storage requirements for no-wait operations.
- CMGMRT – Gets hold buffer space for no-wait operations.
- CMSET – Formats hold buffer space for put-no-wait operations.

Invite Buffer Analysis Routine (CMIVGM)

ENTRY POINT: CMIVGM
CHART: QK
FUNCTIONS: Analyzes and obtains Getmain buffer for invite input operations in the line queue.
OPERATIONS:
- If BSCA poll for status is in the queue, sets all polling skip bits on.
- If LCB currently has a hold area, Freemains it.
- If get operation to be scheduled, sets up to handle it.
- If system invite to BSCA switched line owned by a user program, ignores the system request for now.
- Determines storage requirements to schedule each invite operation.
- If storage not available, ignores the request.
- If storage is available, sets up to handle it.
- If BSCA poll for status, schedules it by itself.
- Getmains the space needed to schedule operations on the line.
- Sets up the LCB hold area to reflect the space acquired.
- Sets up Getmain parameter list in the acquired area.

INPUT: At least one parm list in the line queue.
OUTPUT:
- XR1 – Address of the last TP parameter list in the queue.
- XR2 – Address of the DTF.
- CMSPL – Address of last get/invite parameter list not in CCPERP.

EXIT, NORMAL: To NSI of caller.
EXIT, ERROR: To CPHALT with a 4 halt if the Getmain fails.
EXTERNAL REFERENCES:
- CMFMR2 – Frees LCB hold buffer before starting analysis.
- CMSTOR – Determines storage need for invites.
- CMBSKP – Sets BSCA poll skip bits on/off.
- CMONSK/CMOFSK – Sets MLTA poll skip bits on/off.
- CMGMRT – Getmains hold buffer to schedule invites on the line.

Getmain Size Determination Routine (CMSTOR)

ENTRY POINT: CMSTOR
CHART: QL
FUNCTION: This routine calculates the amount of main storage needed for a particular type of TP request.
OPERATIONS:
- Always adds four bytes for the Getmain parm list.
- If a put operation, adds PLOUTL.
- If a put then get operation, adds 16 for the TP parameter list.
- If a poll for status operation, adds 20.
- If system invite only, adds CC#MCL and moves CC#MCL into PLINL.
- If user invite only, add PLINL.

INPUT:
- XR1 – Address of the TP parameter list.
- XR2 – Address of the DTF for the TP parameter list.

OUTPUT:
- XR1 – Not altered.
- XR2 – Not altered.
- #BUFND – Total amount of storage need for the TP request.

EXIT: To NSI of caller.

Format Put-No-Wait Area Routine (CMSET)

ENTRY POINT: CMSET
CHART: QM
FUNCTION: This routine formats a storage area for a put-no-wait operation.
OPERATIONS:
- If a system put-no-wait then get operation, moves the TP parameter list into the TUB.
- If any other put-no-wait, moves the TP parameter list into the first part of the Getmained area.
- Moves the Getmain parameter list into the acquired area.
- Finally, moves the data into the acquired area.

INPUT:
- XR1 – Address of the TP parameter list.
- XR2 – Saved and restored.
- GMLIST – Getmain parameter list.

OUTPUT:
- XR1 – Address of the TP parameter list.
- XR2 – Restored to entry value.
- Getmain area-formatted as defined above.

EXIT: To NSI of caller.
EXTERNAL REFERENCES: $CC4MV – Moves data into Getmain area from user TP parameter list.

Post TP Scheduled (CMPSRQ)

ENTRY POINT: CMPSRQ
CHART: QS
FUNCTION: Posts requester that his TP operation has been scheduled.
OPERATIONS:
- Posts the requester of the TP operation that it has been scheduled for execution unless the operation to be posted is to a terminal in command interrupt mode.
- Posts the Command Processor ($CC4CP) if the terminal is in command interrupt mode.

INPUT:
- XR1 – Address of the TP parameter list.
- XR2 – Saved and restored.

OUTPUT:
- XR1 – Address of the TP parameter list.
- XR2 – Restored to entry value.

EXIT: To NSI of caller.
EXTERNAL REFERENCES: $CC4PS – Posts the requester that the TP operation has been scheduled.

Operation Input Length Determination Routine (CMGINL)

ENTRY POINT: CMGINL
CHART: QU
FUNCTION:
- Determines the input record length.
- Sets truncated indicator if input length is less than actual data length.

OPERATIONS:
- Sets the input length for the operation.
  - Polls for status, (uses 20).
  - Record mode input
    - Uses PLINL if PLINL less than TUBRCL.
    - Uses TUBRCL if PLINL greater than TUBRCL.
- If not MLMP variable length record support, adjusts the input length if less data is available in the actual line buffer than is requested.
- If more data is available in line buffer than requested, then sets the truncated data indicator.
- If pure get request or a message mode input at other than receive initial time, returns to user after performing the above functions.
- Otherwise, adds four bytes for Getmain parameter list.
- Frees the current LCB hold buffer if it is larger than needed and Getmains the exact hold area needed.
- If Getmain area not available, waits for it to be available.
- Otherwise, sets up the acquired area for the invite input operation.

INPUT: XR2 – Address of the DTF.
OUTPUT:
- XR1 – Not restored.
- XR2 – Address of the DTF.
- $BDREL – Sets up for input length of the next record.

EXIT, NORMAL: To NSI of caller.
EXIT, ERROR: If space is not available, exit to CMPAII and wait until the required space is available.
EXTERNAL REFERENCES:
- CMFMRT – Freemains current hold buffer if larger than is needed.
- CMGMRT – Getmains exact buffer needed.

BSCA Trace Interface ($CC4BT)

ENTRY POINTS: $CC4BT – Initial and only entry to the routine.
CHART: QZ
FUNCTION: The function of this module is to provide the interface between the CCP MLMP routines and the BSCA trace routine. The entry point specified as an EXTRN in the MLMP routines is now an entry to $CC4BT and the BSCA trace routine is an O–module loaded by CCP Startup. $CC4BT tests for the BSCA Trace routine being in main storage by testing @BTRAC field in $CCCOM. If the field is non-zero, the field contains the entry address of the actual BSCA Trace Routine, and it is called. The Trace Routine returns control through $CC4BT and back to the user via ARR.
INPUT: None
OUTPUT: None
EXIT: To $$BSMT (BSCA trace) if the Trace is present in the system. When control returns, exit is to the caller via the ARR.
EXTERNAL REFERENCES: CC@BTR – Entry name of @BTRAC in $CCCOM which contains the address of the BSCA Trace Routine.

BSCA Set Polling Skip Bit Routine (CMBSKP)

ENTRY POINTS:
- CMBSKP -- All functions below.
- CMASCH – Only find the entry in the addressing list.
- CMPSCH -- Only find the entry in the polling/switched ID list.

CHART: RF
FUNCTION: Sets the skip entry indicator on/off in the BSCA polling list or the BSCA switched line ID verification list.
OPERATIONS:
- If this DTF or this particular operation does not involve one of the BSCA list, returns without setting skip bit.
- Otherwise, finds the entry in the appropriate list.
- Sets the skip bit as specified for all entries for the same terminal.

INPUT:
- XR1 – Address of the TP parameter list.
- XR2 – Saved and restored.
- CMB#SB – SBN2 (X'BA') Set the skip bit on, or SBF2 (X'BB') Set the skip bit off.
- TUBSID – ID of the entry(s) to perform the operation on.

OUTPUT:
- XR1 – Address of the TP parameter list.
- XR2 – Restored to entry value.
- Polling/switched ID list–skip bit for appropriate ID set as specified.

EXIT, NORMAL: To NSI of caller.
EXIT, ERROR: CMASCH/CMPSCH will return to NSI+3 if the entry is not found.

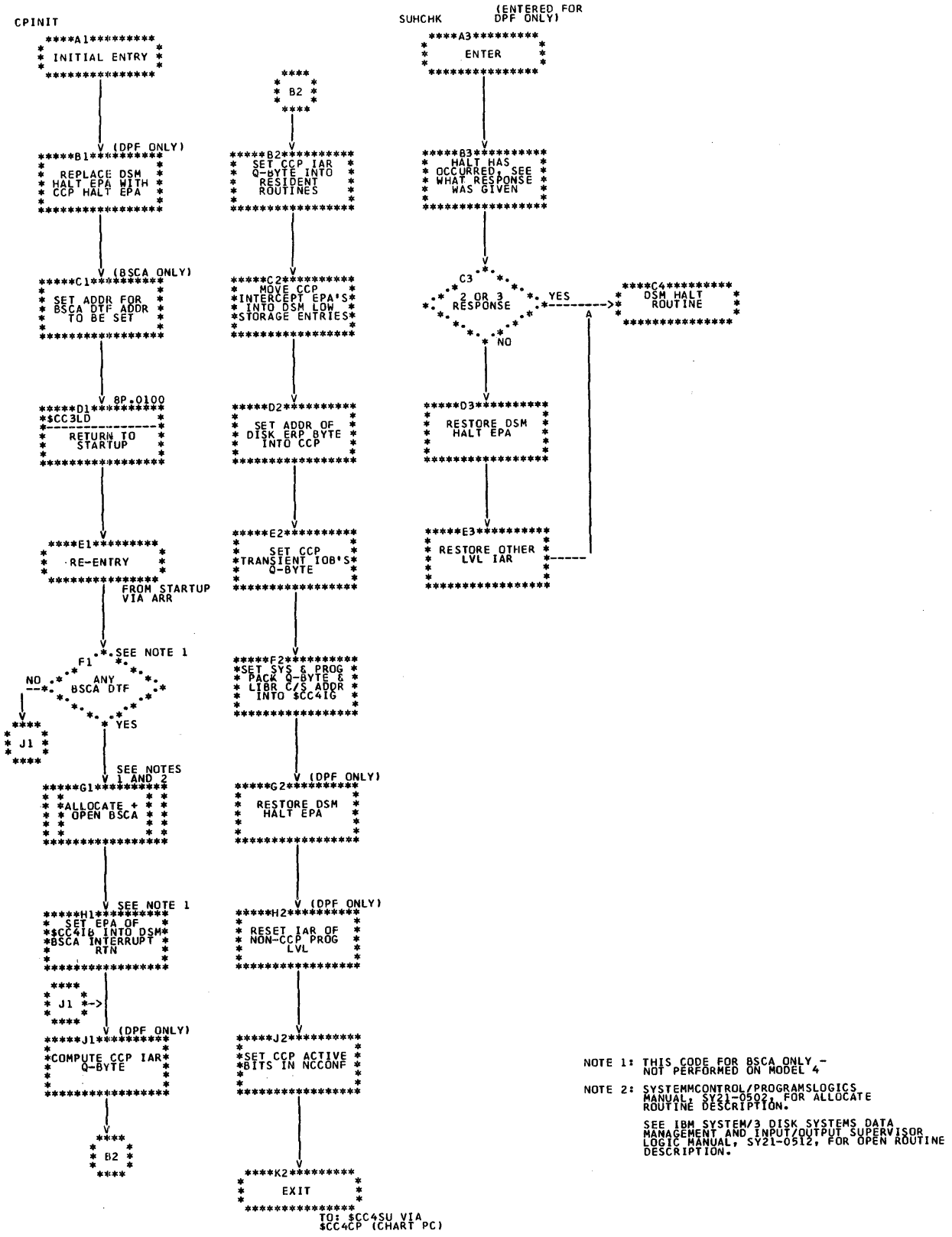


Chart OA. Resident Startup Initialization Routine (CPINIT)

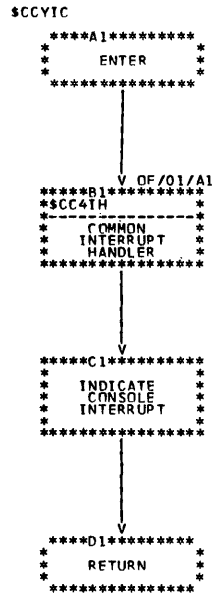


Chart OC. Console Interrupt Intercept ($CC4IC)

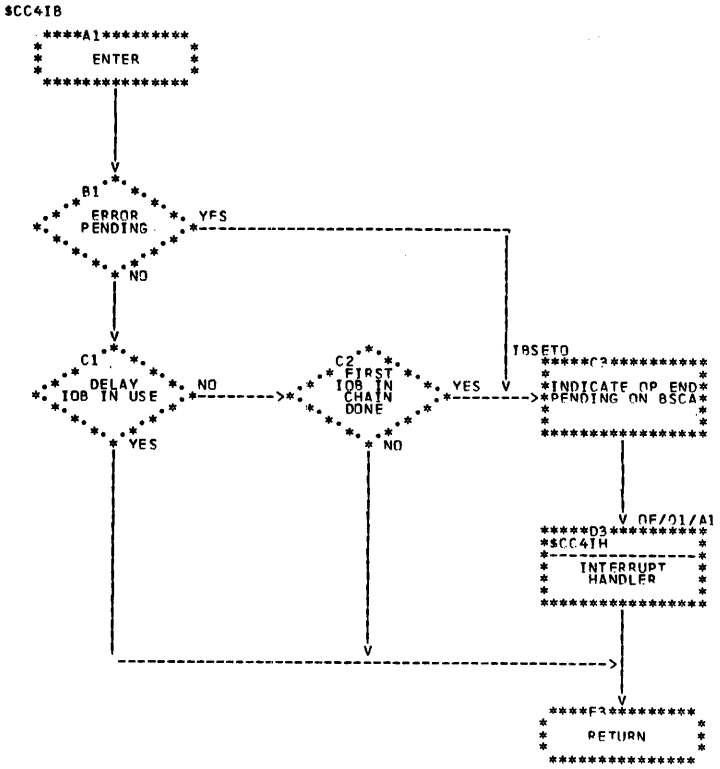


Chart OD. BSCA Interrupt Intercept ($CC4IB)

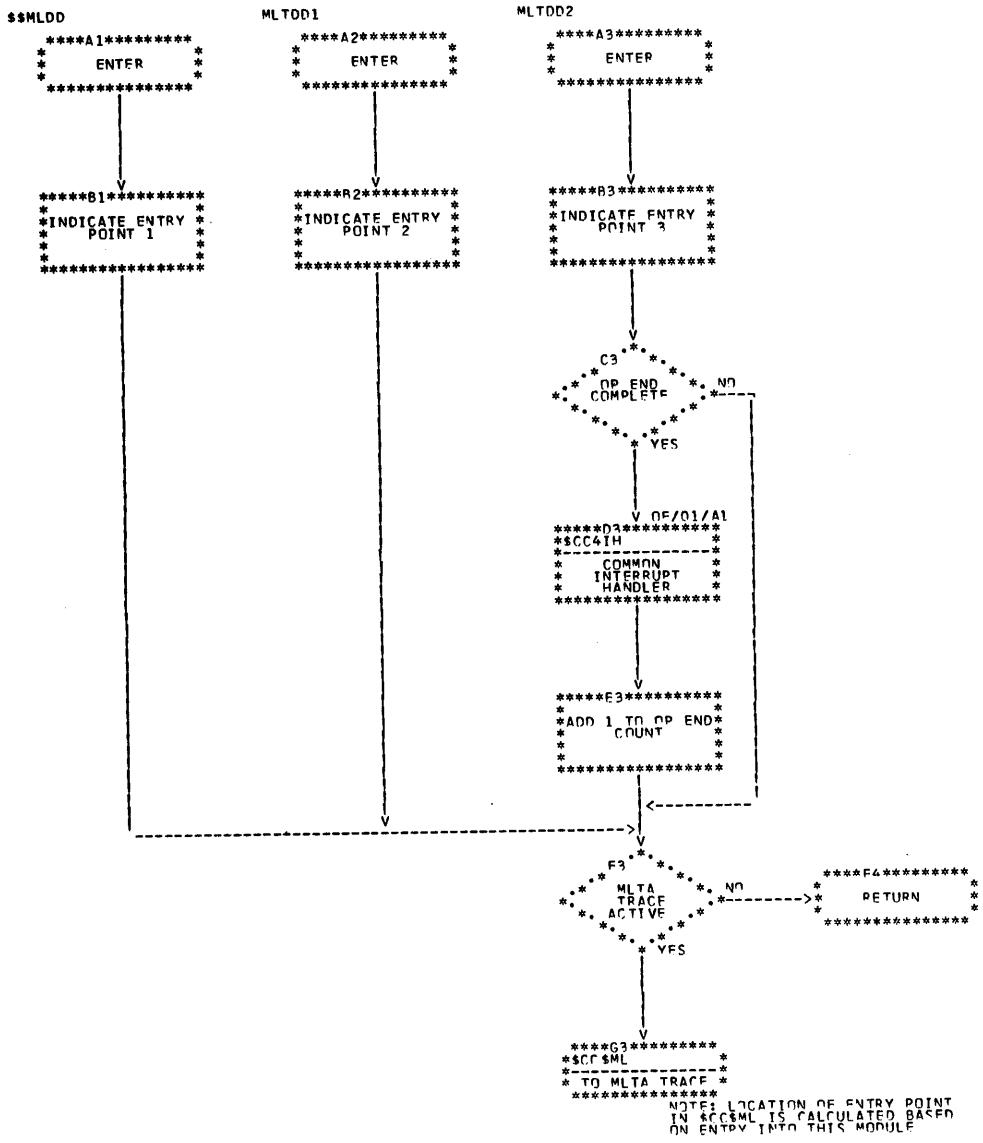


Chart OE. MLTA Interrupt Intercept ($CC4IM) (Models 8, 10, and 12 Only)

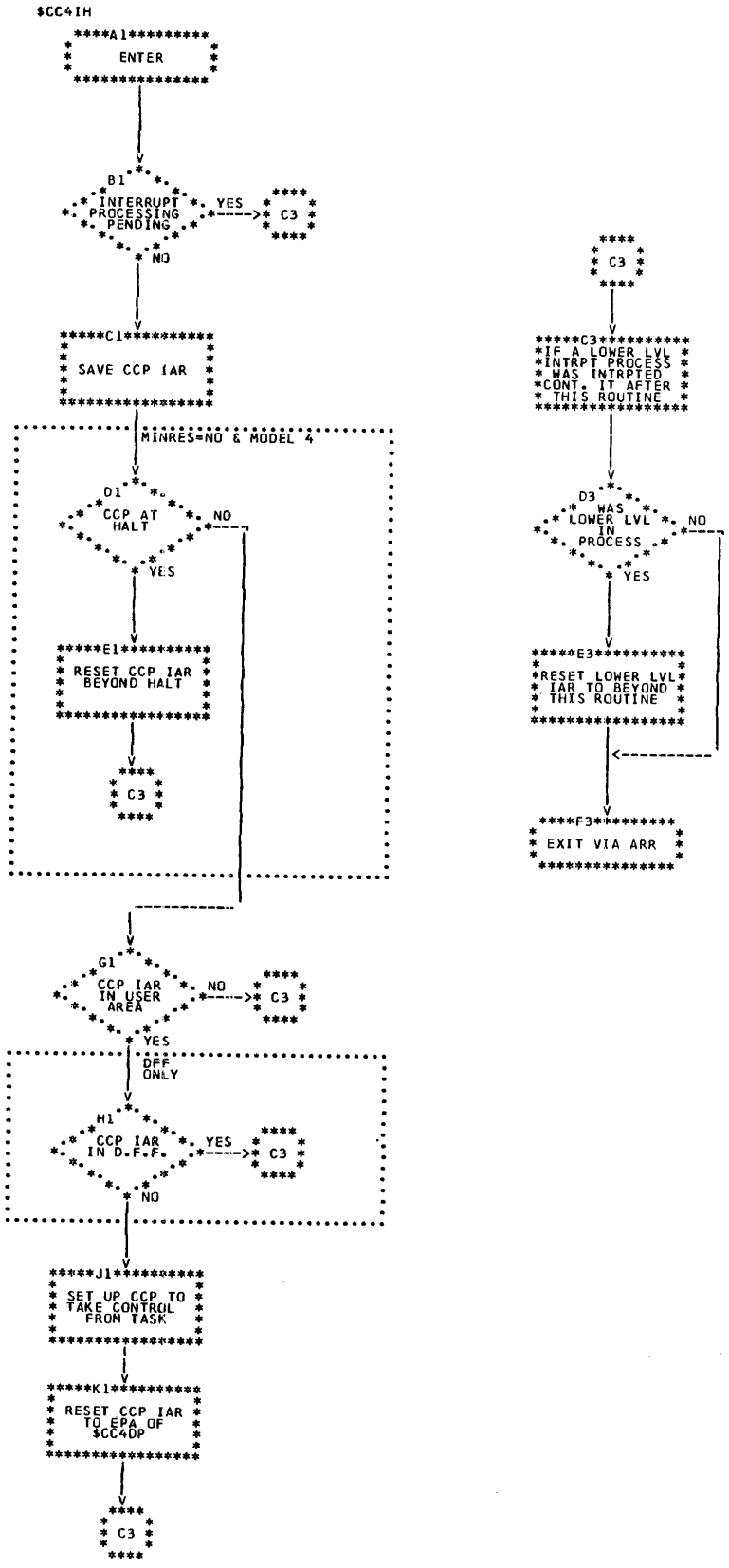


Chart OF. Common Interrupt Handler ($CC4IH)

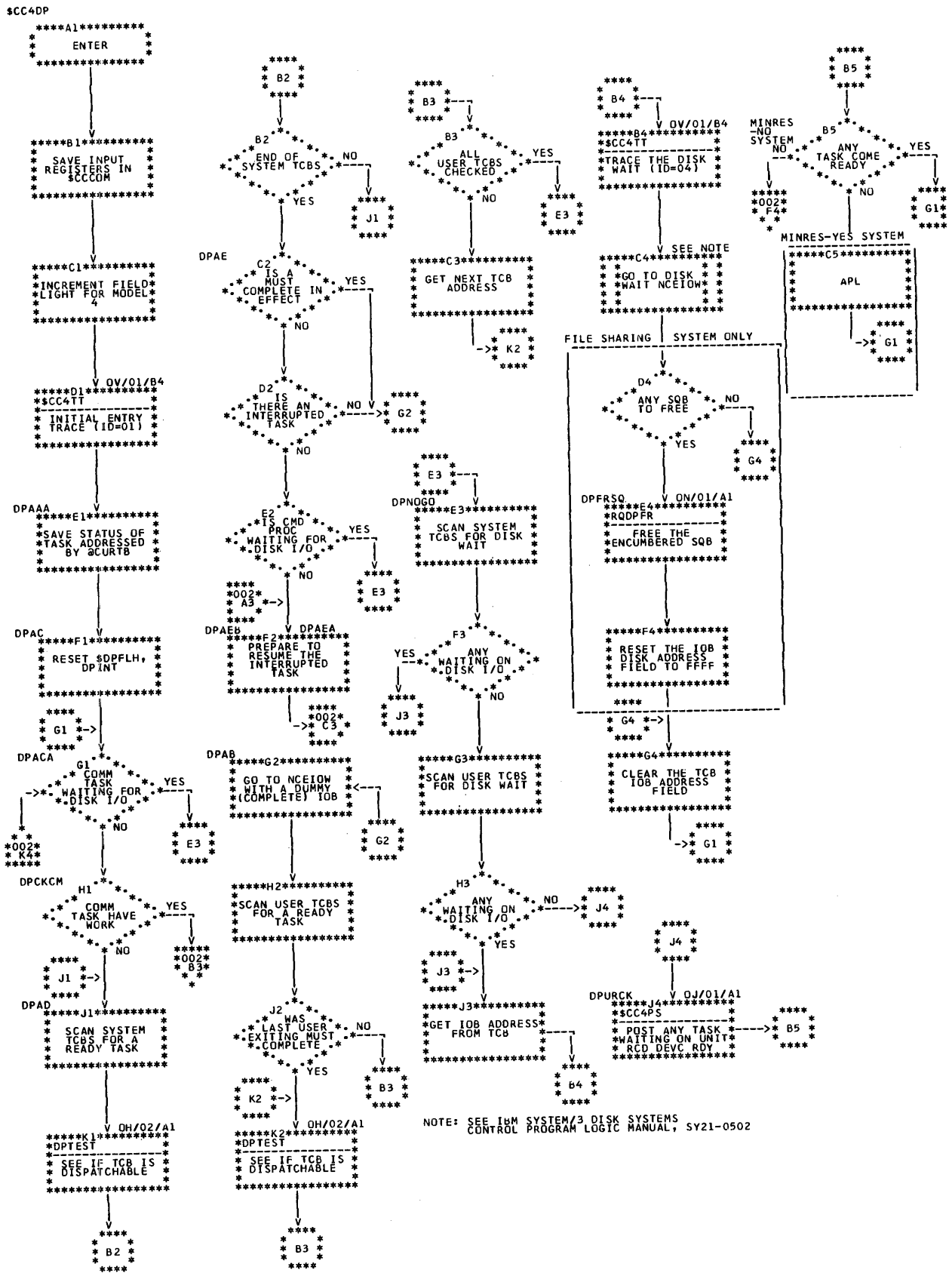


Chart OH (Part 1 of 2). CCP Task Dispatcher ($CC4DP)

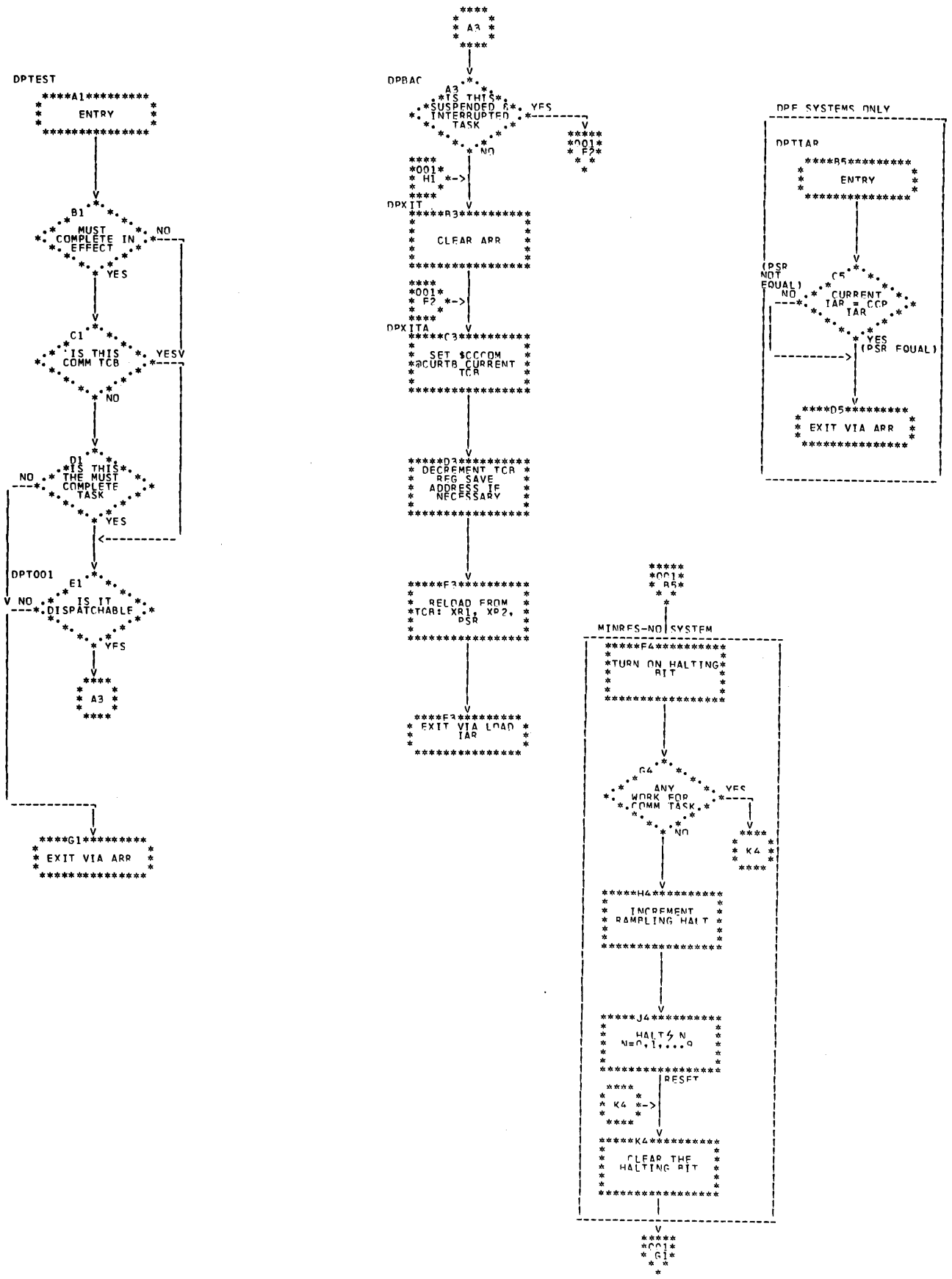


Chart OH (Part 2 of 2). CCP Task Dispatcher ($CC4DP)

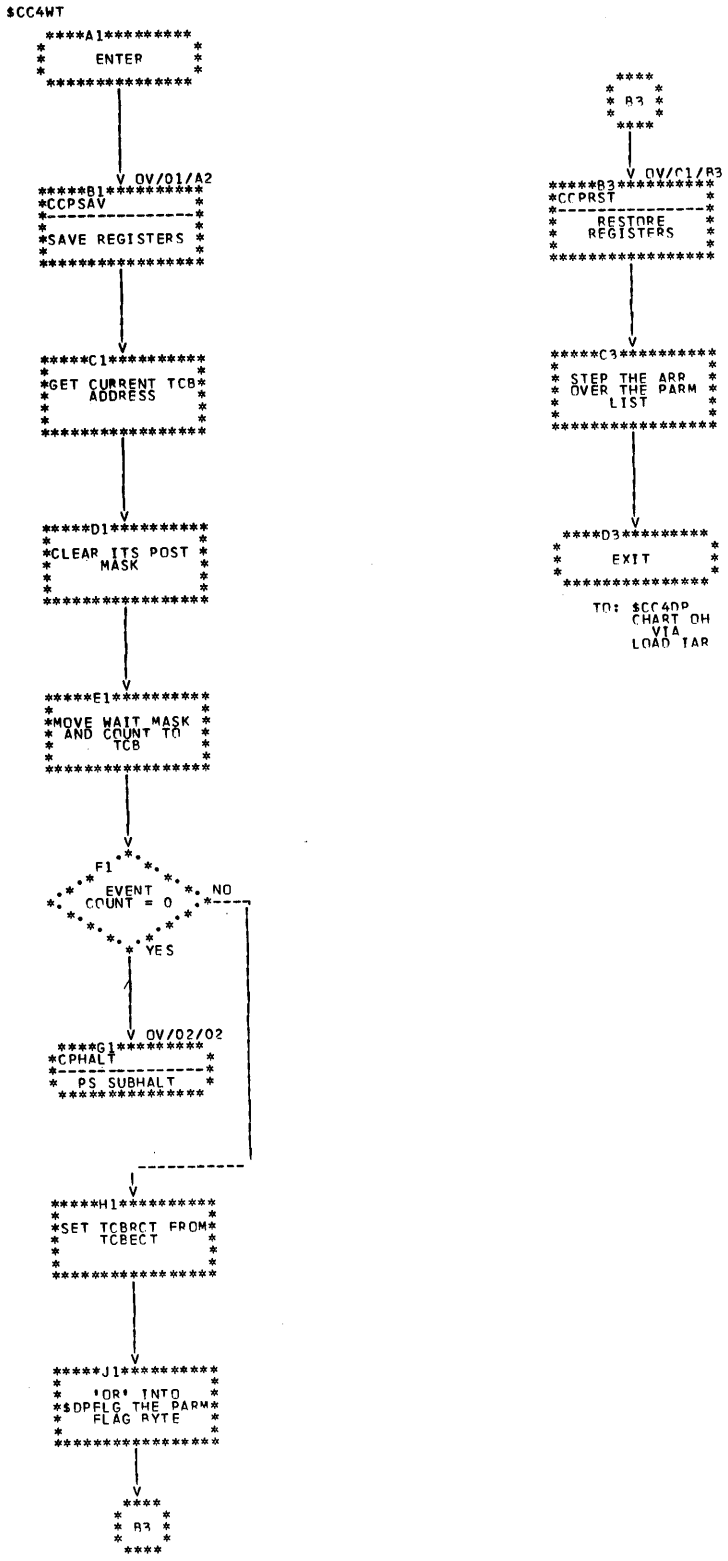


Chart OI. CCP Wait Routine ($CC4WT)

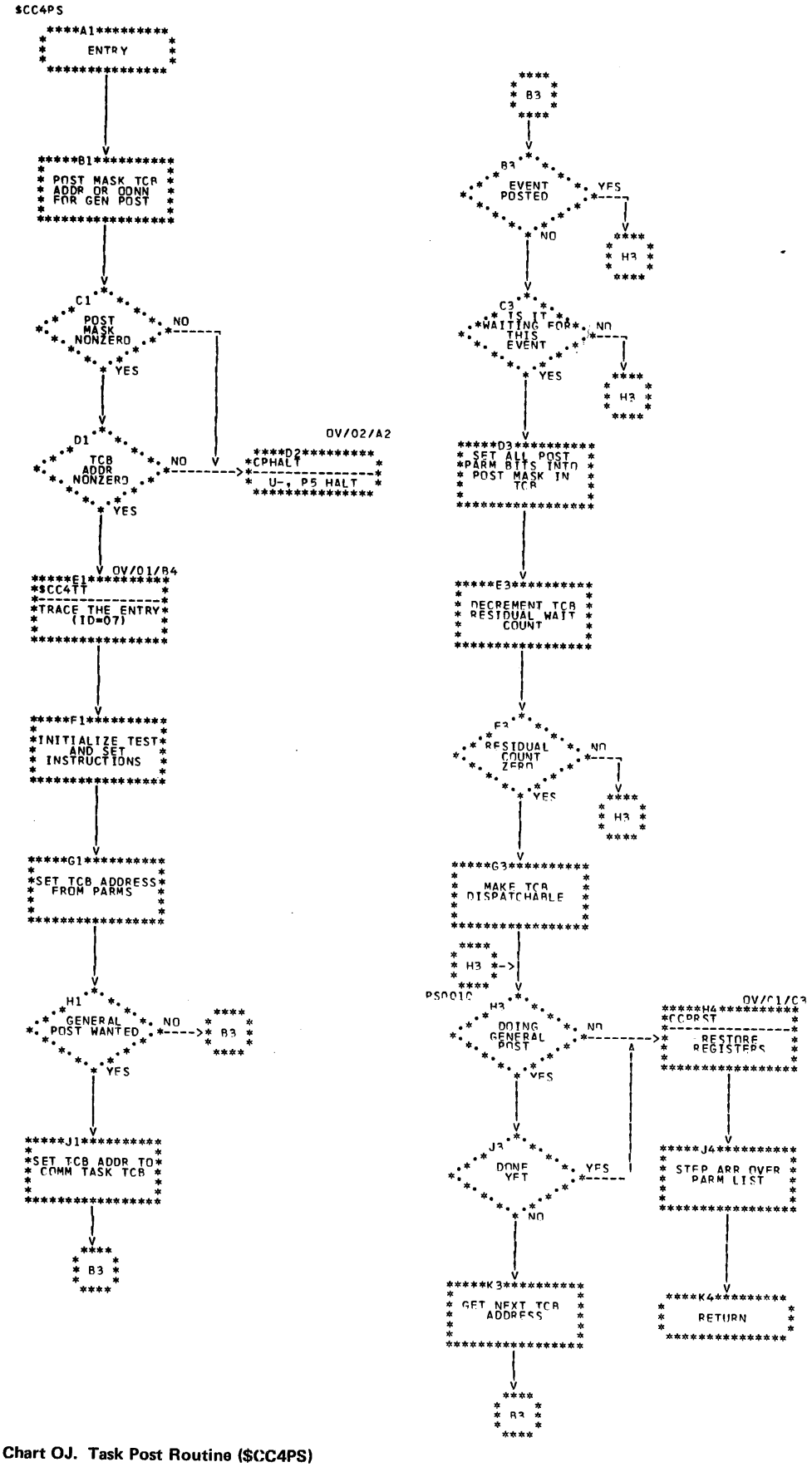


Chart OJ. Task Post Routine ($CC4PS)

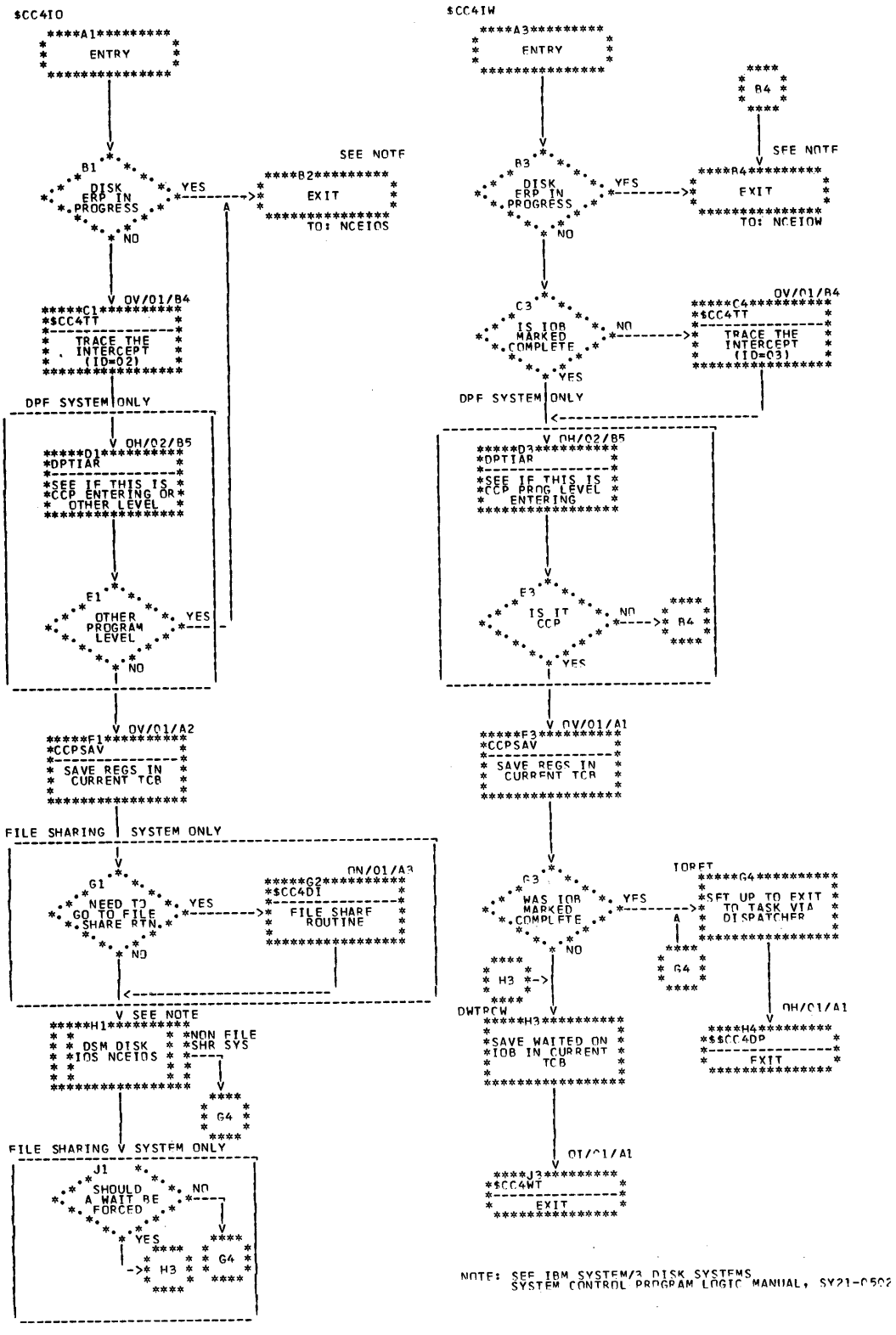


Chart OM. CCP Disk Intercept ($CC4IO, $CC4IW)

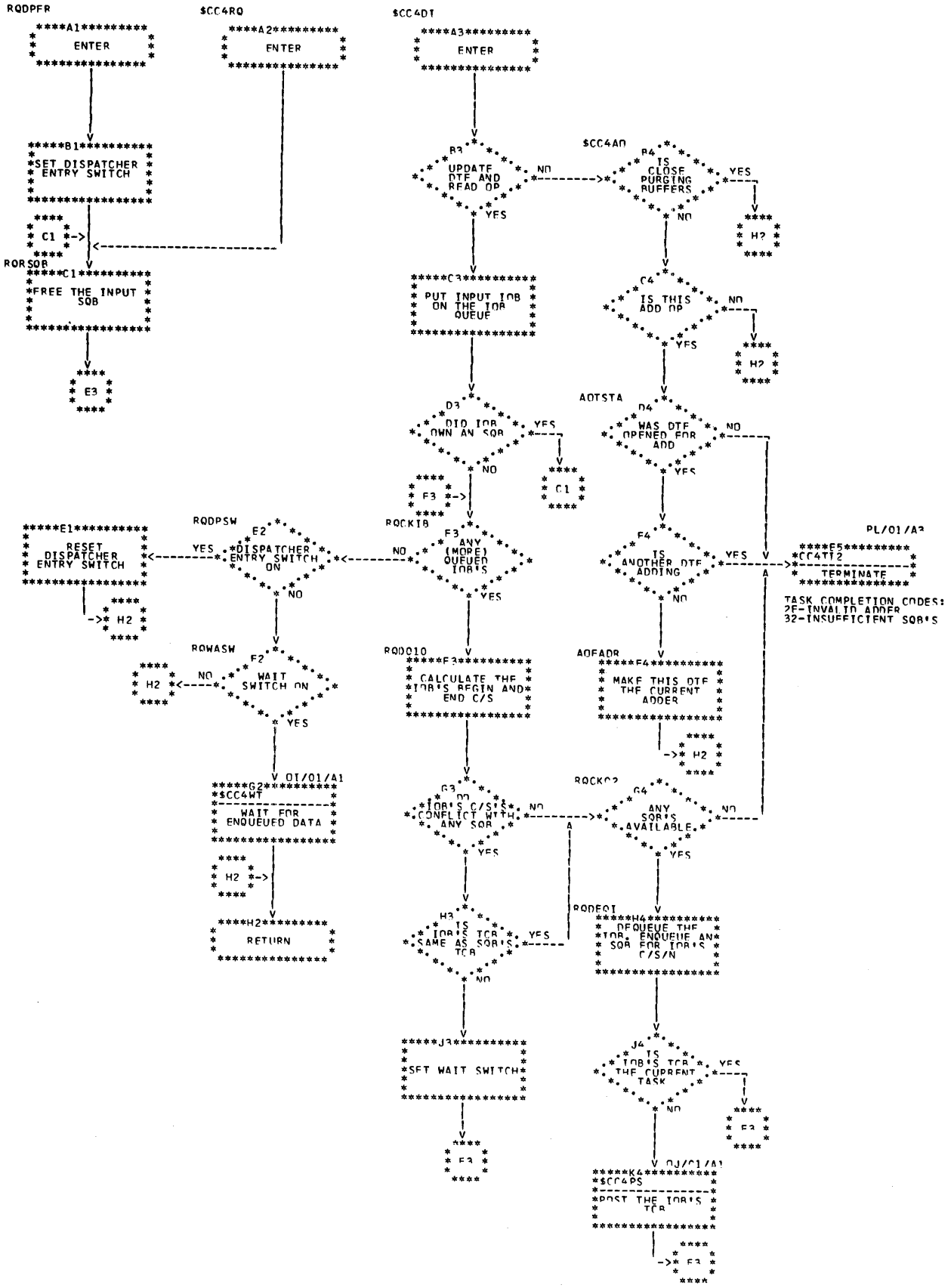


Chart ON. CCP File Sharing Enqueue — Dequeue Routines ($CC4DI)

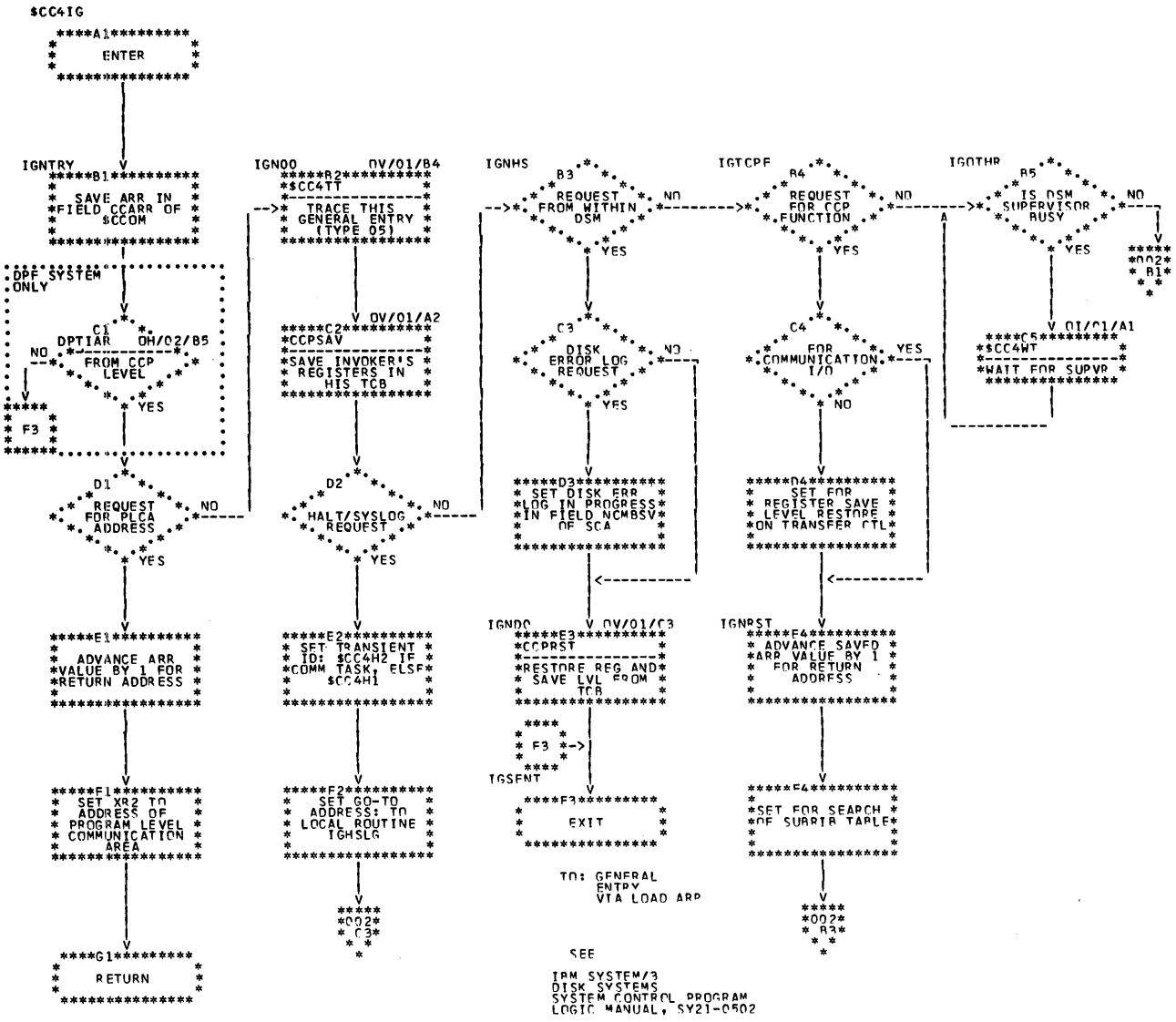


Chart OR (Part 1 of 2). General Entry Intercept ($CC4IG) (Models 8, 10, and 12 Only)

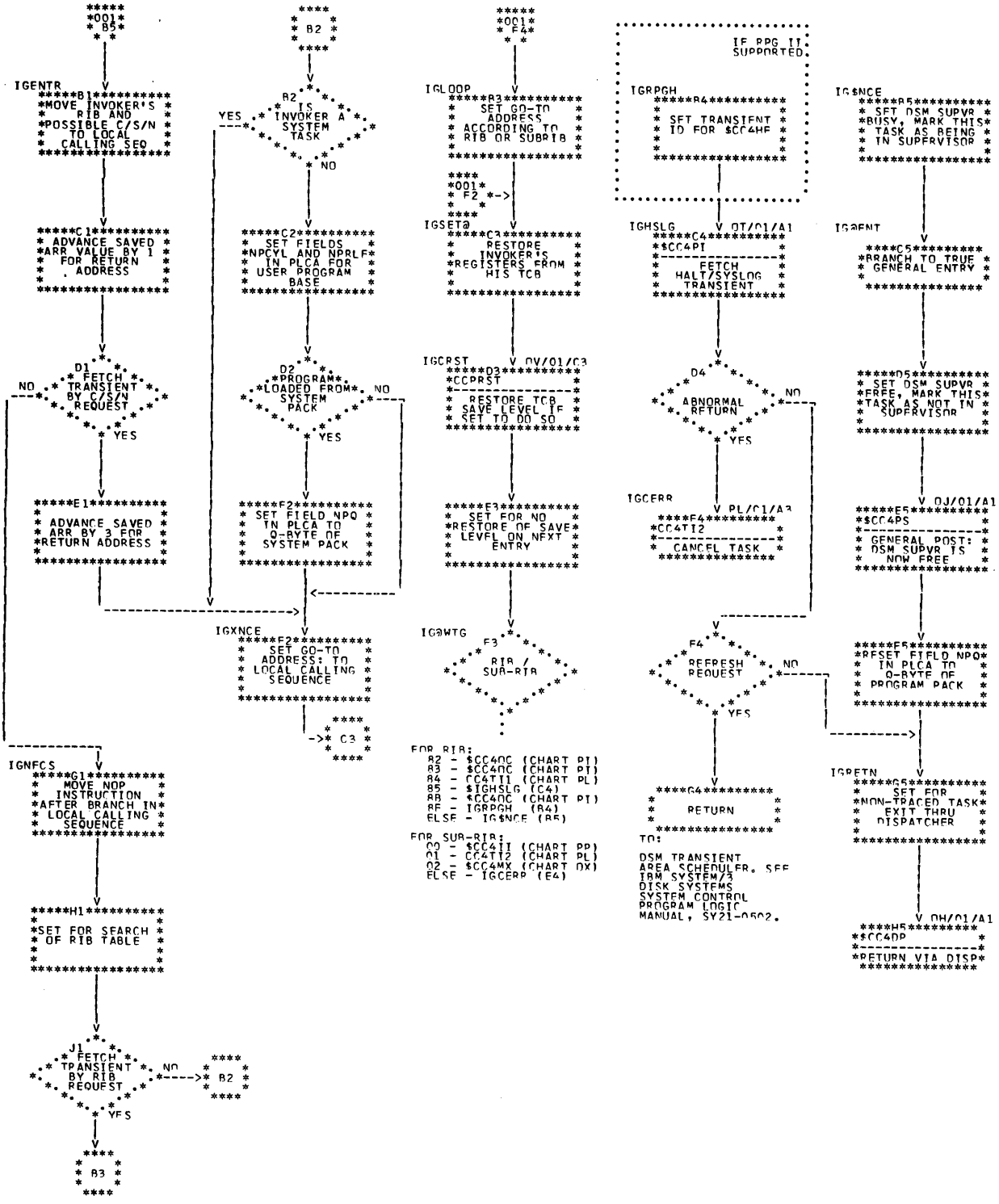


Chart OR (Part 2 of 2). General Entry Intercept ($CC4IG) (Models 8, 10, and 12 Only)

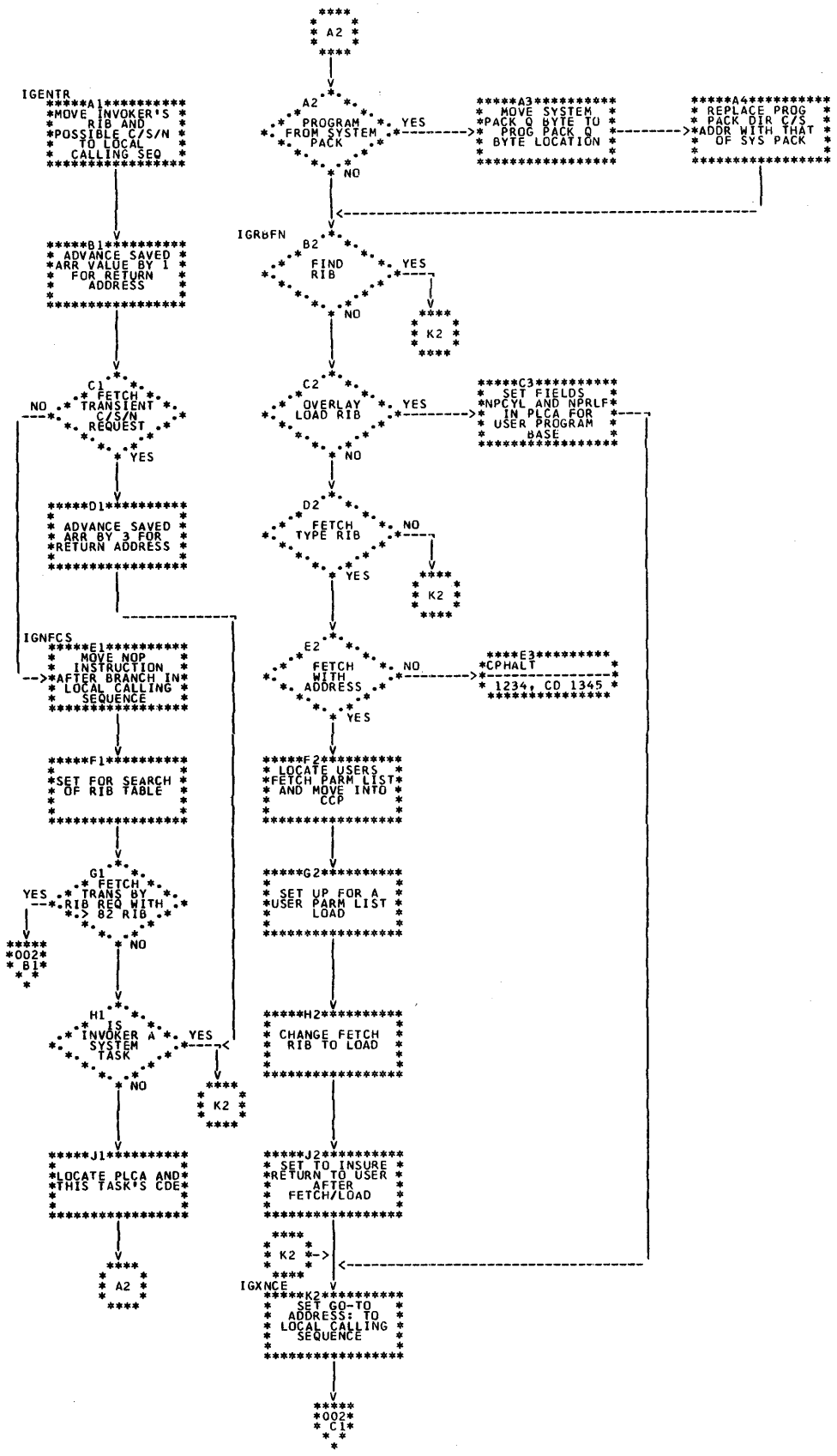


Chart OS (Part 1 of 2). General Entry Intercept ($CC4IG) (Model 4 Only)

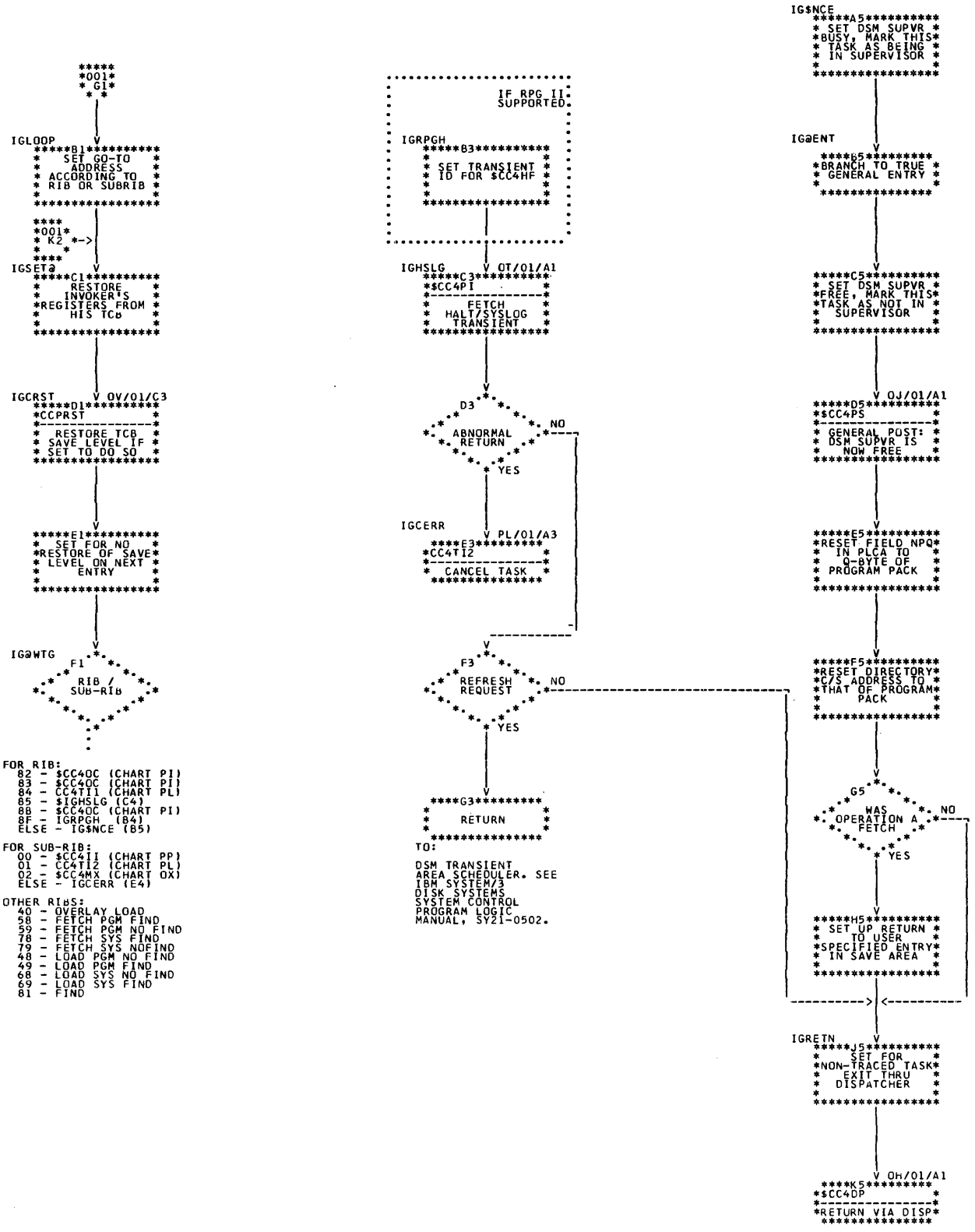


Chart OS (Part 2 of 2). General Entry Intercept ($CC4IG) (Model 4 Only)

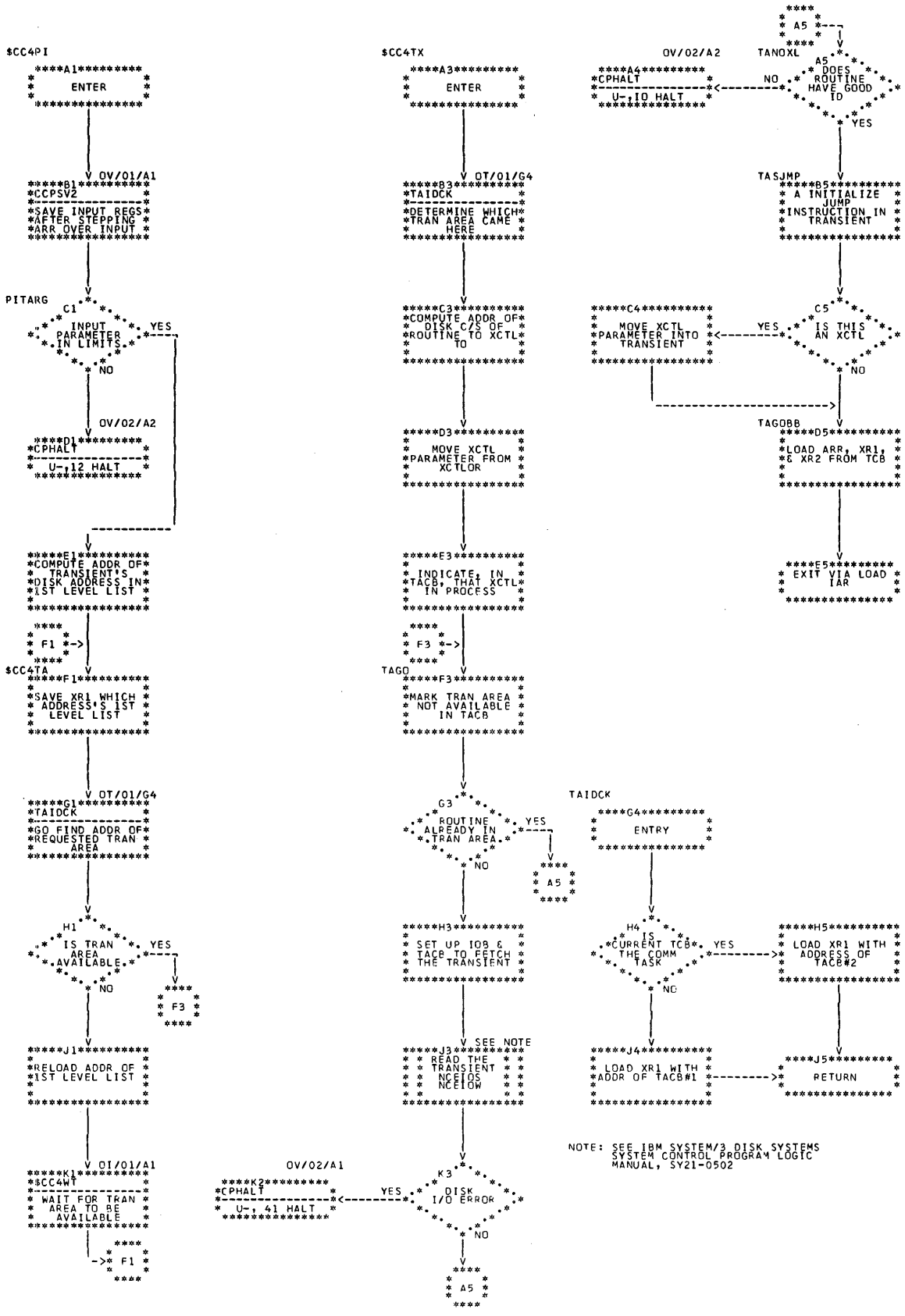


Chart OT (Part 1 of 2). Transient Area Handler ($CC4PI, $CC4TX, $CC4TR)

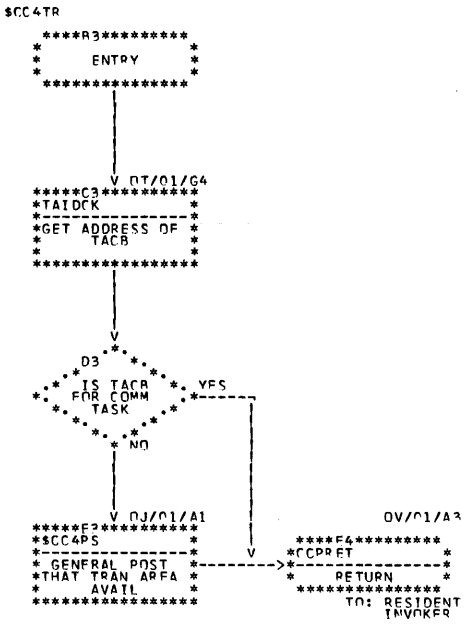


Chart OT (Part 2 of 2). Transient Area Handler ($CC4PI, $CC4TX, $CC4TR)

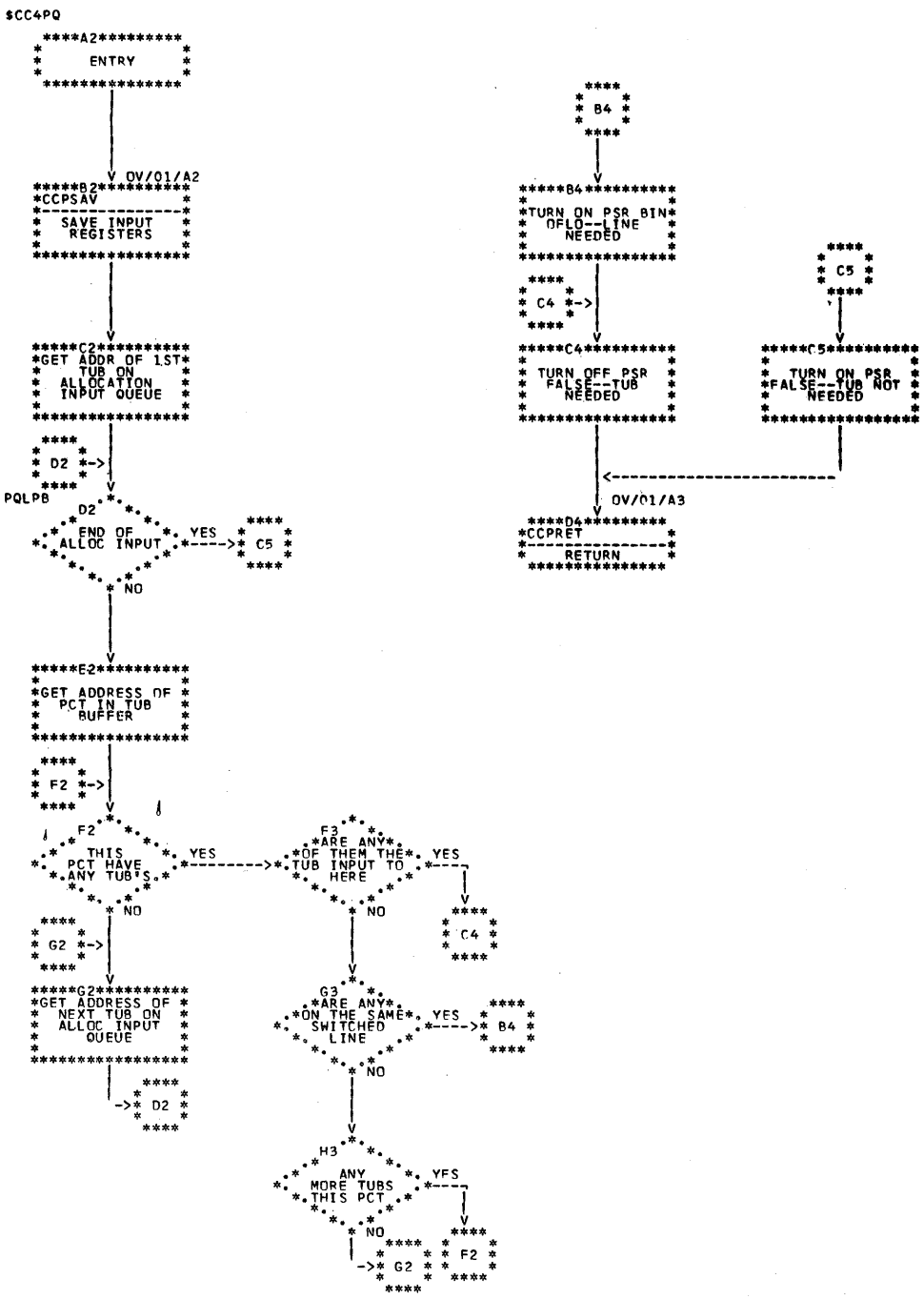


Chart OU. Program Request Queue Test ($CC4PQ)

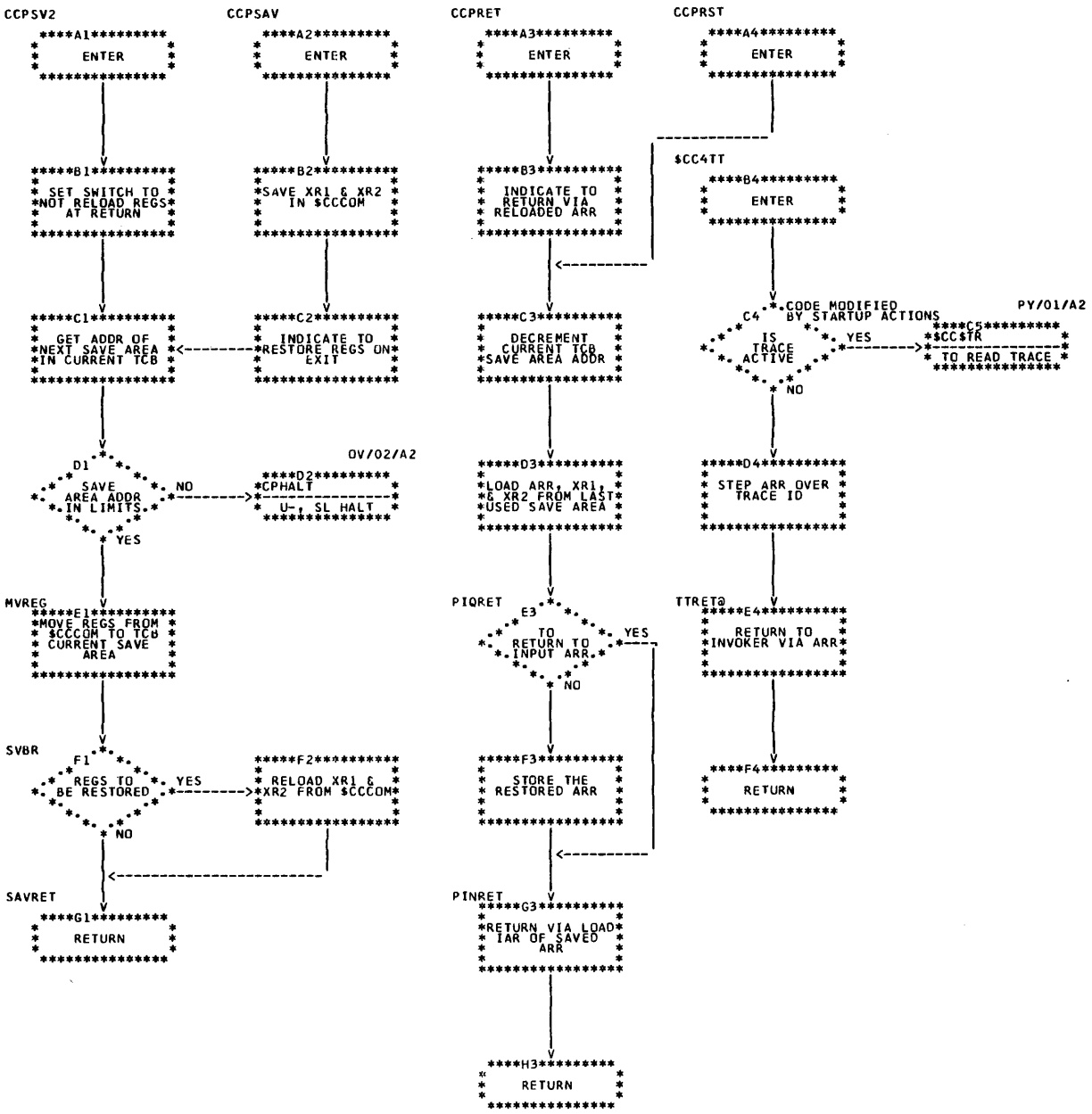


Chart OV (Part 1 of 2). Register Save and Restore, Trace, and Halt (CCPSAV, $CC4TT, CPHALT)

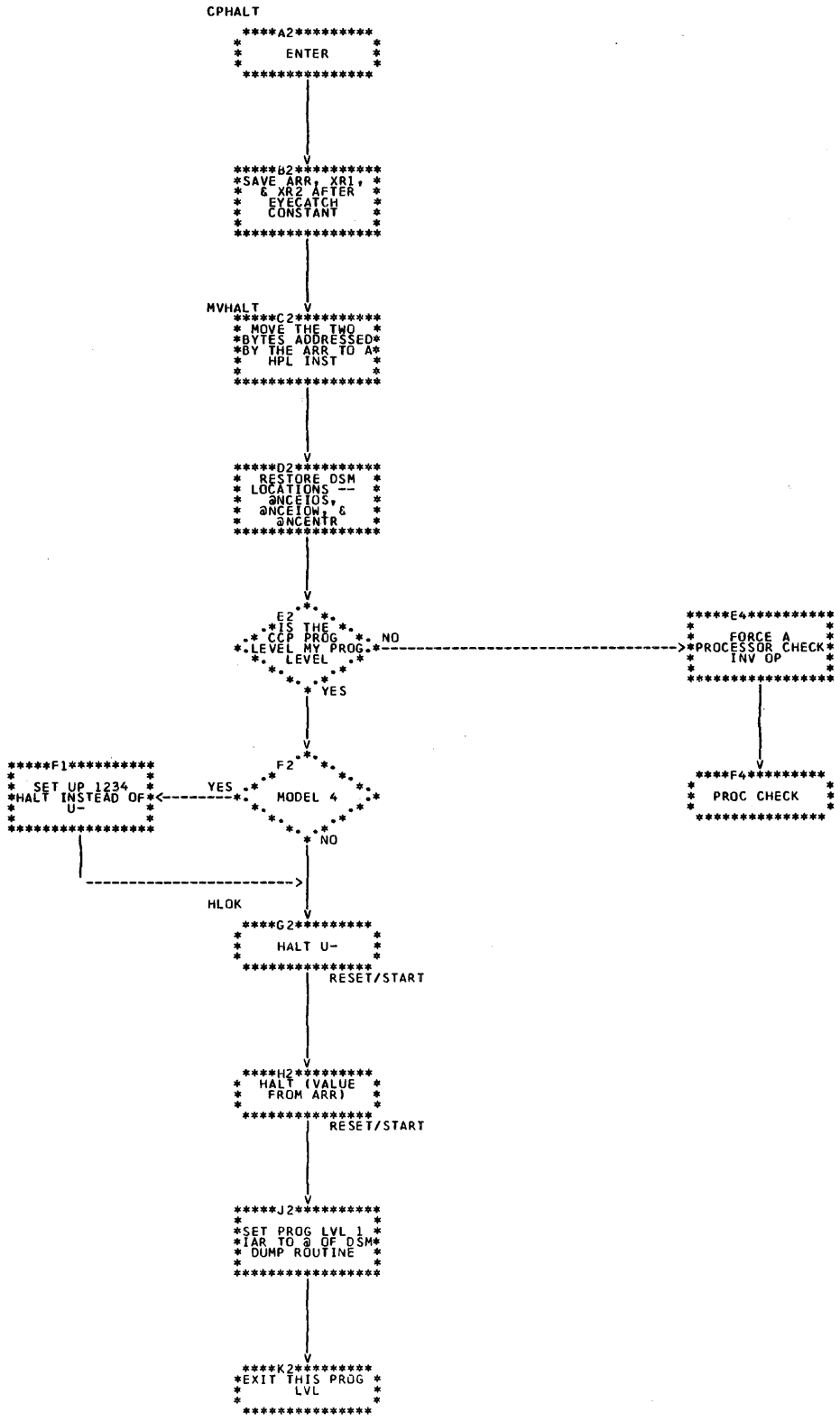


Chart OV (Part 2 of 2). Register Save and Restore, Trace, and Halt (CCPSAV, $CC4TT, CPHALT)

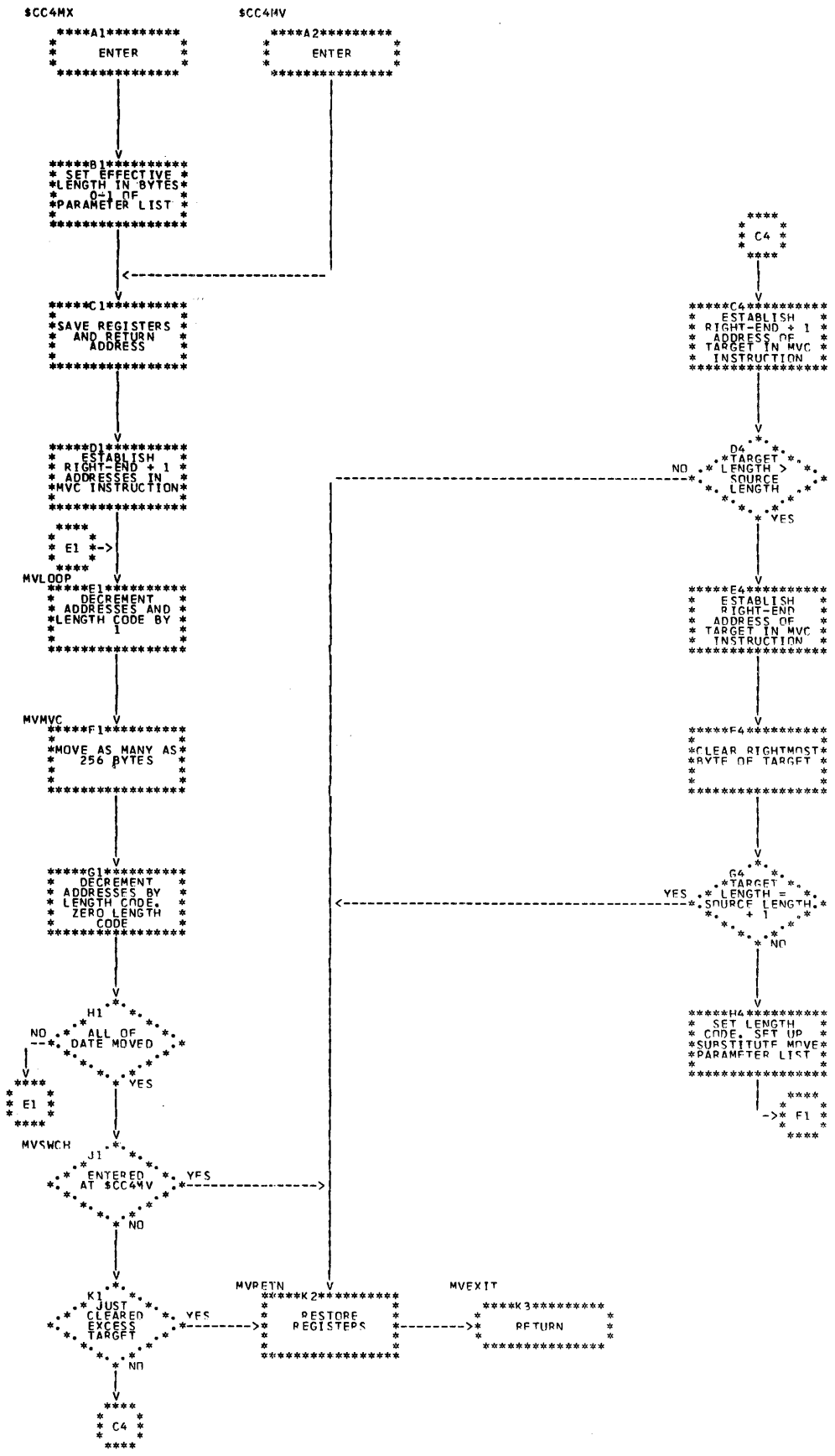


Chart OX. Generalized Move-Service Routine ($CC4MX)

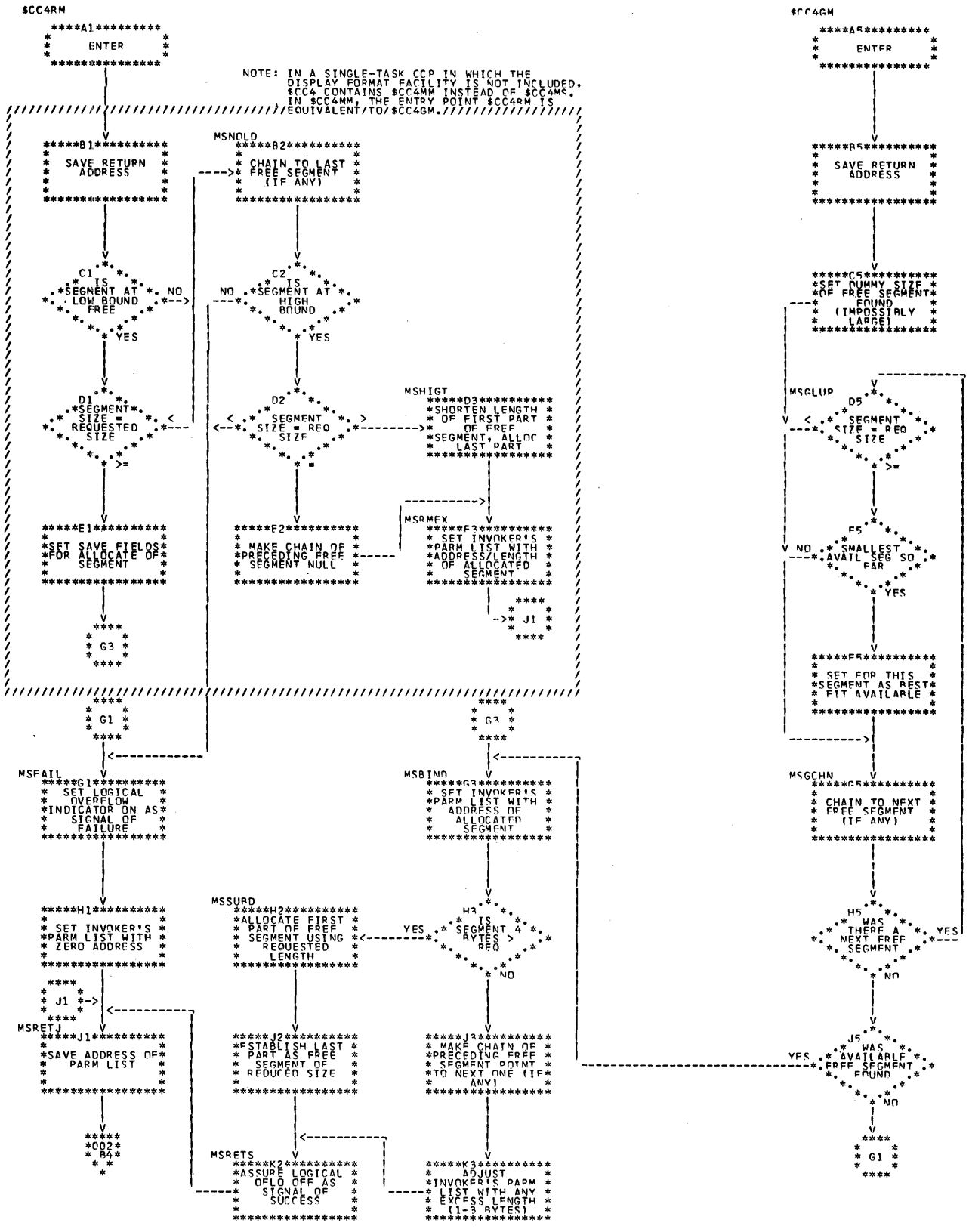


Chart OZ (Part 1 of 2). Getmain/Freemain Service Routine ($CC4MS, $CC4MM)

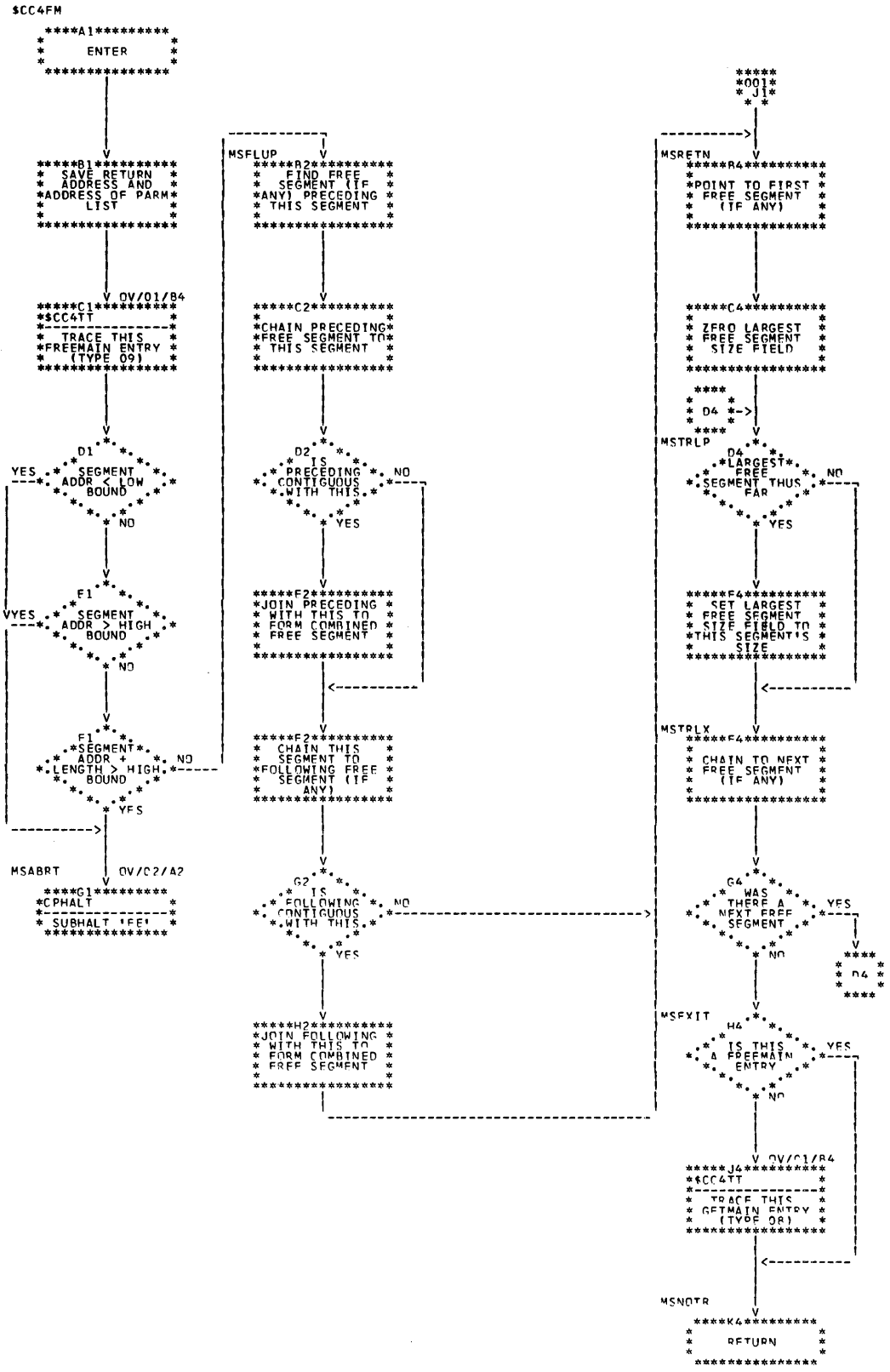


Chart OZ (Part 2 of 2). Getmain/Freemain Service Routine ($CC4MS, $CC4MM)

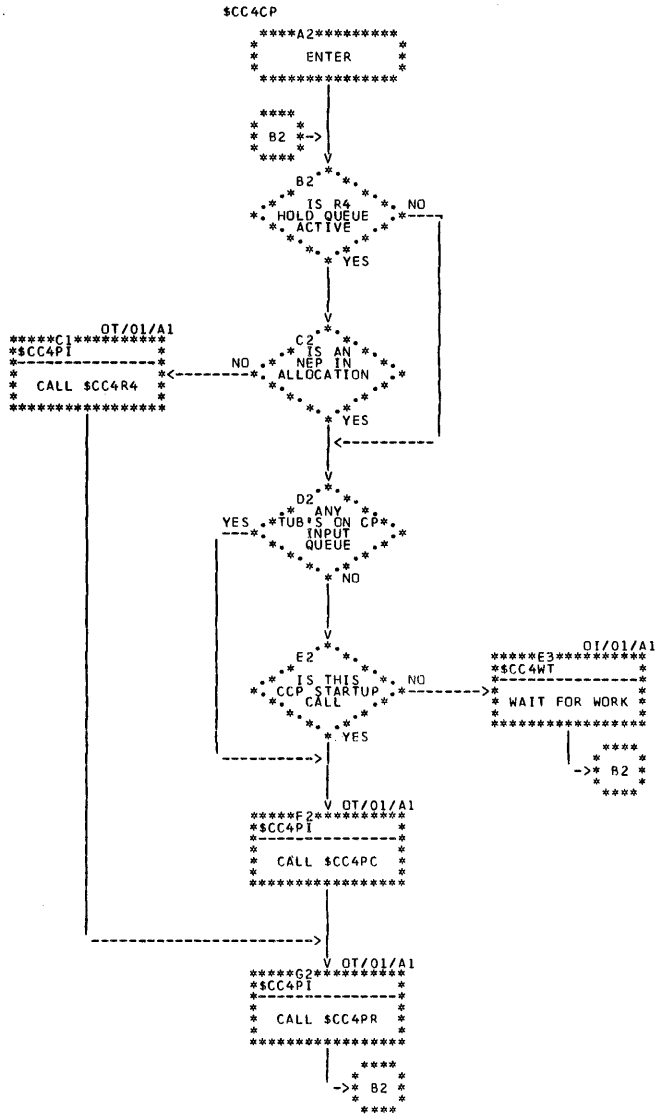


Chart PC. Command Processor Mainline ($CC4CP)

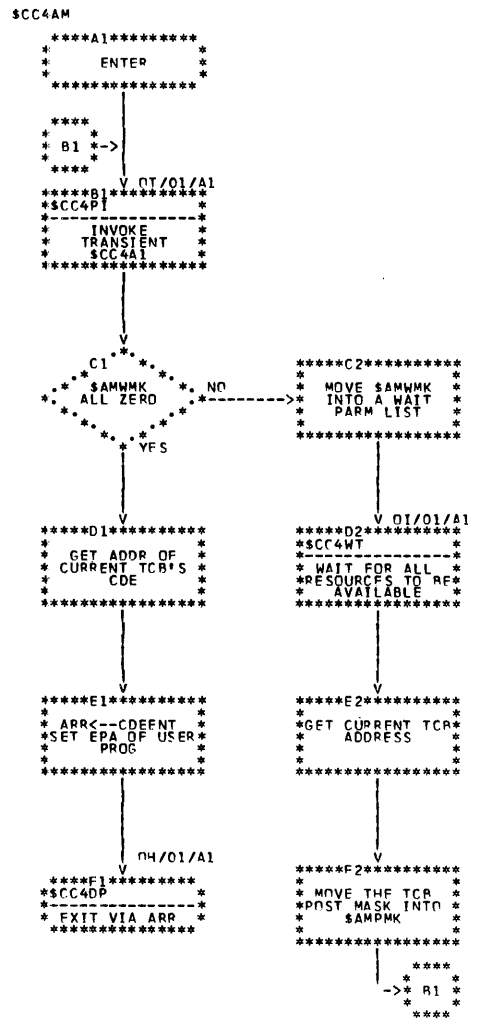


Chart PF. Allocation Task Resident Control Routine ($CC4AM)

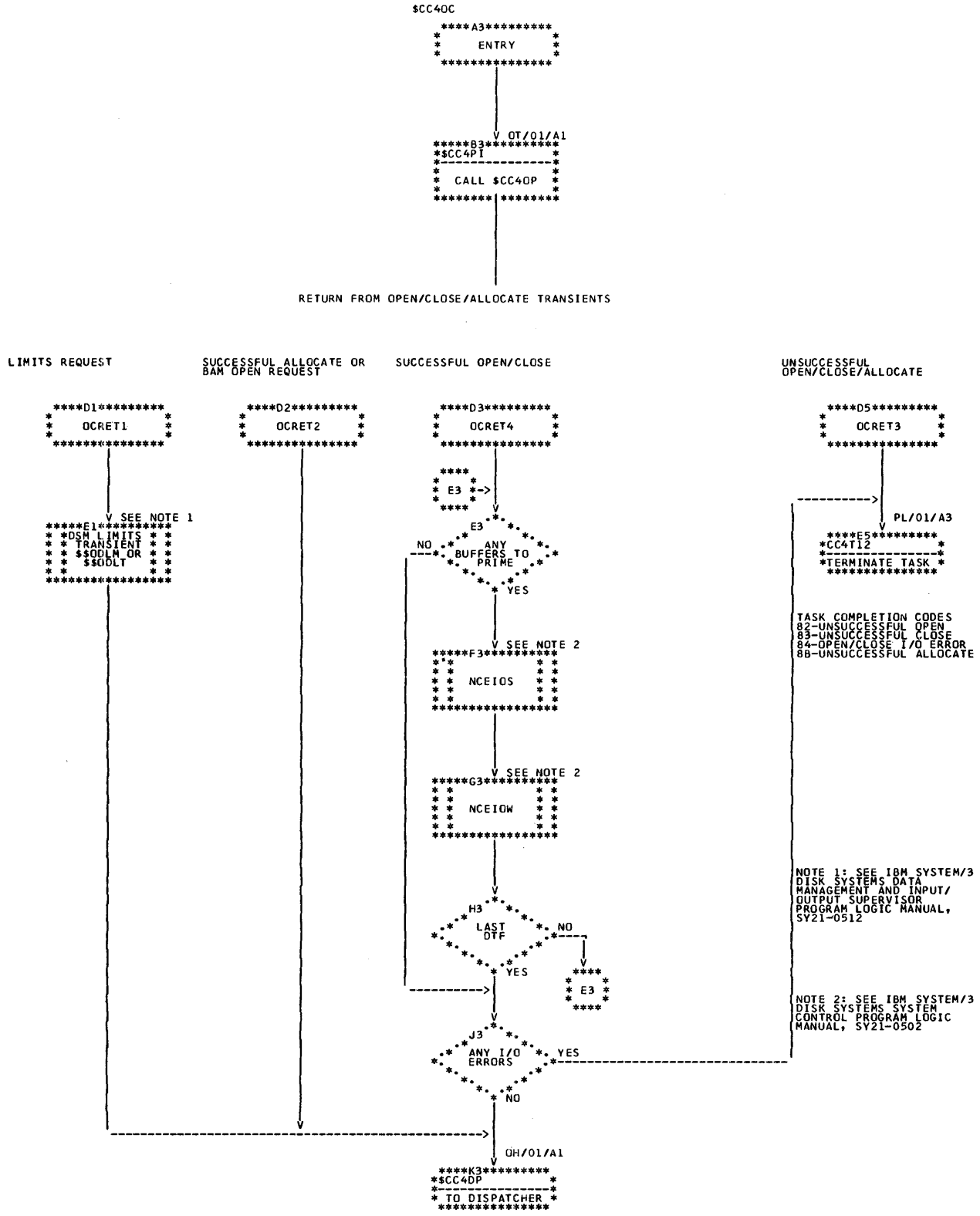


Chart PI. Open/Close/Allocate Mainline ($CC4OC)

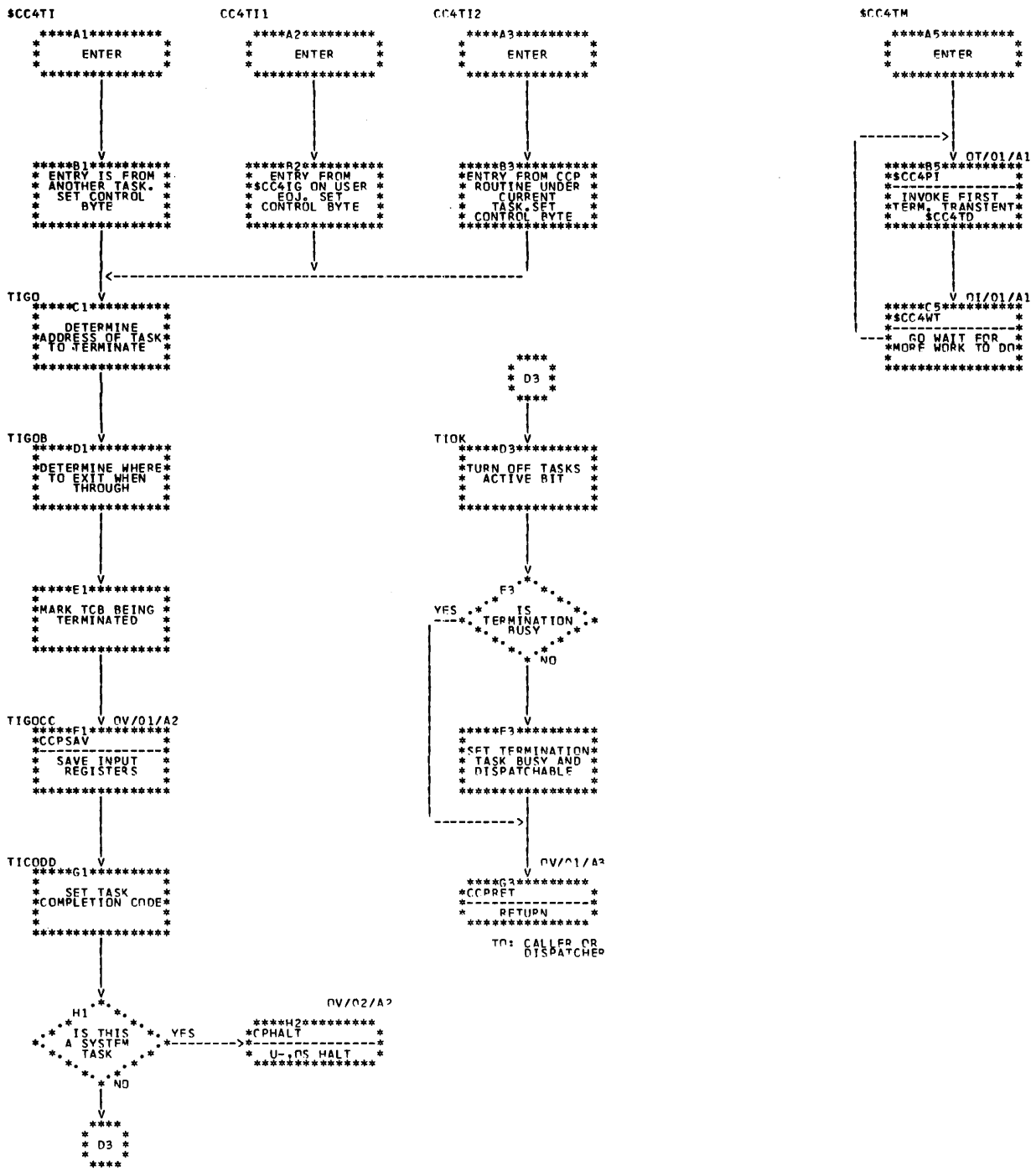


Chart PL. Termination Interface ($CC4TI/$CC4TM)

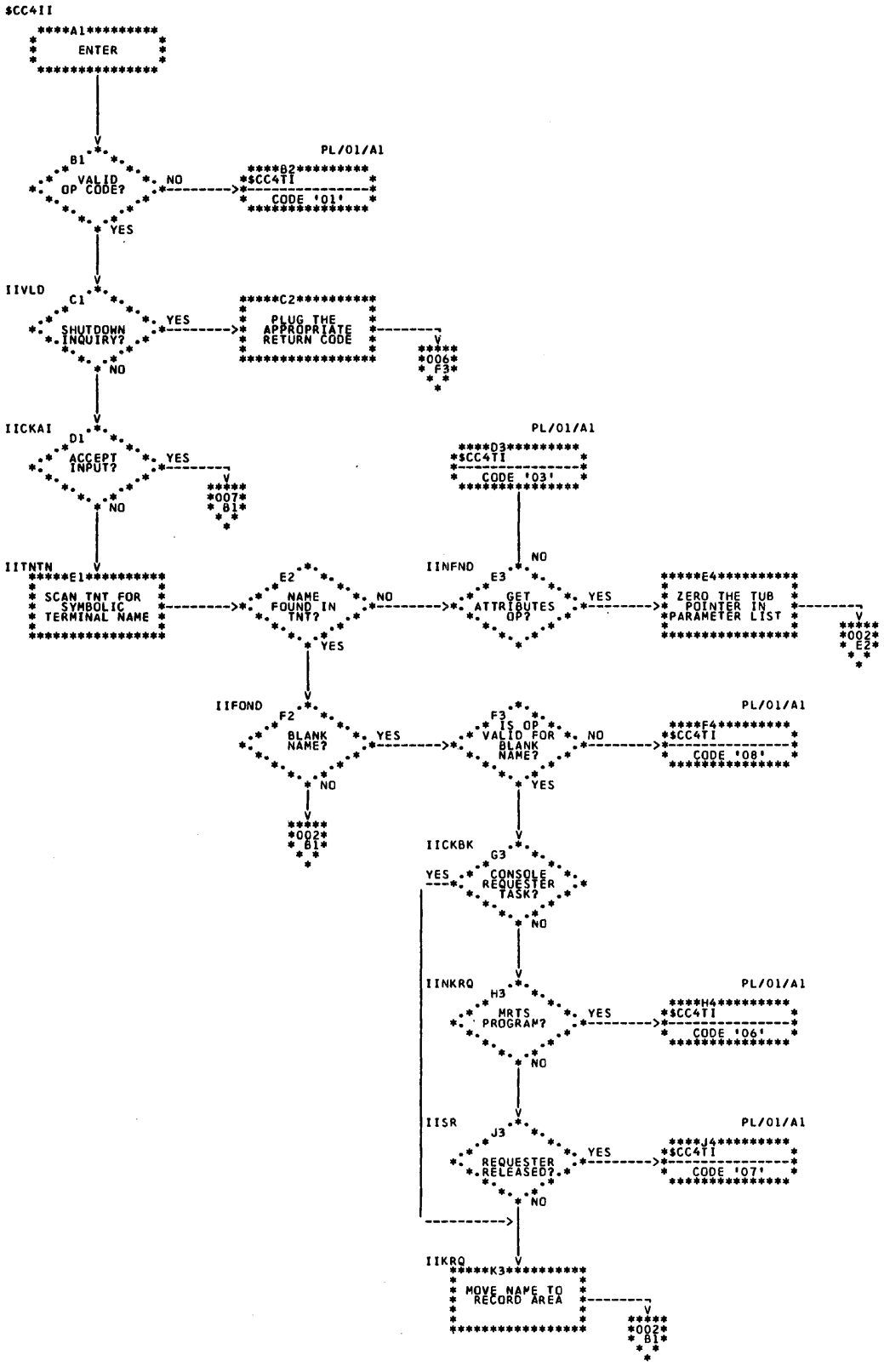


Chart PP (Part 1 of 10). I/O Interface Mainline ($CC4II/$CC4IS)

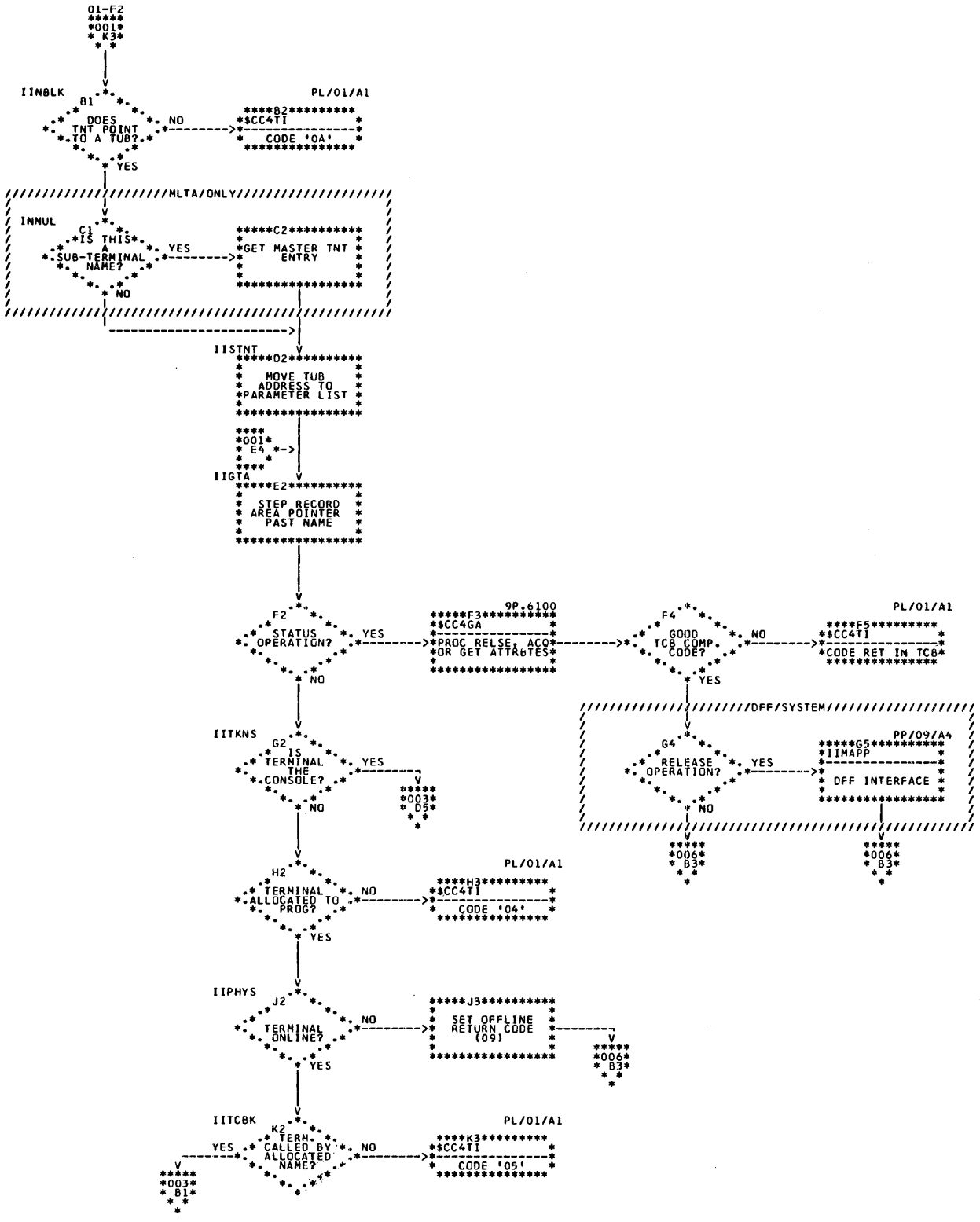


Chart PP (Part 2 of 10). I/O Interface Mainline ($CC4II/$CC4IS)

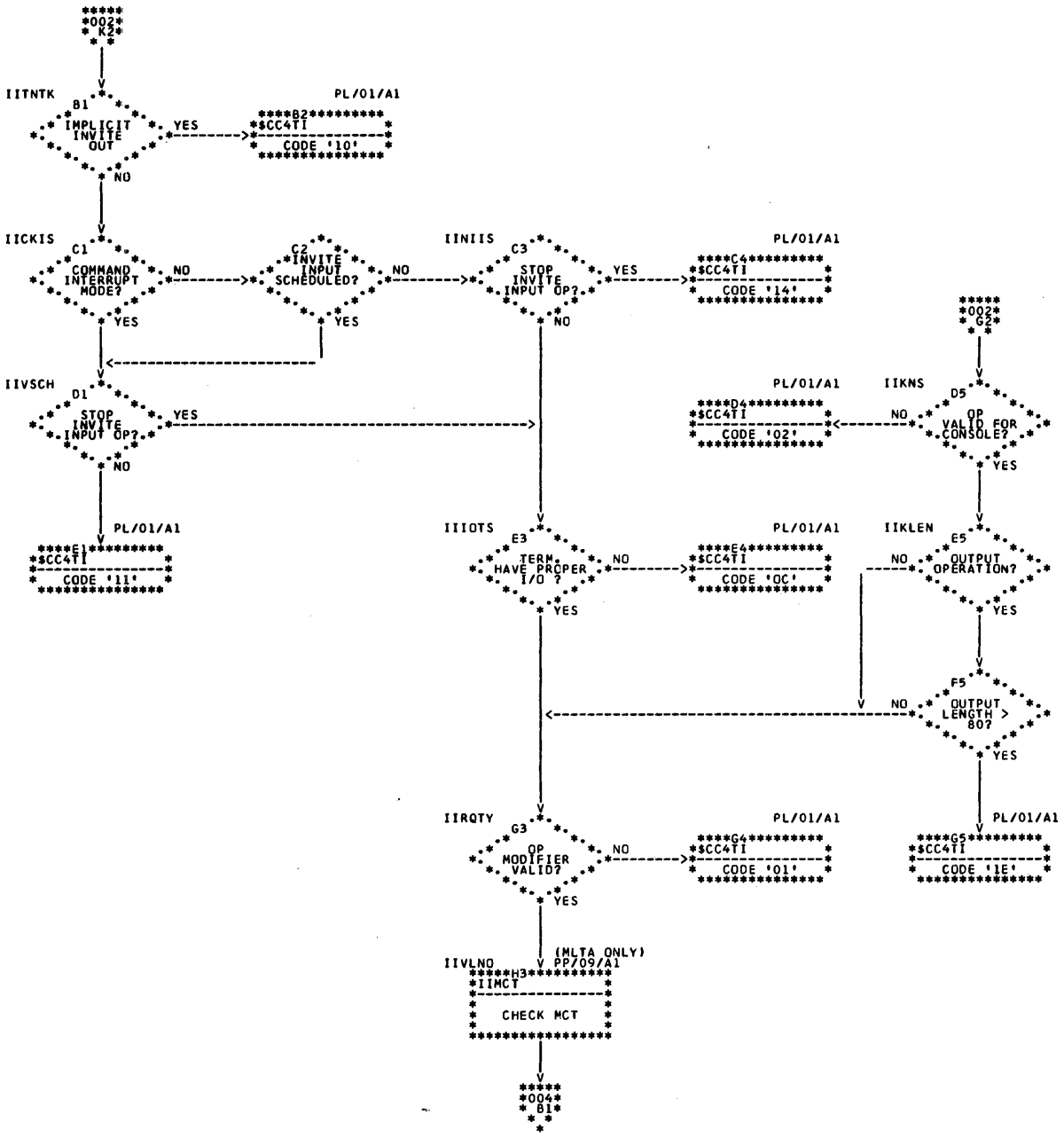


Chart PP (Part 3 of 10). I/O Interface Mainline ($CC4II/$CC4IS)

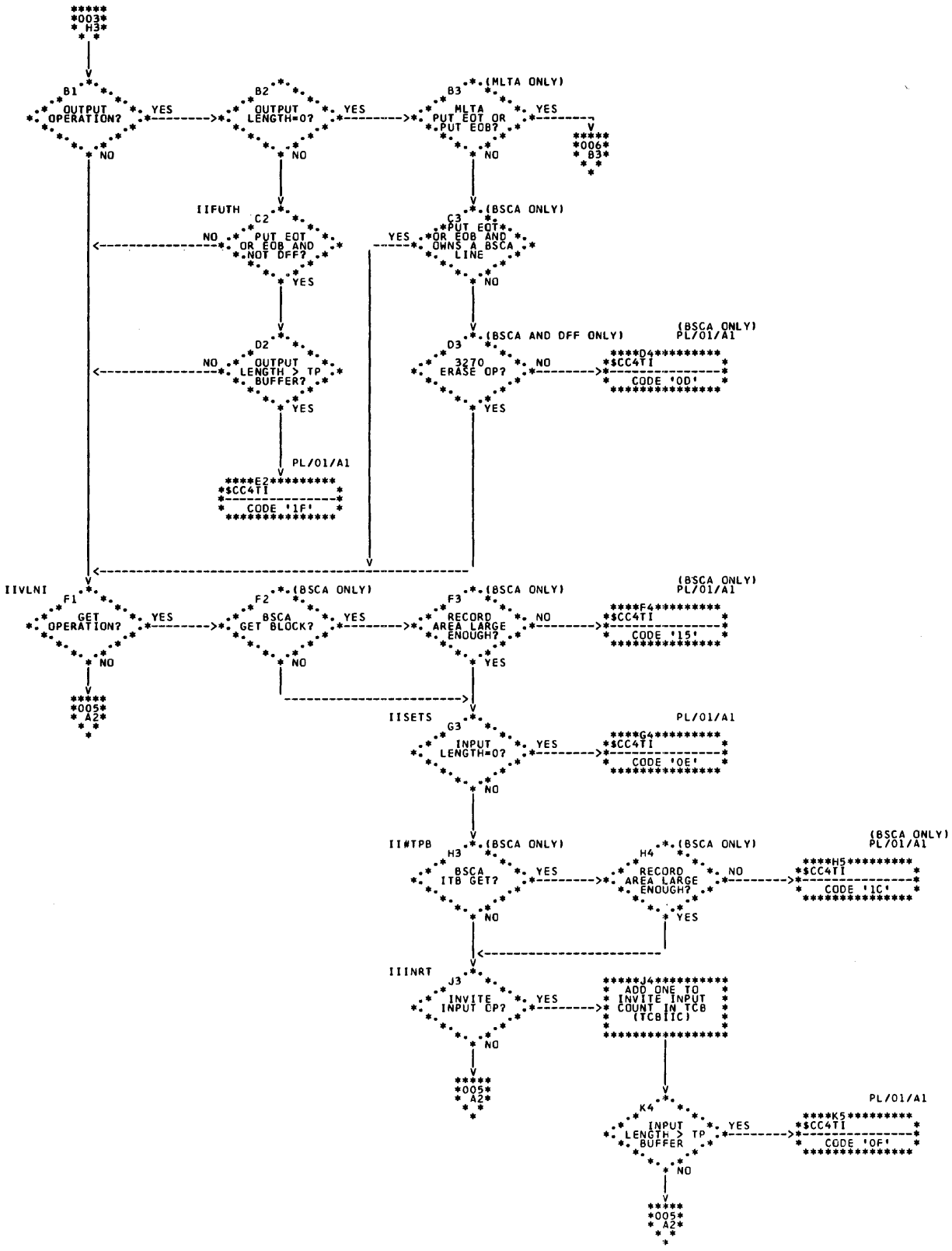


Chart PP (Part 4 of 10). I/O Interface Mainline ($CC4II/$CC4IS)

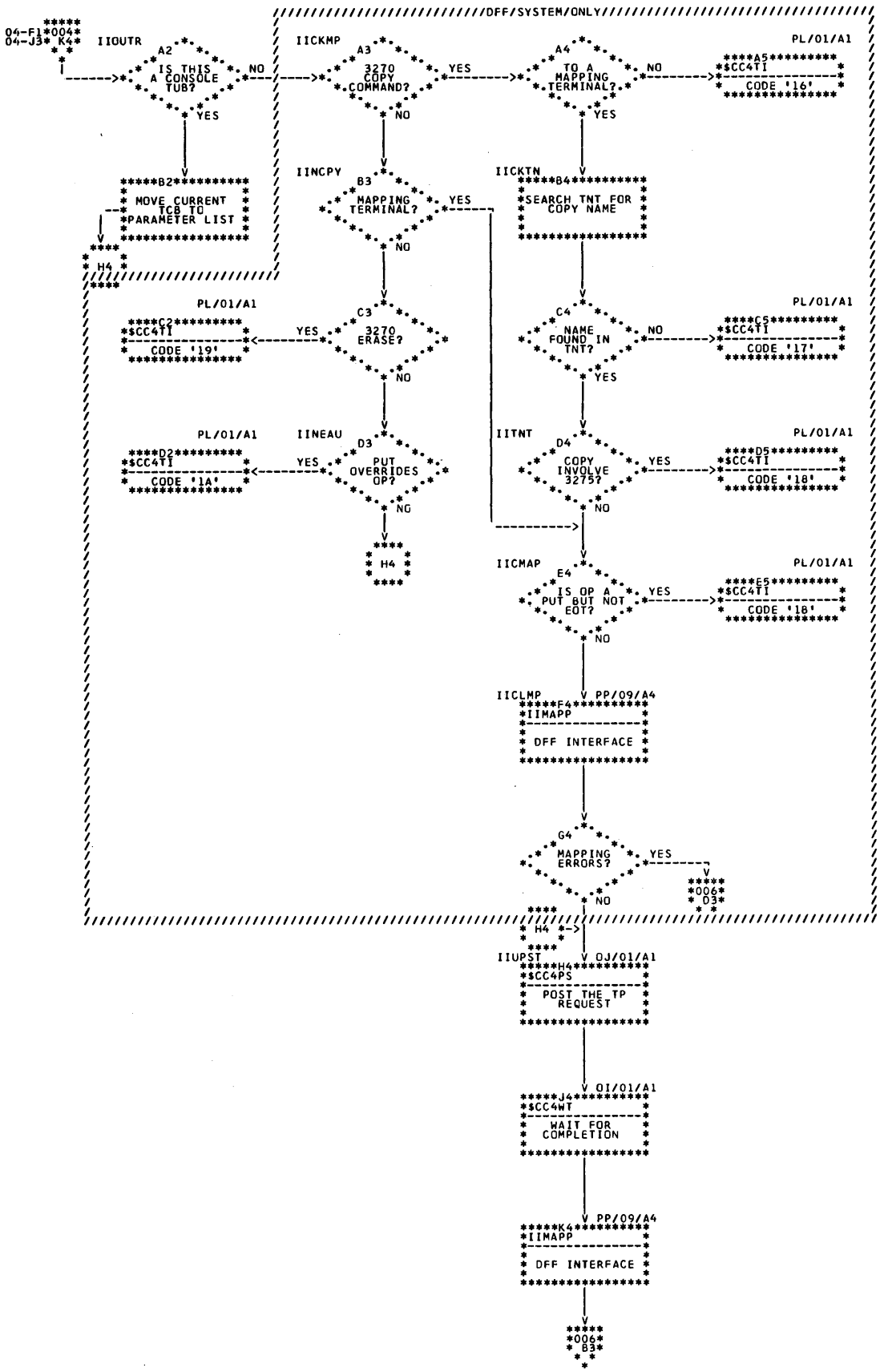


Chart PP (Part 5 of 10). I/O Interface Mainline ($CC4II/$CC4IS)

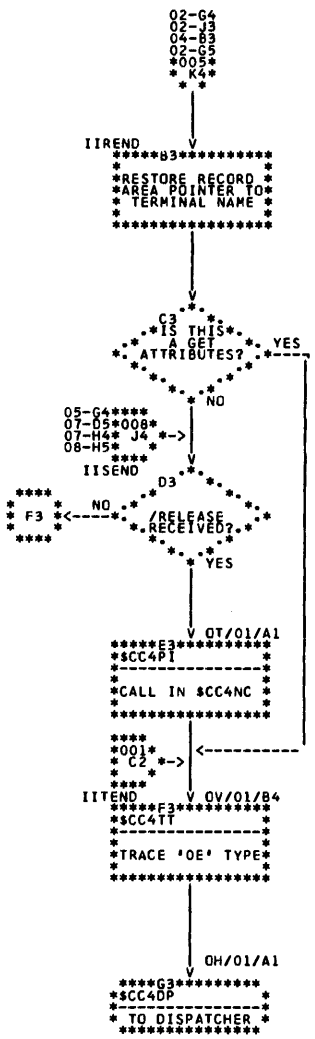


Chart PP (Part 6 of 10). I/O Interface Mainline ($CC4II/$CC4IS)

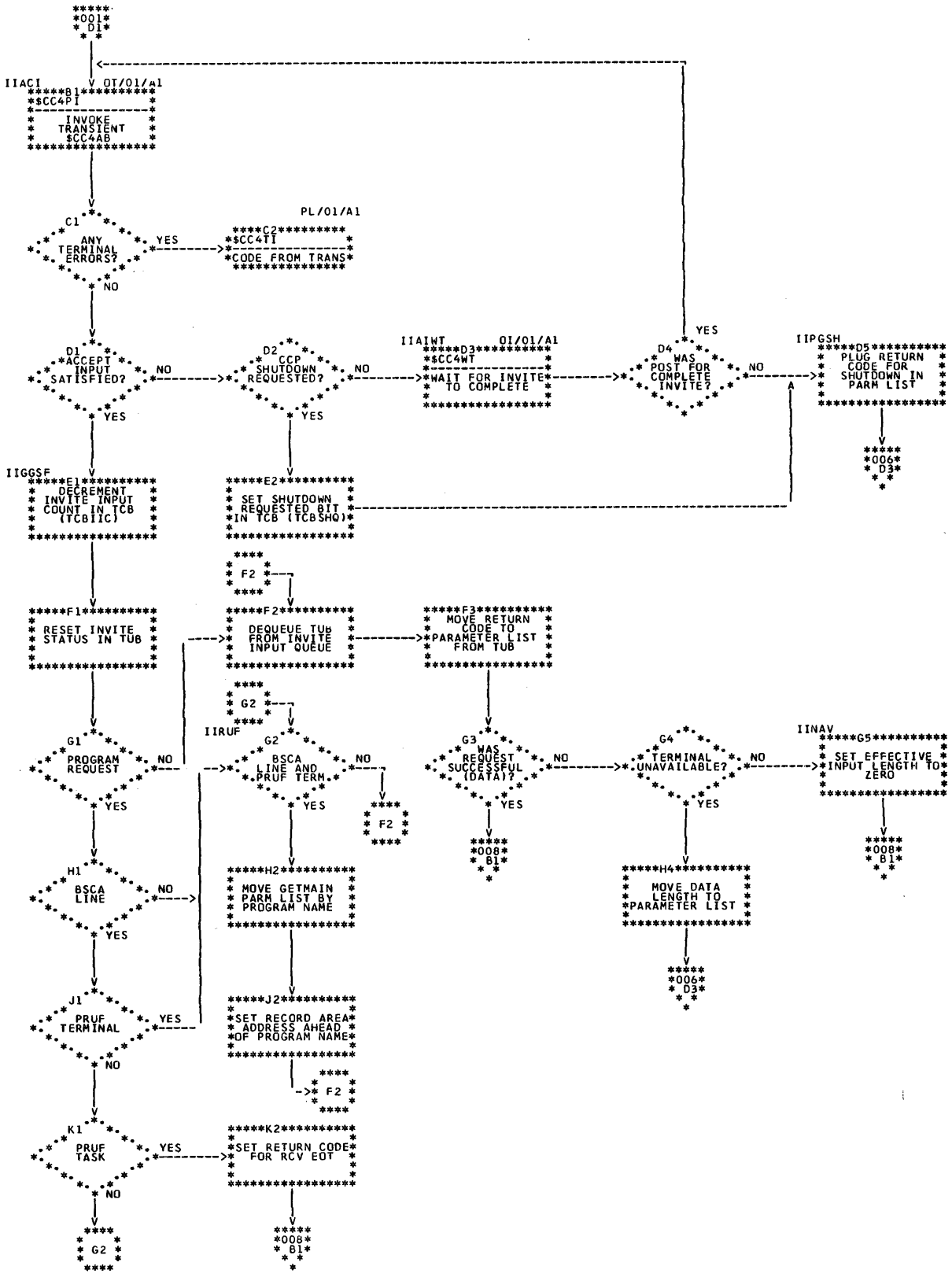


Chart PP (Part 7 of 10). I/O Interface Mainline ($CC4II/$CC4IS)

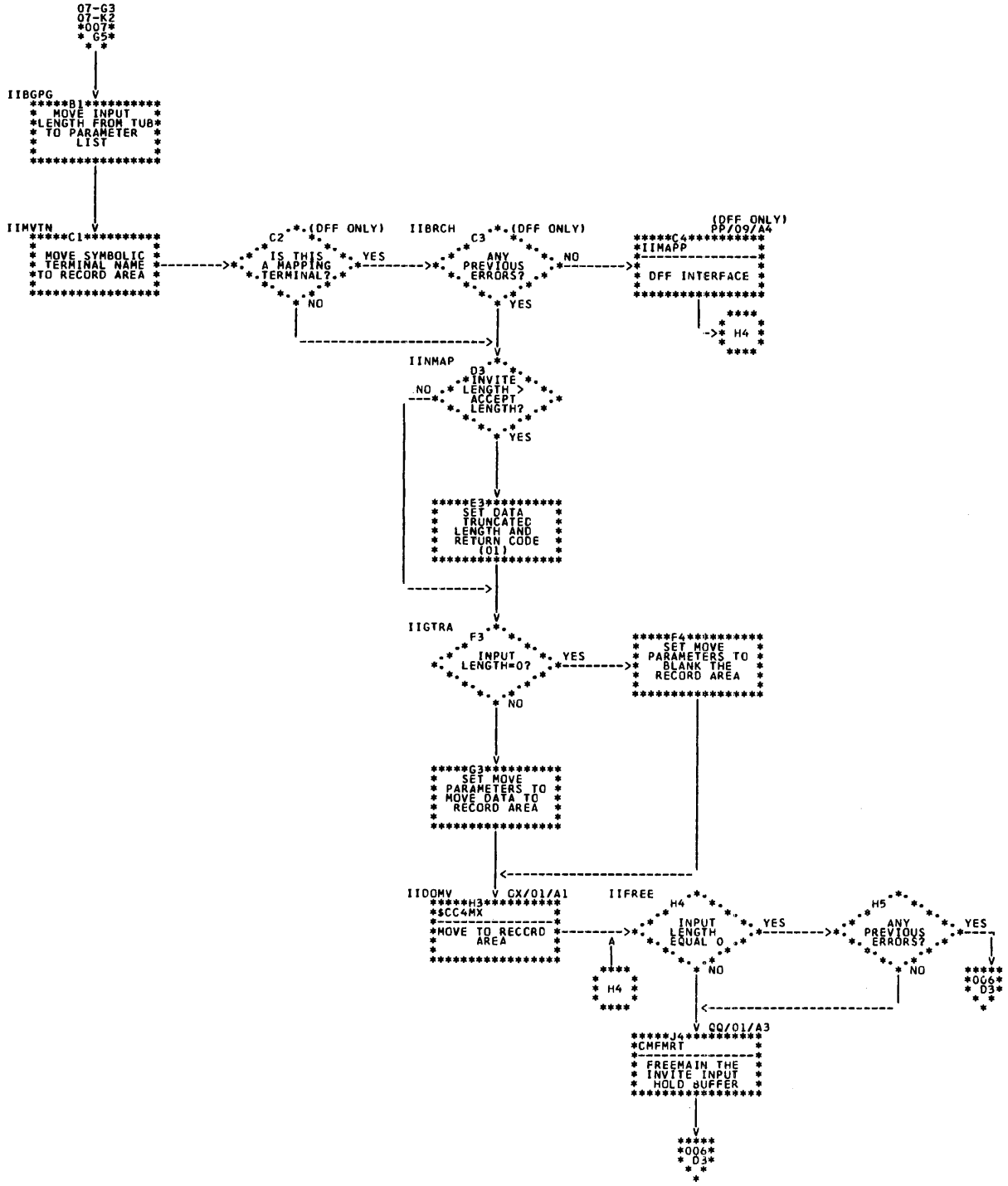


Chart PP (Part 8 of 10). I/O Interface Mainline ($CC4II/$CC4IS)

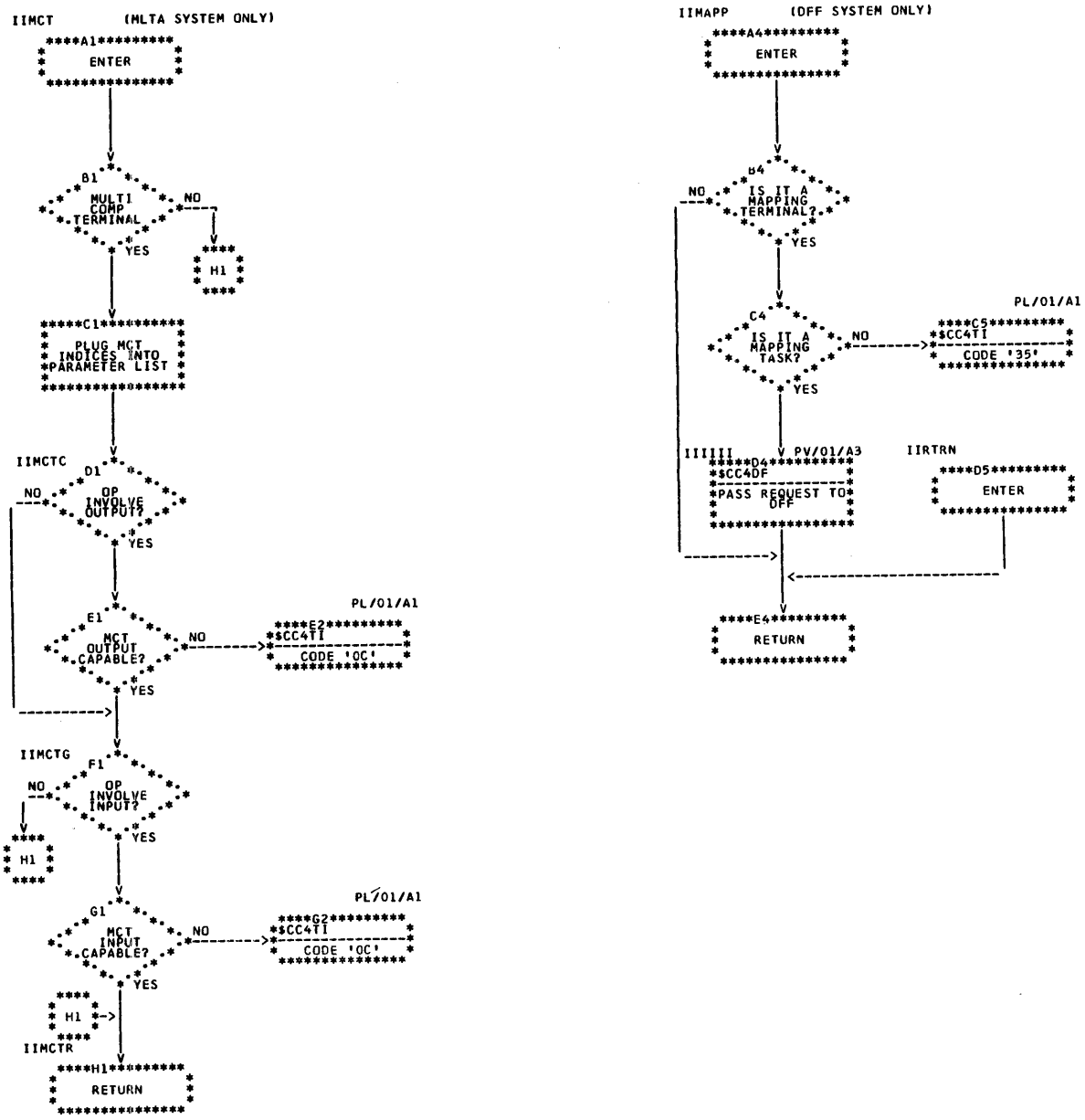


Chart PP (Part 9 of 10). I/O Interface Mainline ($CC4II/$CC4IS)

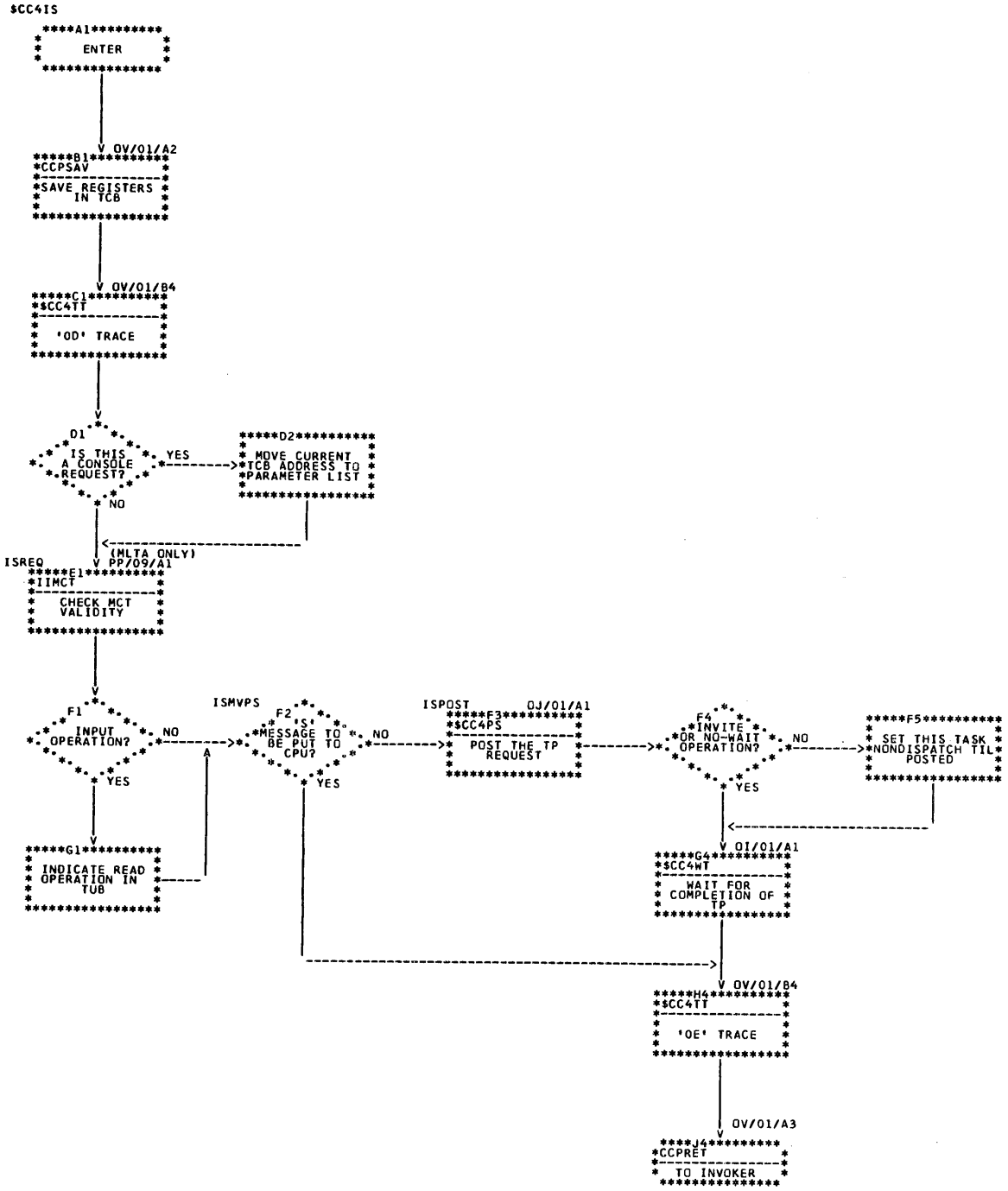


Chart PP (Part 10 of 10). I/O Interface Mainline ($CC4II/$CC4IS)

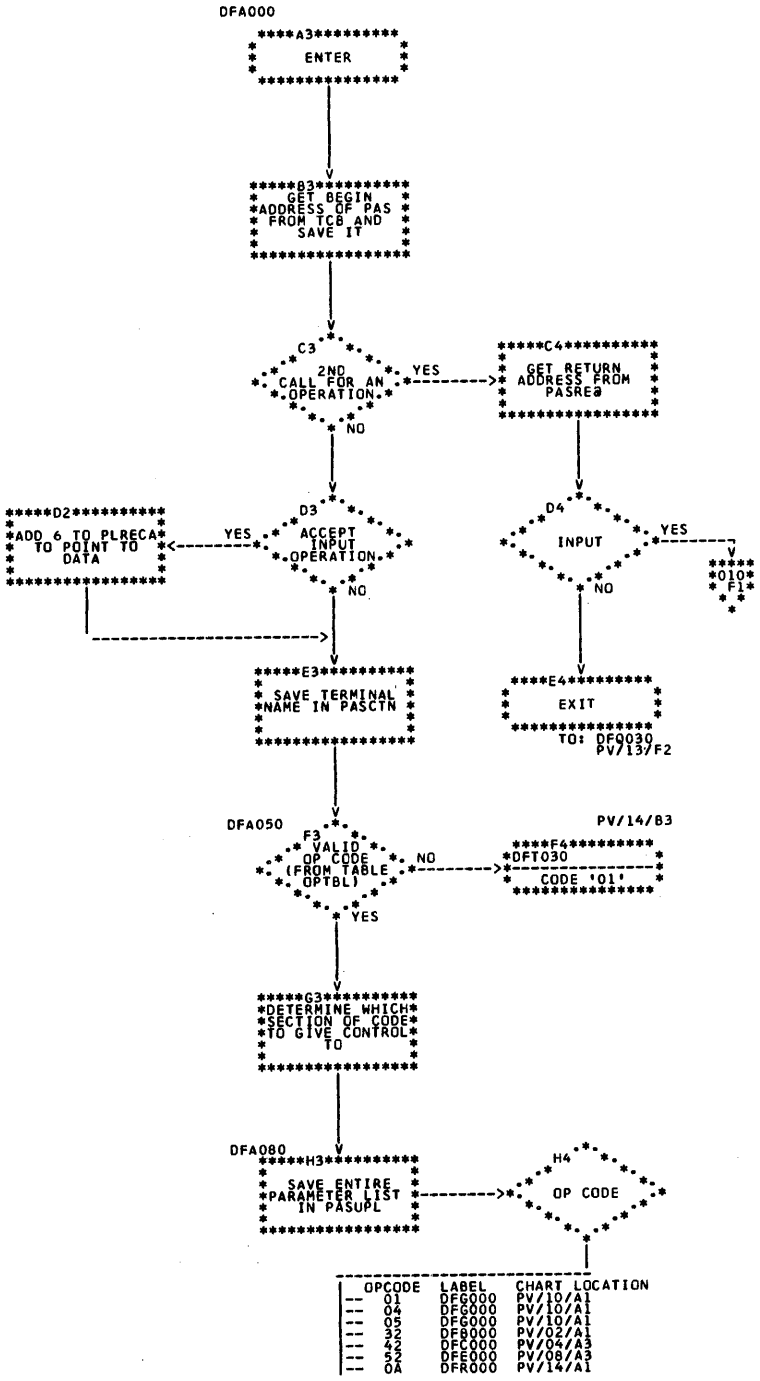


Chart PV (Part 1 of 14). Display Format Control Routine ($CC4DF)

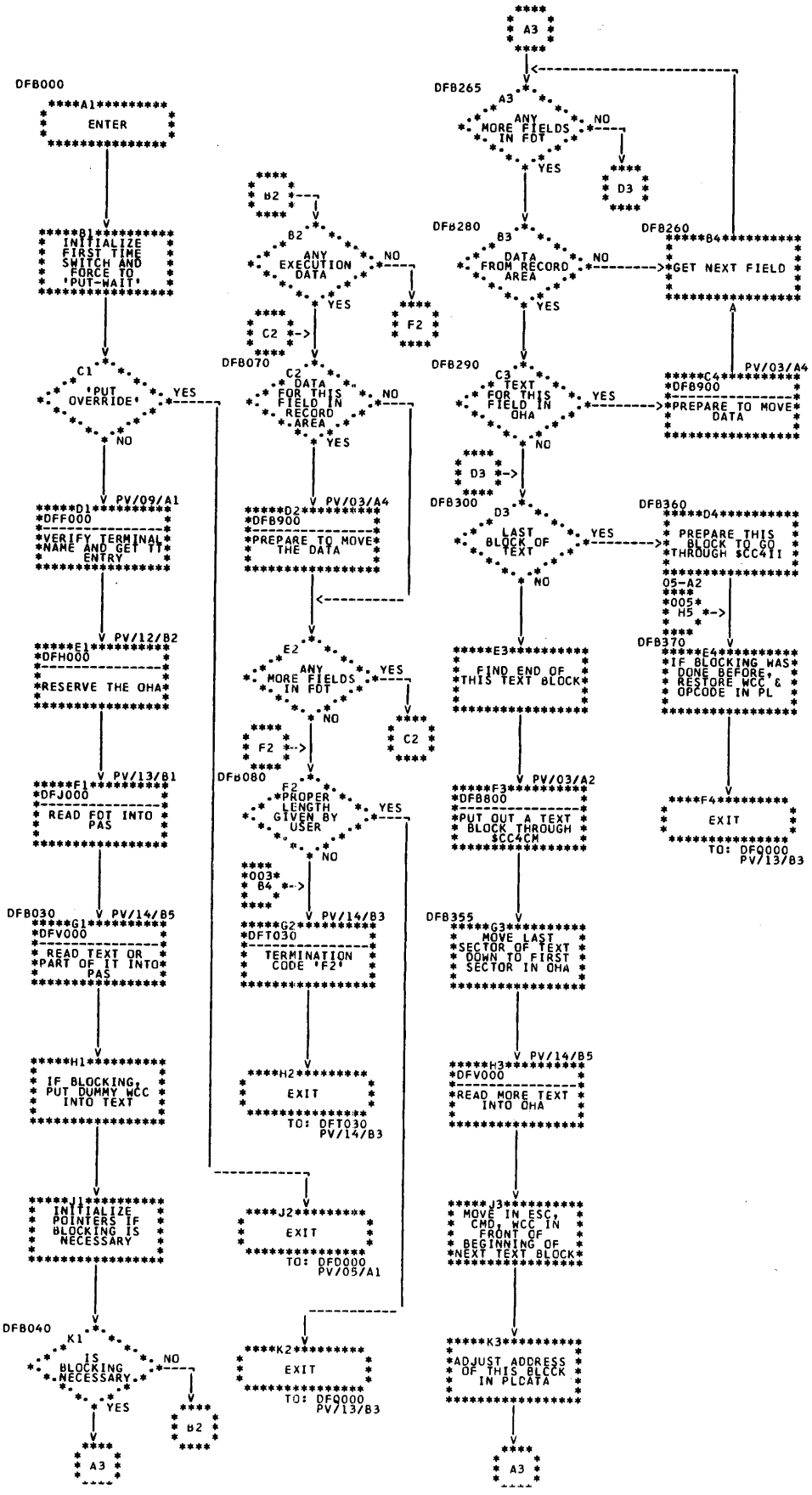


Chart PV (Part 2 of 14). Display Format Control Routine ($CC4DF)

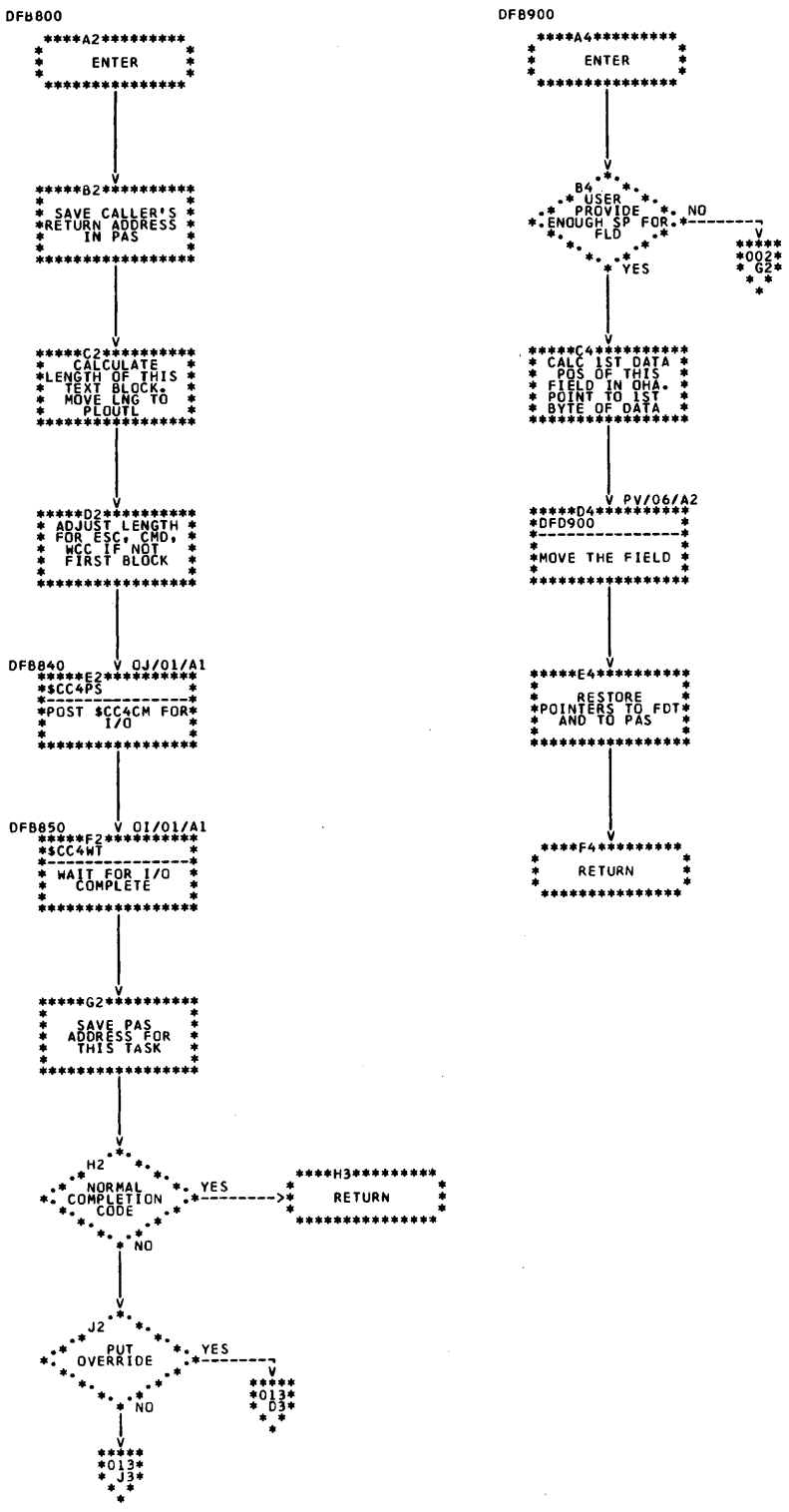


Chart PV (Part 3 of 14). Display Format Control Routine ($CC4DF)

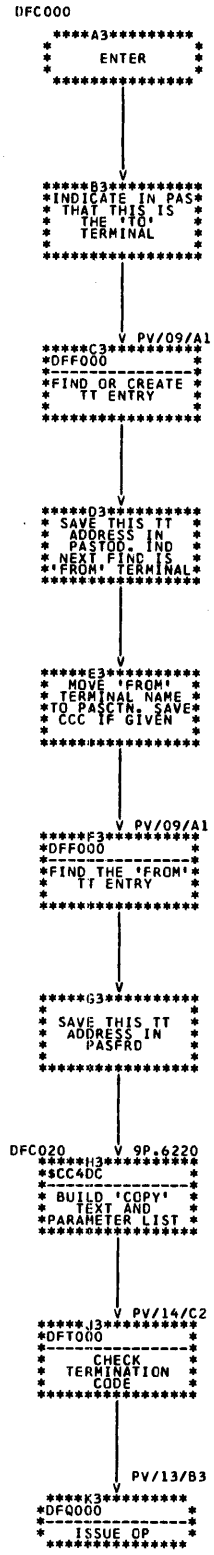


Chart PV (Part 4 of 14). Display Format Control Routine ($CC4DF)

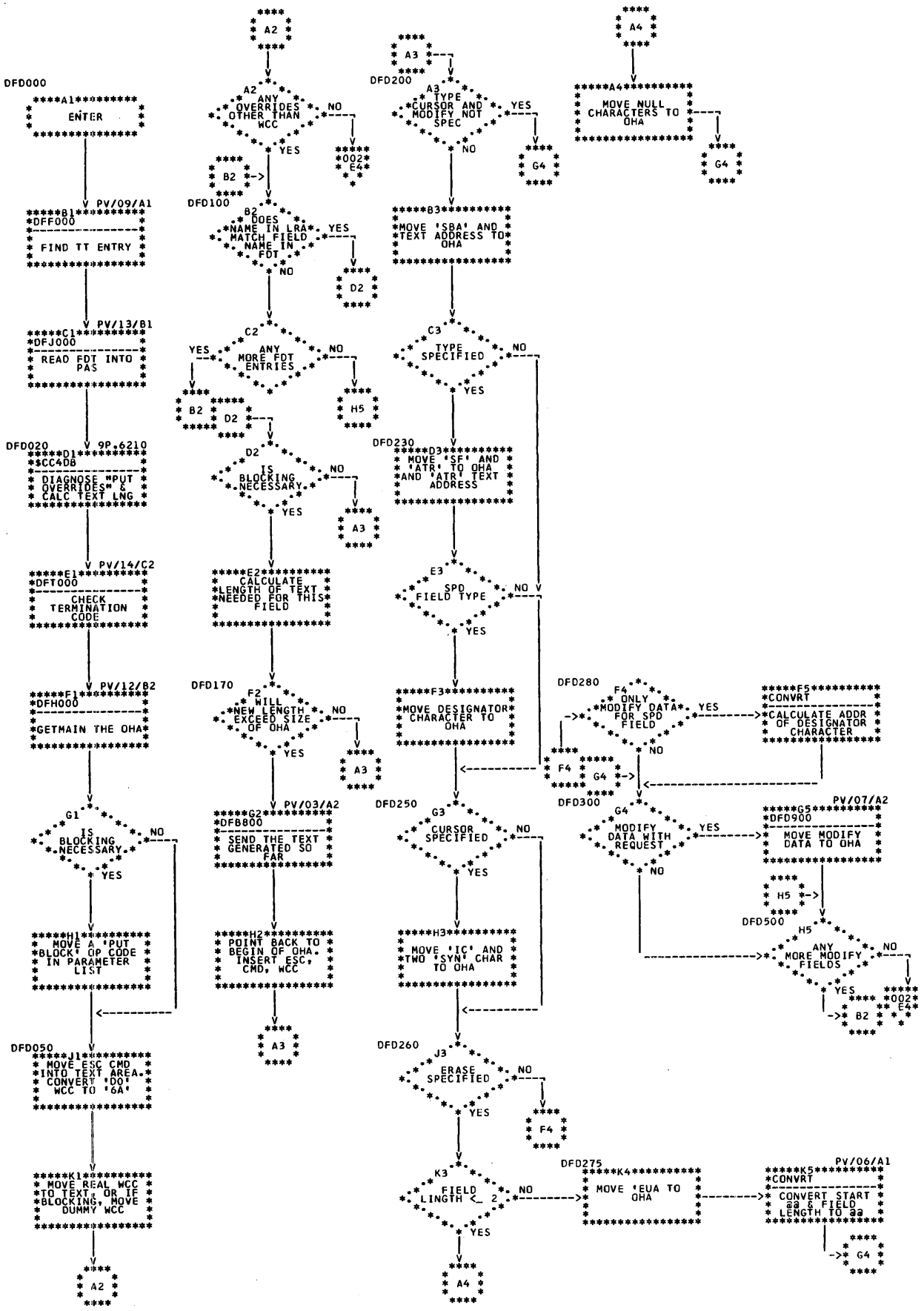


Chart PV (Part 5 of 14). Display Format Control Routine ($CC4DF)

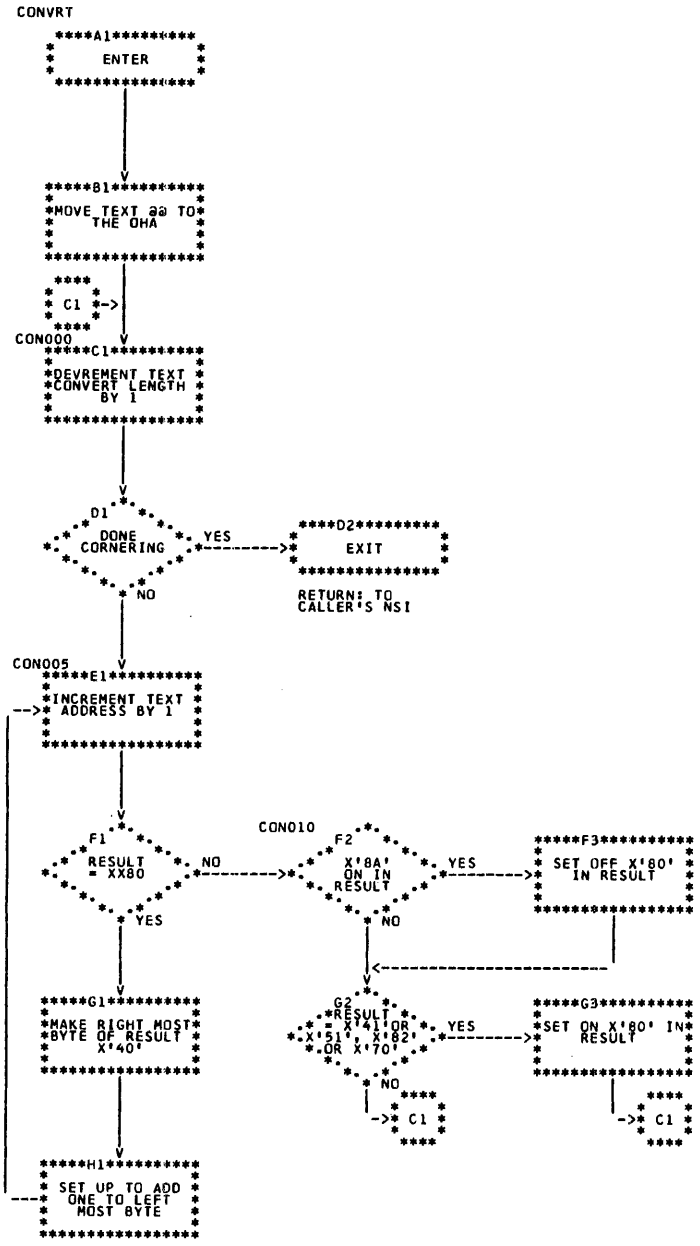


Chart PV (Part 6 of 14). Display Format Control Routine ($CC4DF)

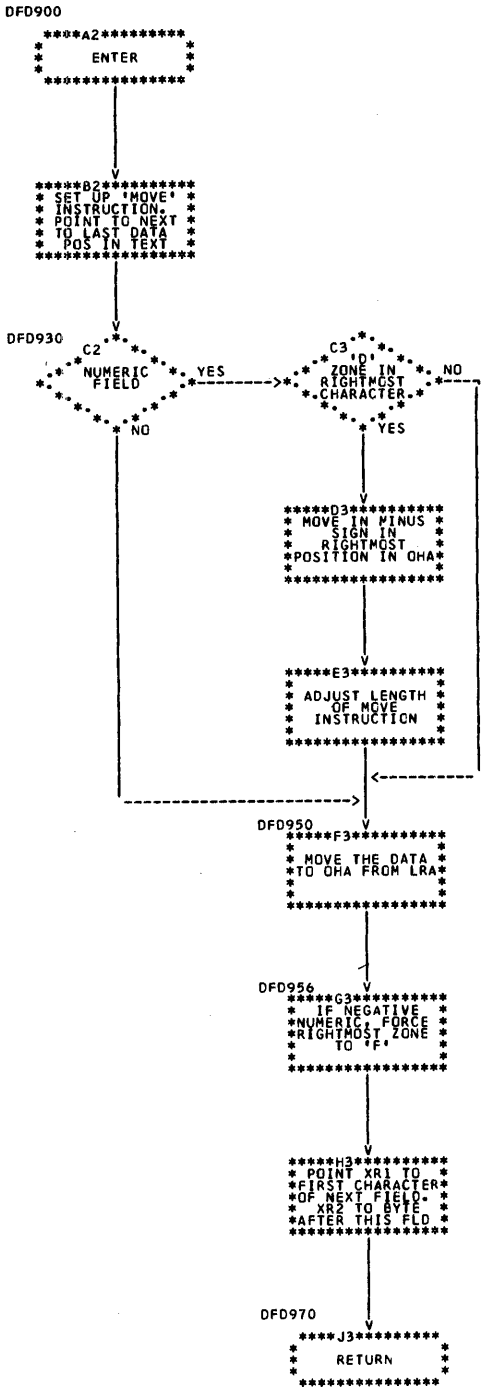


Chart PV (Part 7 of 14). Display Format Control Routine ($CC4DF)

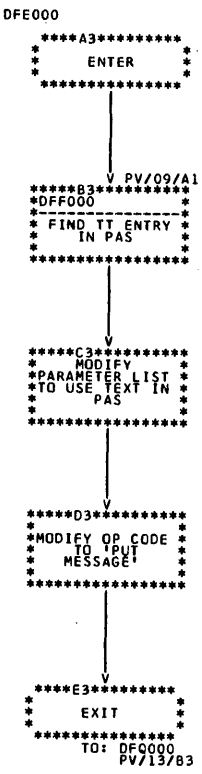


Chart PV (Part 8 of 14). Display Format Control Routine ($CC4DF)

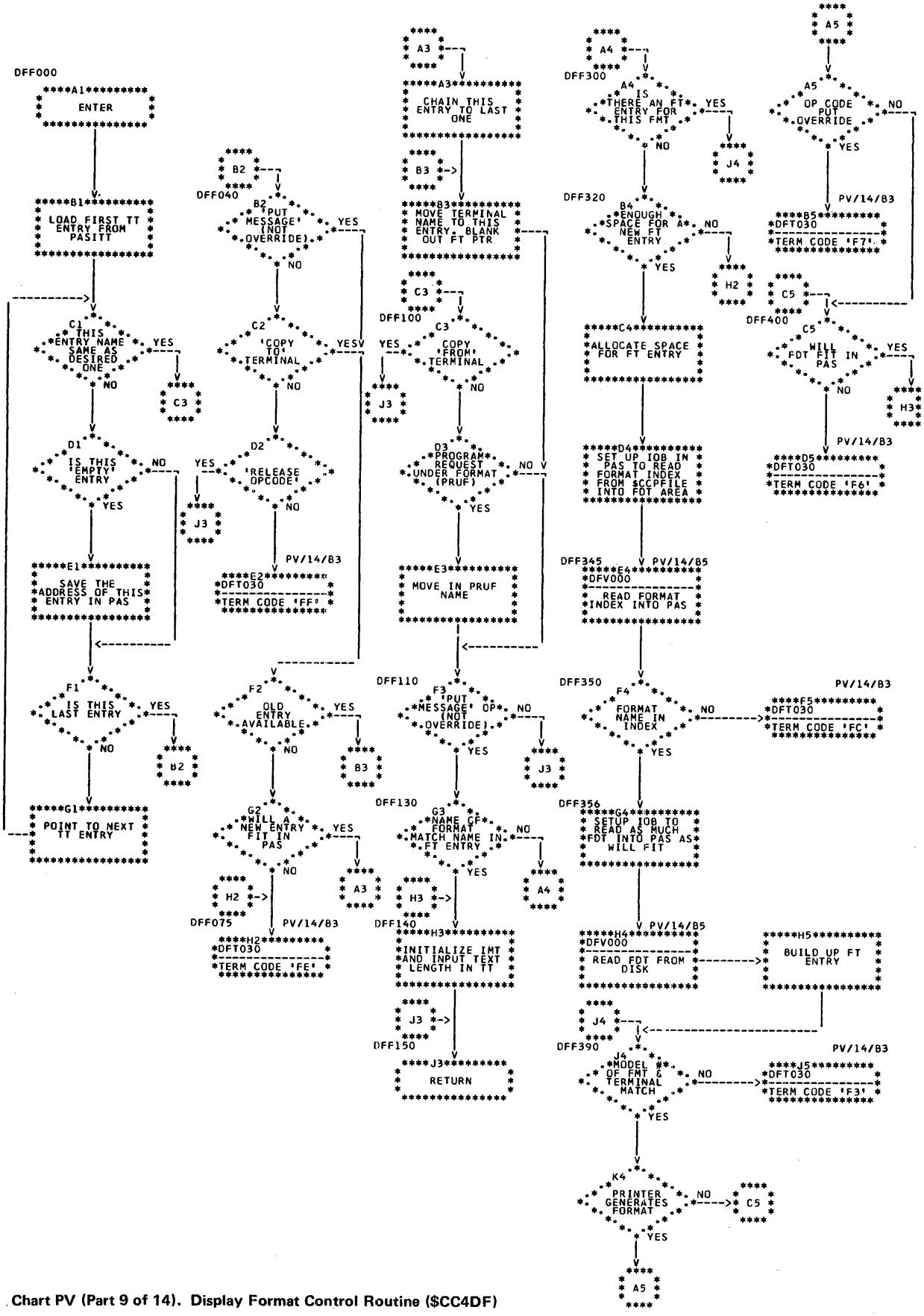


Chart PV (Part 9 of 14). Display Format Control Routine ($CC4DF)

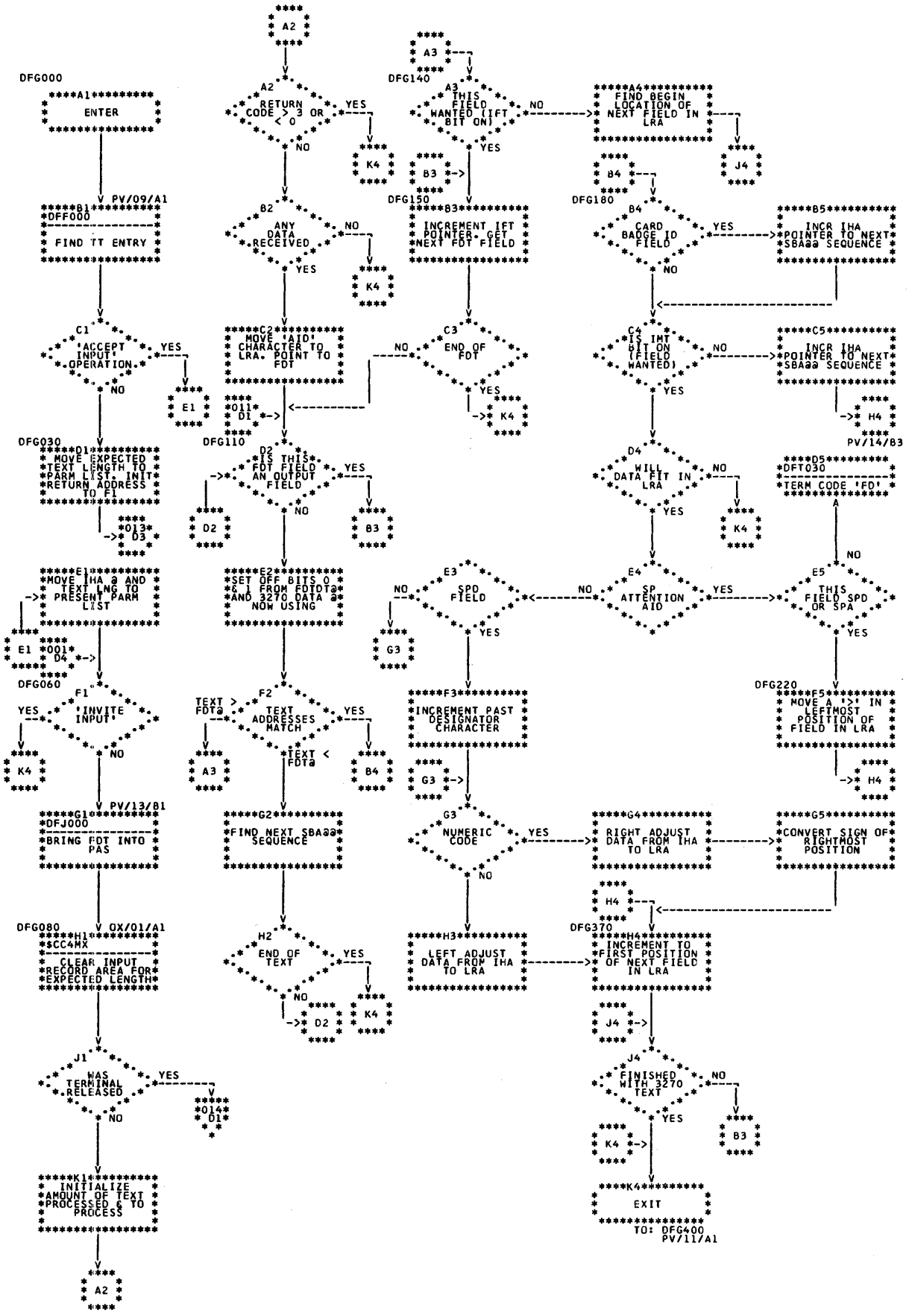


Chart PV (Part 10 of 14). Display Format Control Routine ($CC4DF)

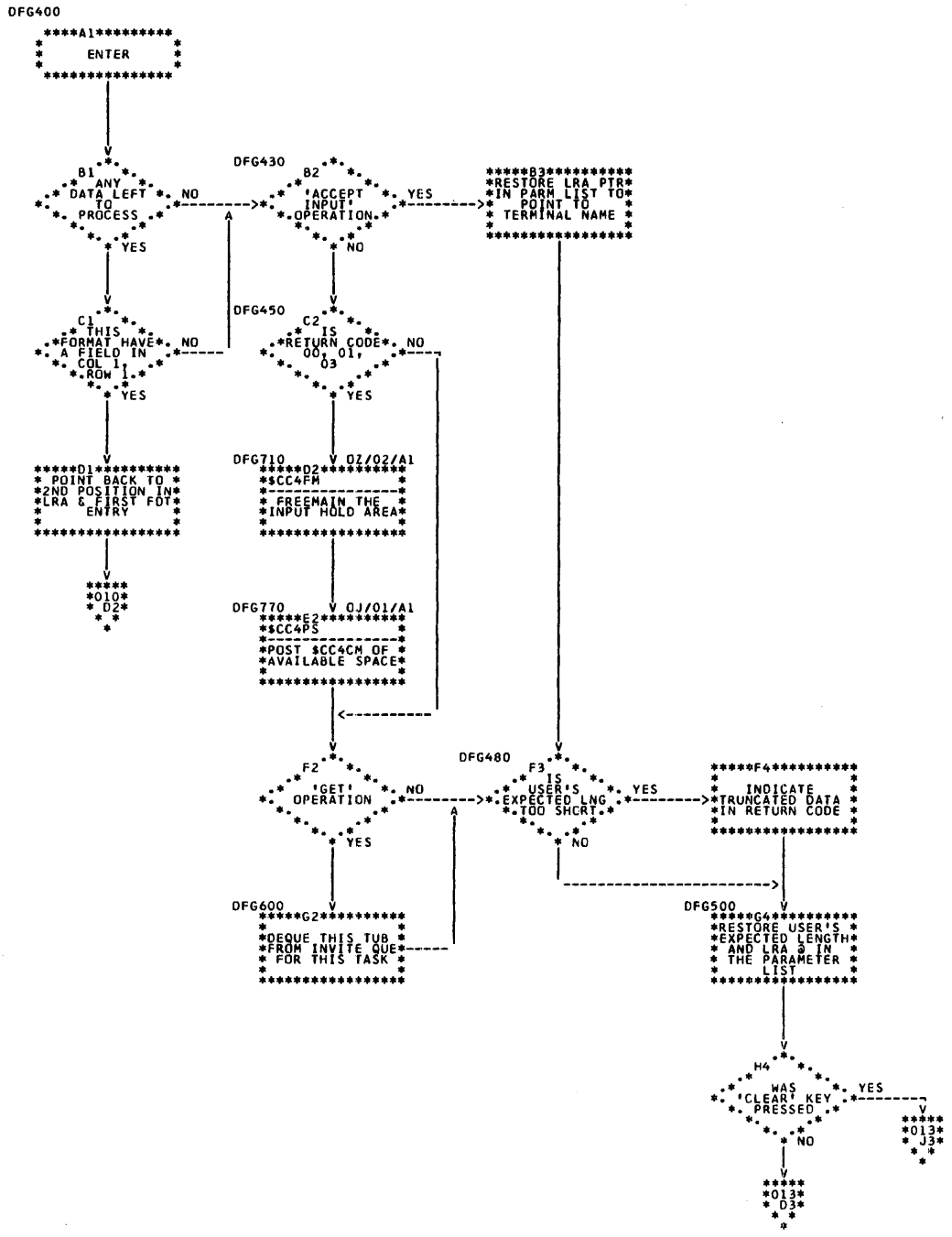


Chart PV (Part 11 of 14). Display Format Control Routine ($CC4DF)

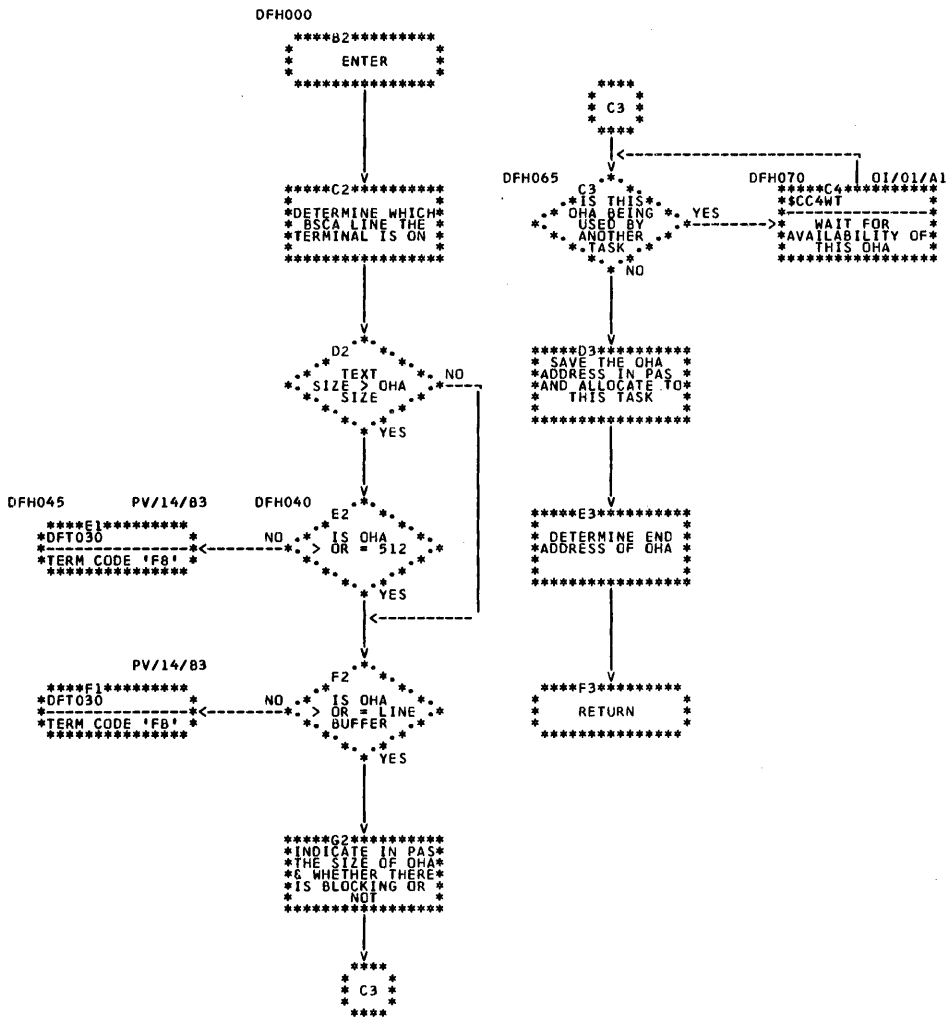


Chart PV (Part 12 of 14). Display Format Control Routine ($CC4DF)

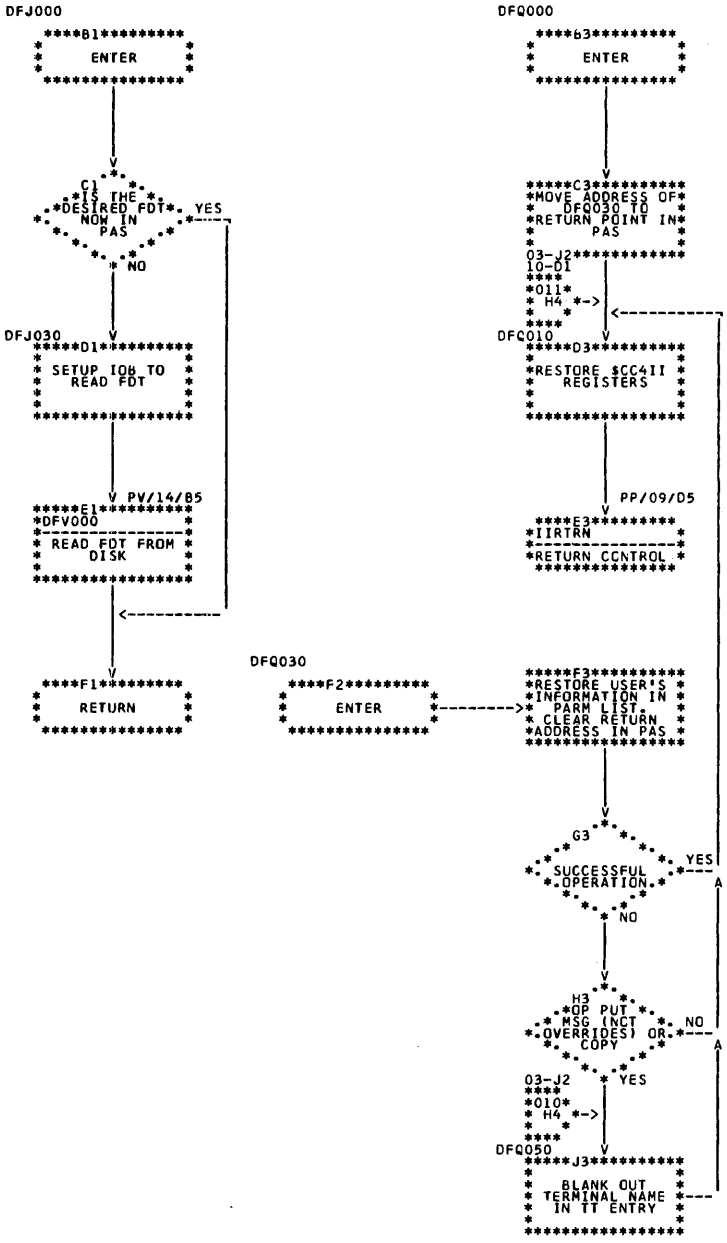


Chart PV (Part 13 of 14). Display Format Control Routine ($CC4DF)

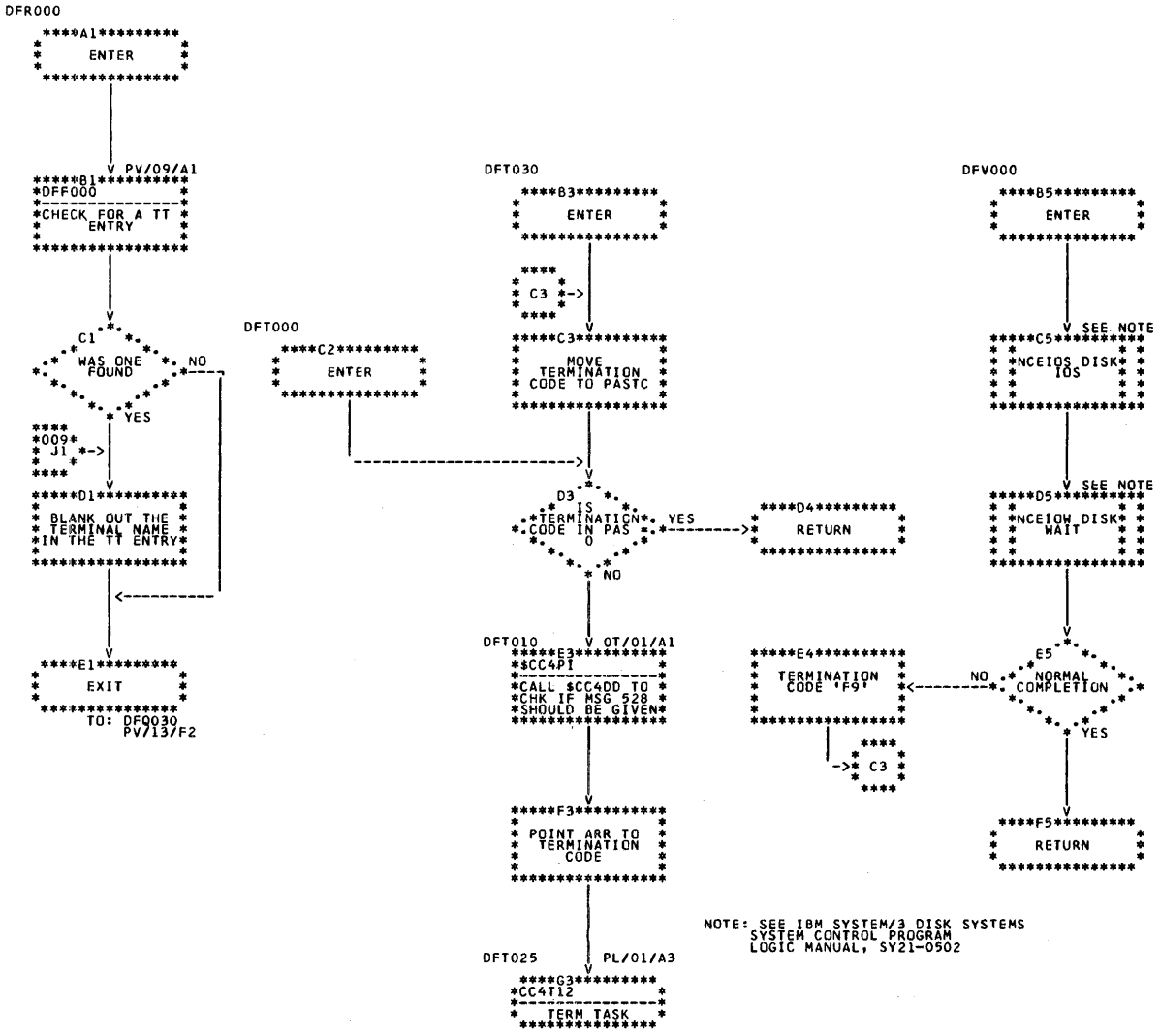


Chart PV (Part 14 of 14). Display Format Control Routine ($CC4DF)

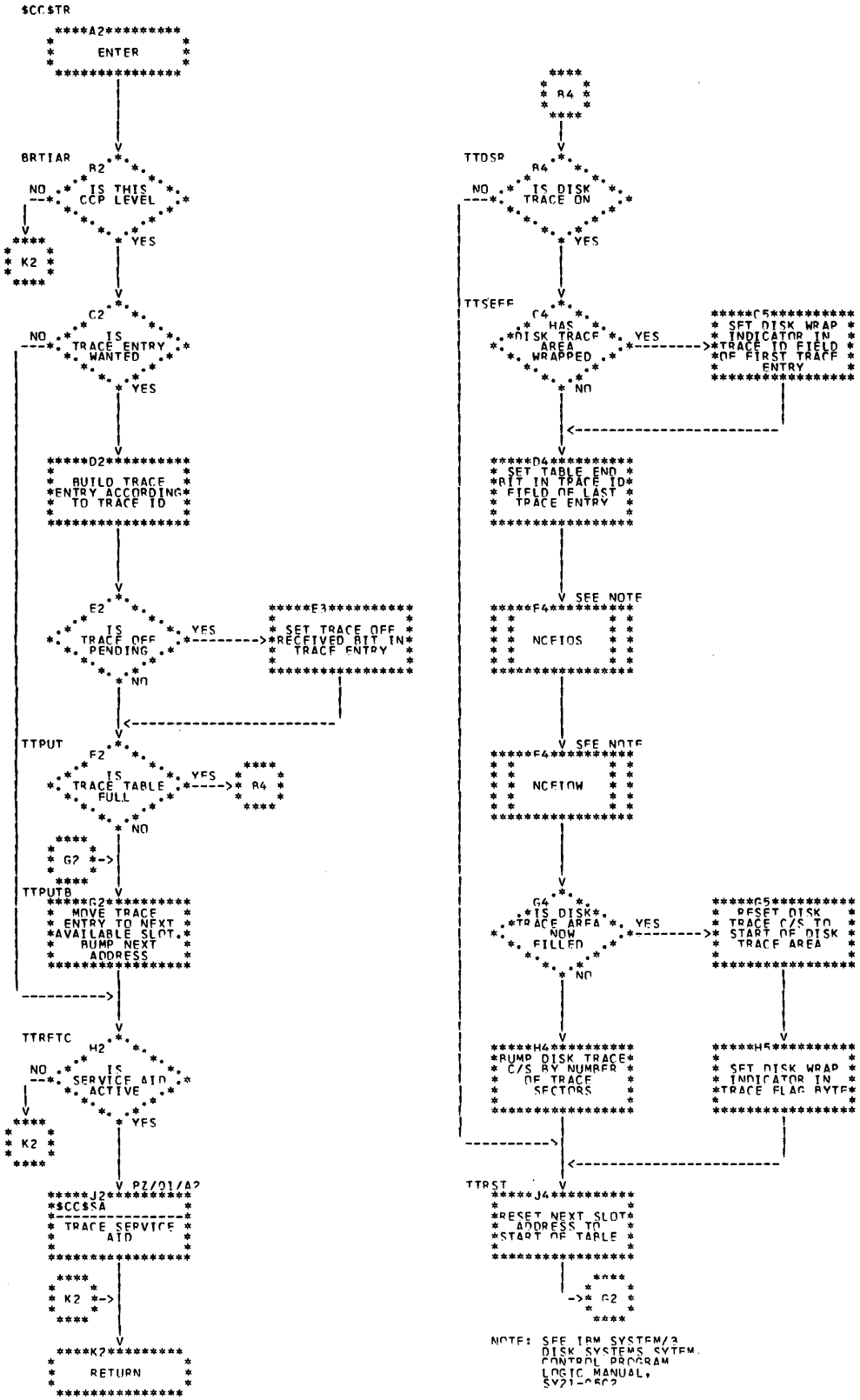


Chart PY. CCP Trace (CCTR)

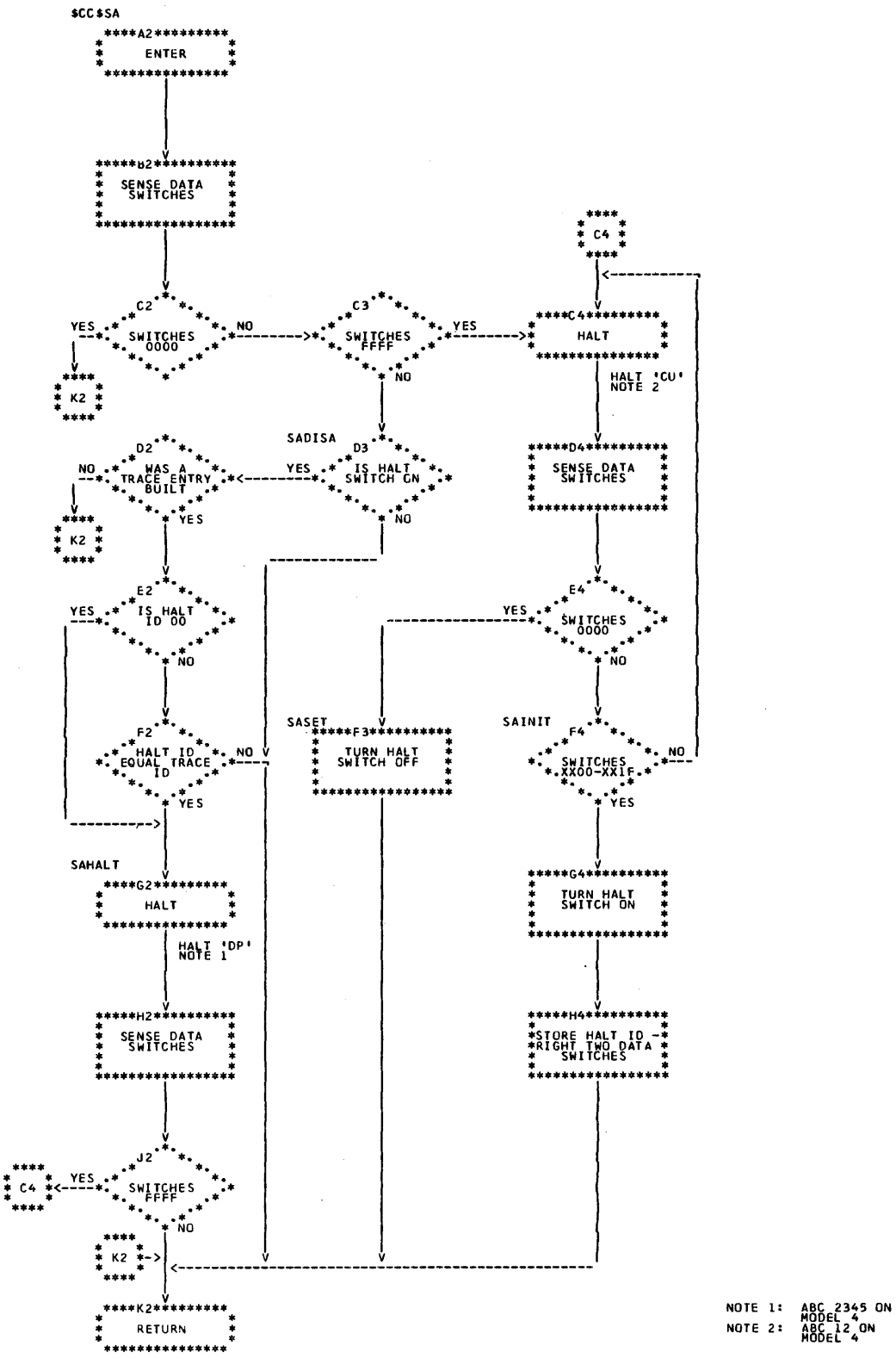


Chart PZ. Trace Halt Service Aid (CCSA)

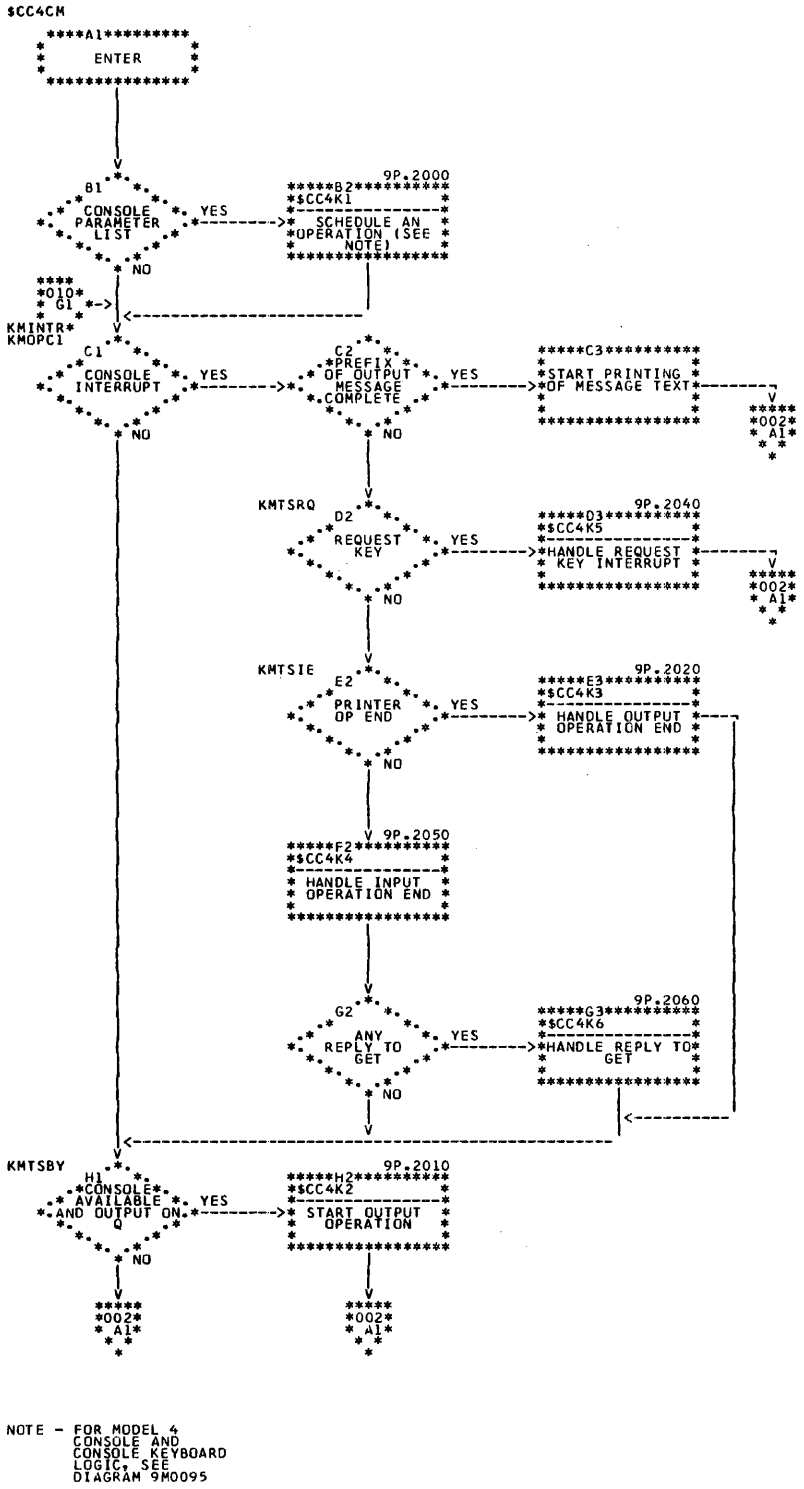


Chart QA (Part 1 of 10). Overview Chart of Communications Management ($CC4CM)

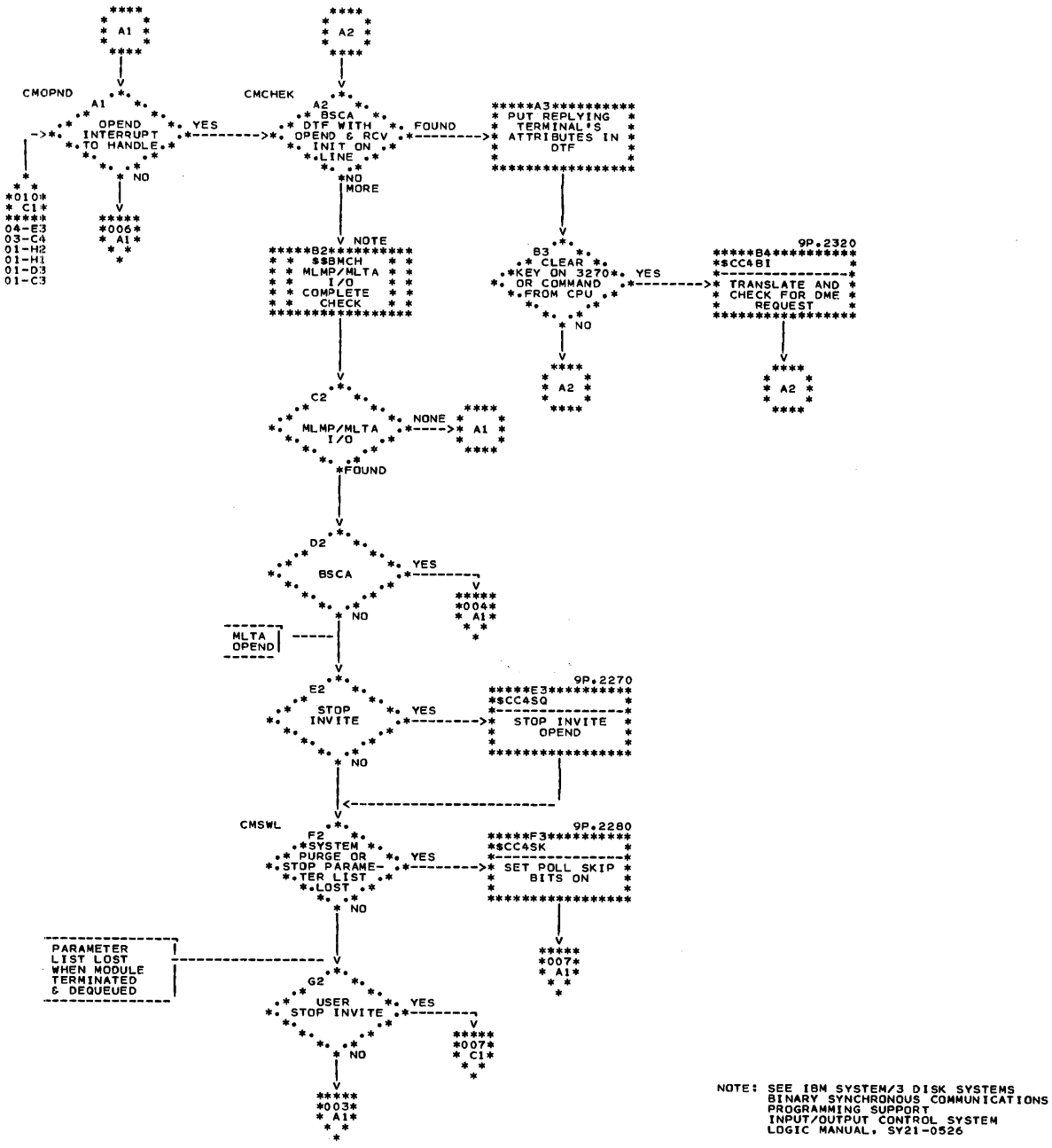


Chart QA (Part 2 of 10). Overview Chart of Communications Management ($CC4CM)

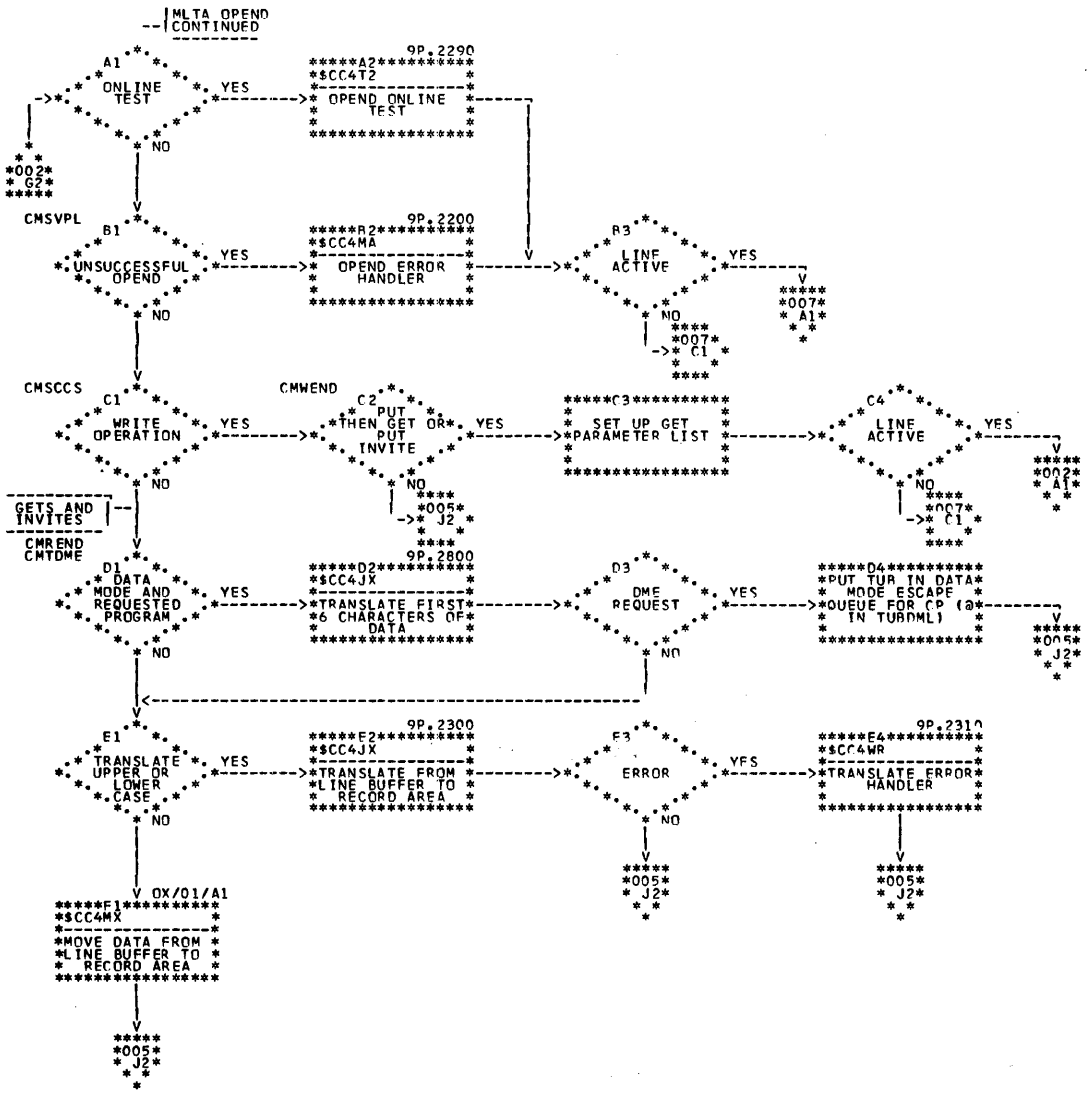


Chart QA (Part 3 of 10). Overview Chart of Communications Management ($CC4CM)

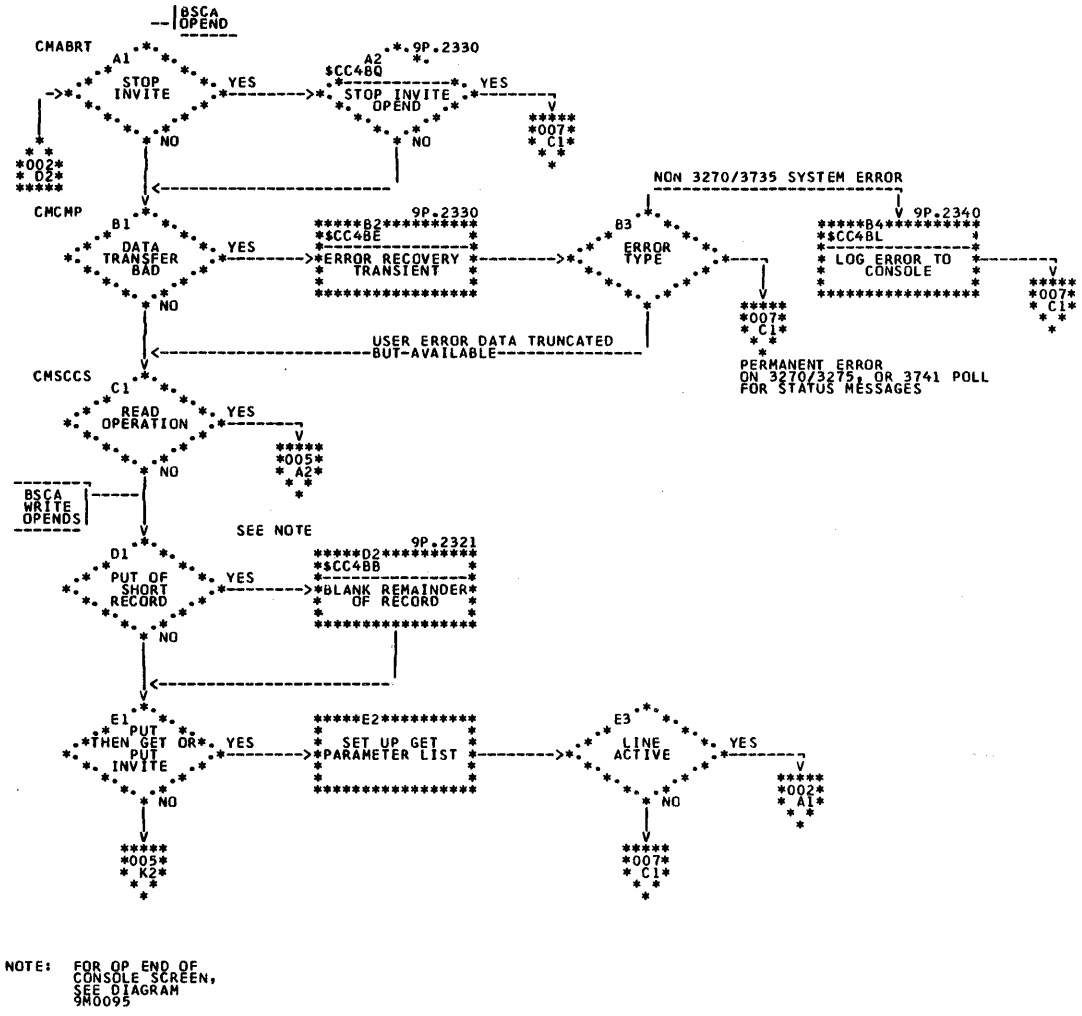


Chart QA (Part 4 of 10). Overview Chart of Communications Management ($CC4CM)

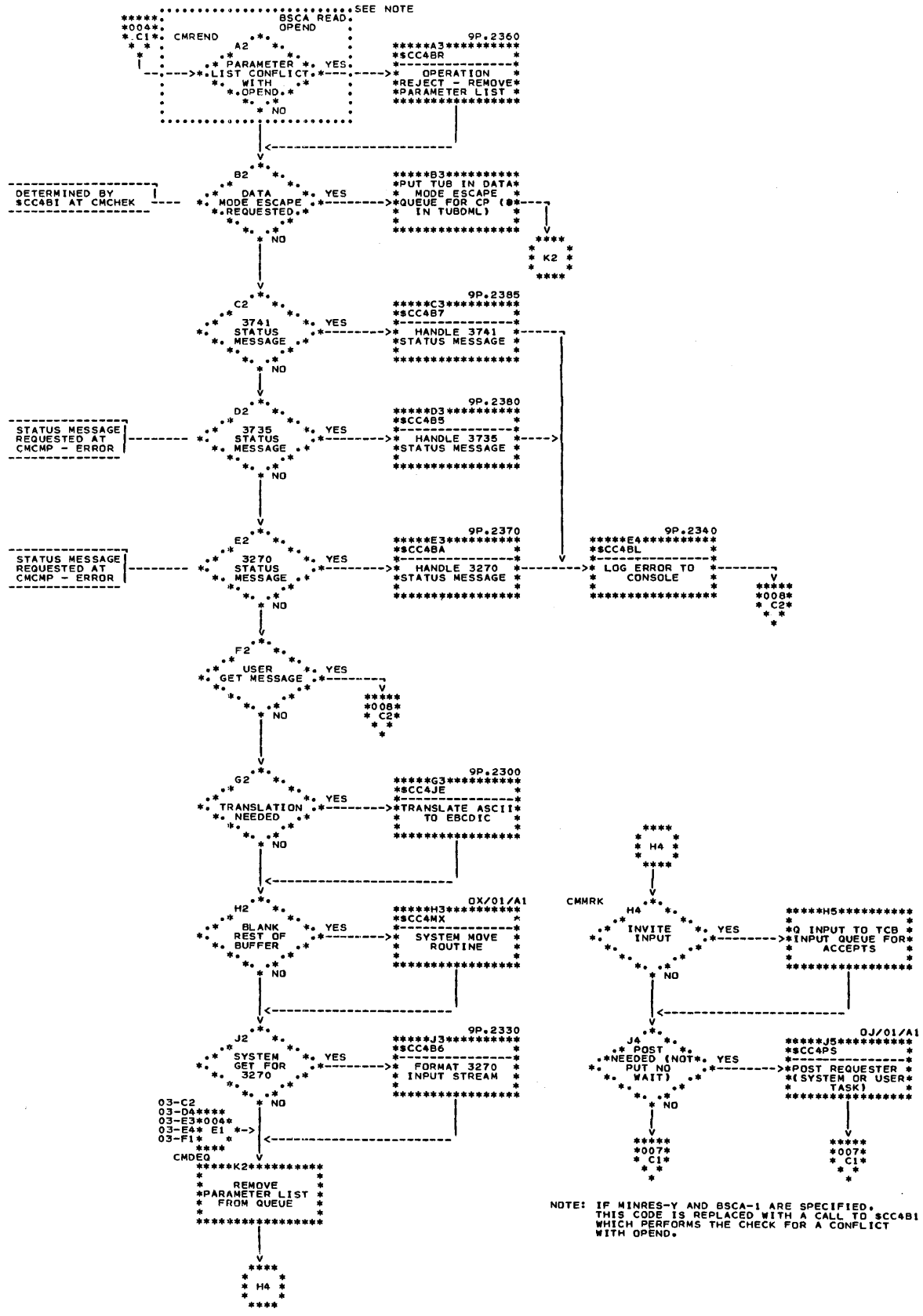


Chart QA (Part 5 of 10). Overview Chart of Communications Management ($CC4CM)

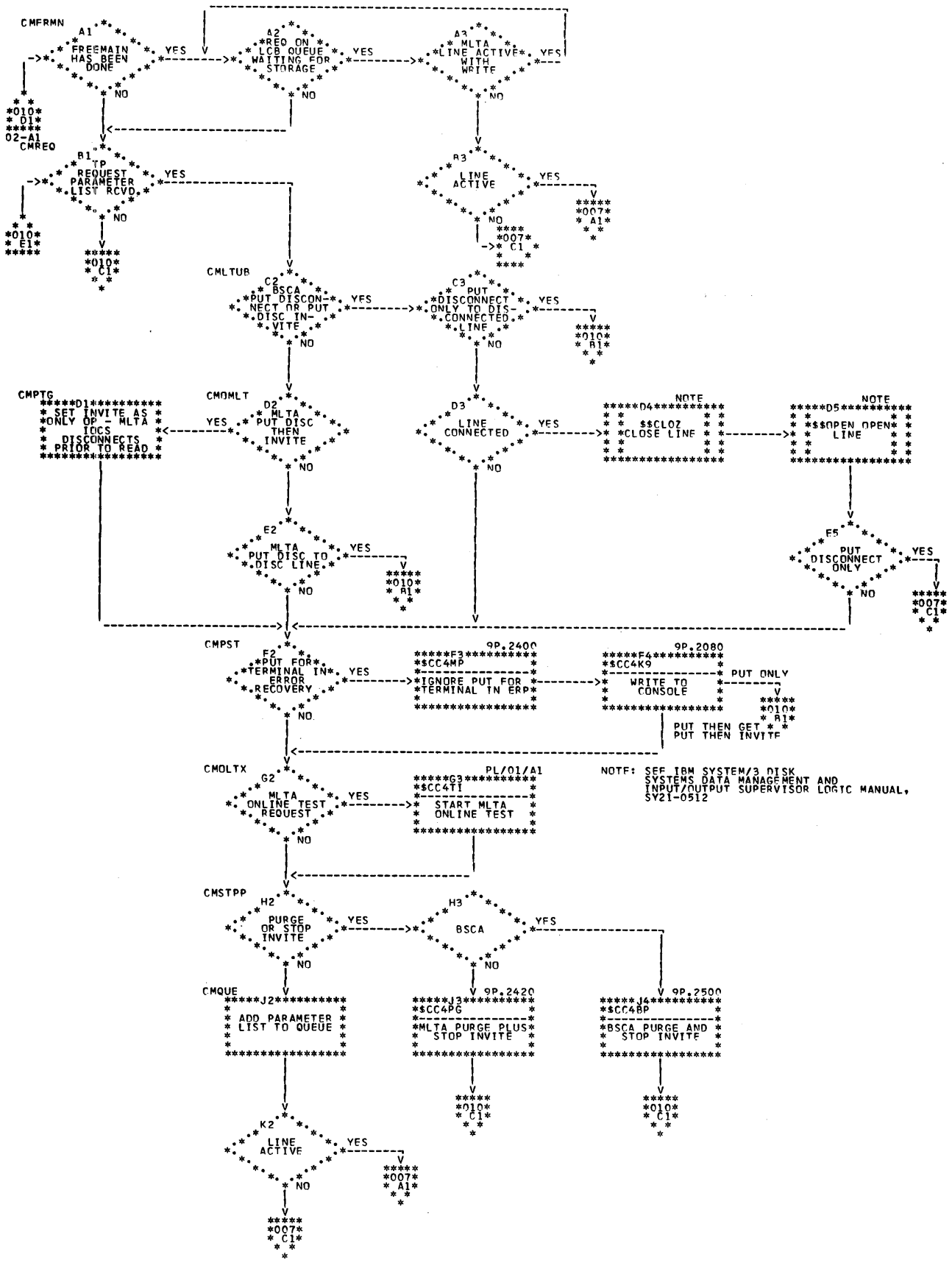


Chart QA (Part 6 of 10). Overview Chart of Communications Management ($CC4CM)

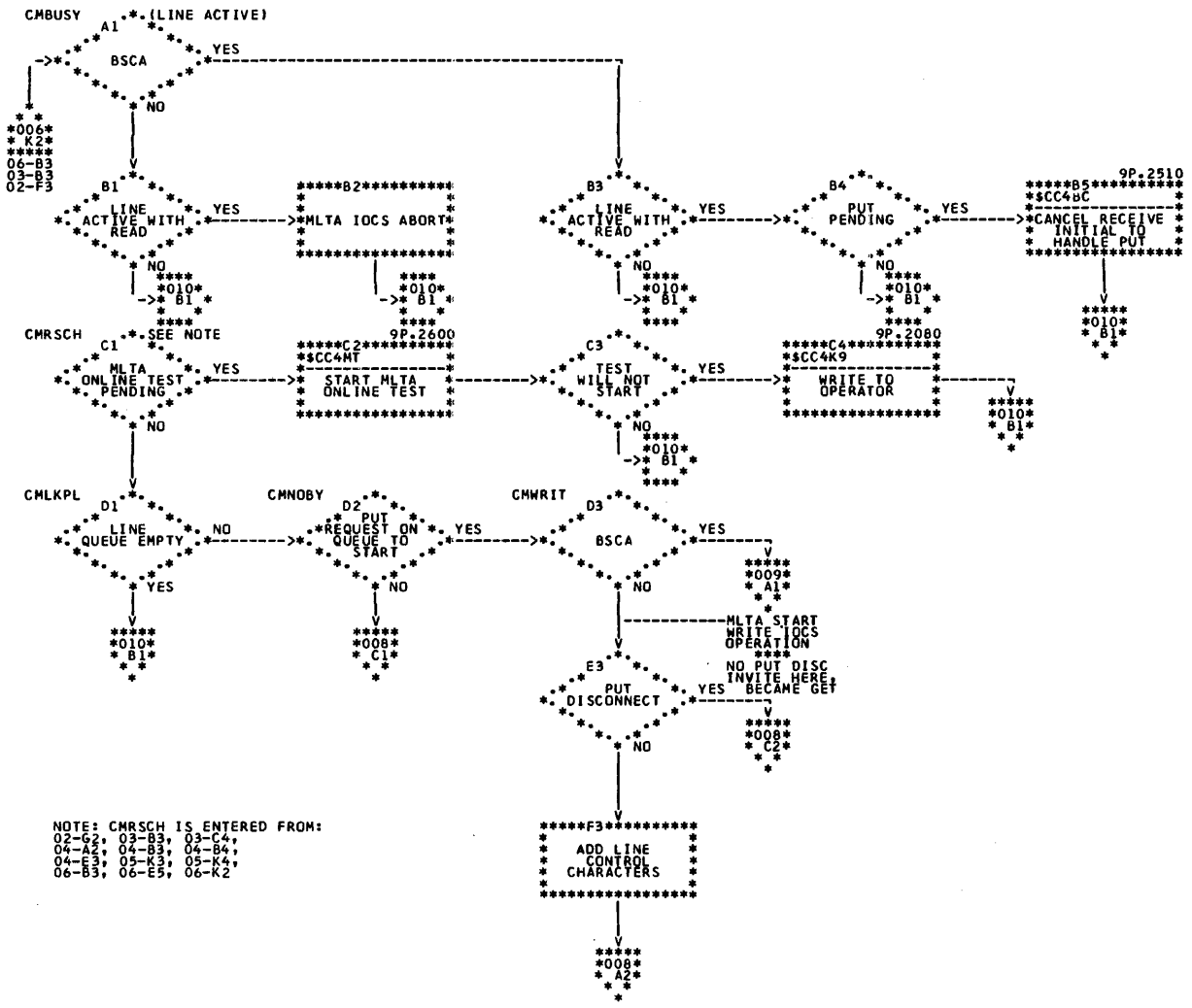


Chart QA (Part 7 of 10). Overview Chart of Communications Management ($CC4CM)

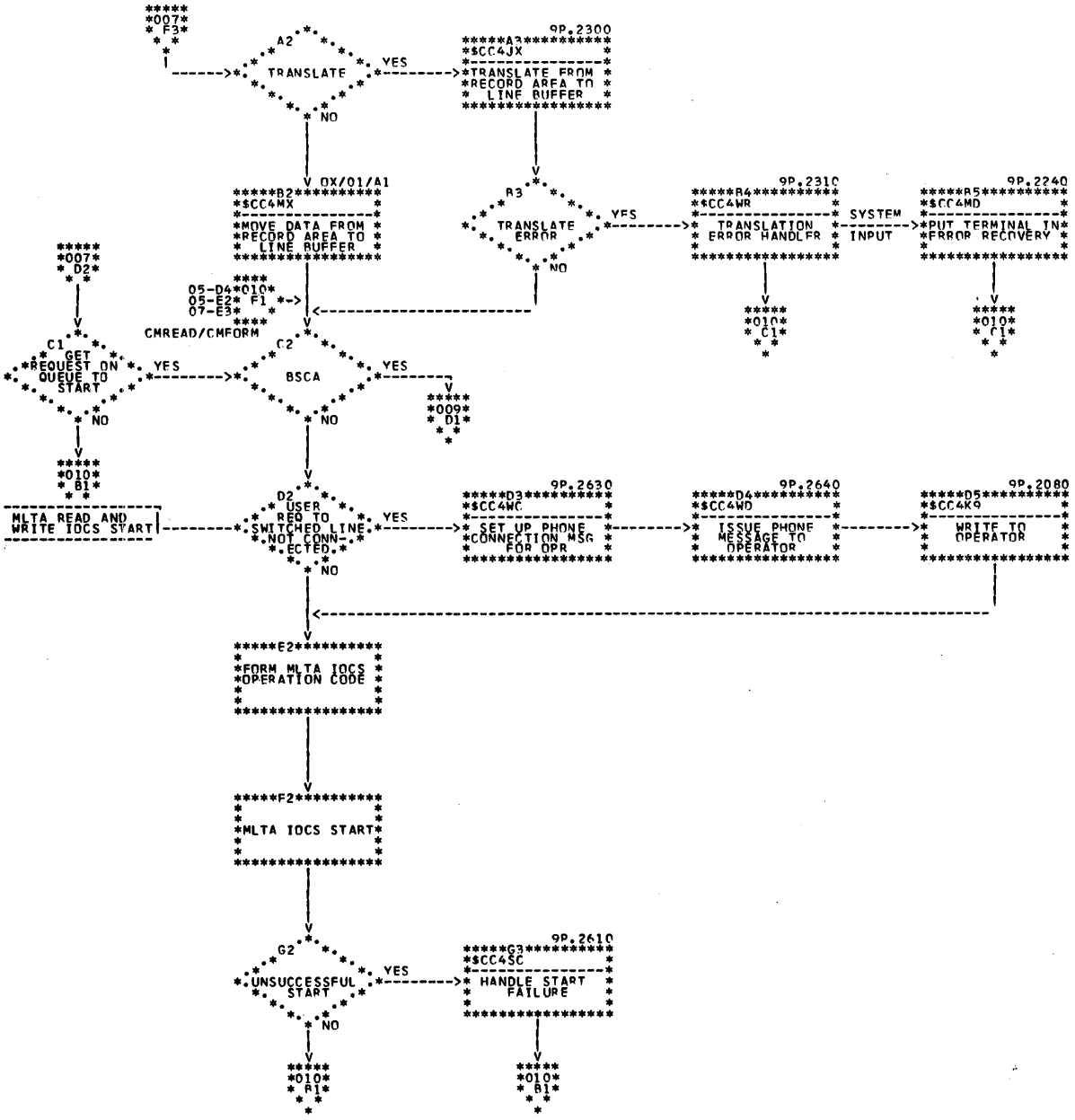
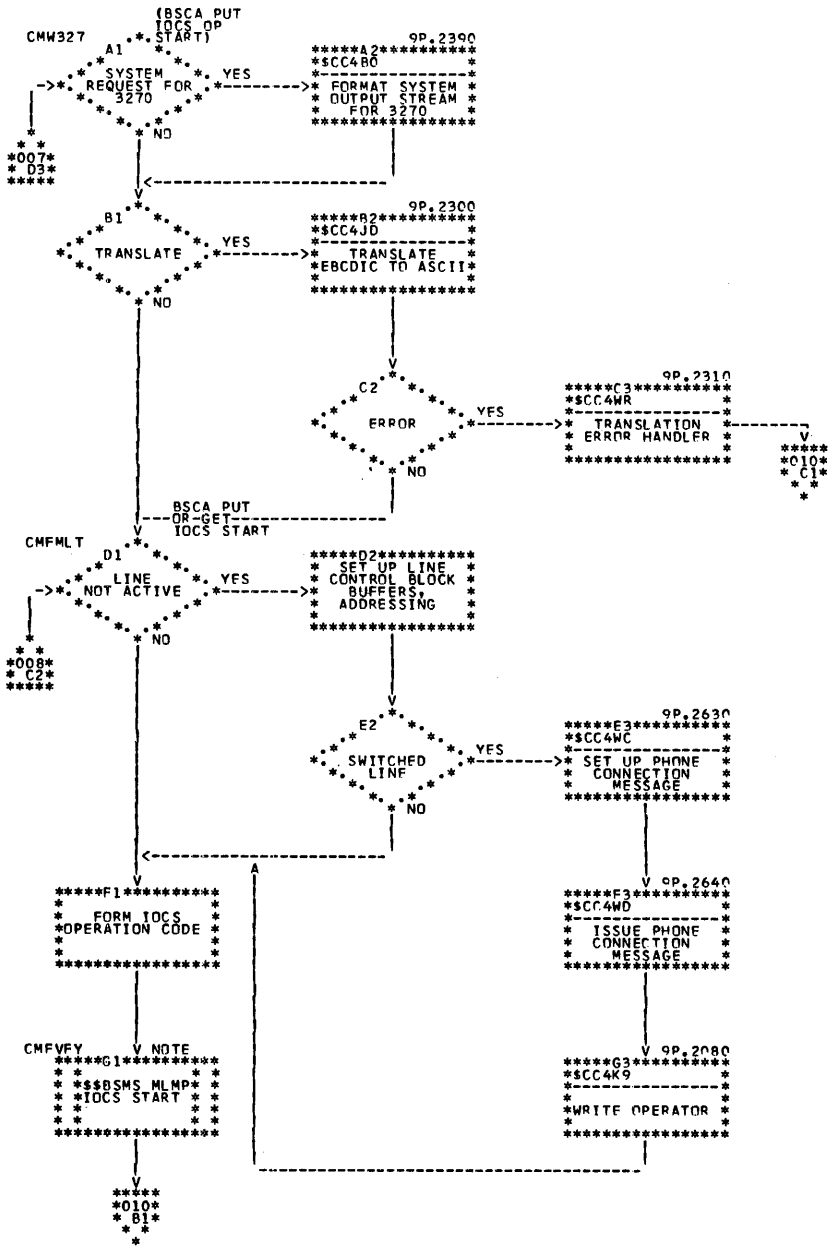


Chart QA (Part 8 of 10). Overview Chart of Communications Management ($CC4CM)



NOTE: SEE IBM SYSTEM/3 DISK SYSTEMS
BINARY SYNCHRONOUS COMMUNICATIONS
PROGRAMMING SUPPORT
INPUT/OUTPUT CONTROL SYSTEM
LOGIC MANUAL, SY21-0526

Chart QA (Part 9 of 10). Overview Chart of Communications Management ($CC4CM)

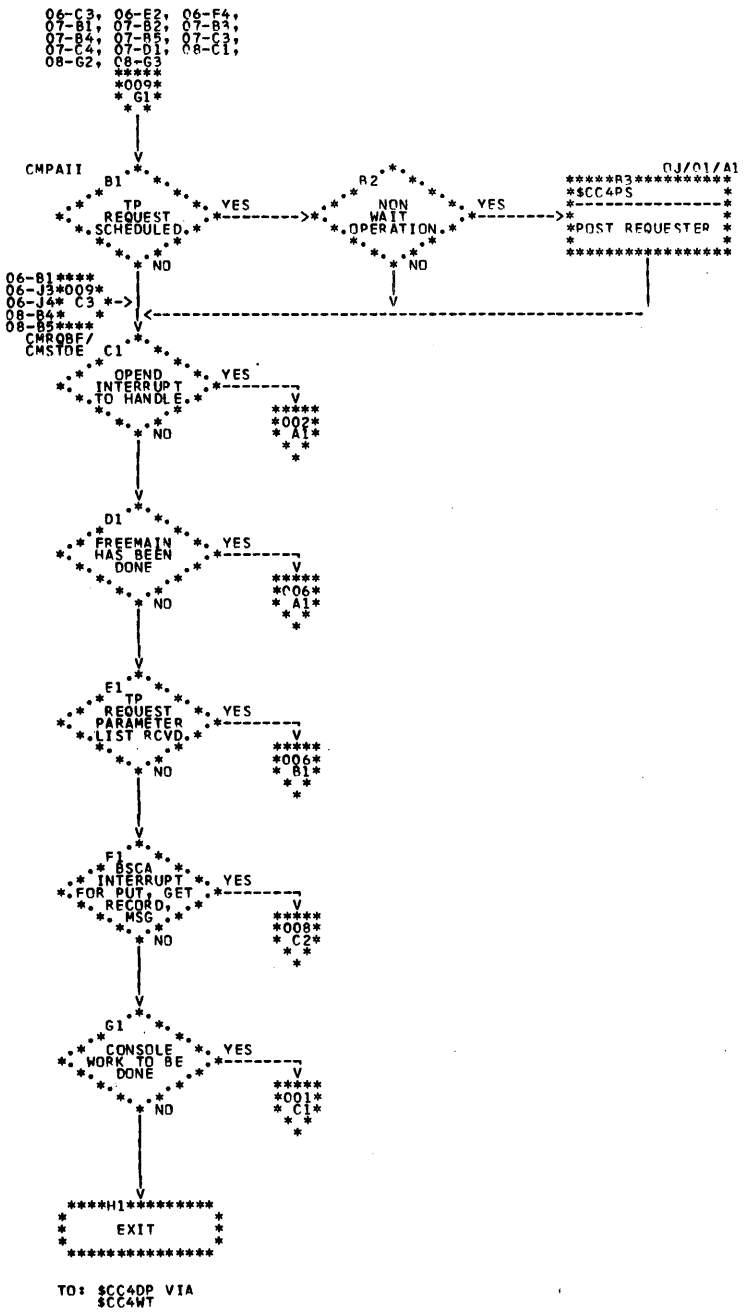
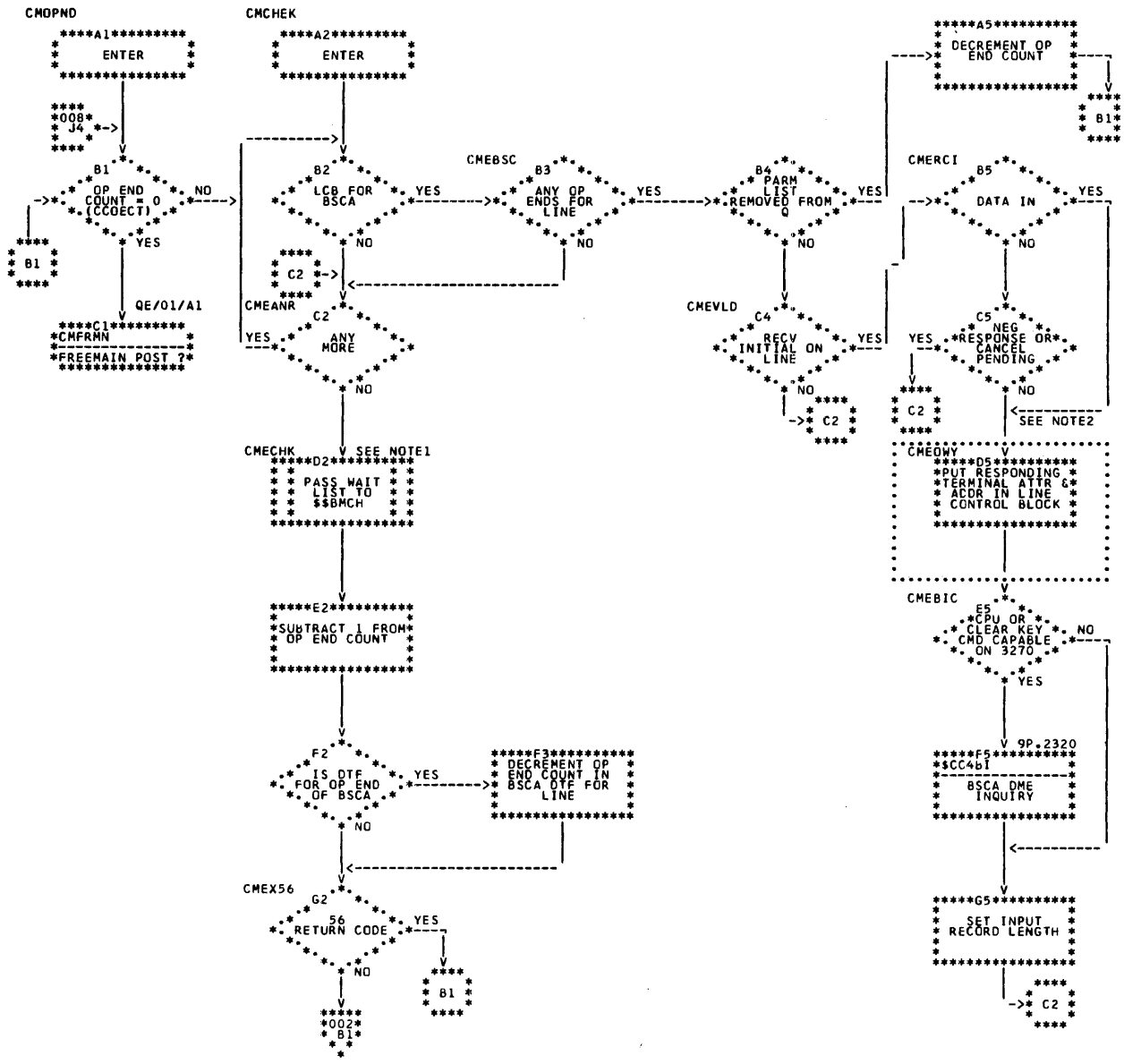


Chart QA (Part 10 of 10). Overview Chart of Communications Management ($CC4CM)



NOTE1: SEE IBM SYSTEM/3 DISK SYSTEMS
BINARY SYNCHRONOUS COMMUNICATIONS
PROGRAMMING SUPPORT INPUT/OUTPUT
CONTROL SYSTEM
LOGIC MANUAL, SY21-0526

NOTE2: IF MINRES-Y AND BSCA-1 ARE SPECIFIED,
THIS CODE IS REPLACED WITH A CALL
TO $CC4B2 WHICH PERFORMS THE CHECK
WITH OPEND.

Chart QC (Part 1 of 8). CM Op End ($CC4CM)

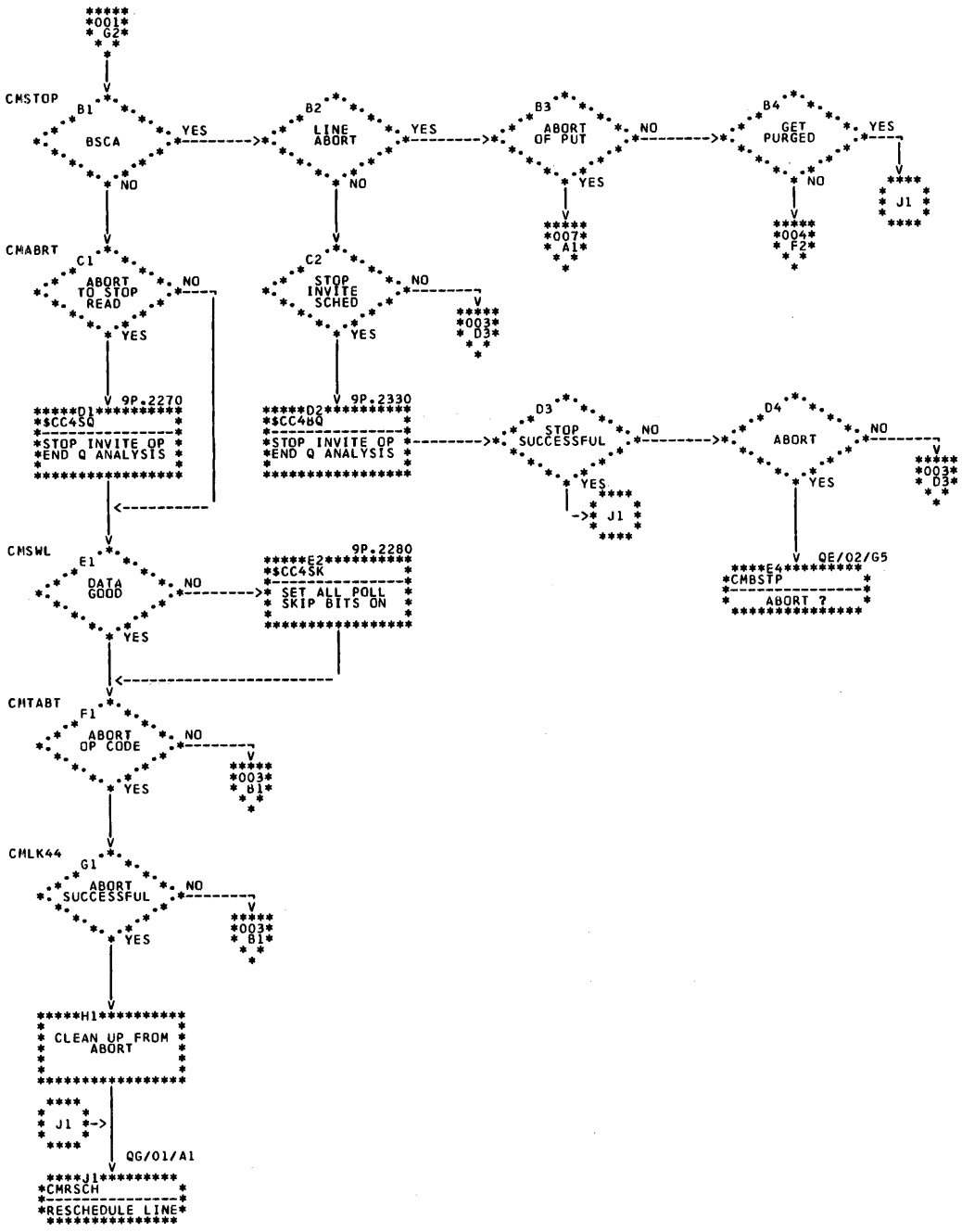


Chart QC (Part 2 of 8). CM Op End ($CC4CM)

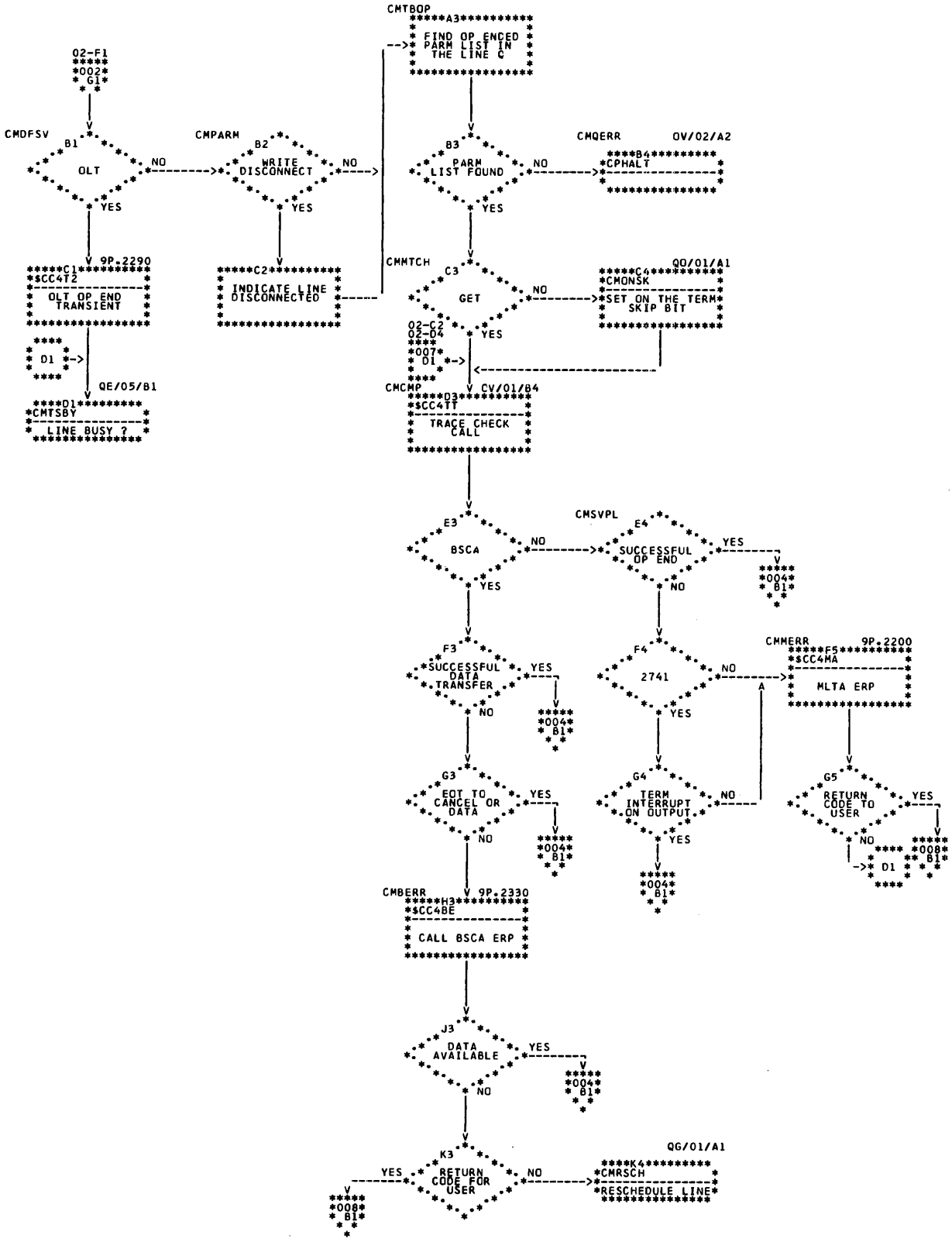


Chart QC (Part 3 of 8). CM Op End ($CC4CM)

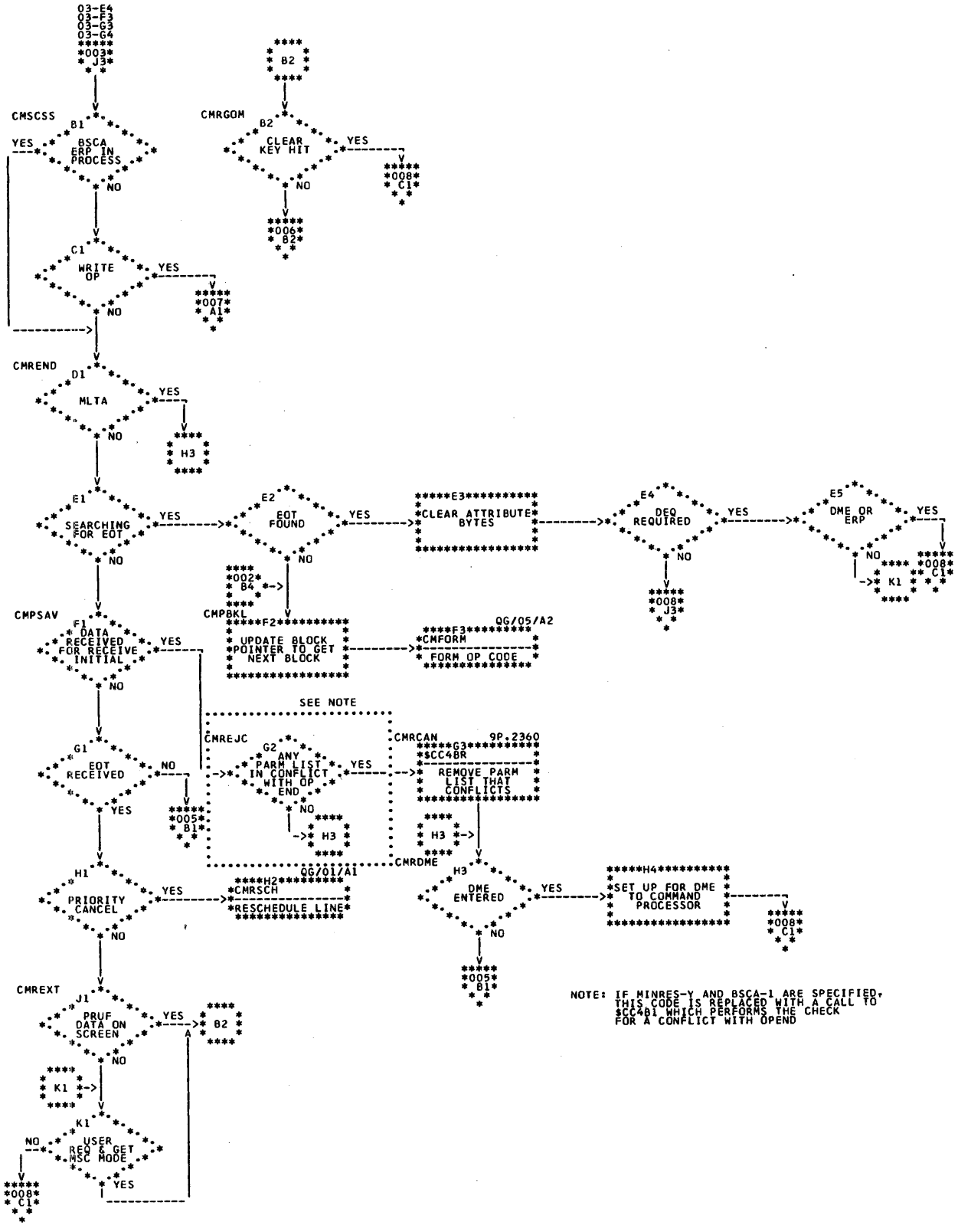


Chart QC (Part 4 of 8). CM Op End ($CC4CM)

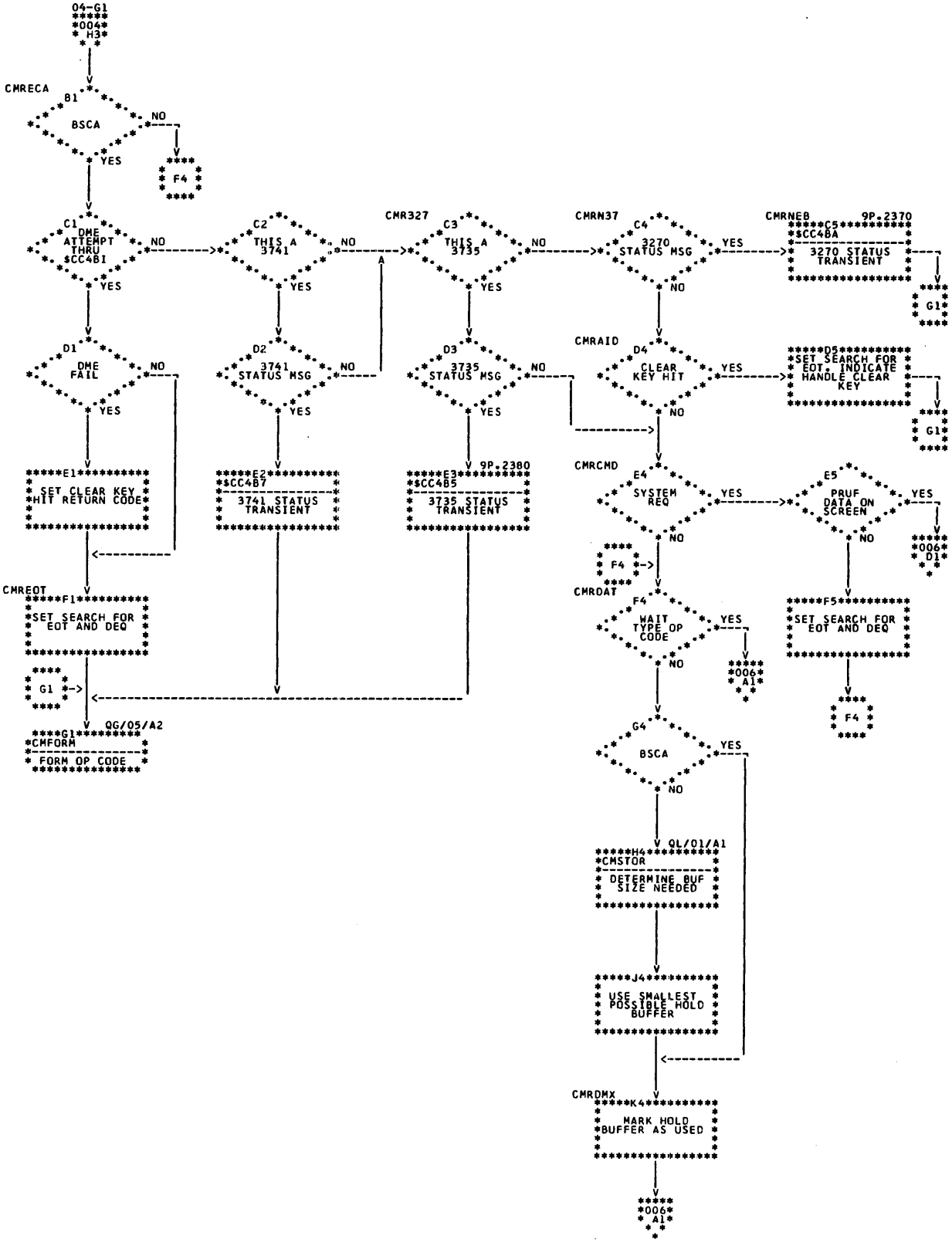


Chart QC (Part 5 of 8). CM Op End ($CC4CM)

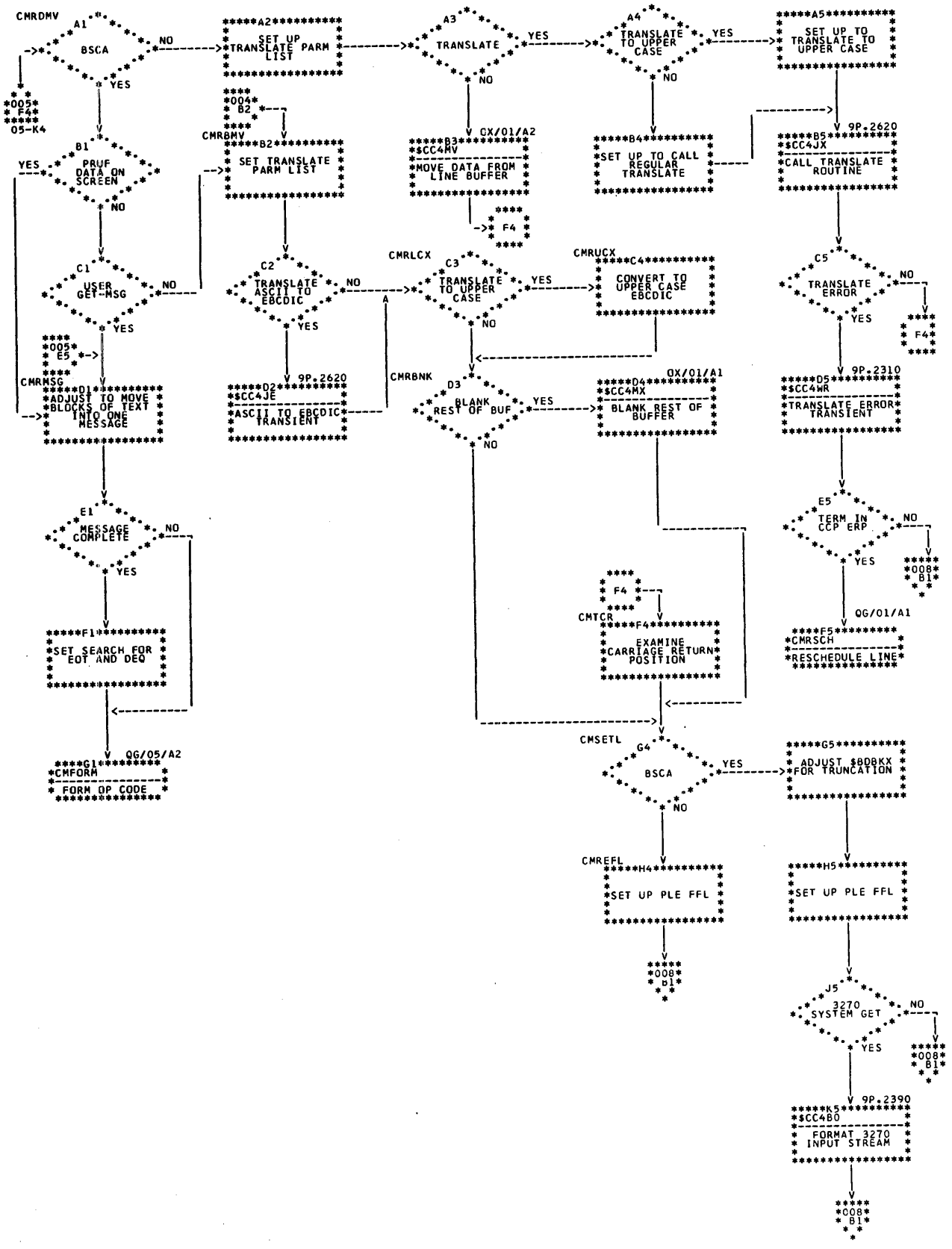


Chart QC (Part 6 of 8). CM Op End ($CC4CM)

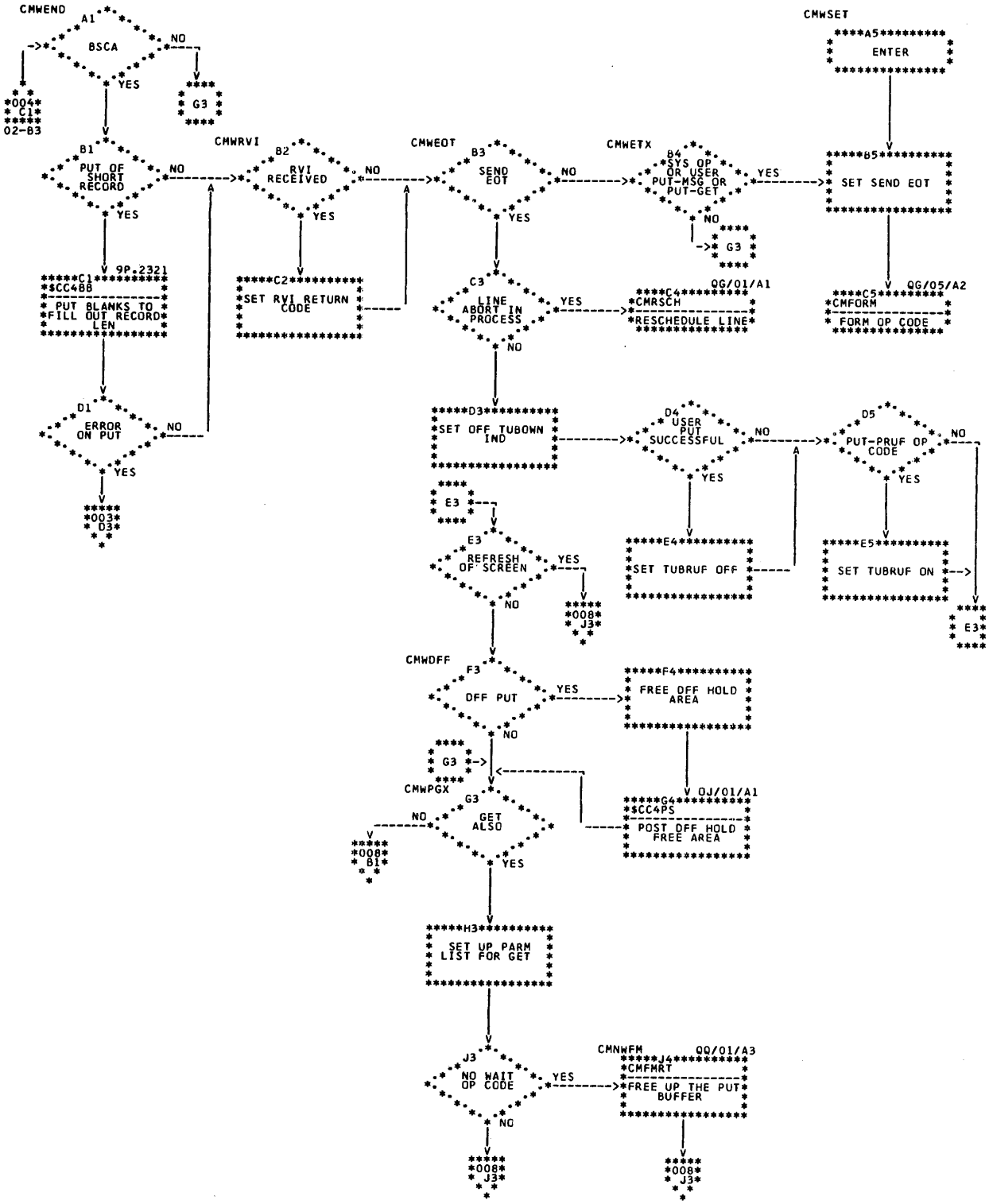


Chart QC (Part 7 of 8). CM Op End ($CC4CM)

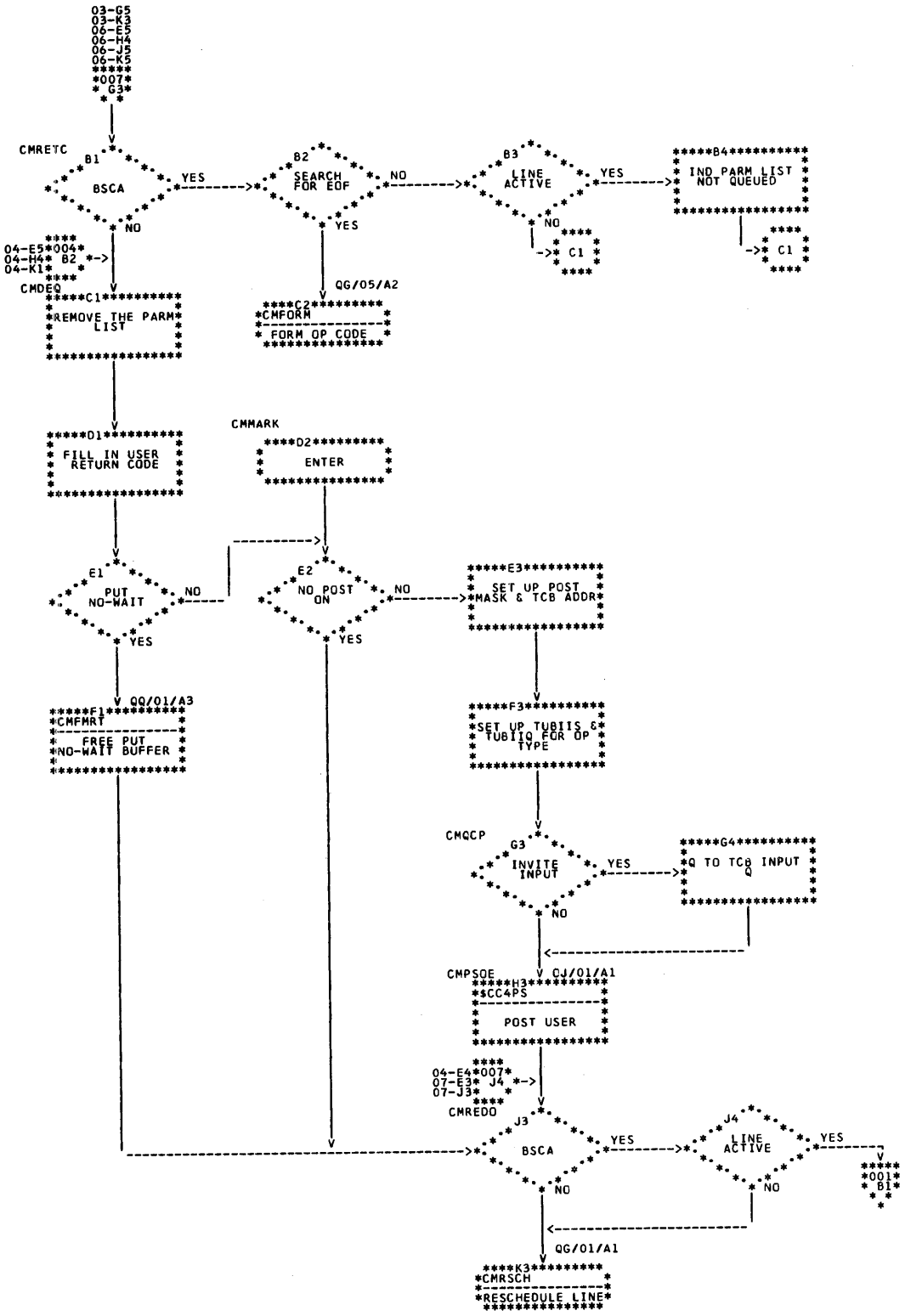


Chart QC (Part 8 of 8). CM Op End ($CC4CM)

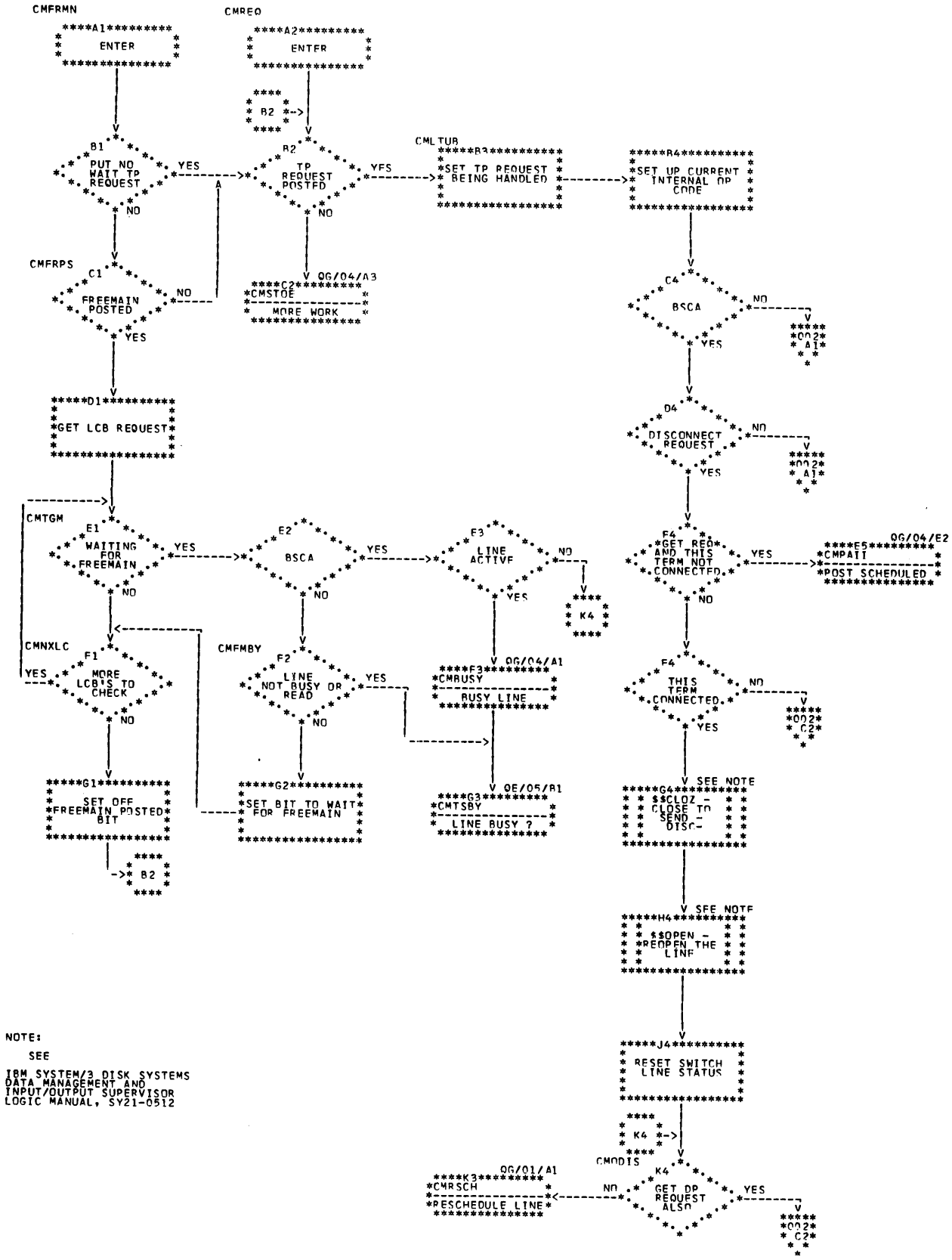


Chart QE (Part 1 of 5). CM Accept Request ($CC4CM)

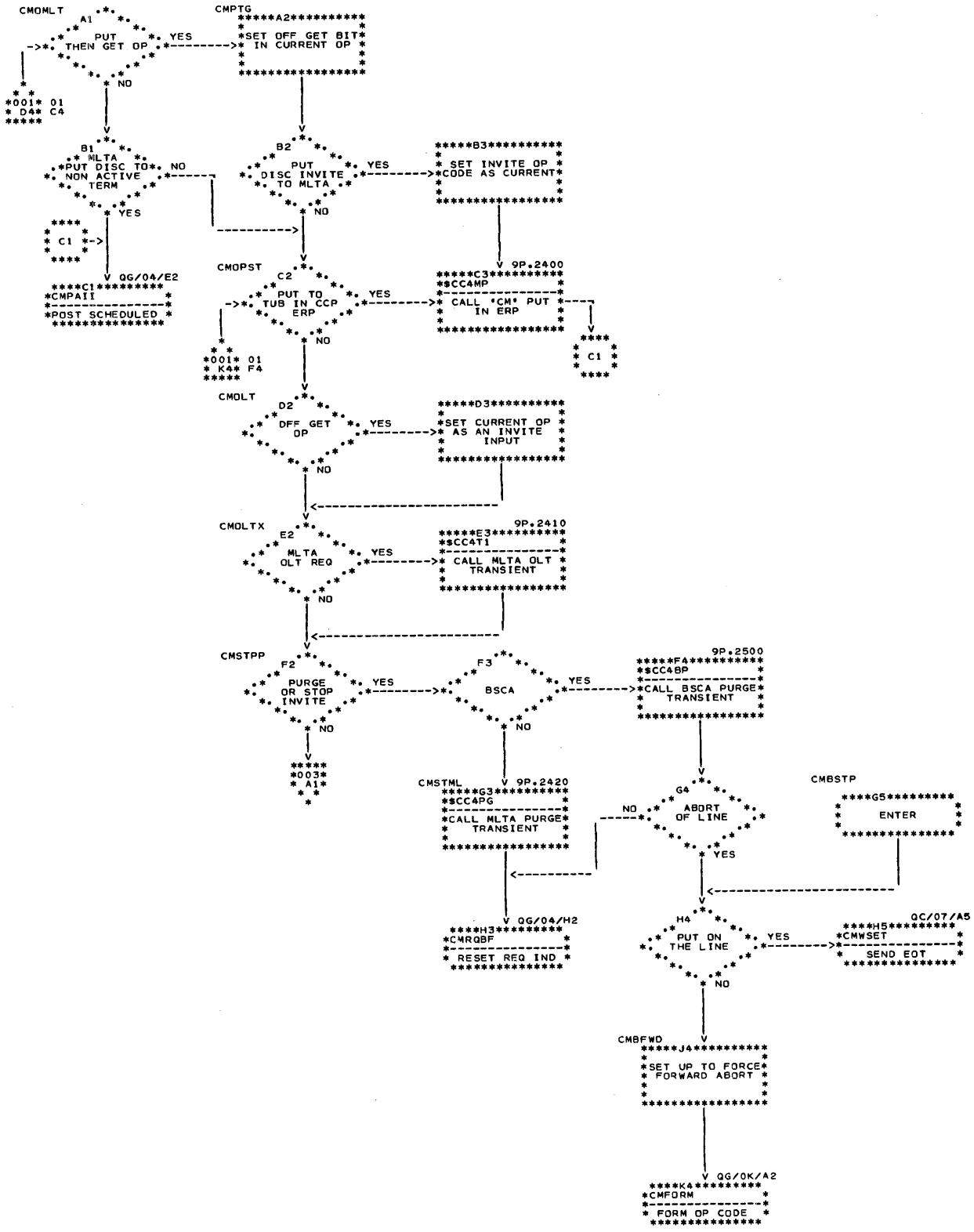


Chart QE (Part 2 of 5). CM Accept Request ($CC4CM)

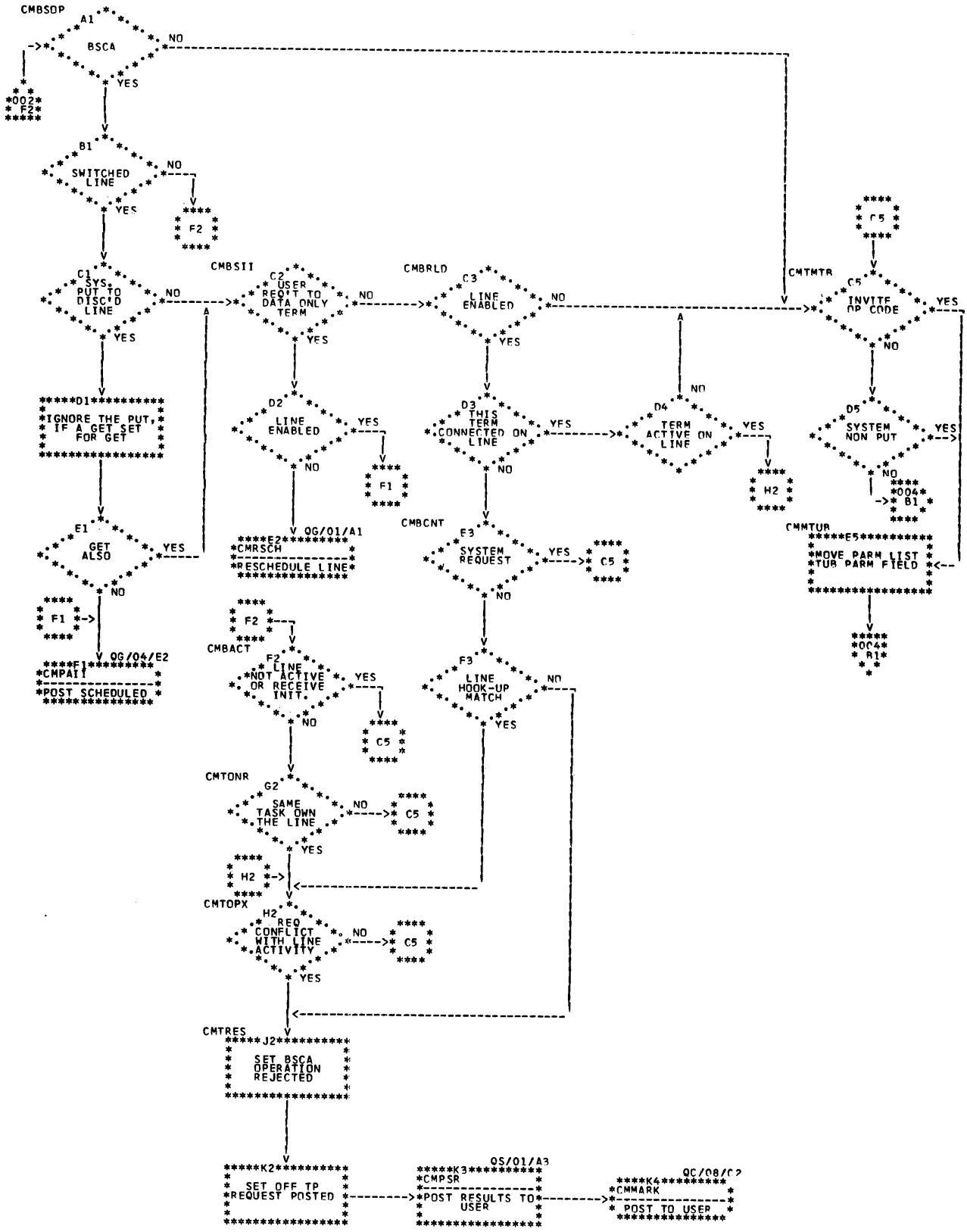


Chart QE (Part 3 of 5). CM Accept Request ($CC4CM)

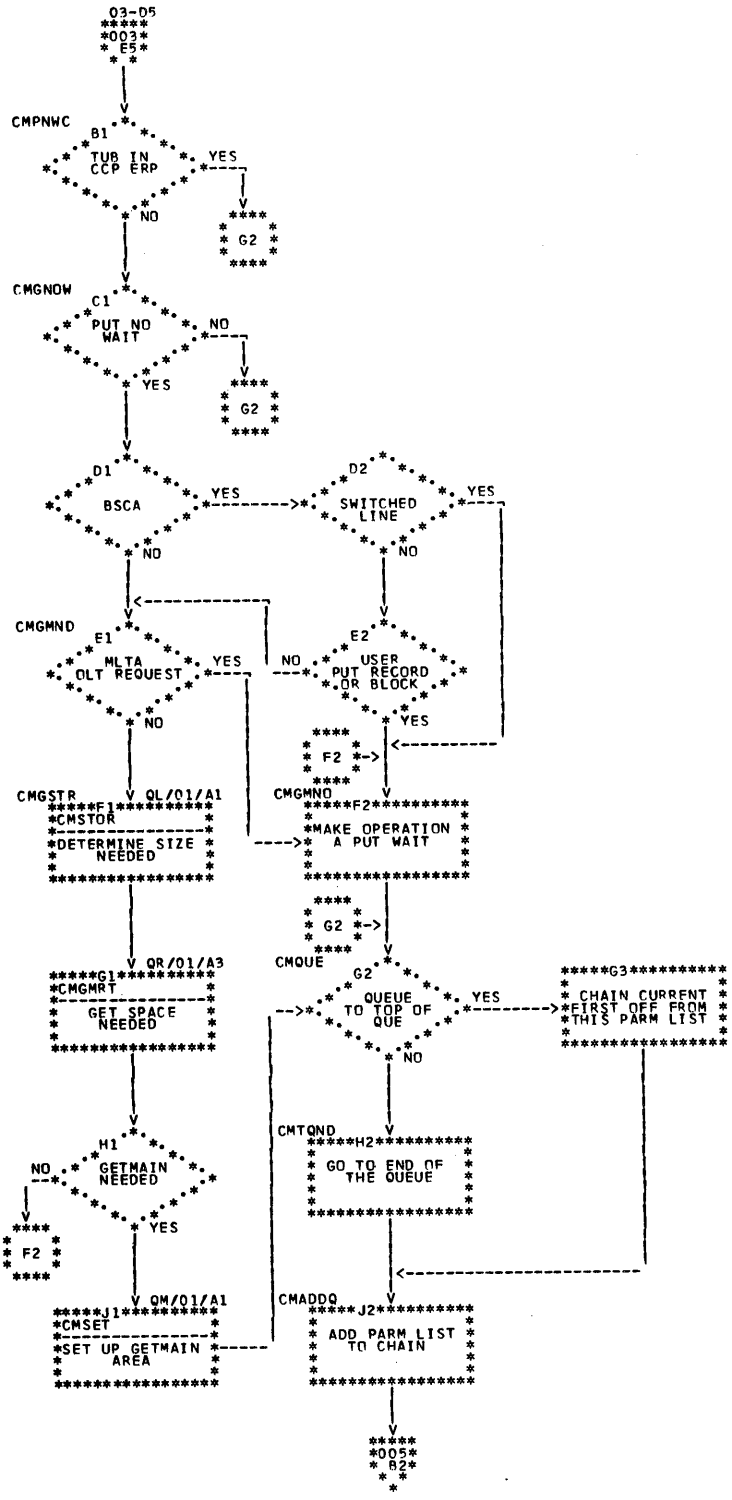


Chart QE (Part 4 of 5). CM Accept Request ($CC4CM)

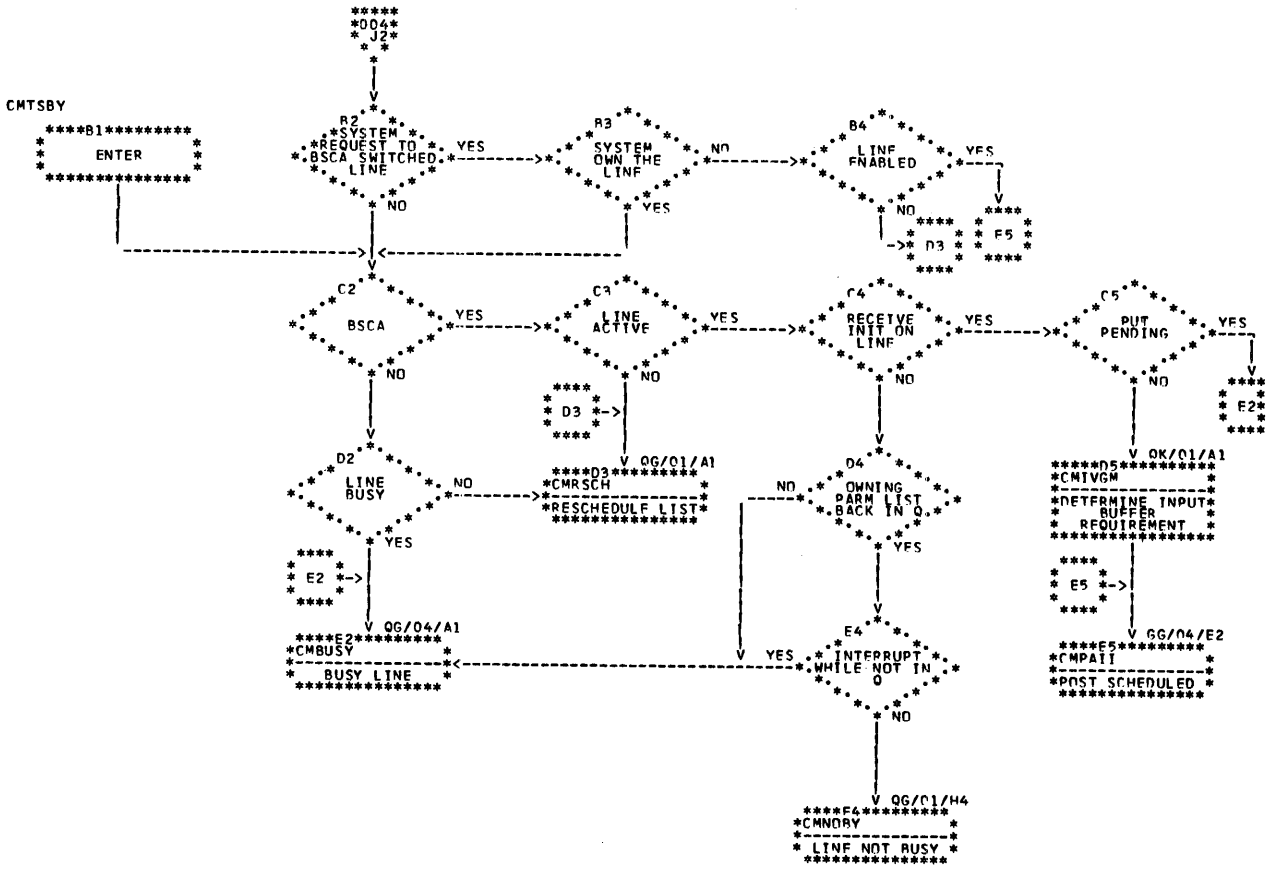


Chart QE (Part 5 of 5). CM Accept Request ($CC4CM)

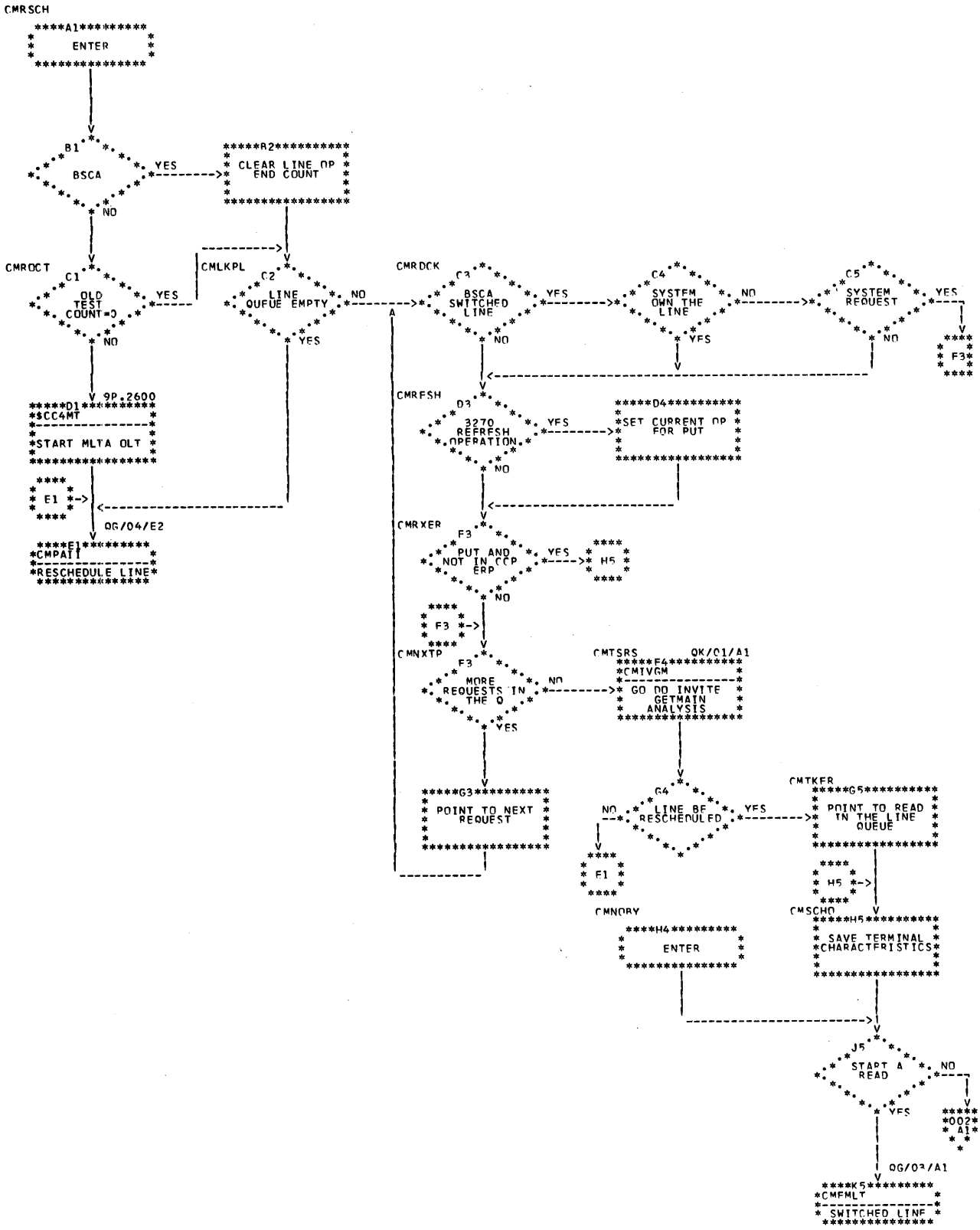


Chart QG (Part 1 of 6). CM Start Operation ($CC4CM)

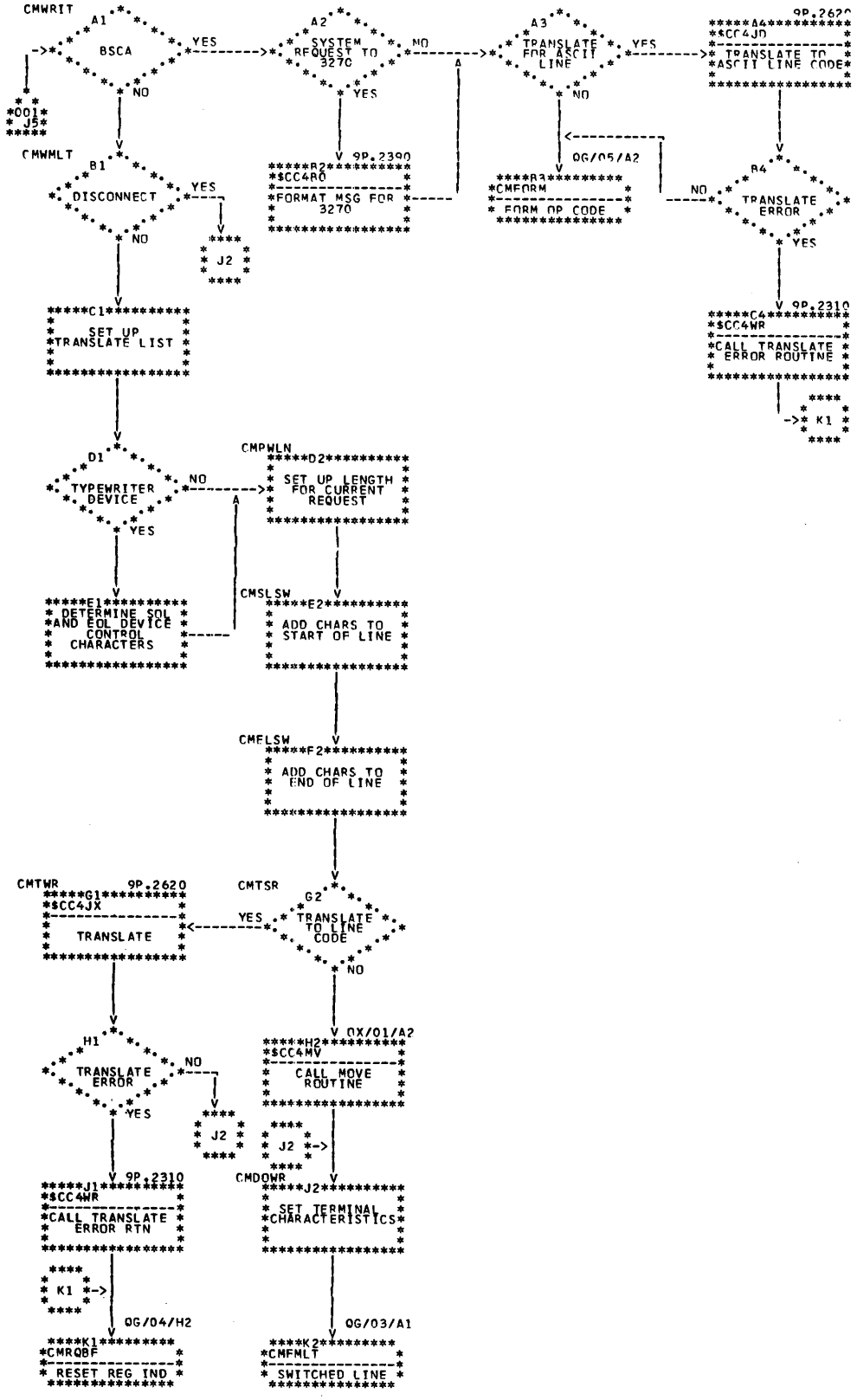


Chart QG (Part 2 of 6). CM Start Operation ($CC4CM)

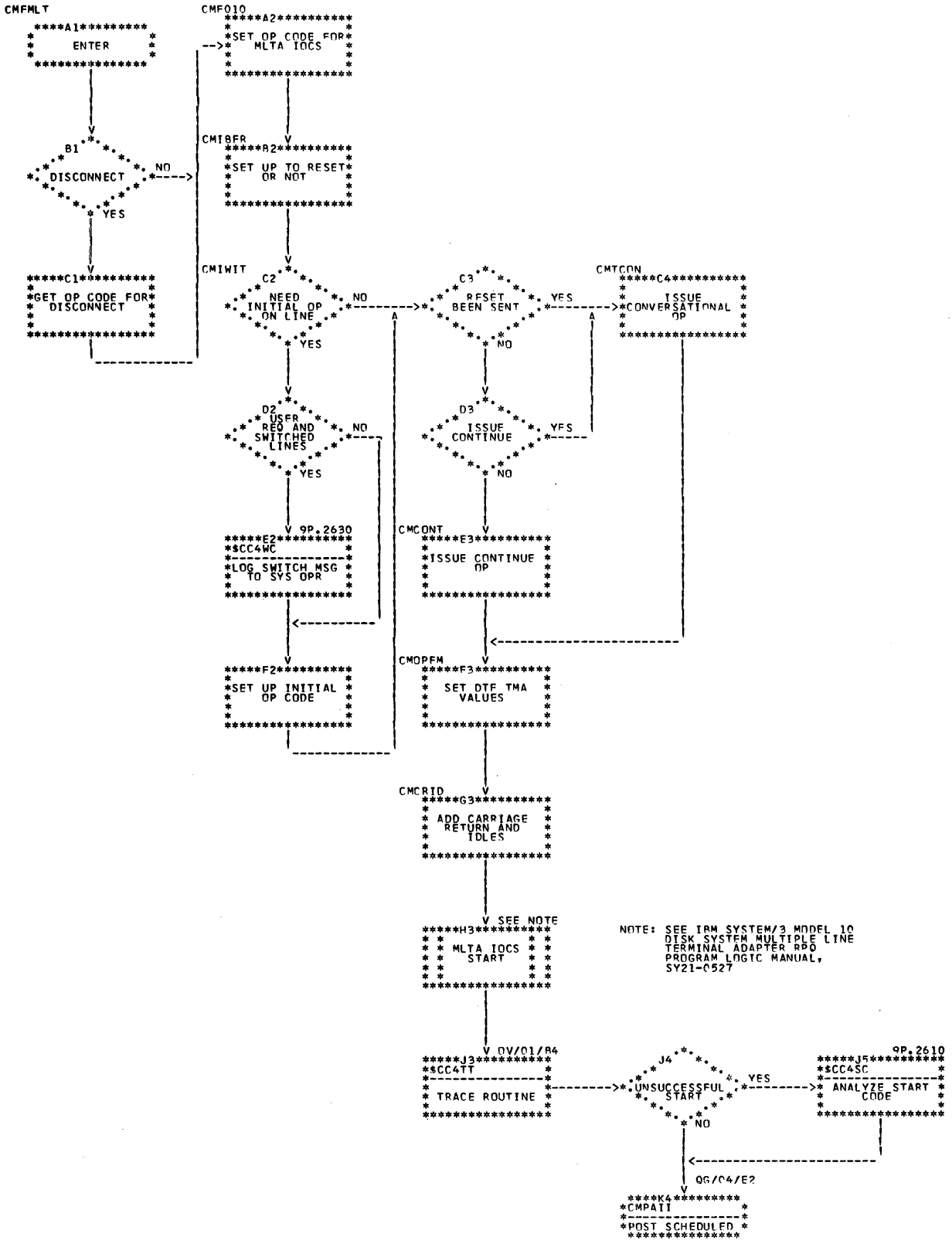


Chart QG (Part 3 of 6). CM Start Operation ($CC4CM)

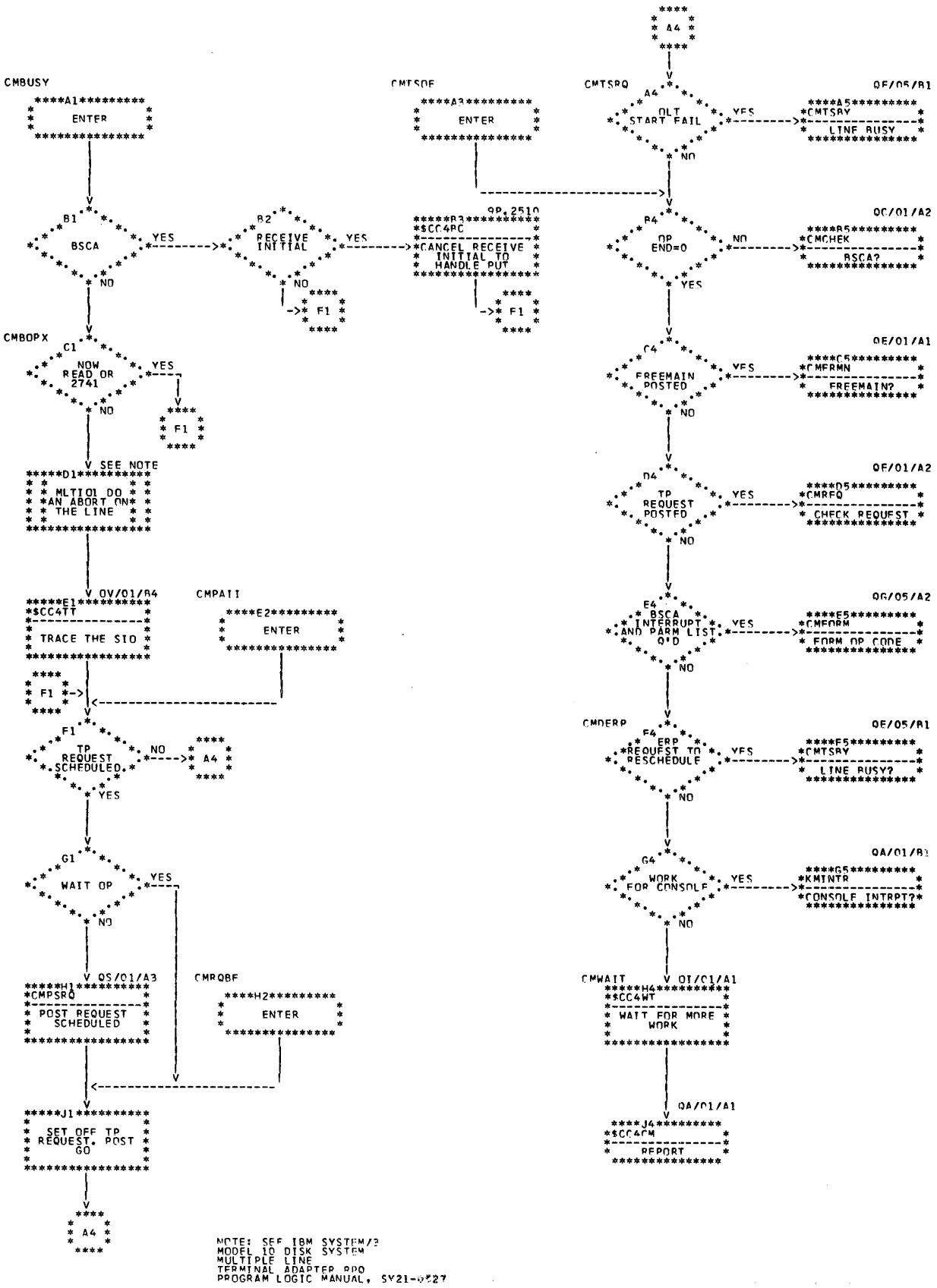


Chart QG (Part 4 of 6). CM Start Operation ($CC4CM)

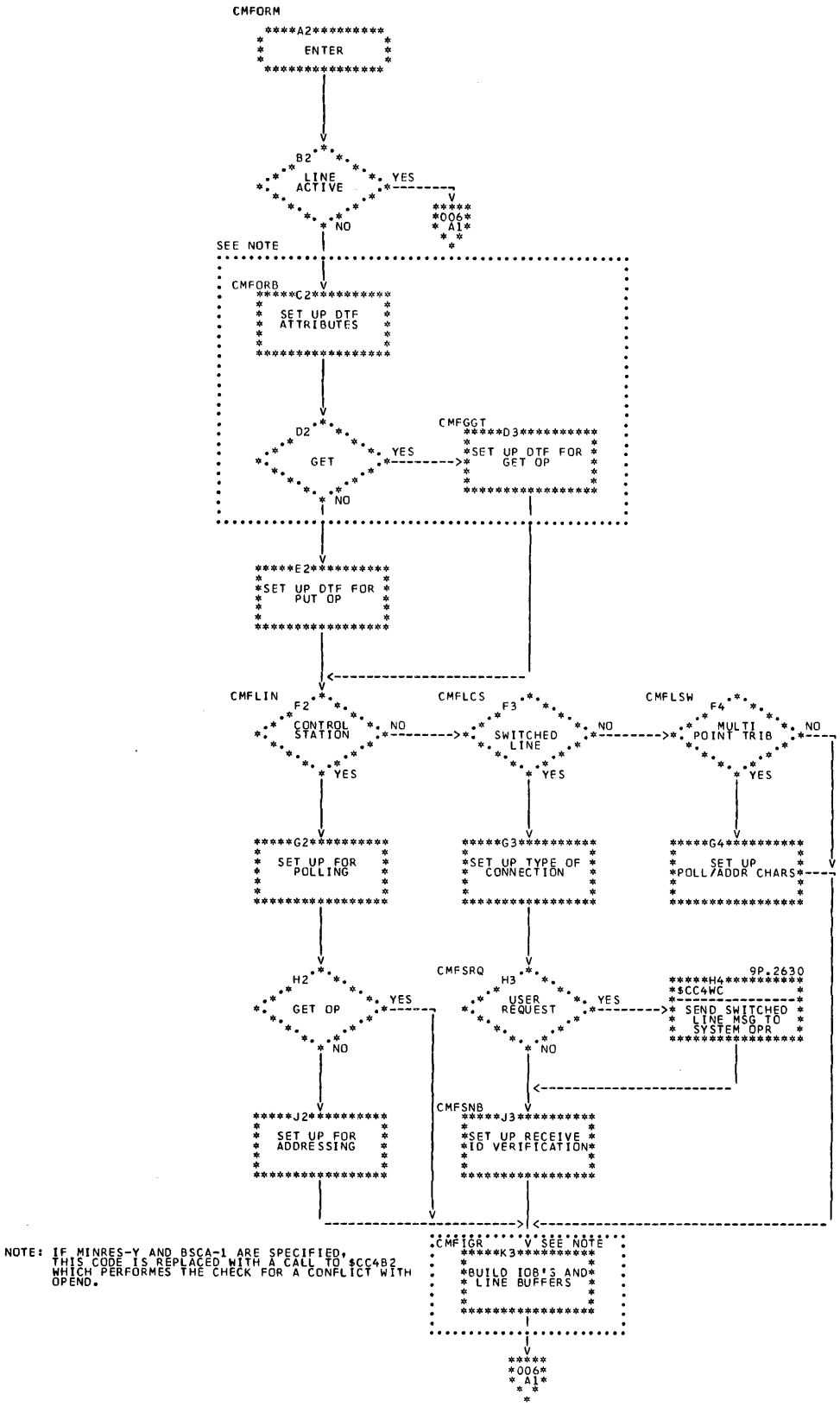


Chart QG (Part 5 of 6). CM Start Operation ($CC4CM)

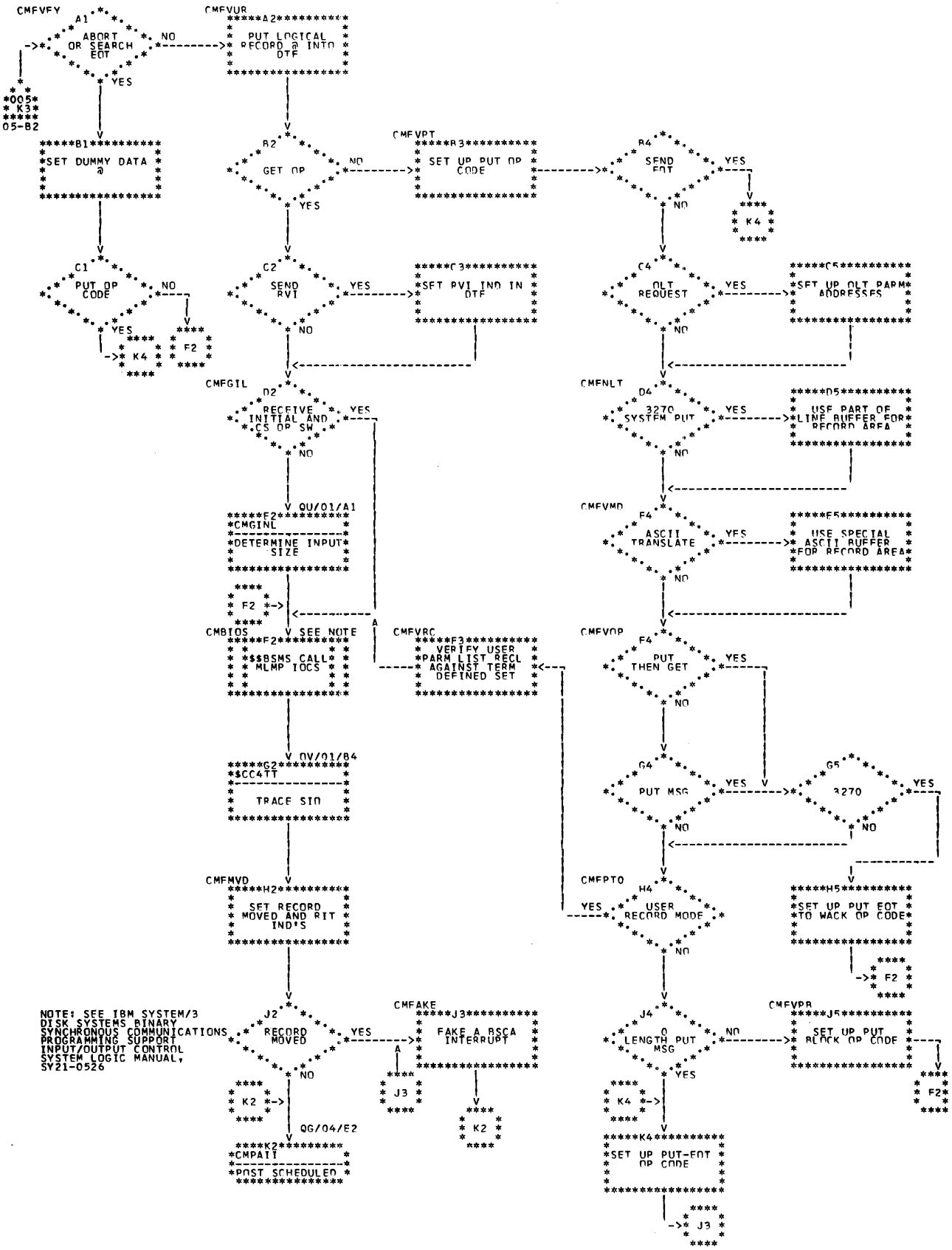


Chart QG (Part 6 of 6). CM Start Operation ($CC4CM)

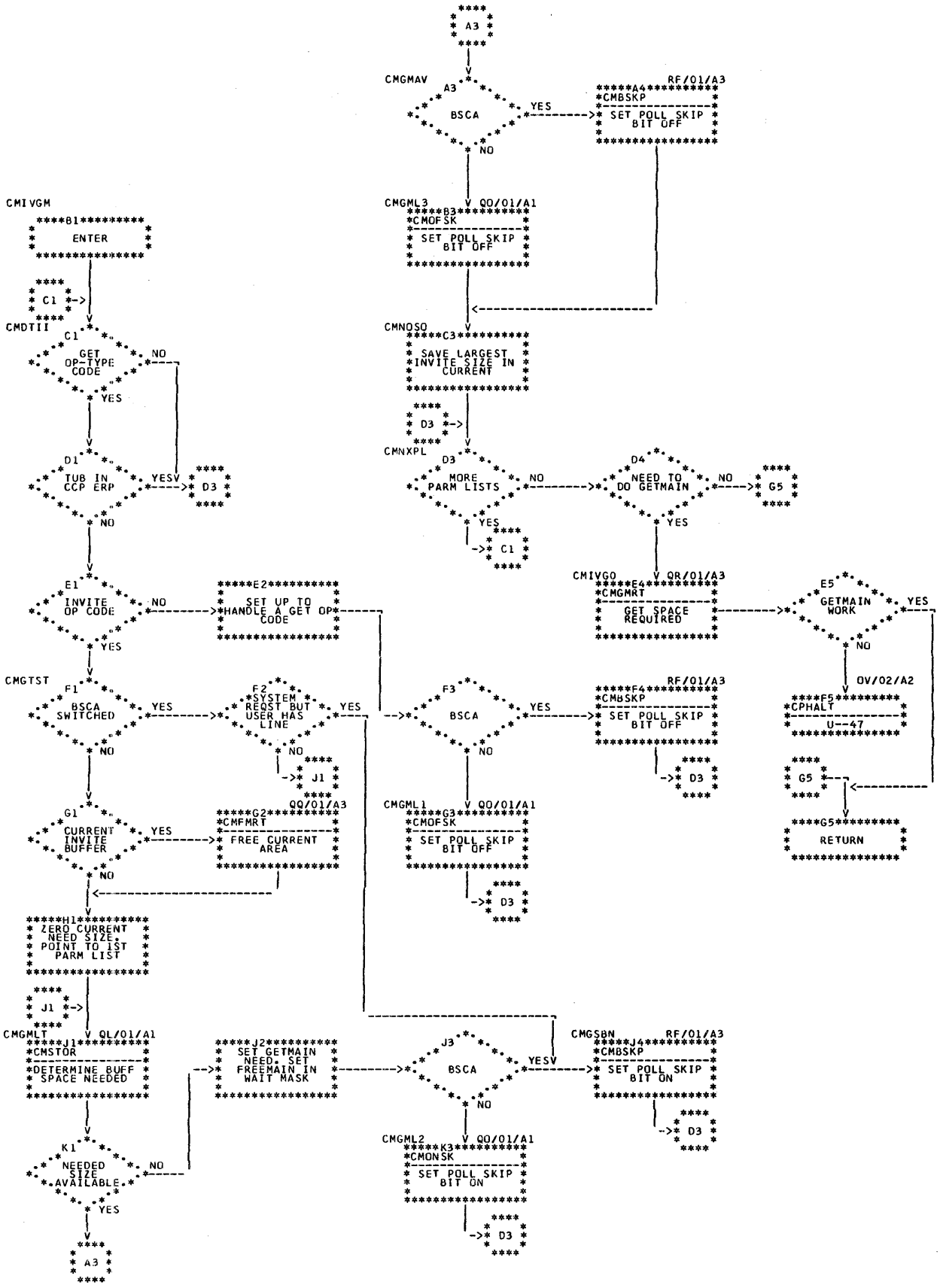


Chart QK. CM Invite Buffer Analysis Routine (CMIVGM)

$CC4V1

This module is the resident code of the CCP for a Model 4 3270 only CCP system. See the following modules for a description of the functions:

$CC4II/$CC4IS	TP I/O interface mainline
$CC4CM	Communications manager
$CC4IC	Console interrupt intercept
$CC4PI	Transient area handler
$CC4AM	Allocation resident routine
$CC4TI	User task termination routine
$CC4CP	Command processor resident routine
$CC4OC	Open/close/allocate interface
$CC4MX	Generalized move routine
$CC4MS	Getmain/freemain interface
$CC4IB	BSCA interrupt appendage
$CC4BT	BSCA trace interface
$CC4BN	BSCA first level interrupt handler
$CC4M1	3270 only MLMP IOS
$$BSAT	BSCA line #2 work area
$$BSLG	BSCA error log
$$BSMA	BSCA line init
$$BSMB	BSCA polling routine
$$BSMC	BSCA resident close routine
$$BSMF	BSCA log routine
$$BSID	DA polling routine

$CC4V2

This module is the resident code of the full BSCA CCP system for the Model 4. See the following modules for a description of the functions:

$CC4II/$CC4IS	TP I/O interface mainline
$CC4CM	Communications manager
$CC4IC	Console interrupt intercept
$CC4PI	Transient area handler
$CC4AM	Allocation resident routine
$CC4TI	User task termination routine
$CC4CP	Command processor resident routine
$CC4OC	Open/close/allocate interface
$CC4MX	Generalized move routine
$CC4MS	Getmain/freemain interface
$CC4IB	BSCA interrupt appendage
$CC4BT	BSCA trace interface
$CC4BN	BSCA first level interrupt handler
$$BSMS	MLMP IOS
$$BSAT	BSCA line #2 work area
$$BSLG	BSCA error log
$$BSMA	BSCA line init
$$BSMB	BSCA polling routine
$$BSMC	BSCA resident close routine
$$BSMF	BSCA log routine
$$BSID	DA polling routine

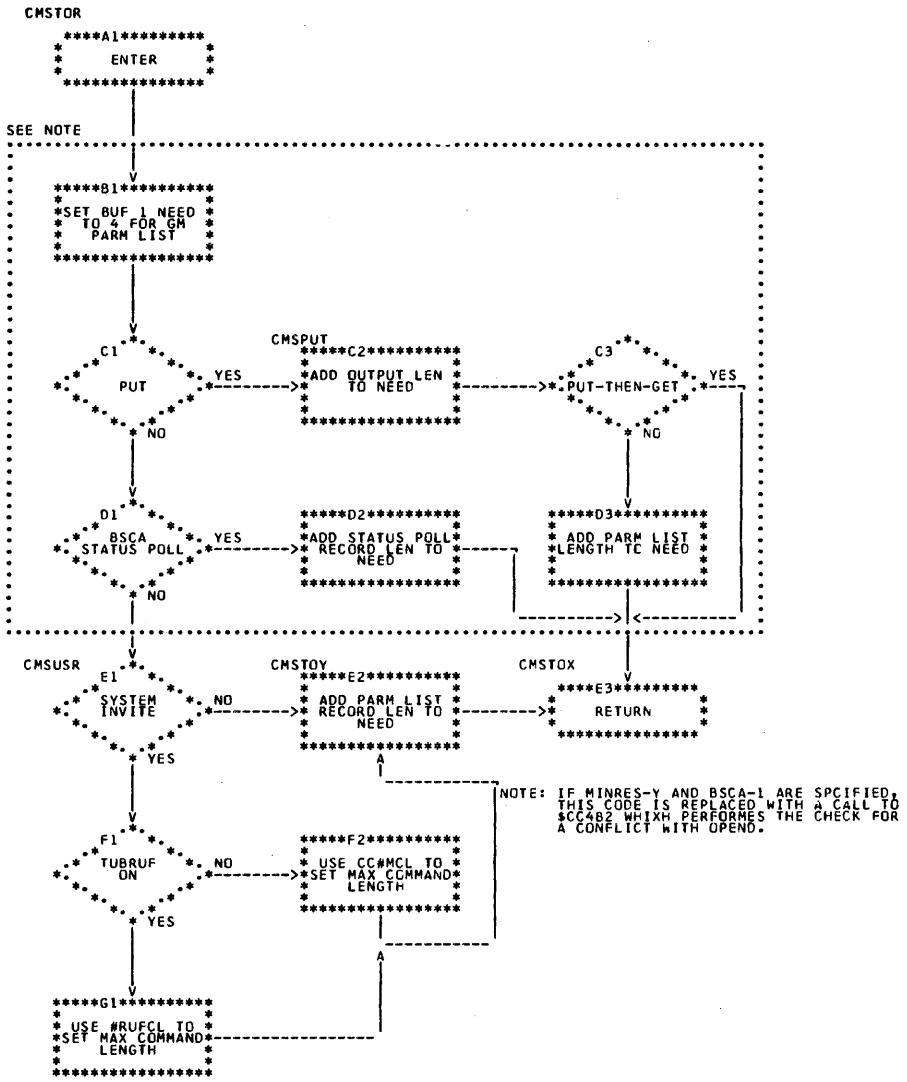


Chart QL. CM Getmain Size Determination Routine (CMSTOR)

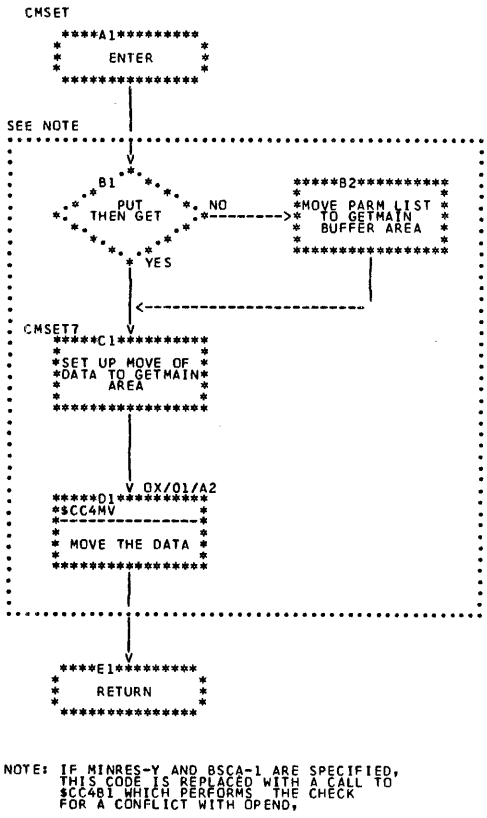


Chart QM. CM Format Put-No-Wait Area Routine (CMSET)

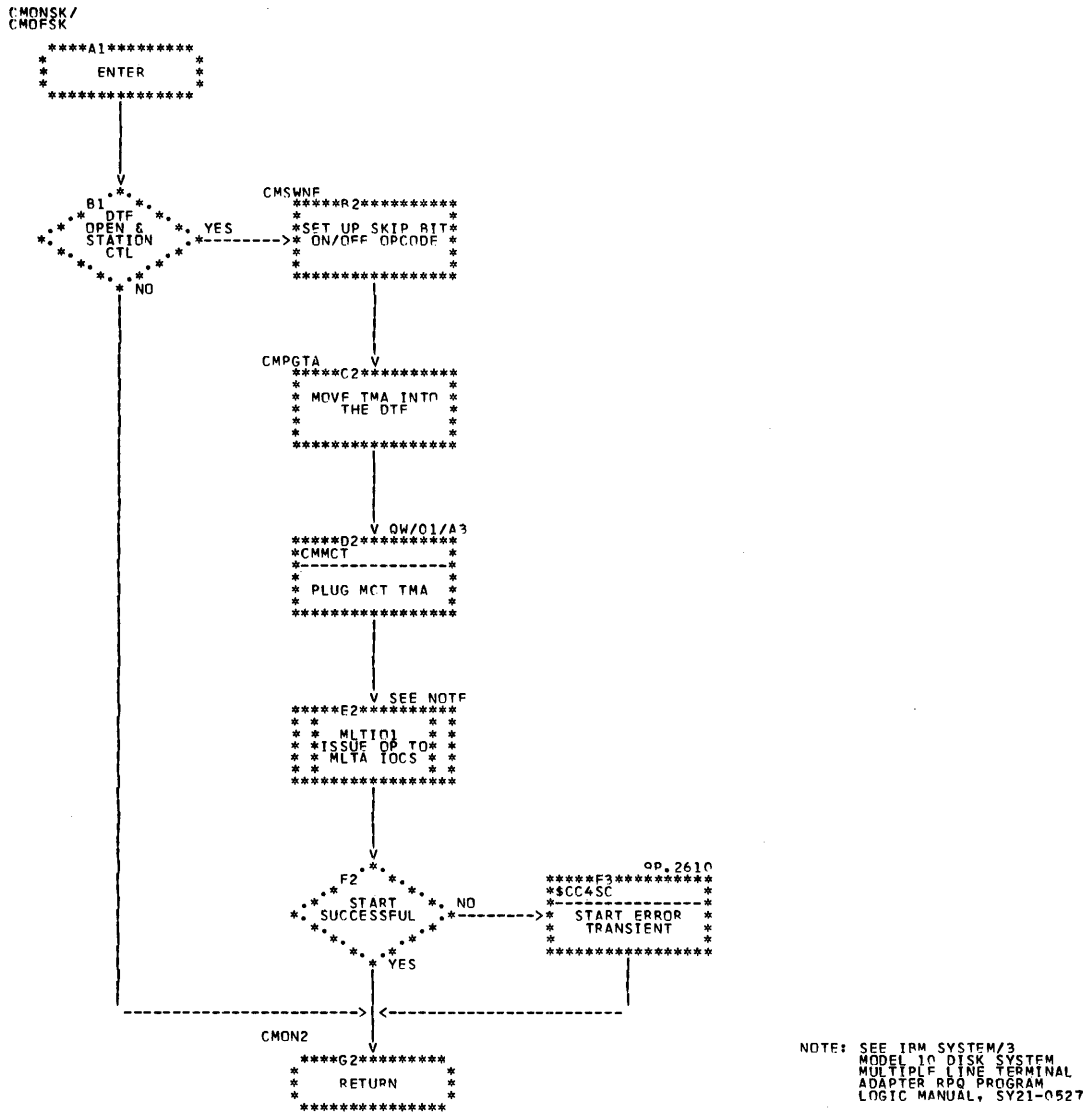


Chart QO. CM MLTA Set Polling Skip Bits Routine (CMONEK, CMOFSK)

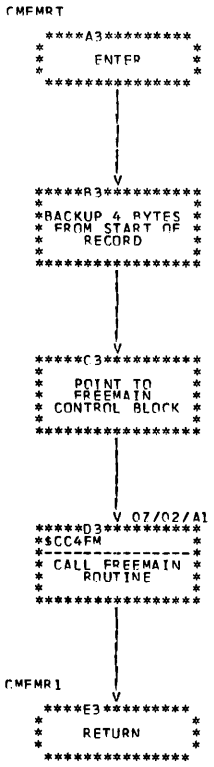


Chart QQ. CM Freemain Interface (CMFMRT)

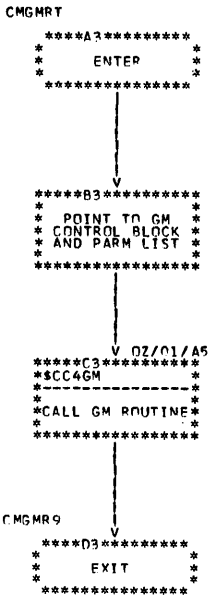


Chart QR. CM Getmain Interface (CMGMRT)

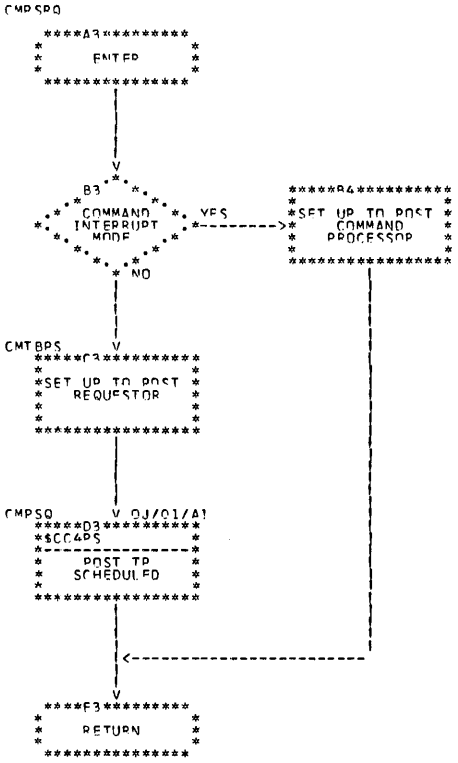


Chart QS. CM Post TP Scheduled (CMPSRQ)

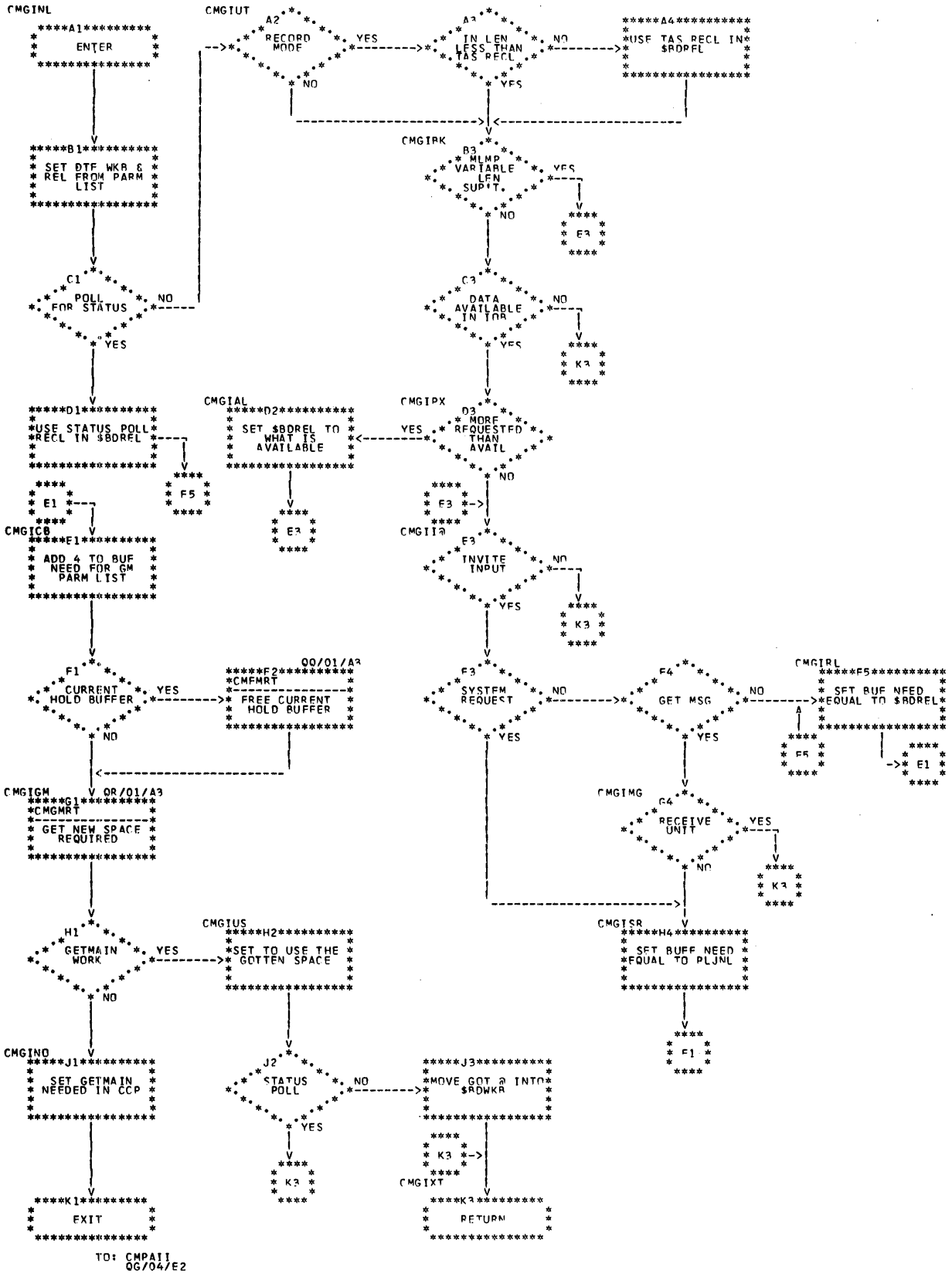


Chart QU. CM Operation Input Length Determination Routine (CMGINL)

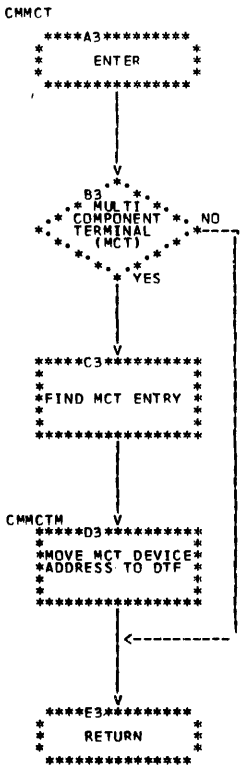
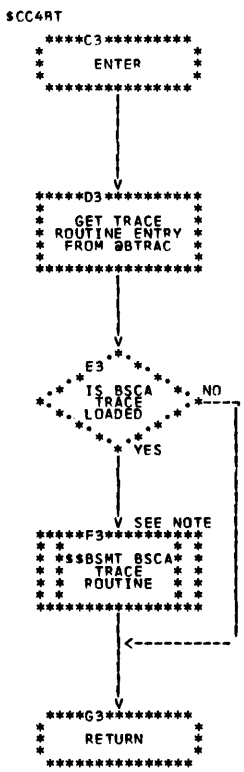


Chart QW. CM MLTA Multicomponent Terminal Routine (CMMCT)



NOTE: SEE IBM SYSTEM/3 DISK SYSTEMS
BINARY SYNCHRONOUS COMMUNICATIONS
PROGRAMMING SUPPORT
INPUT/OUTPUT CONTROL SYSTEM
LOGIC MANUAL, SY21-0526

Chart QZ. CM BSCA Trace Interface ($CC4BT)

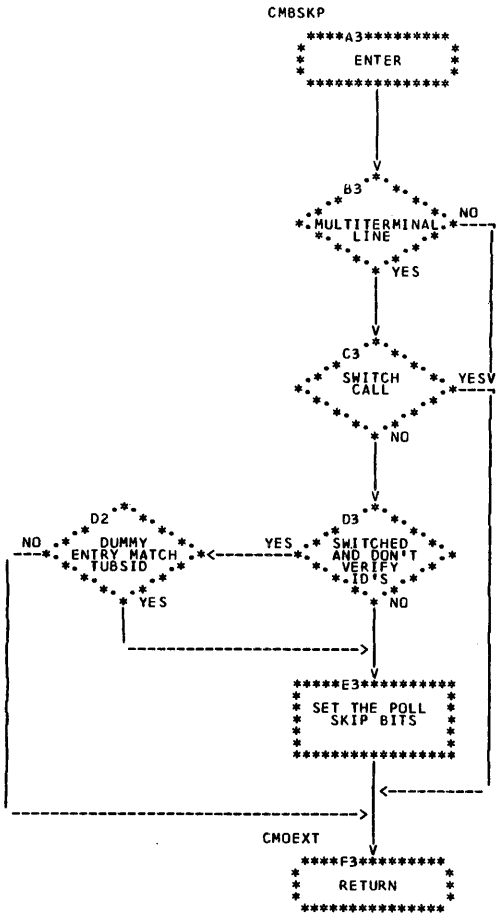


Chart RF. CM BSCA Set Polling Skip Bit Routine ($CMBSKP)

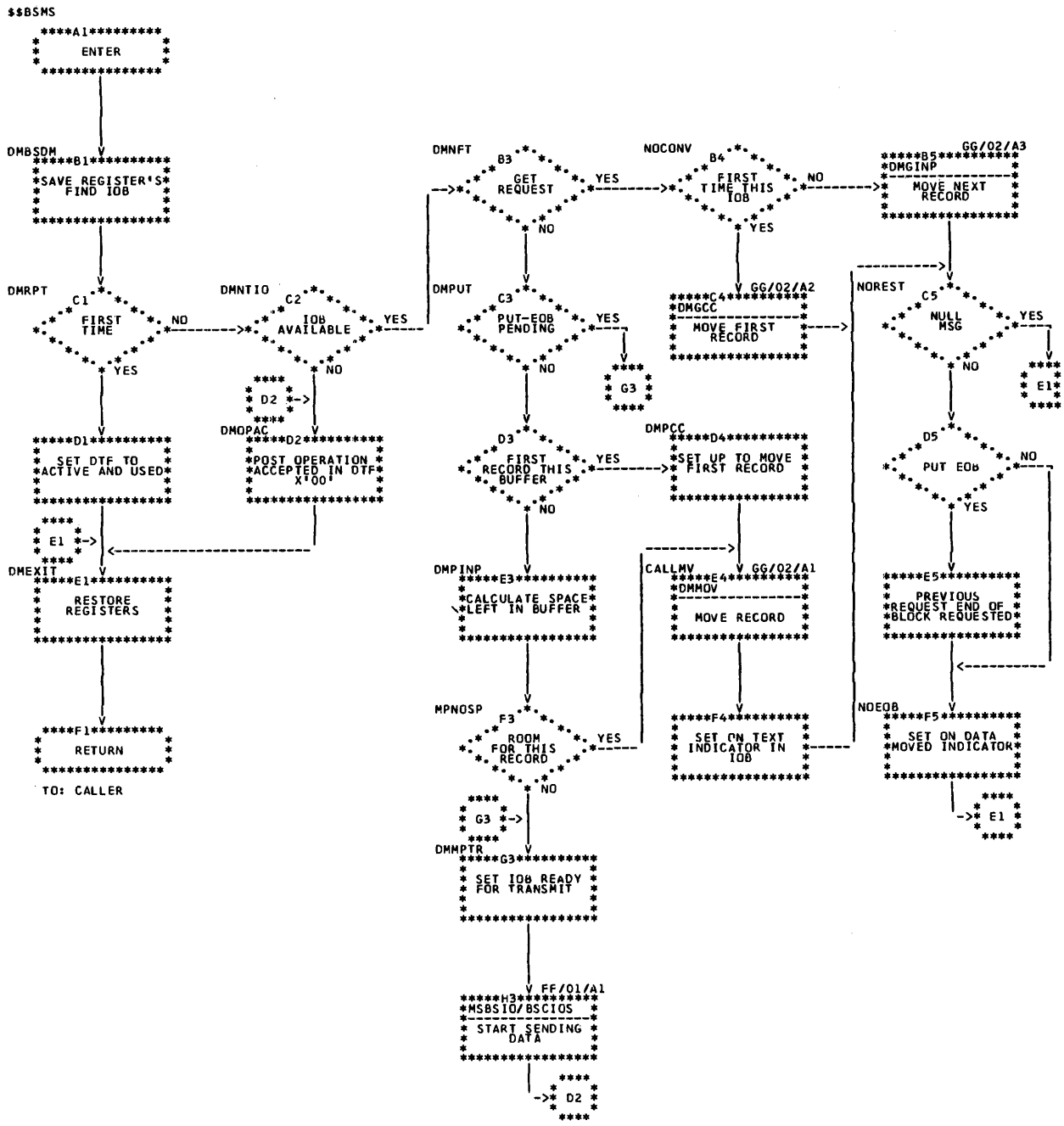


Chart RG (Part 1 of 2). Data Management ($CC4M1)

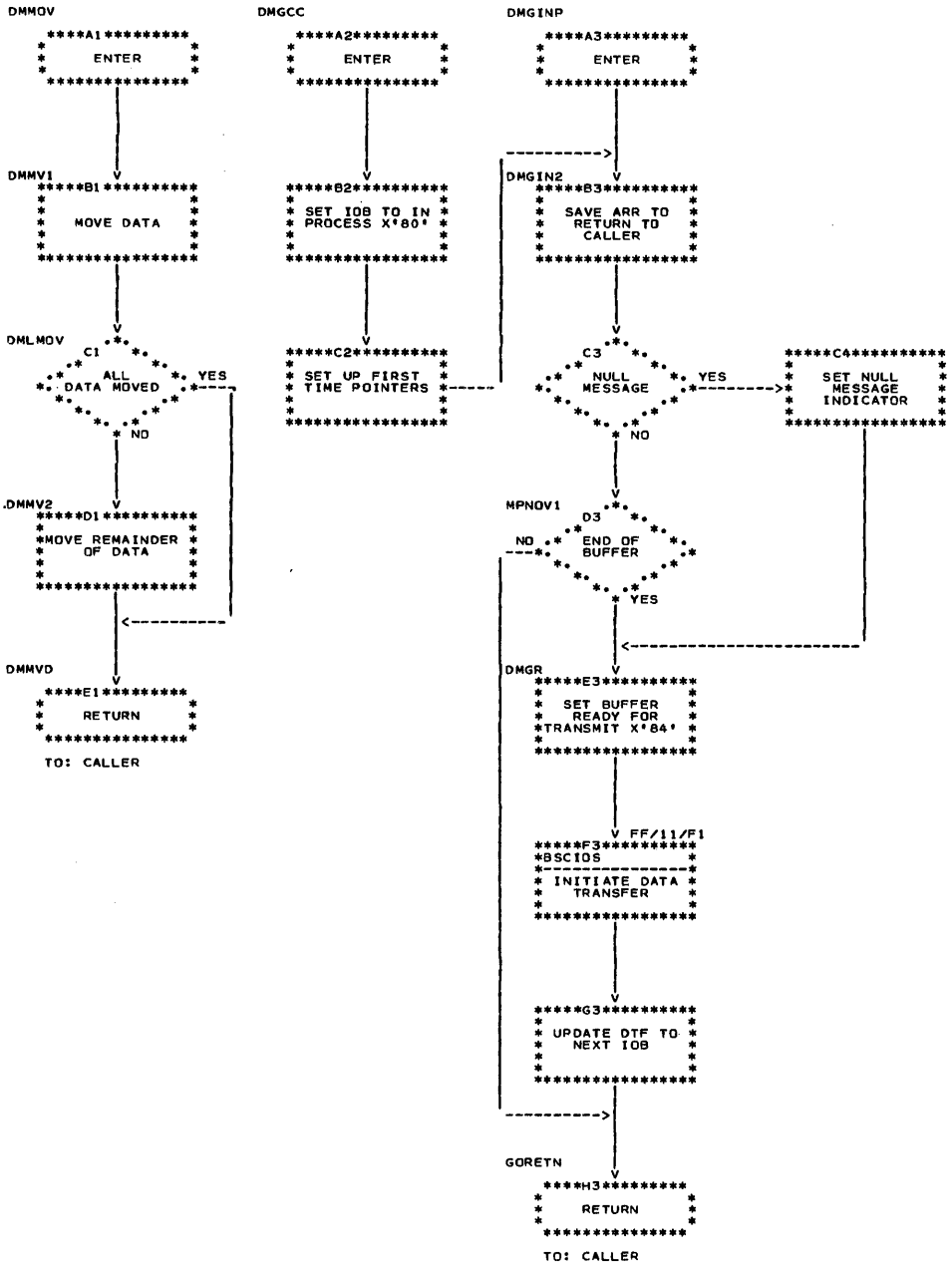


Chart RG (Part 2 of 2). Data Management ($CC4M1)

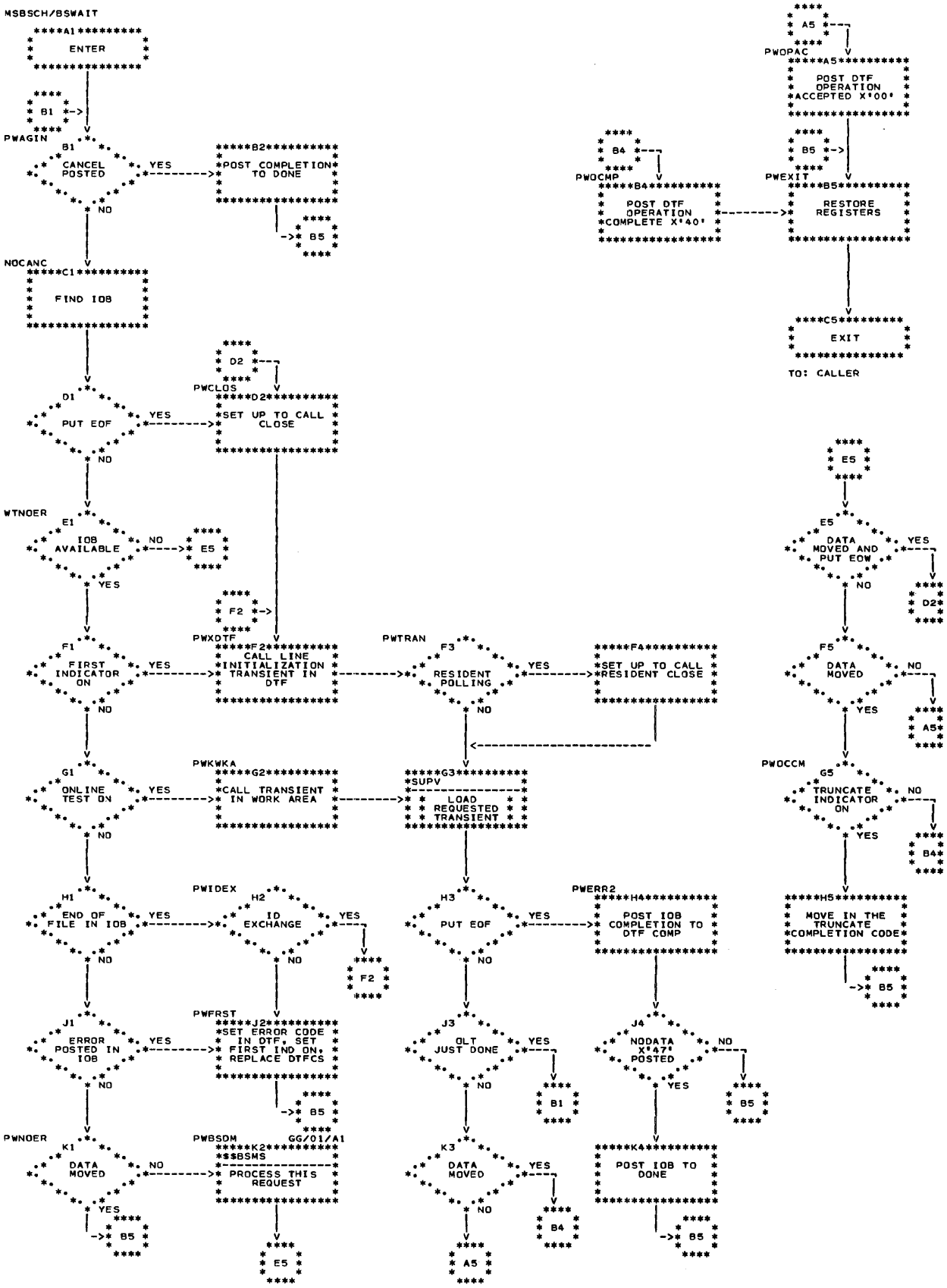


Chart RH. Wait ($CC4M1)

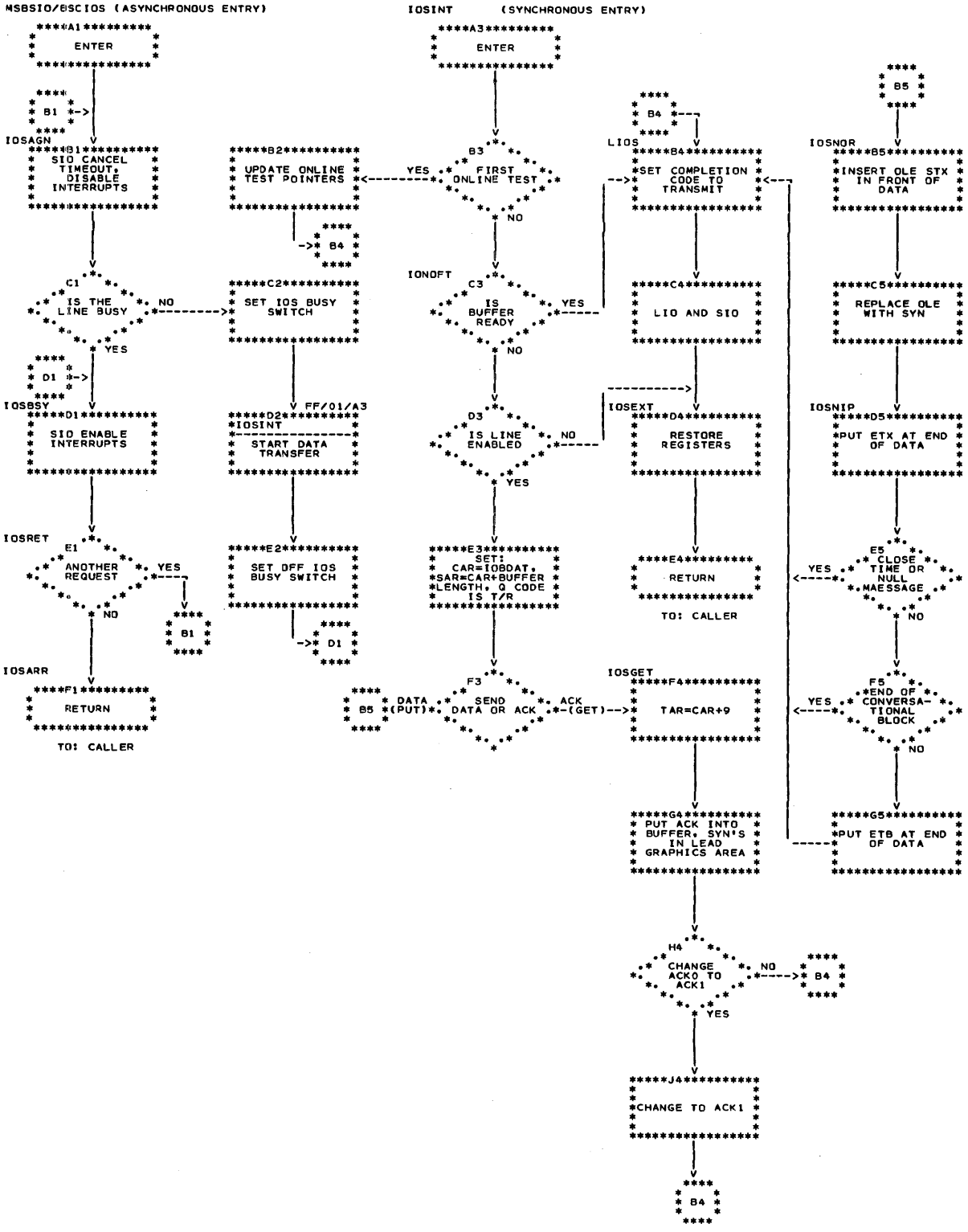


Chart RI. IOS ($CC4M1)

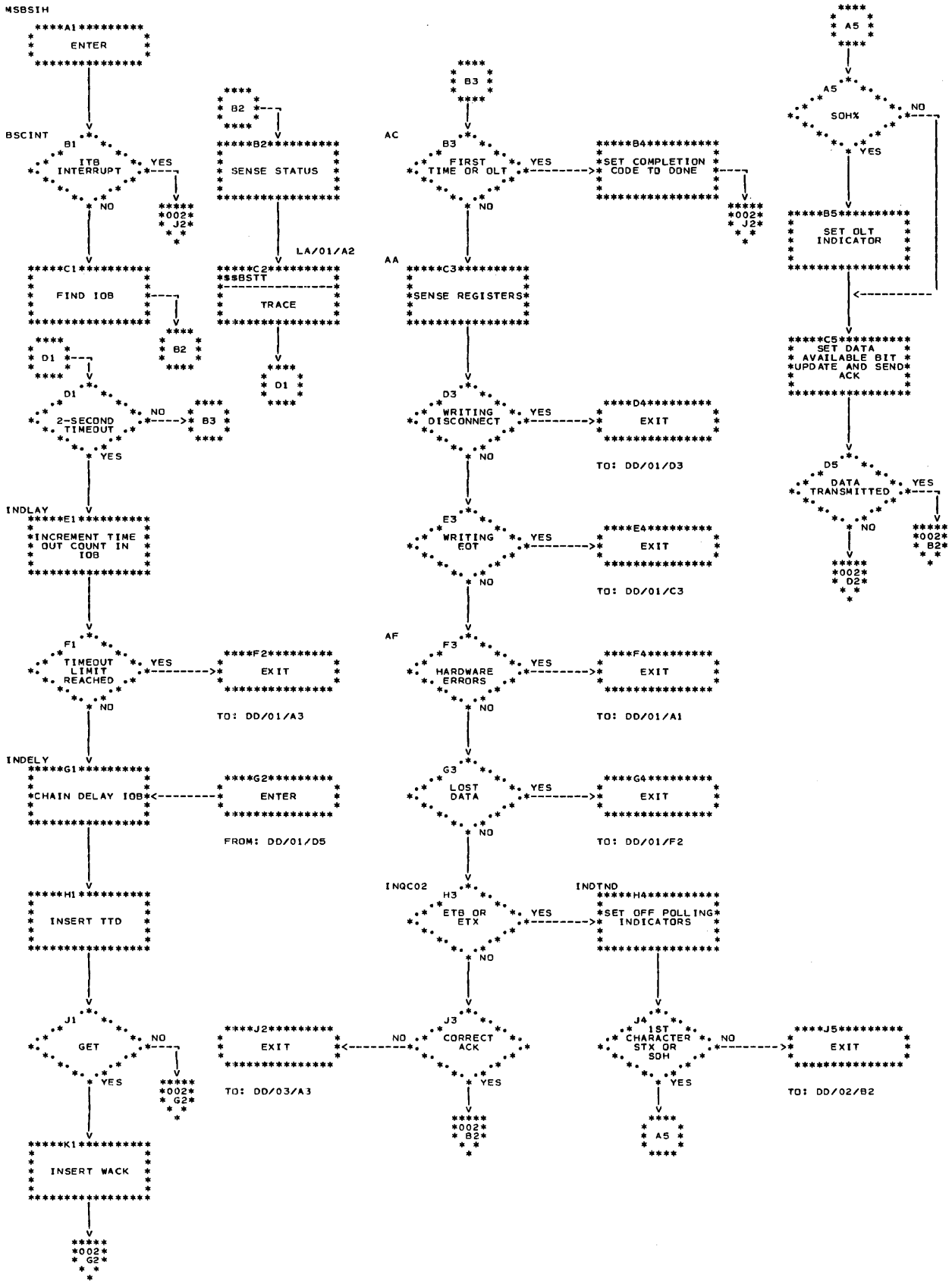


Chart RJ (Part 1 of 2). Interrupt ($CC4M1)

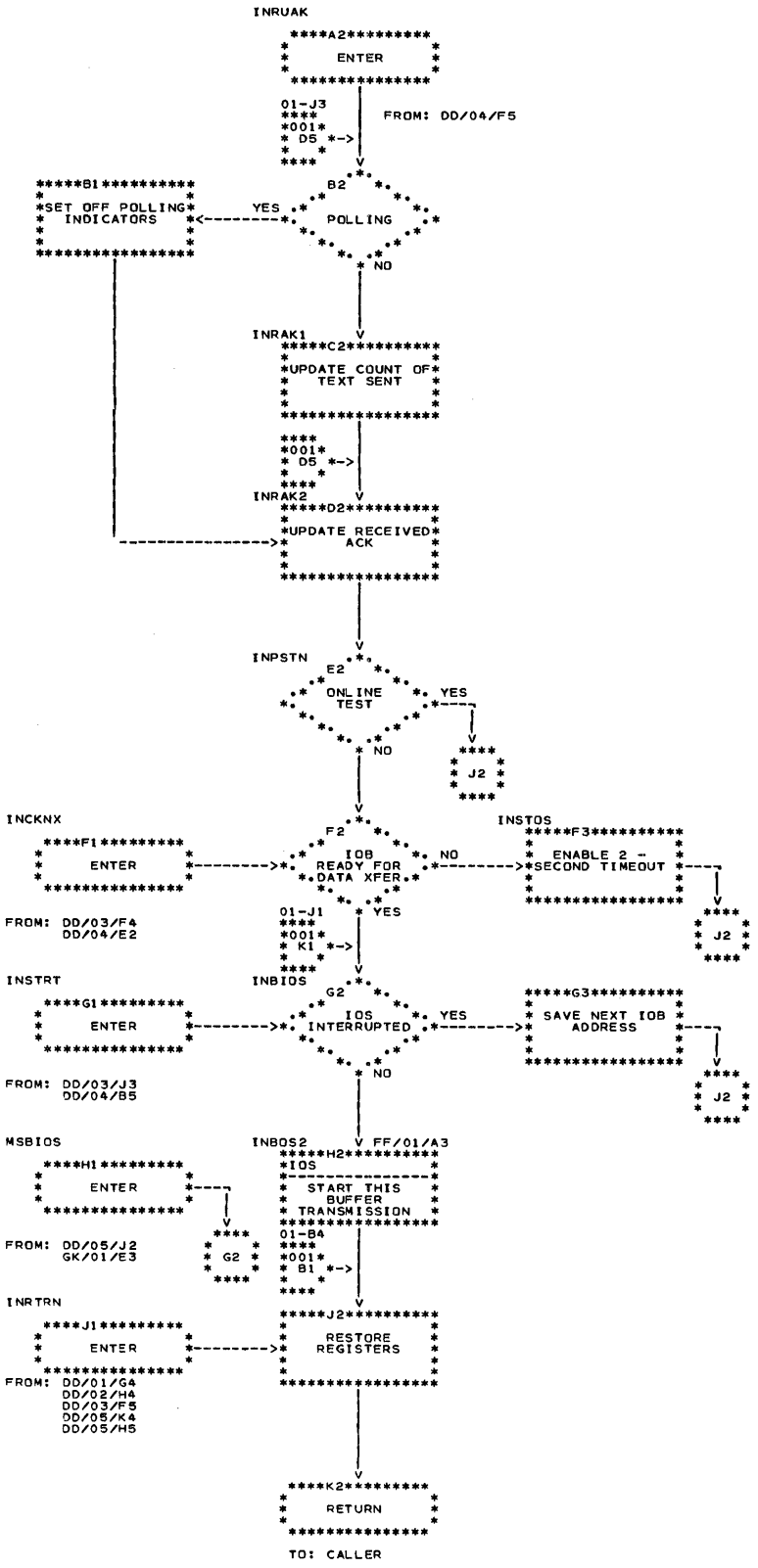


Chart RJ (Part 2 of 2). Interrupt ($CC4M1)

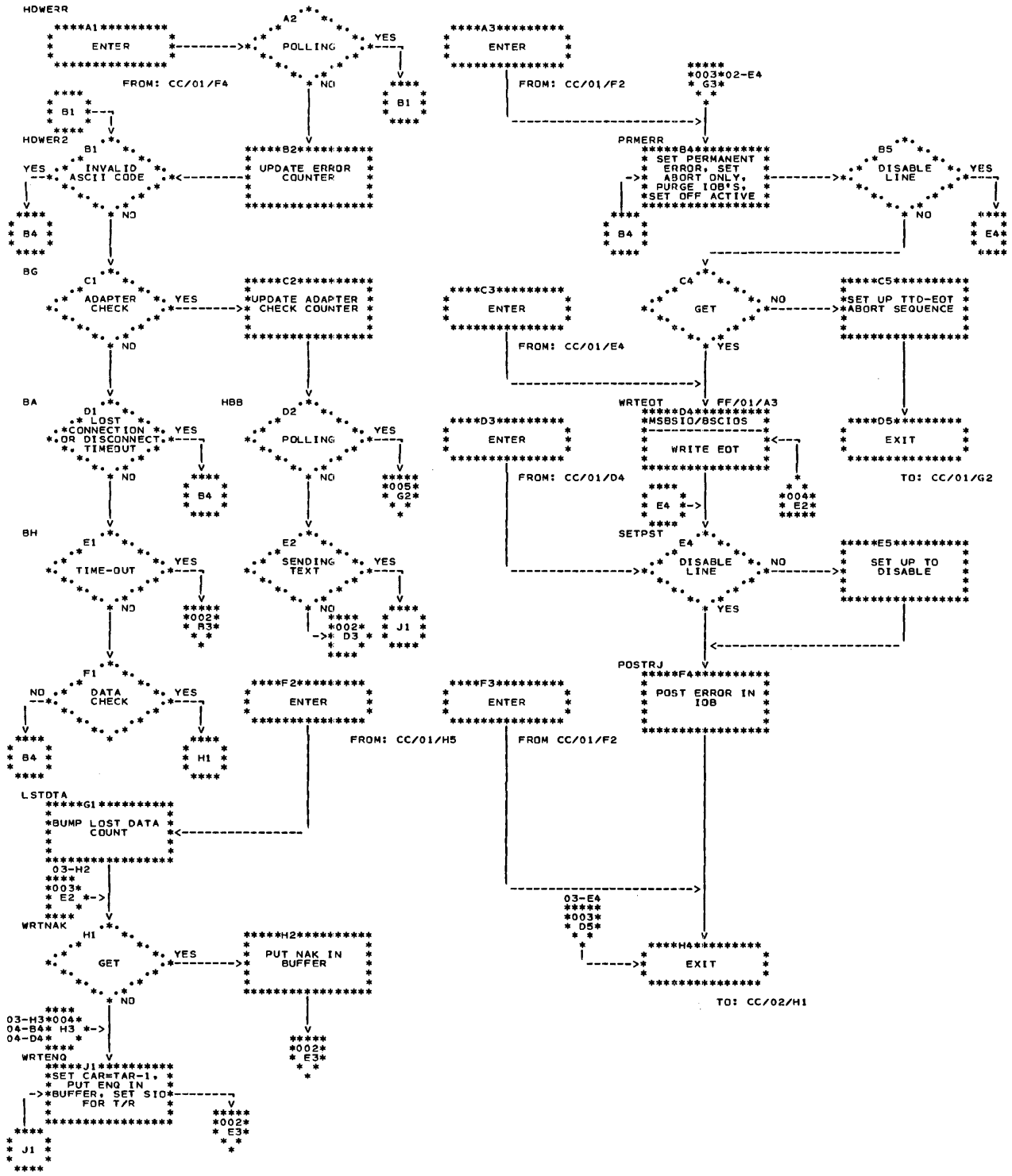


Chart RK (Part 1 of 5). Error Recovery Chart ($CC4M1)

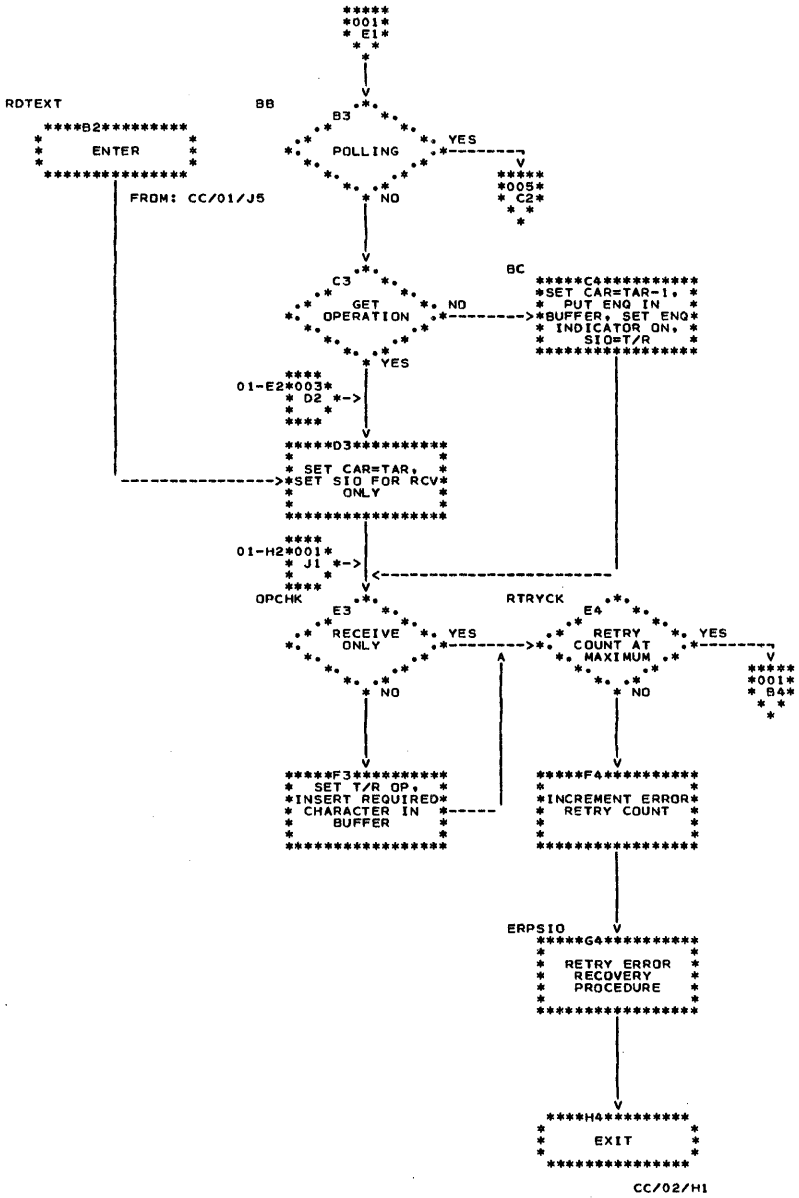


Chart RK (Part 2 of 5). Error Recovery Chart ($CC4M1)

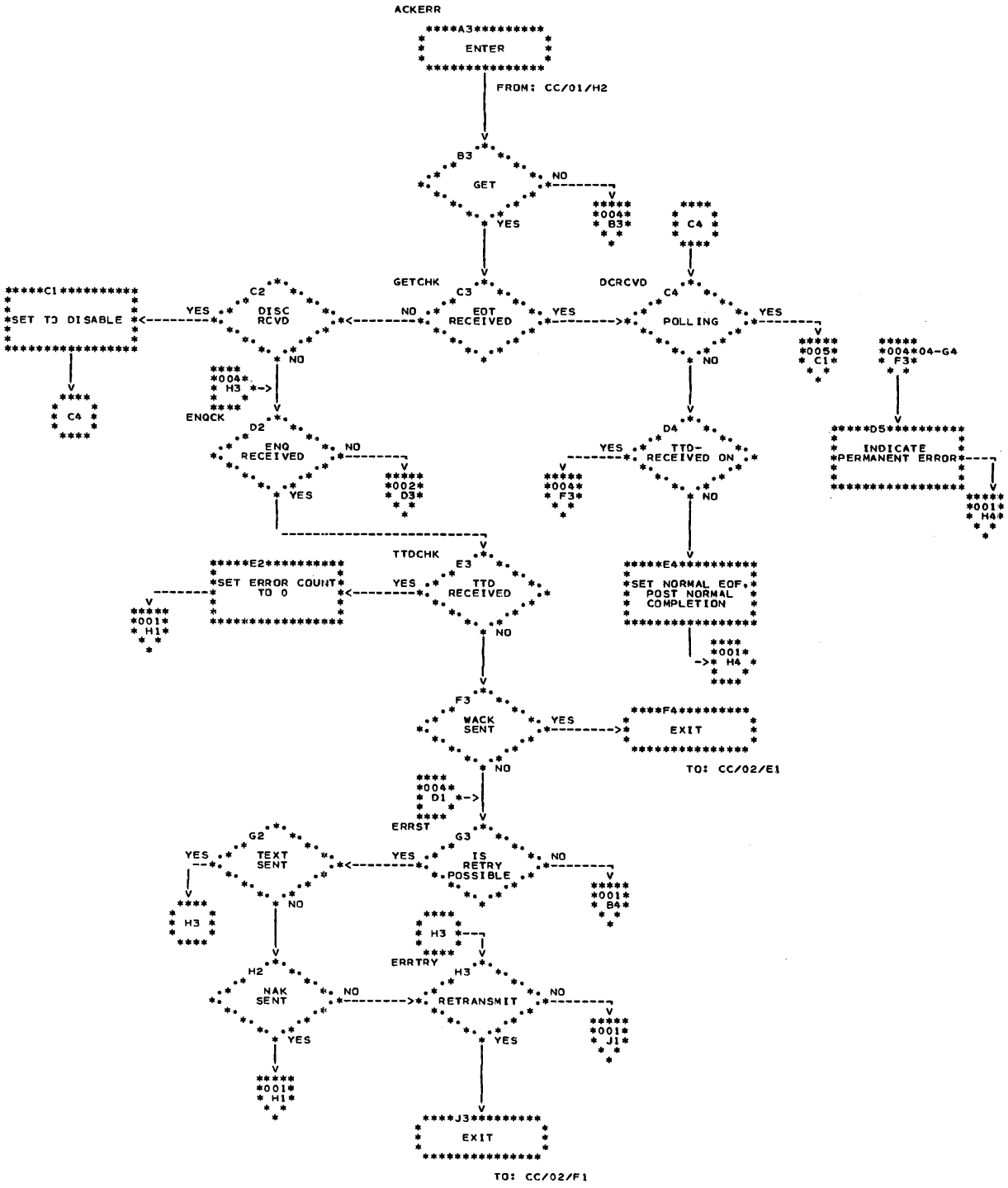


Chart RK (Part 3 of 5). Error Recovery Chart ($CC4M1)

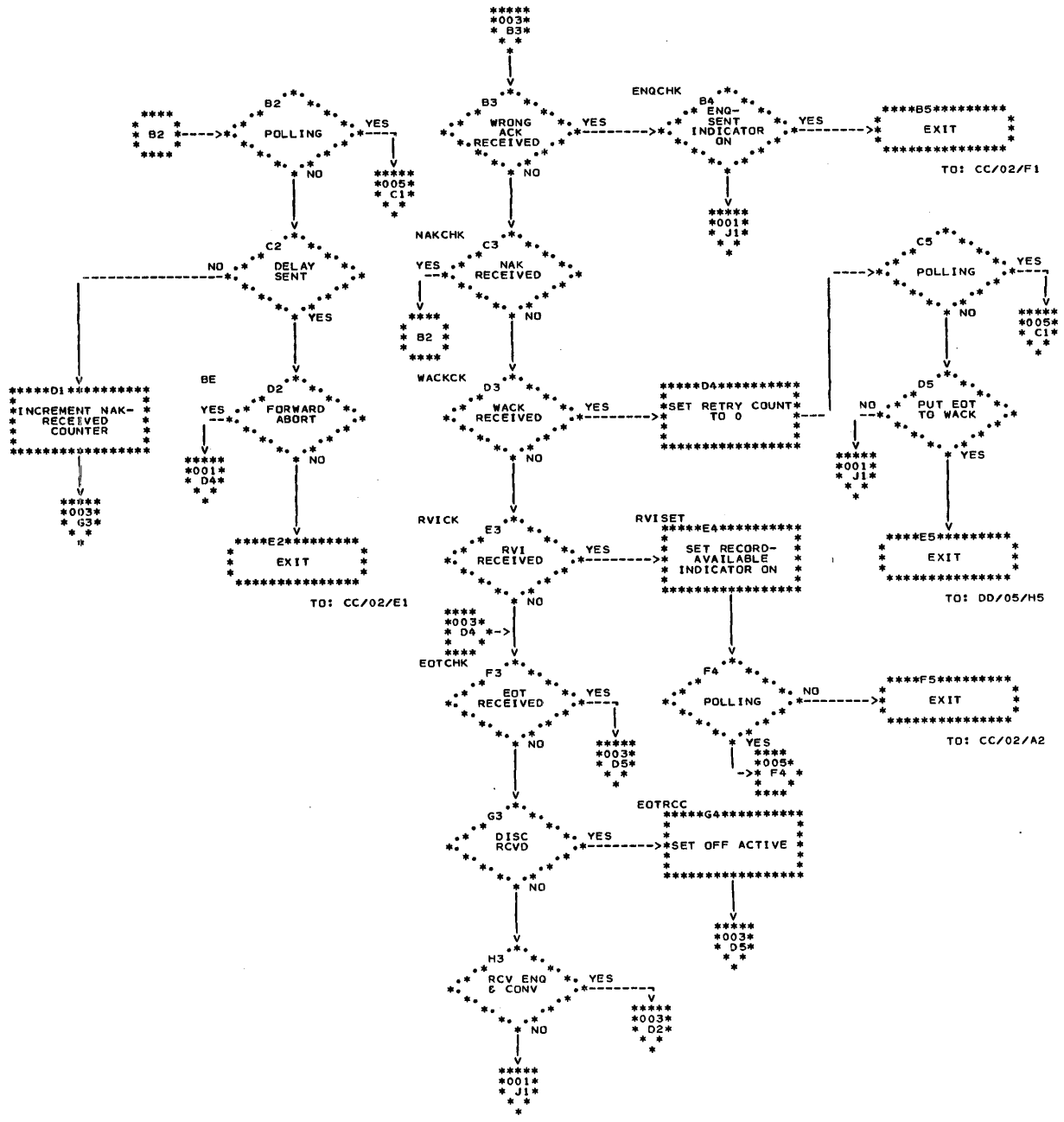


Chart RK (Part 4 of 5). Error Recovery Chart ($CC4M1)

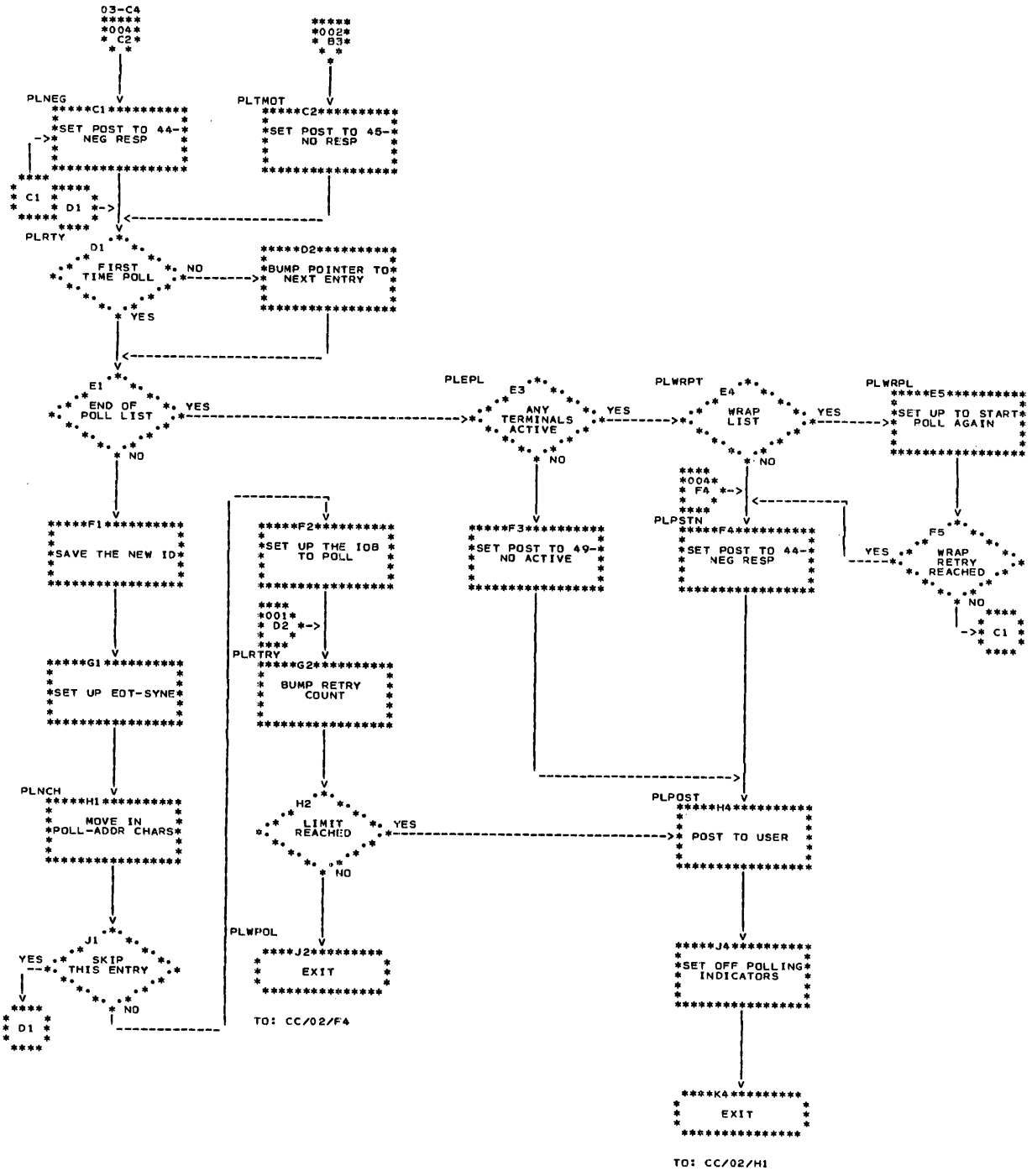


Chart RK (Part 5 of 5). Error Recovery Chart ($CC4M1)

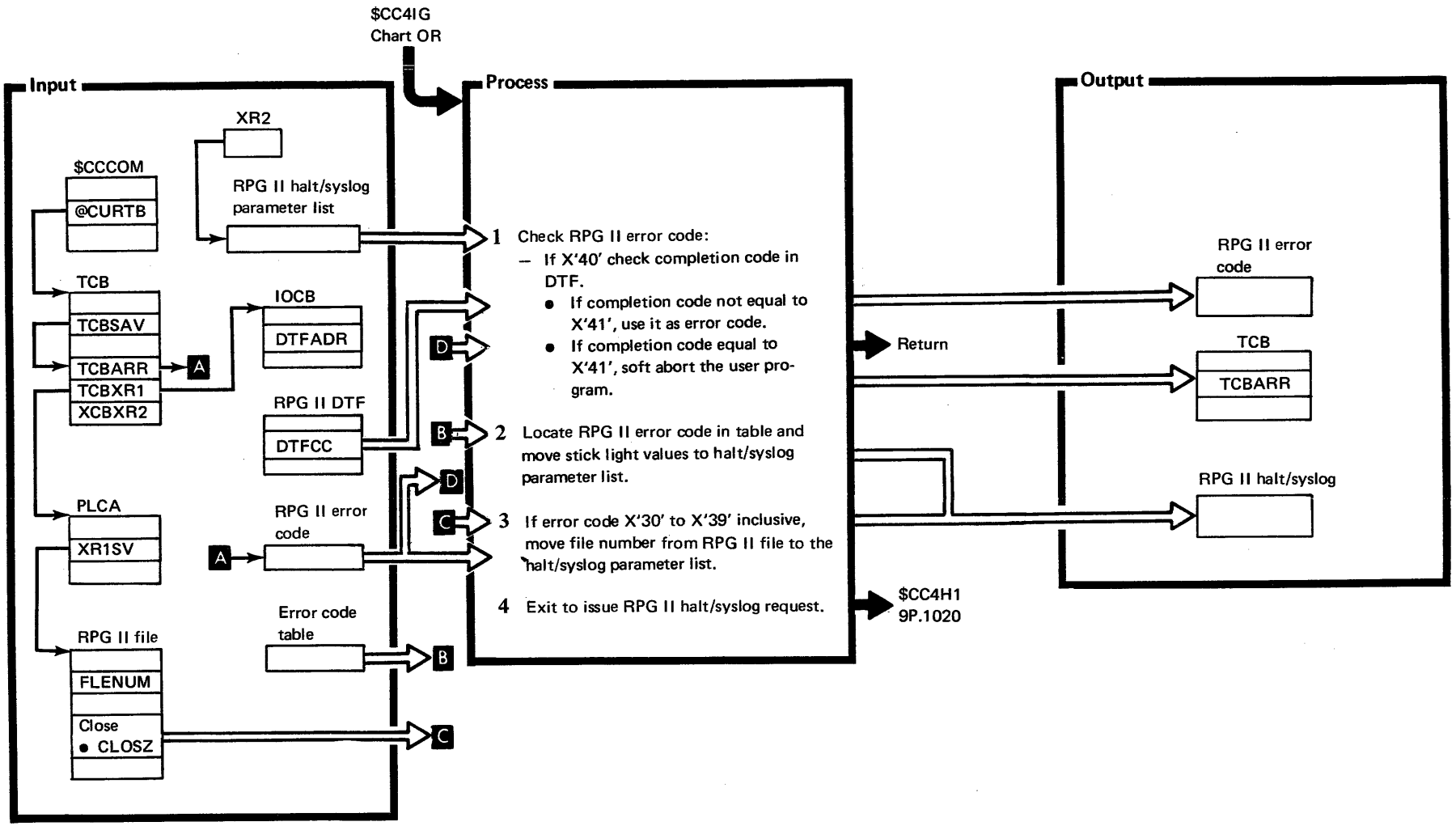


Diagram 9P.1010. $CC4HF

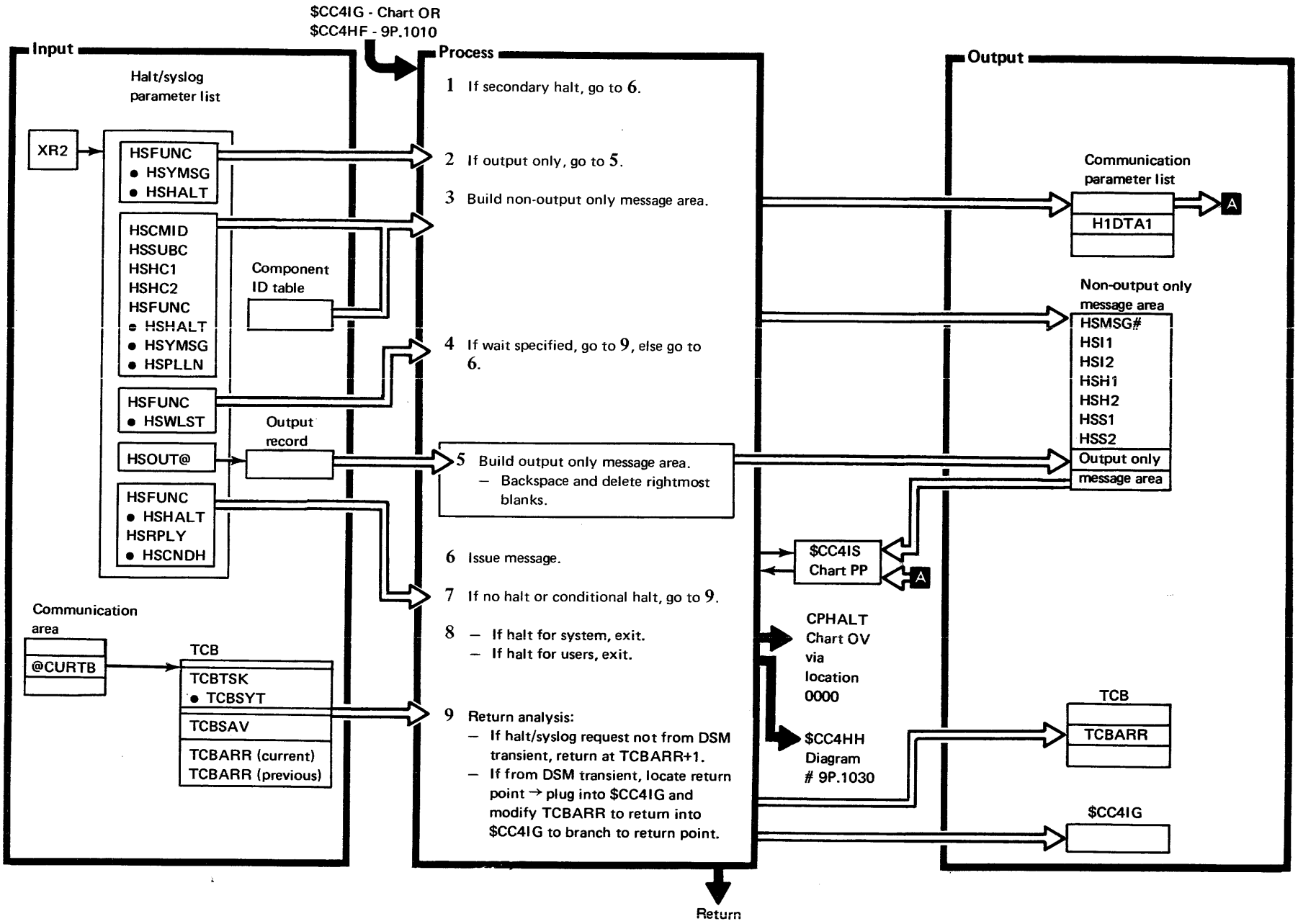


Diagram 9P.1020. $CC4H1 (Models 8, 10, and 12)

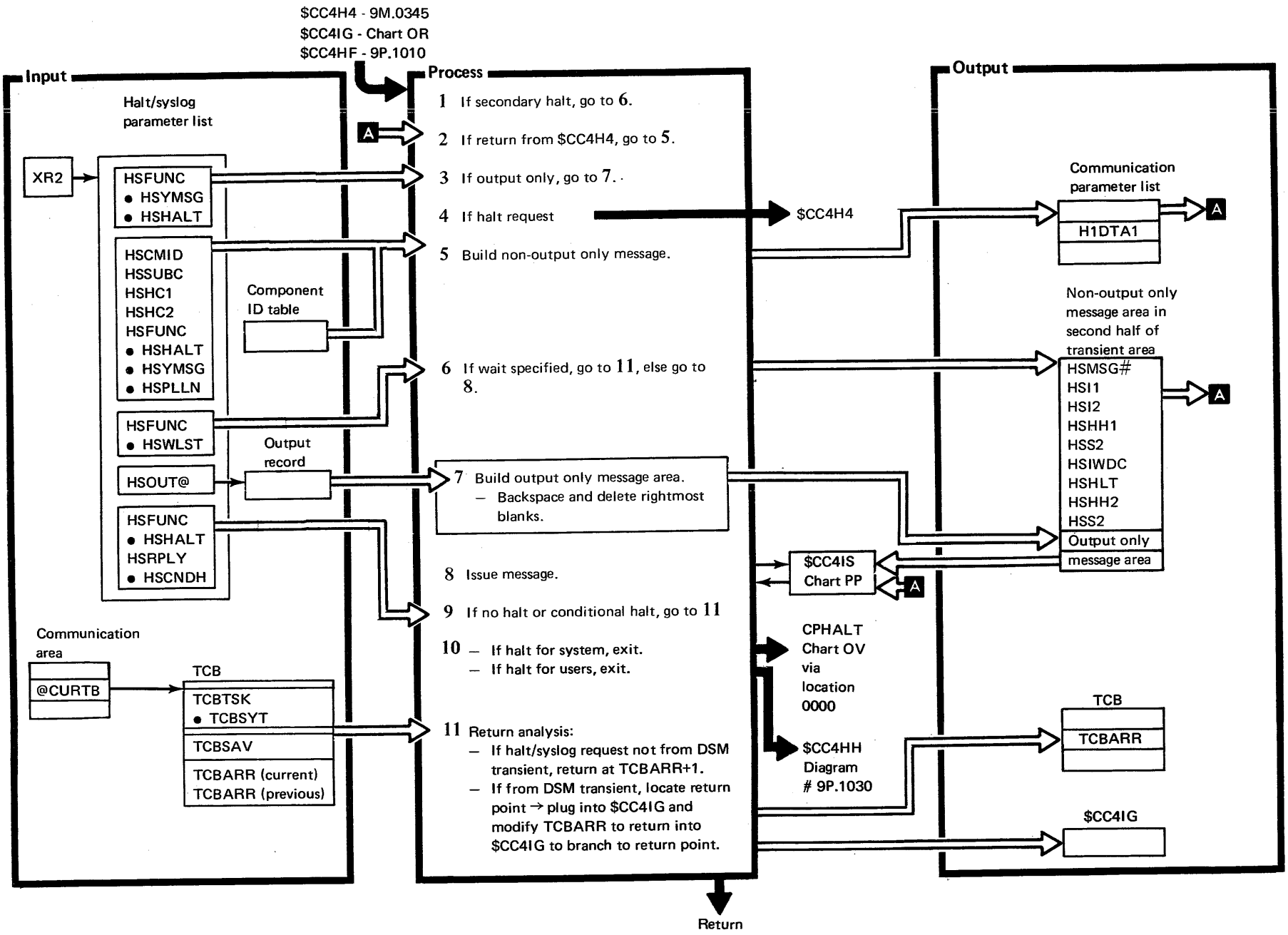


Diagram 9P.1025. $CC4H1 (Model 4)

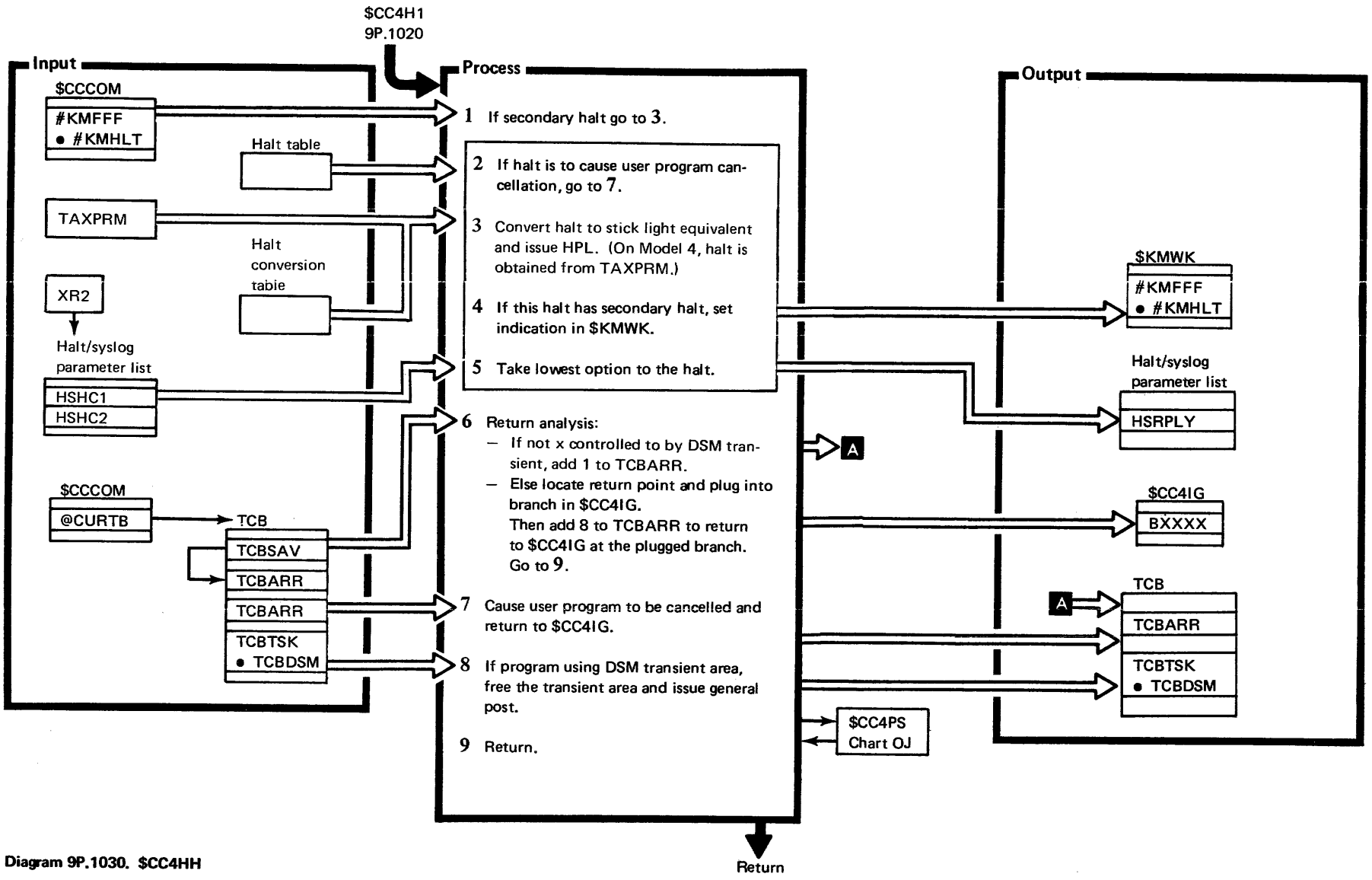


Diagram 9P.1030. $CC4HH

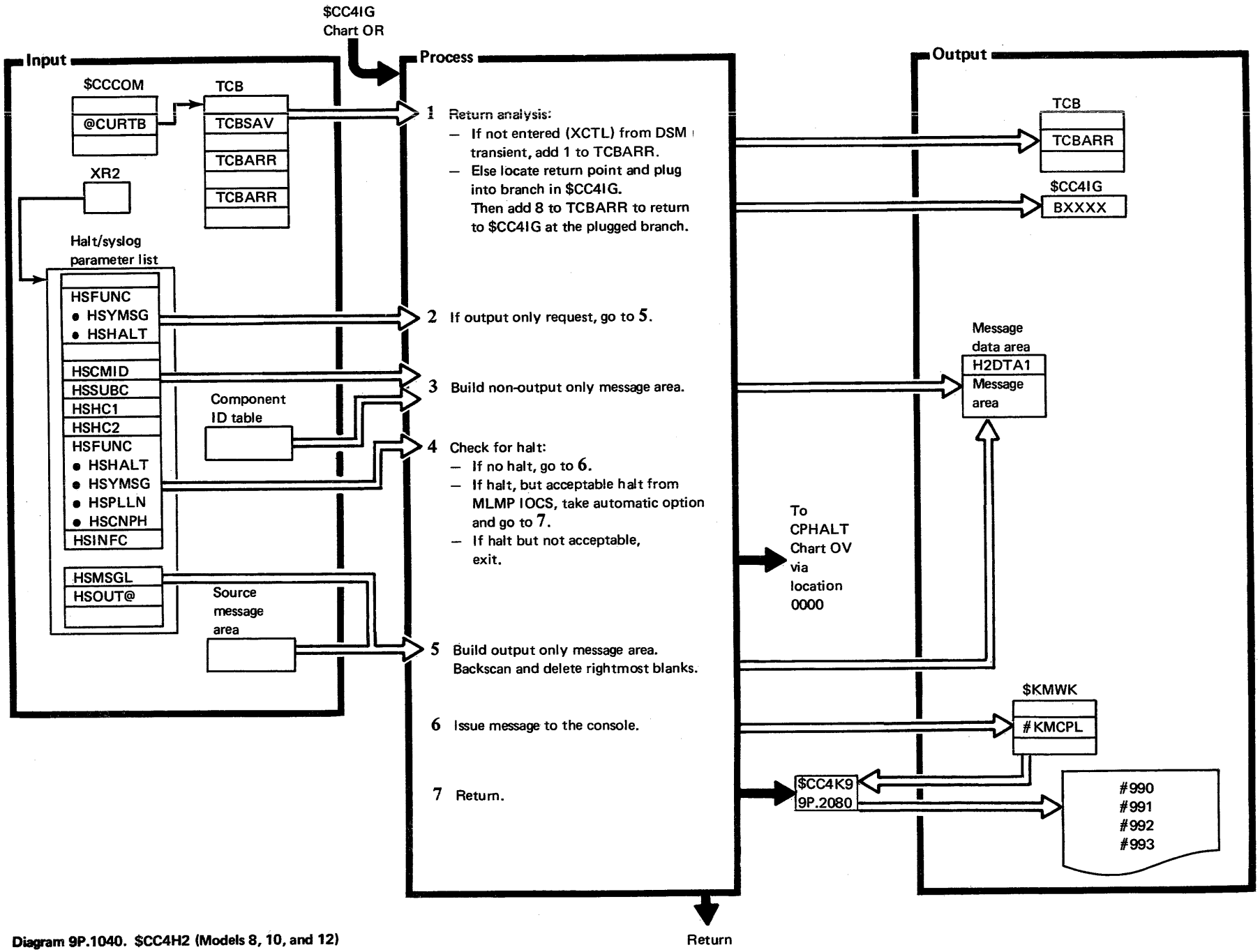
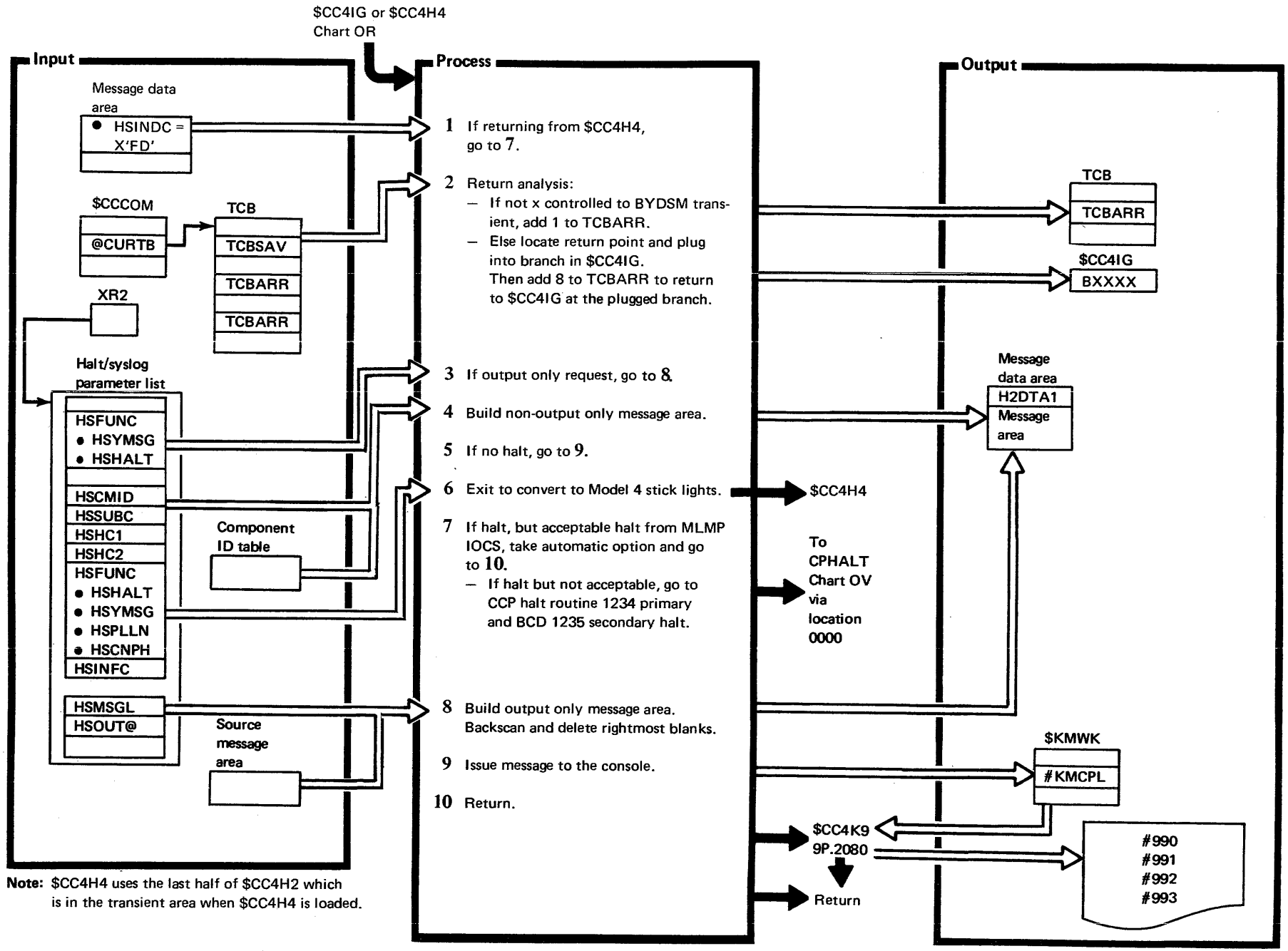


Diagram 9P.1040. $CC4H2 (Models 8, 10, and 12)



Note: $CC4H4 uses the last half of $CC4H2 which is in the transient area when $CC4H4 is loaded.

Diagram 9P.1045. $CC4H2 (Model 4)

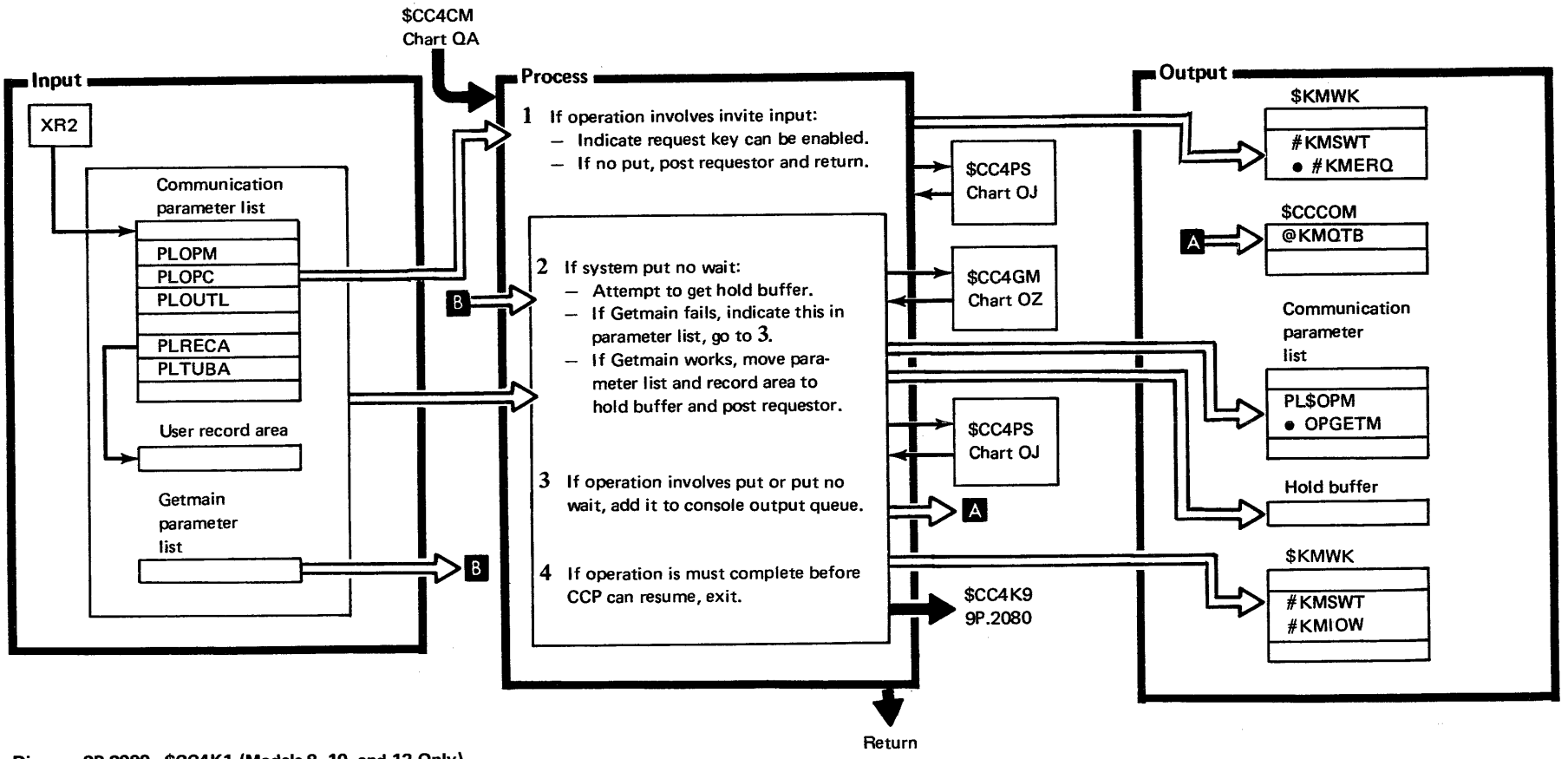


Diagram 9P.2000. $CC4K1 (Models 8, 10, and 12 Only)

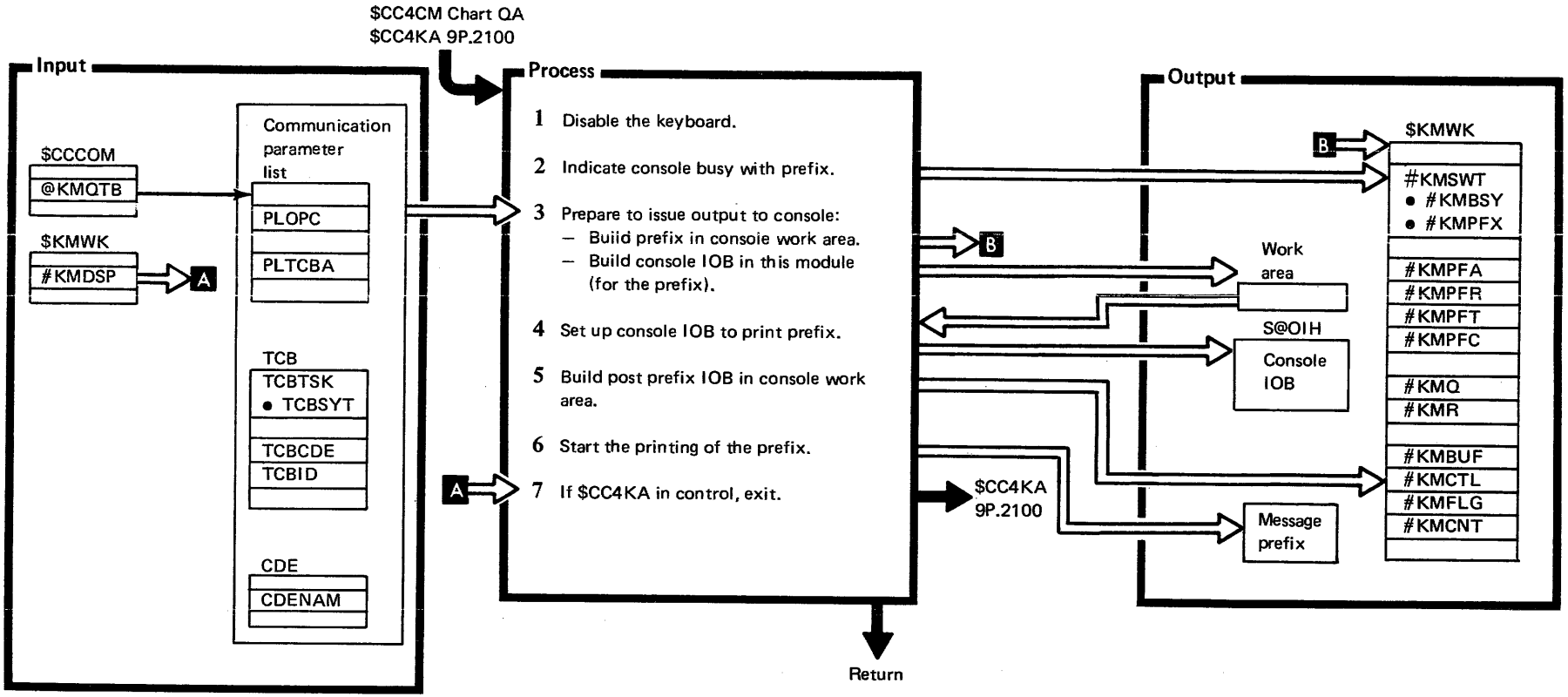


Diagram 9P.2010. $CC4K2 (Models 8, 10, and 12 Only)

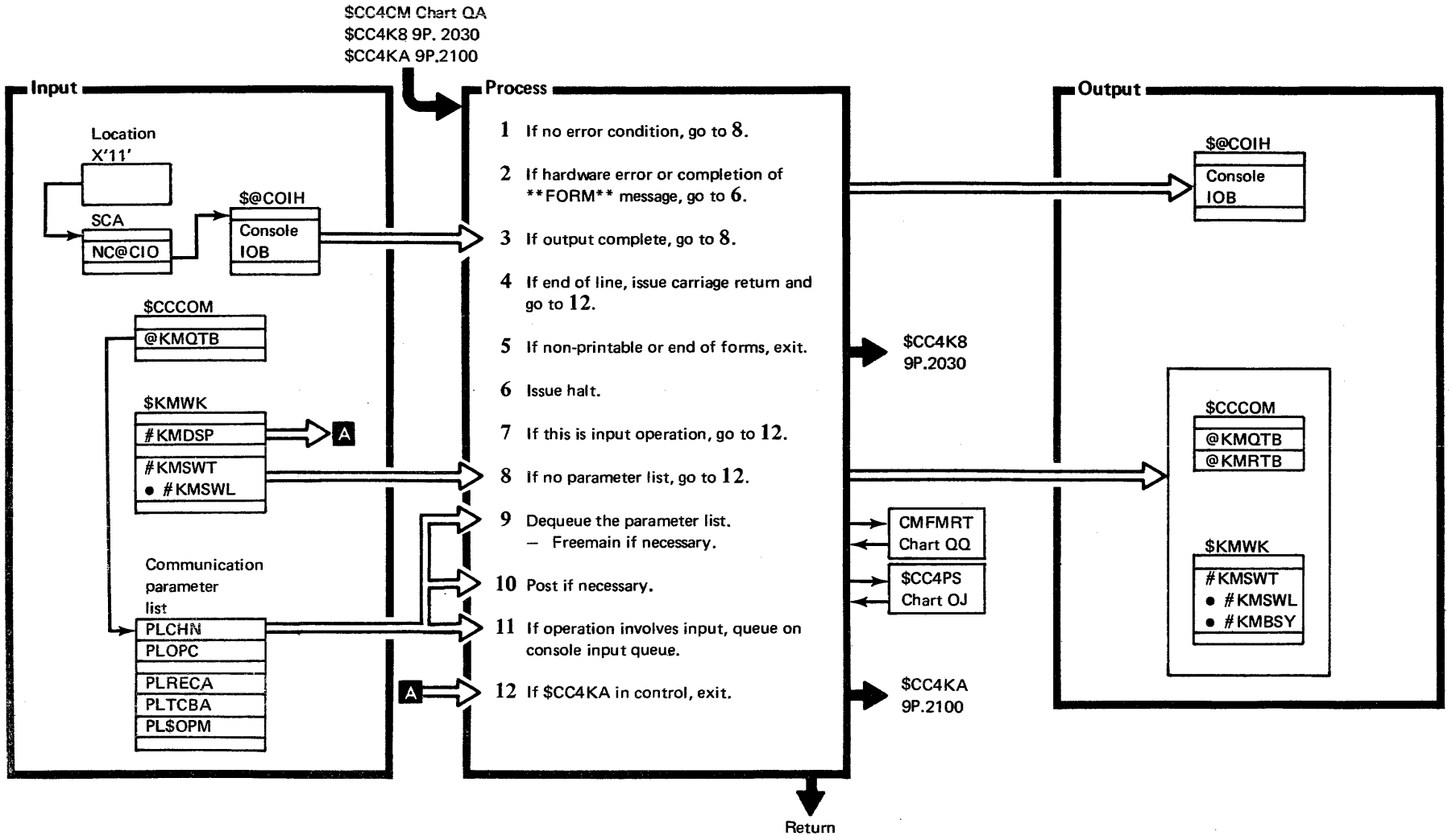


Diagram 9P.2020. $CC4K3 (Models 8, 10, and 12 Only)

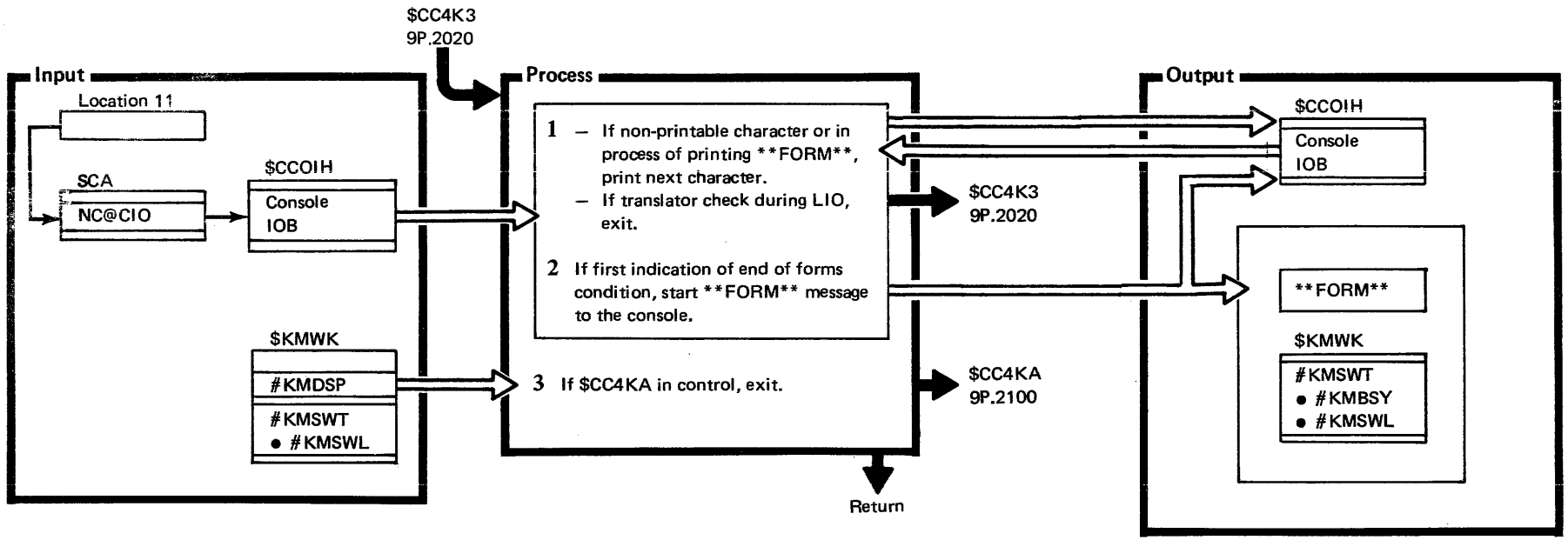


Diagram 9P.2030. $CC4K8 (Models 8, 10, and 12 Only)

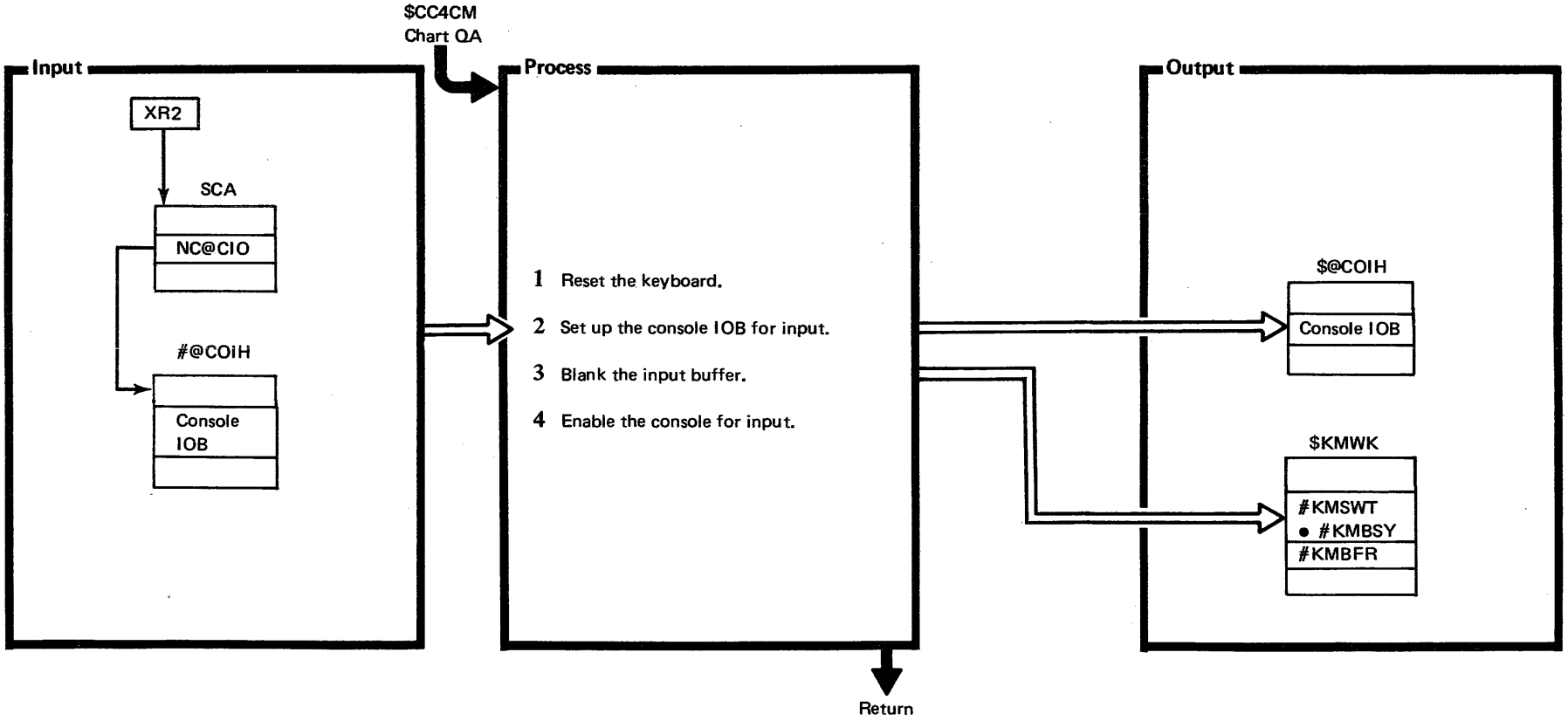


Diagram 9P.2040. $CC4K5 (Models 8, 10, and 12 Only)

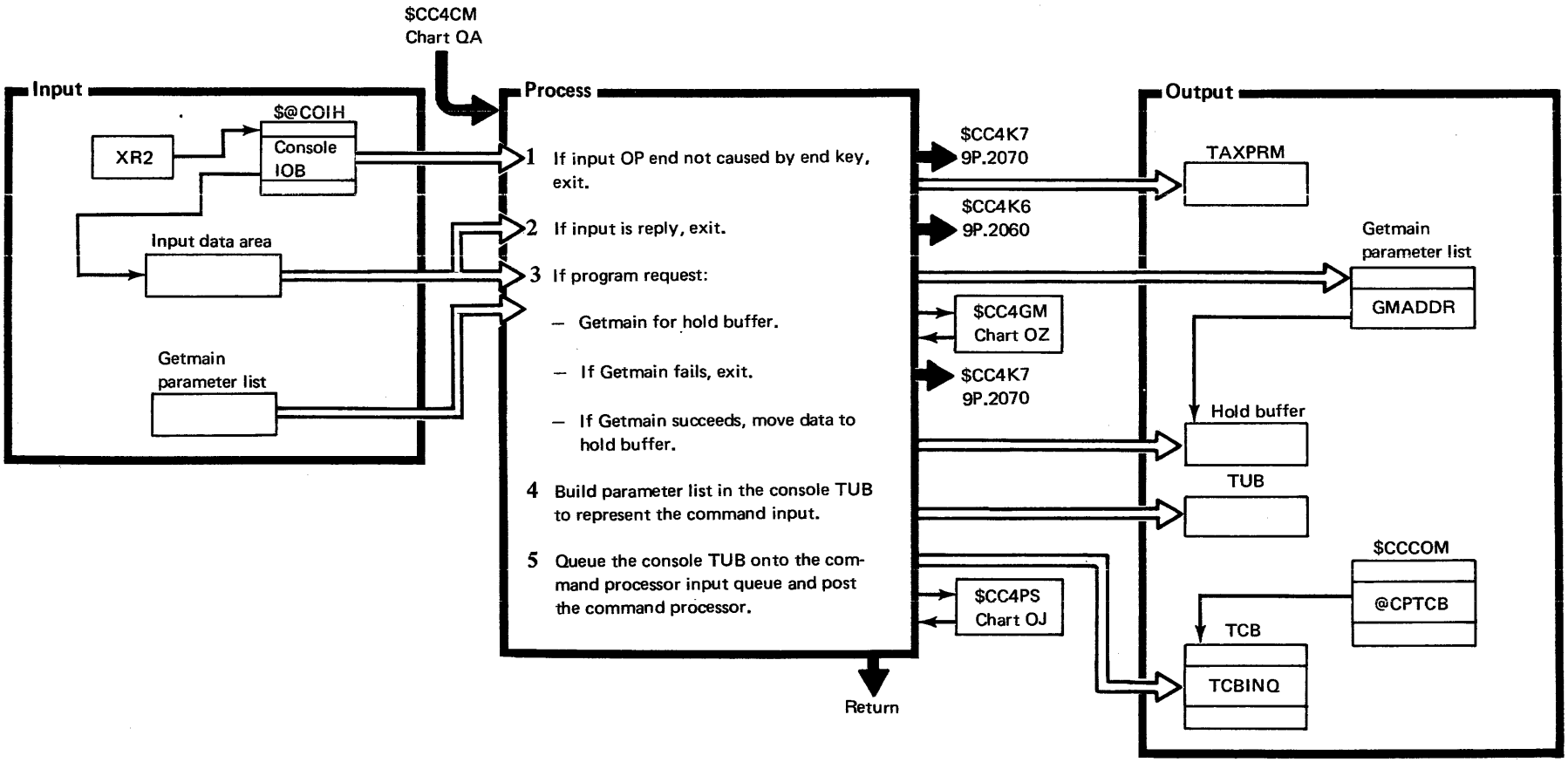


Diagram 9P.2050. $CC4K4 (Models 8, 10, and 12 Only)

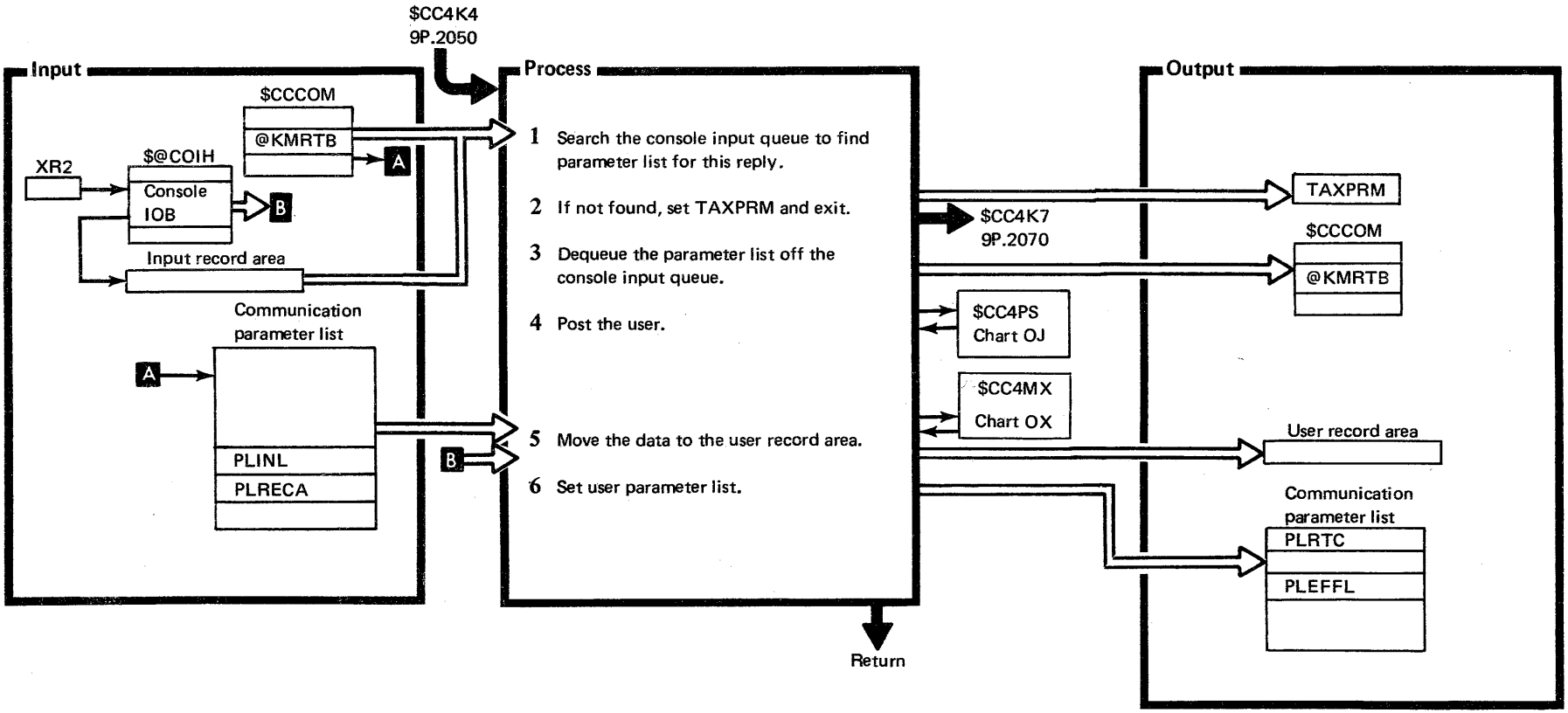


Diagram 9P.2060. $CC4K6 (Models 8, 10, and 12 Only)

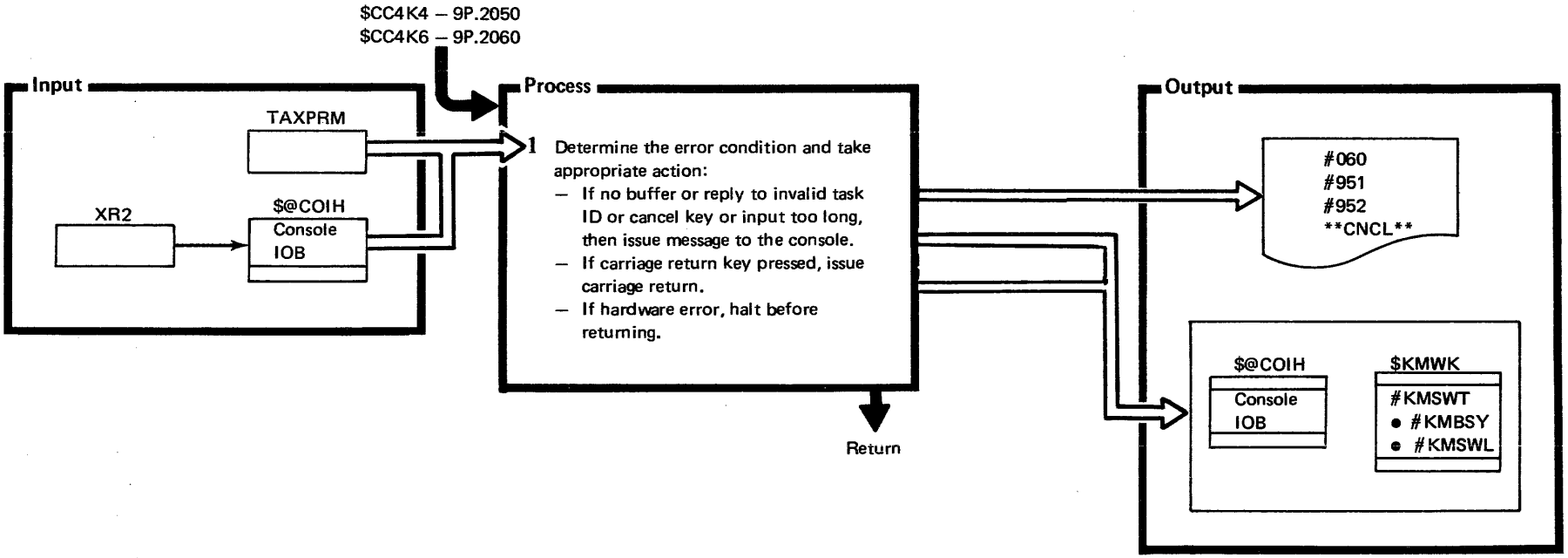
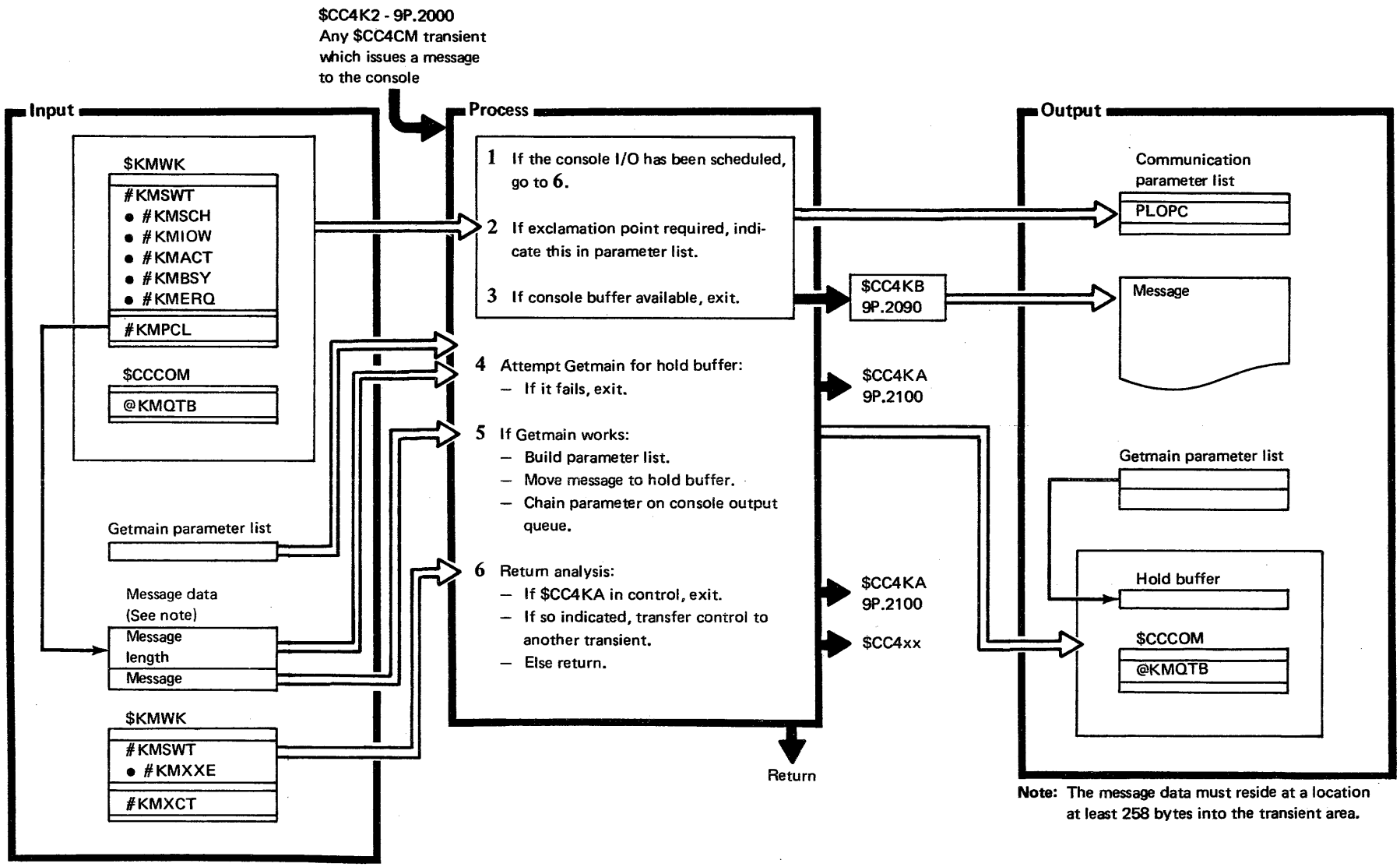
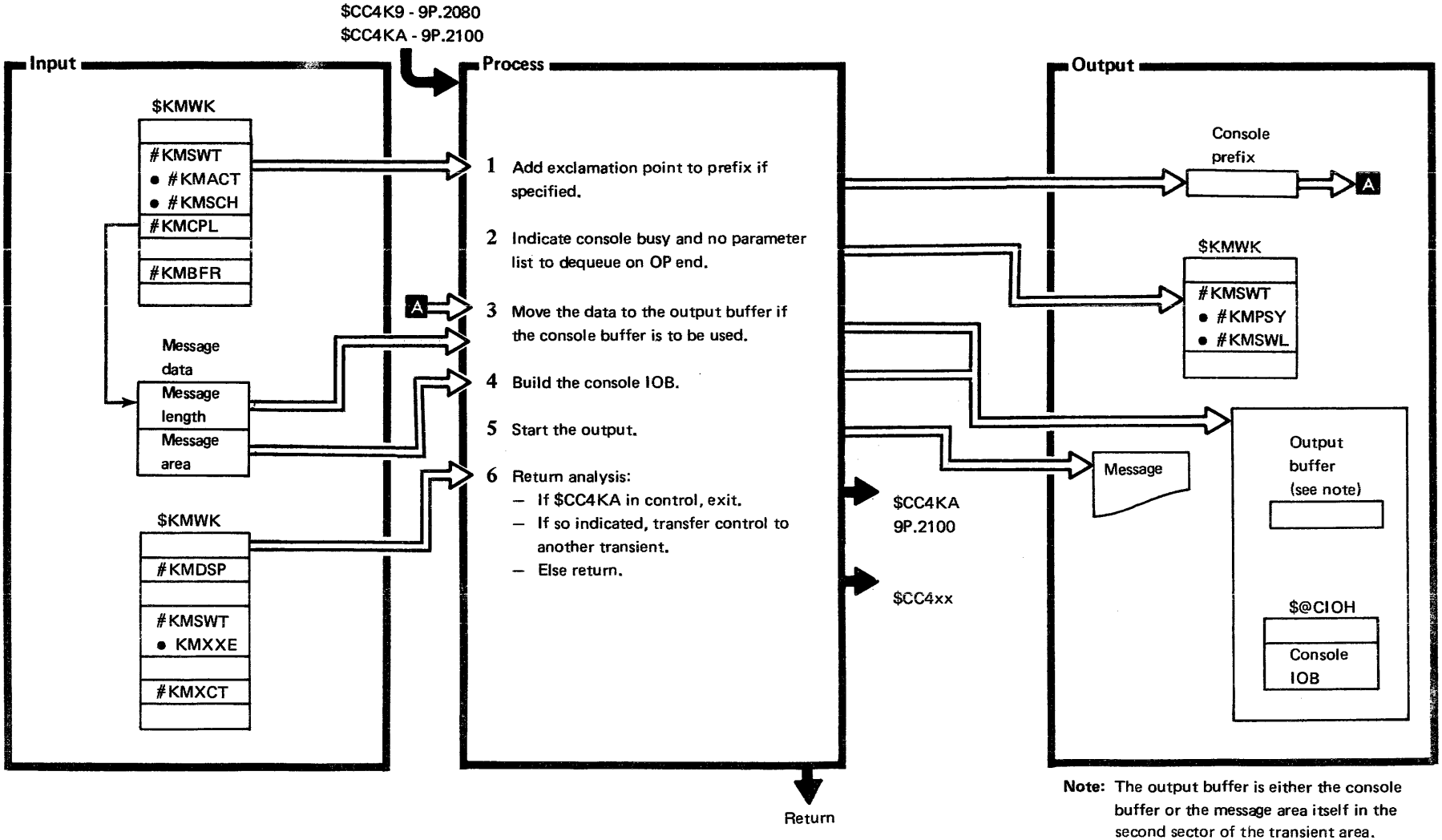


Diagram 9P.2070. $CC4K7 (Models 8, 10, and 12 Only)



Note: The message data must reside at a location at least 258 bytes into the transient area.

Diagram 9P.2080. $CC4K9 (Models 8, 10, and 12 Only)



Note: The output buffer is either the console buffer or the message area itself in the second sector of the transient area.

Diagram 9P.2090. $CC4KB (Models 8, 10, and 12 Only)

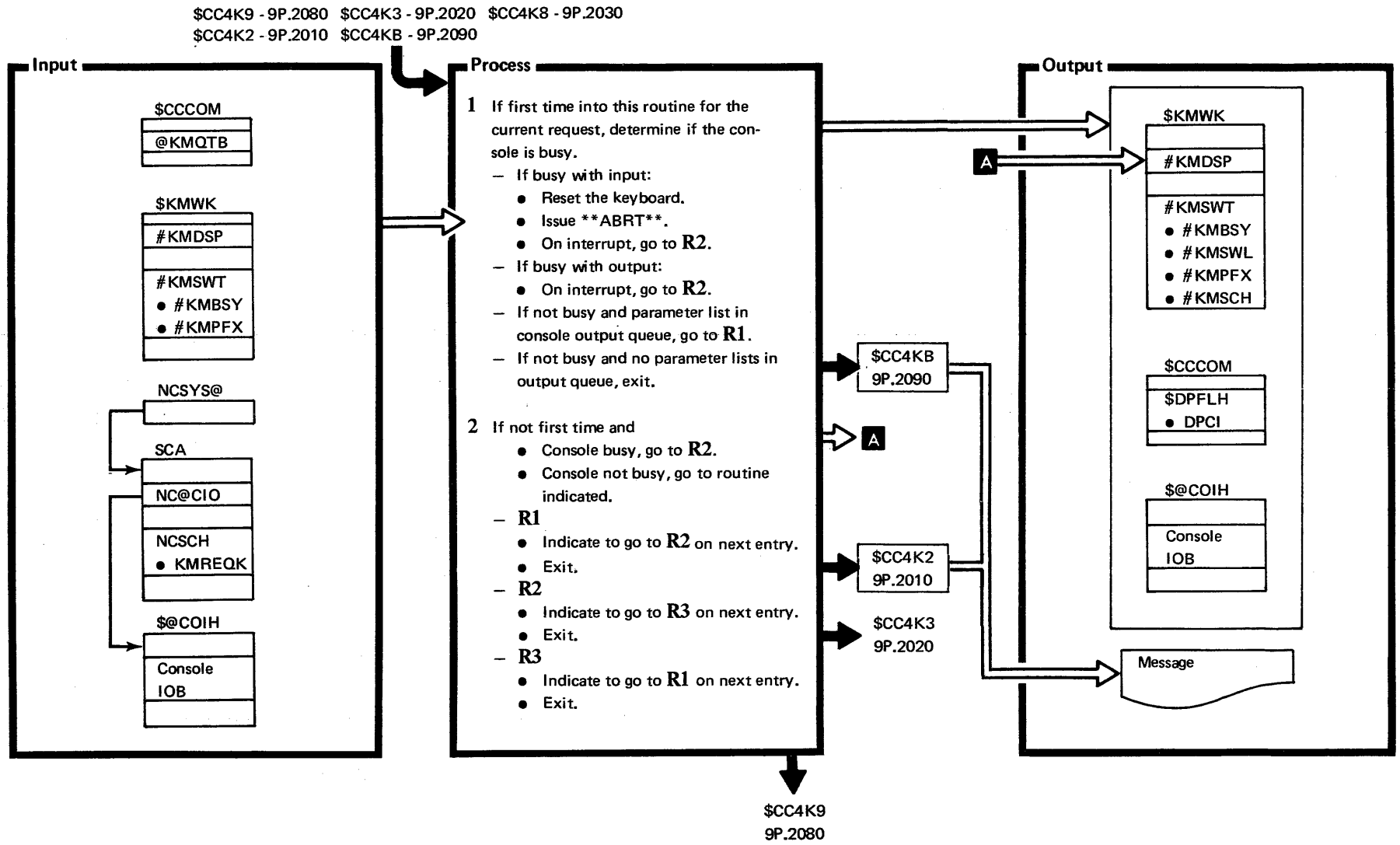


Diagram 9P.2100. $CC4KA (Models 8, 10, and 12 Only)

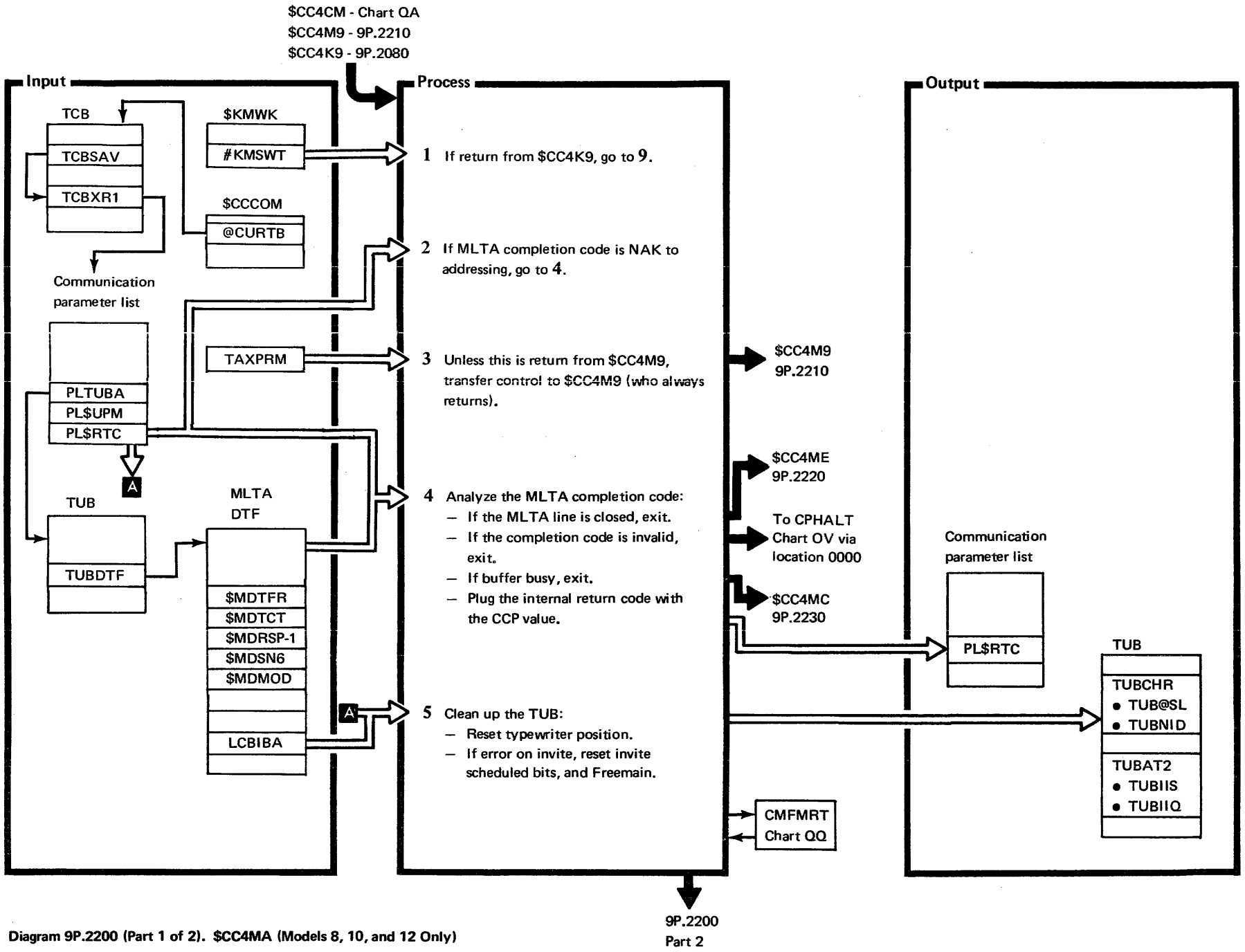


Diagram 9P.2200 (Part 1 of 2). $CC4MA (Models 8, 10, and 12 Only)

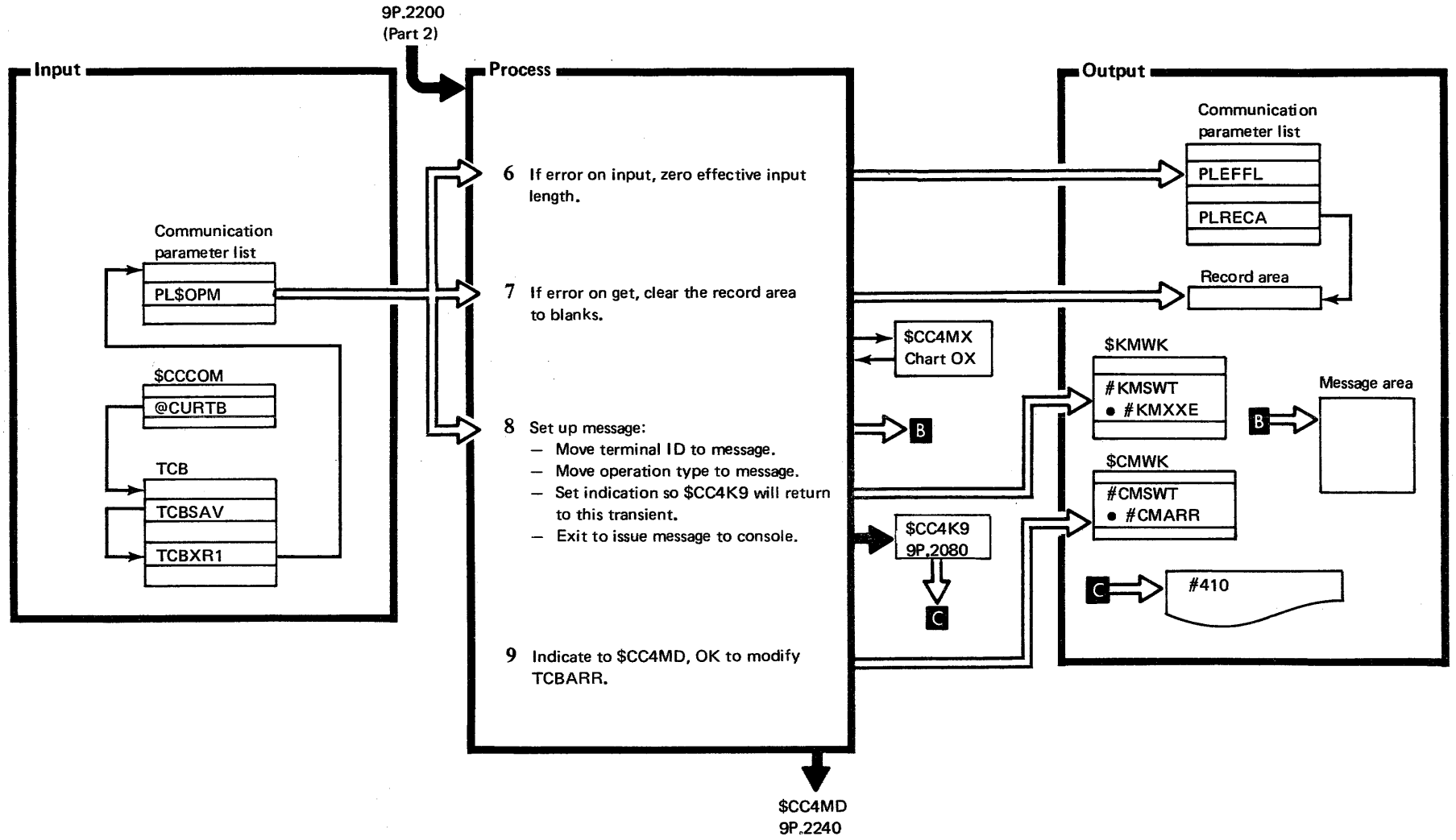


Diagram 9P.2200 (Part 2 of 2). $CC4MA (Models 8, 10, and 12 Only)

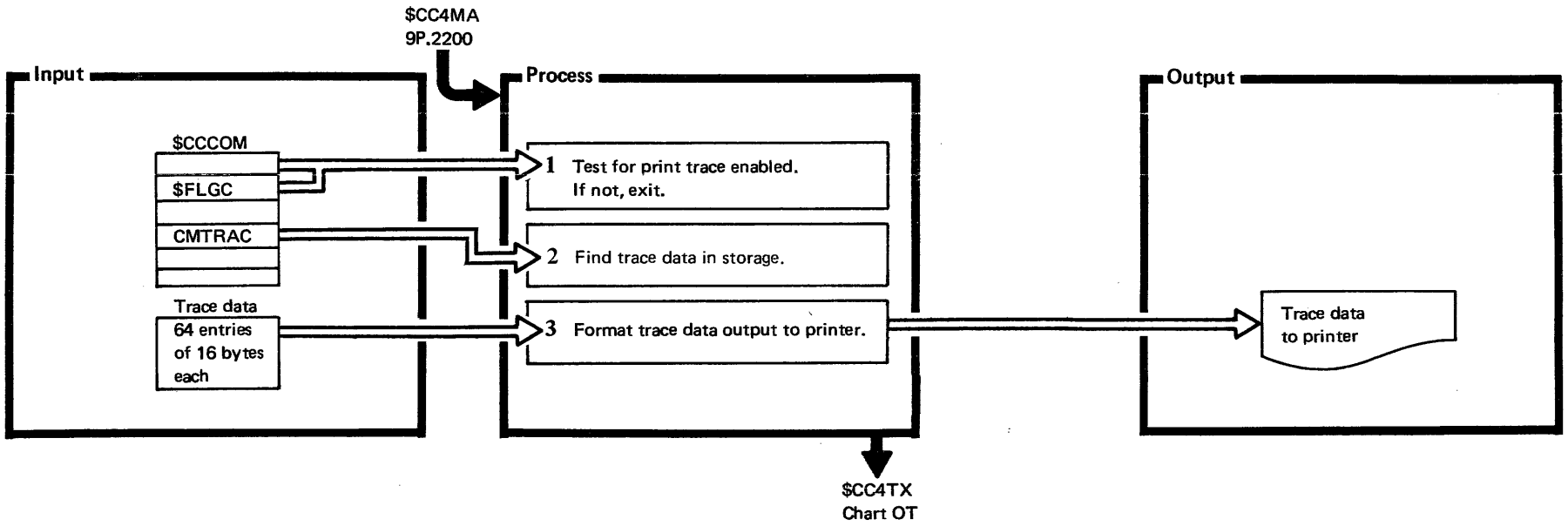


Diagram 9P.2210. $CC4M9 (Models 8, 10, and 12 Only)

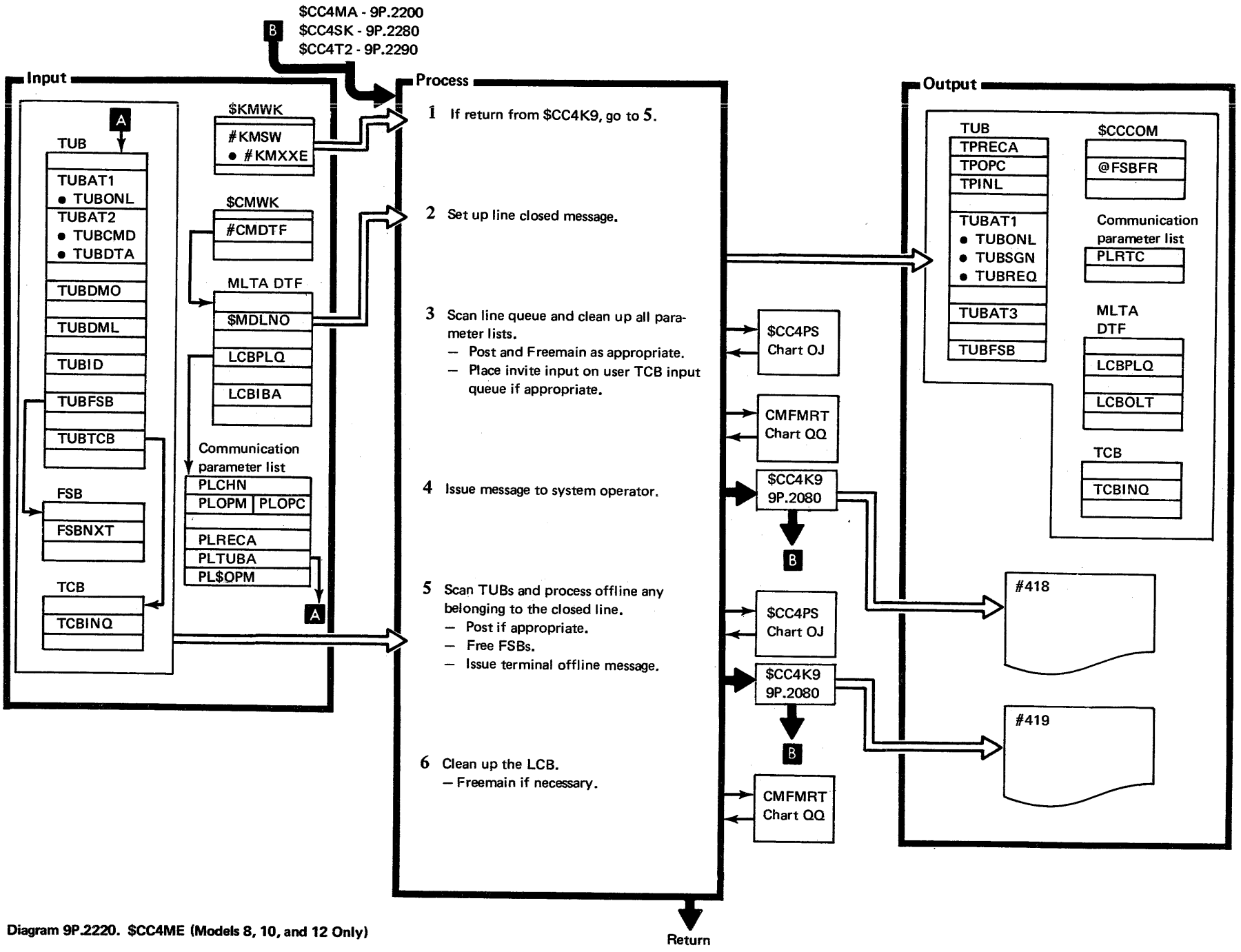
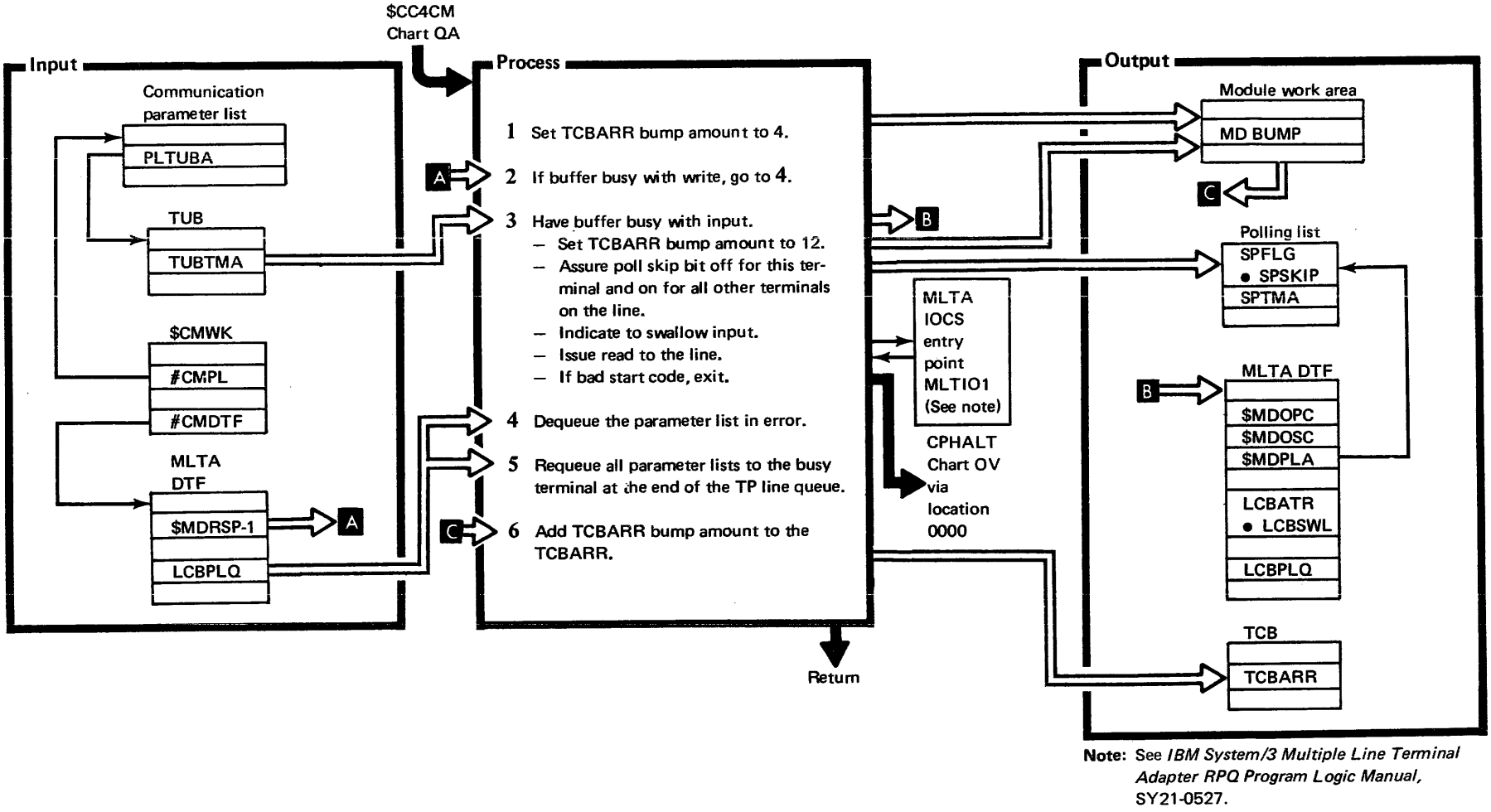


Diagram 9P.2220. $CC4ME (Models 8, 10, and 12 Only)



Note: See *IBM System/3 Multiple Line Terminal Adapter RPQ Program Logic Manual*, SY21-0527.

Diagram 9P.2230. $CC4MC (Models 8, 10, and 12 Only)

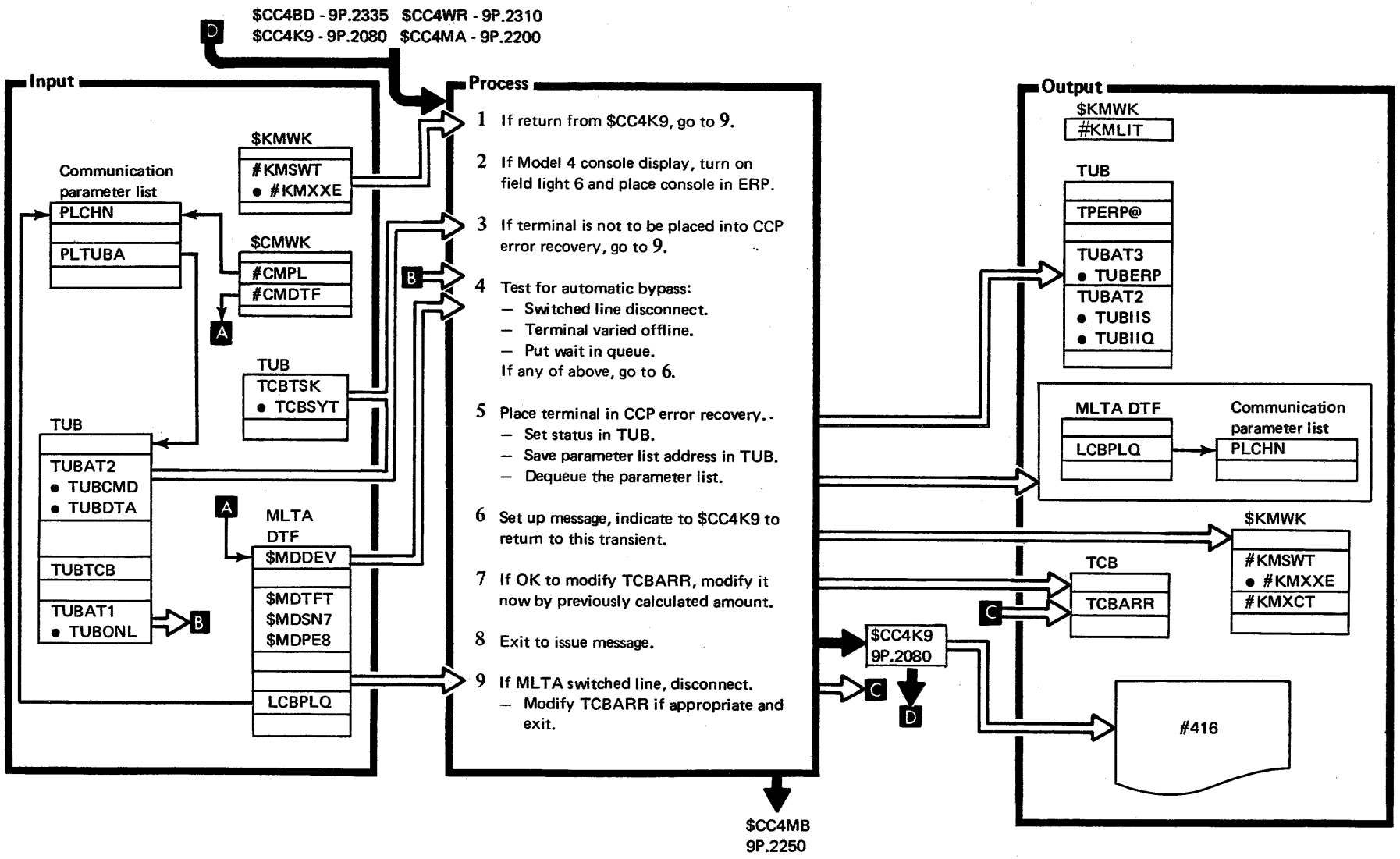


Diagram 9P.2240. $CC4MD (Models 8, 10, and 12 Only)

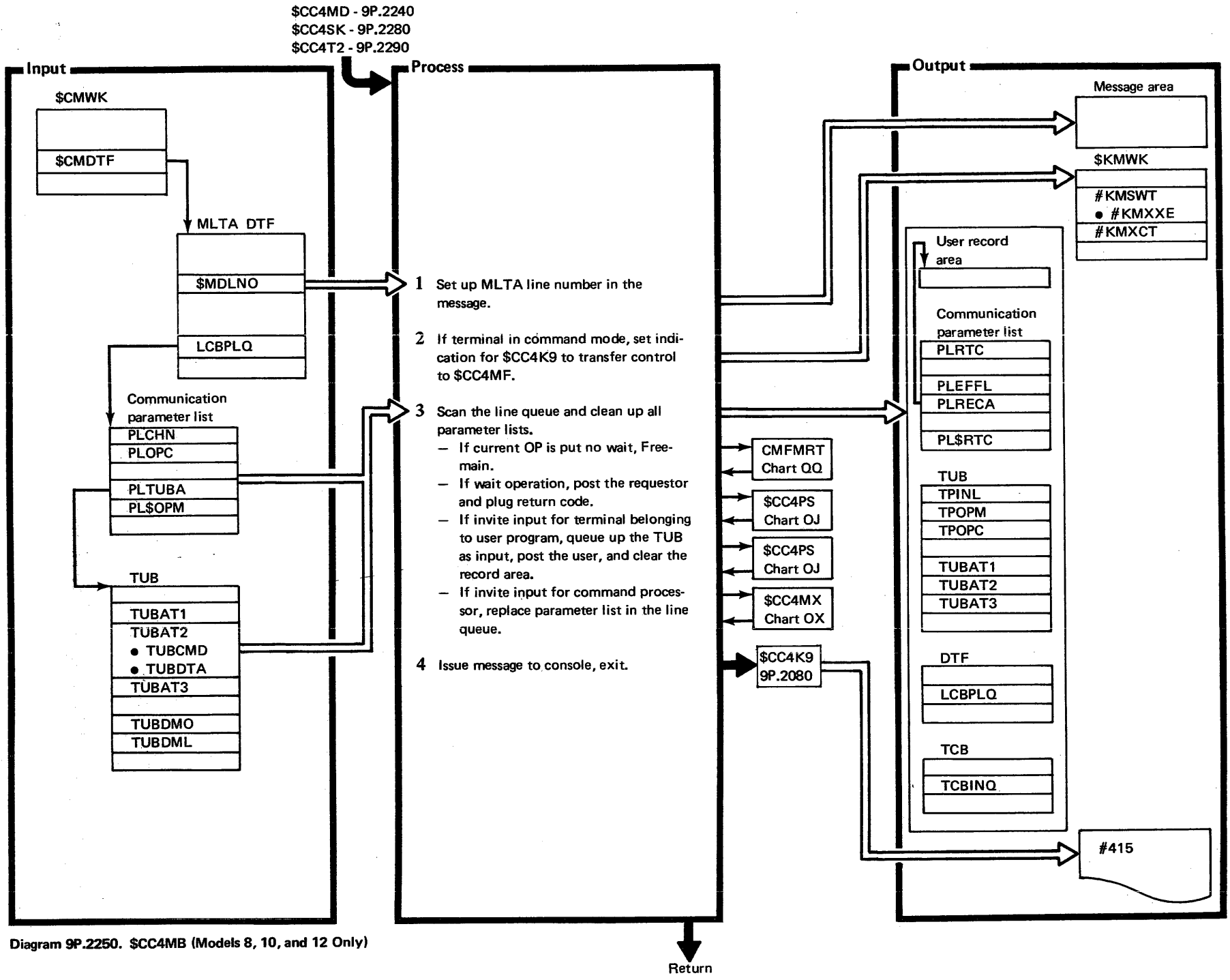


Diagram 9P.2250. $CC4MB (Models 8, 10, and 12 Only)

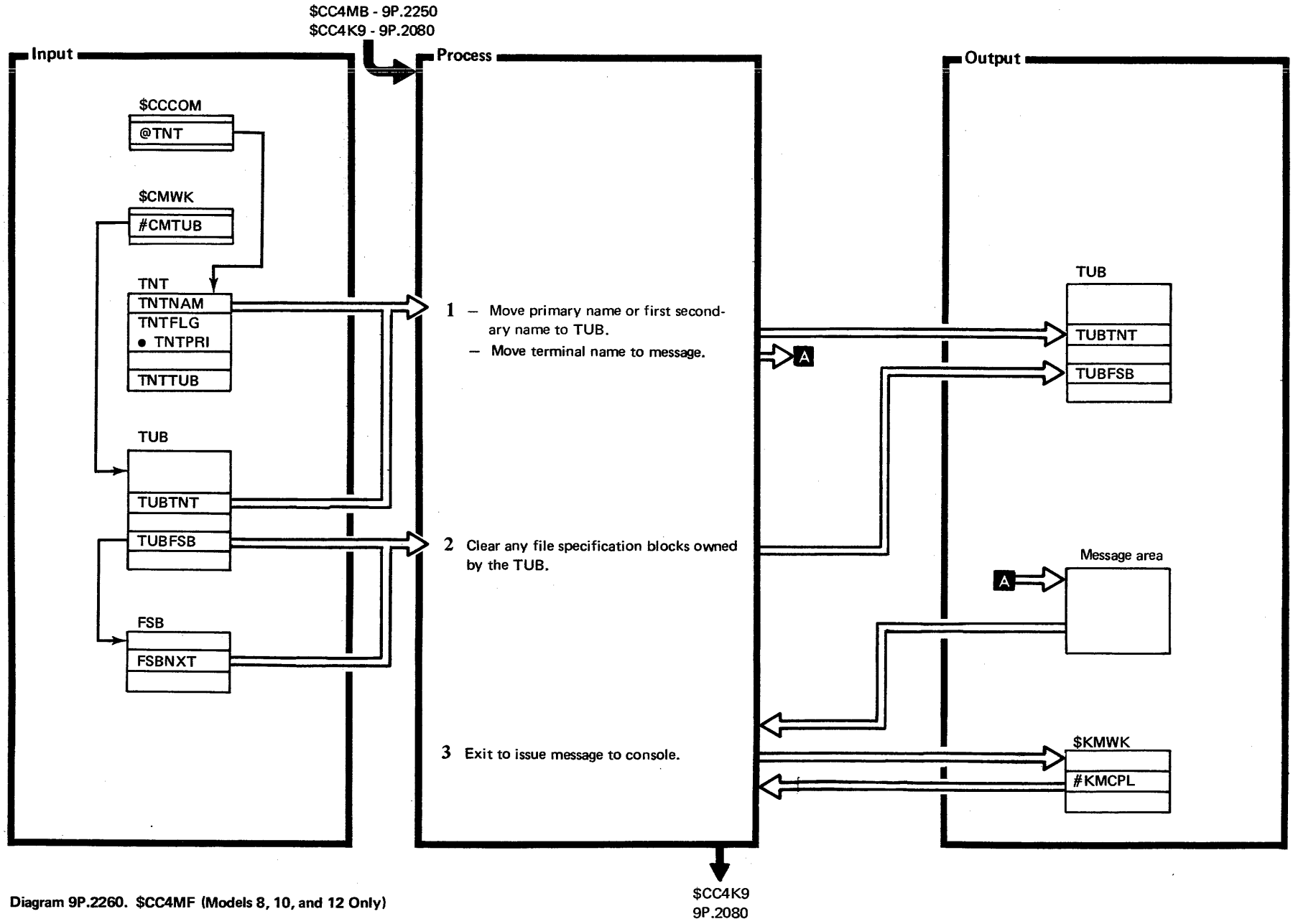


Diagram 9P.2260. $CC4MF (Models 8, 10, and 12 Only)

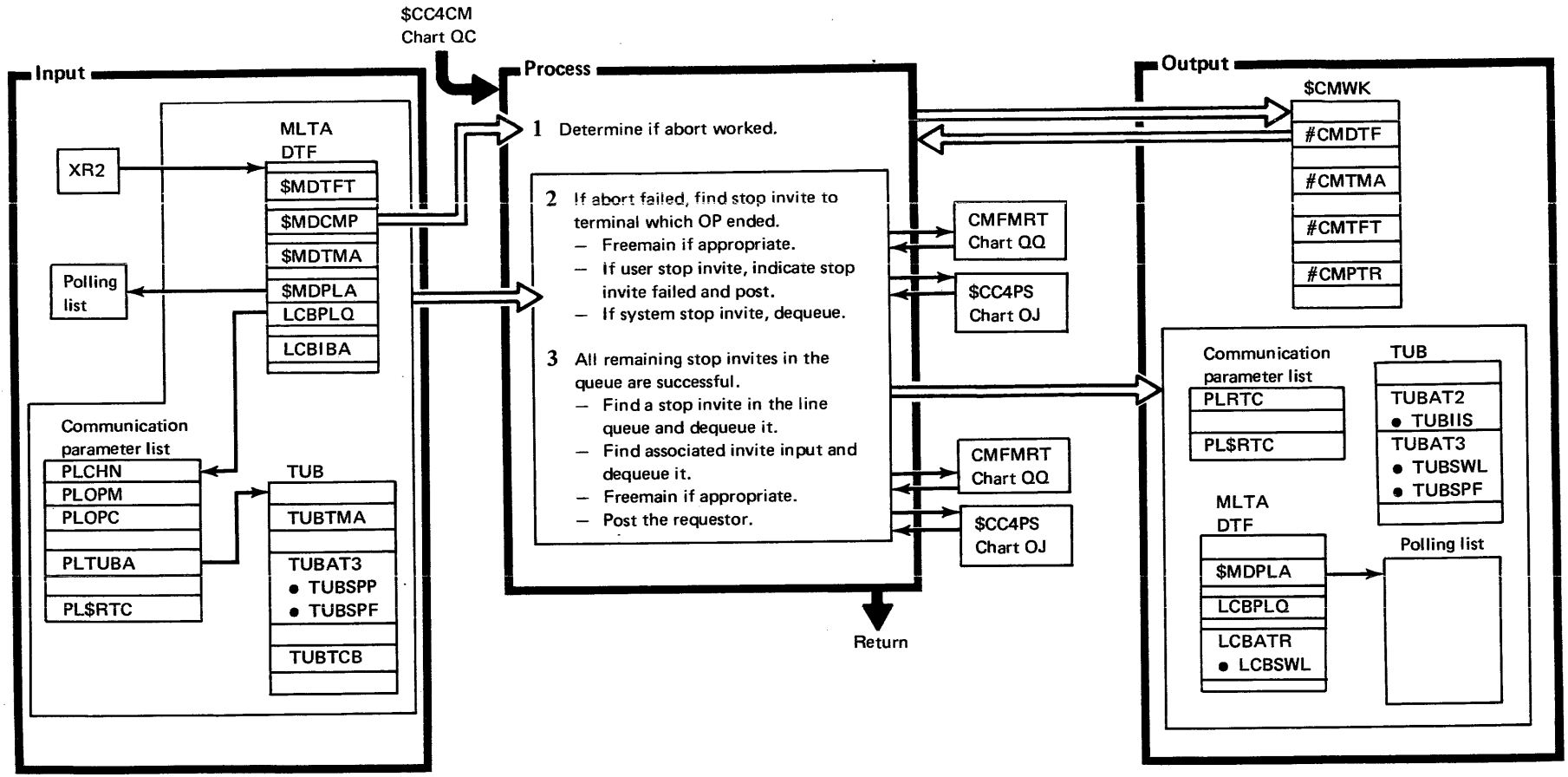


Diagram 9P.2270. $CC4SQ (Models 8, 10, and 12 Only)

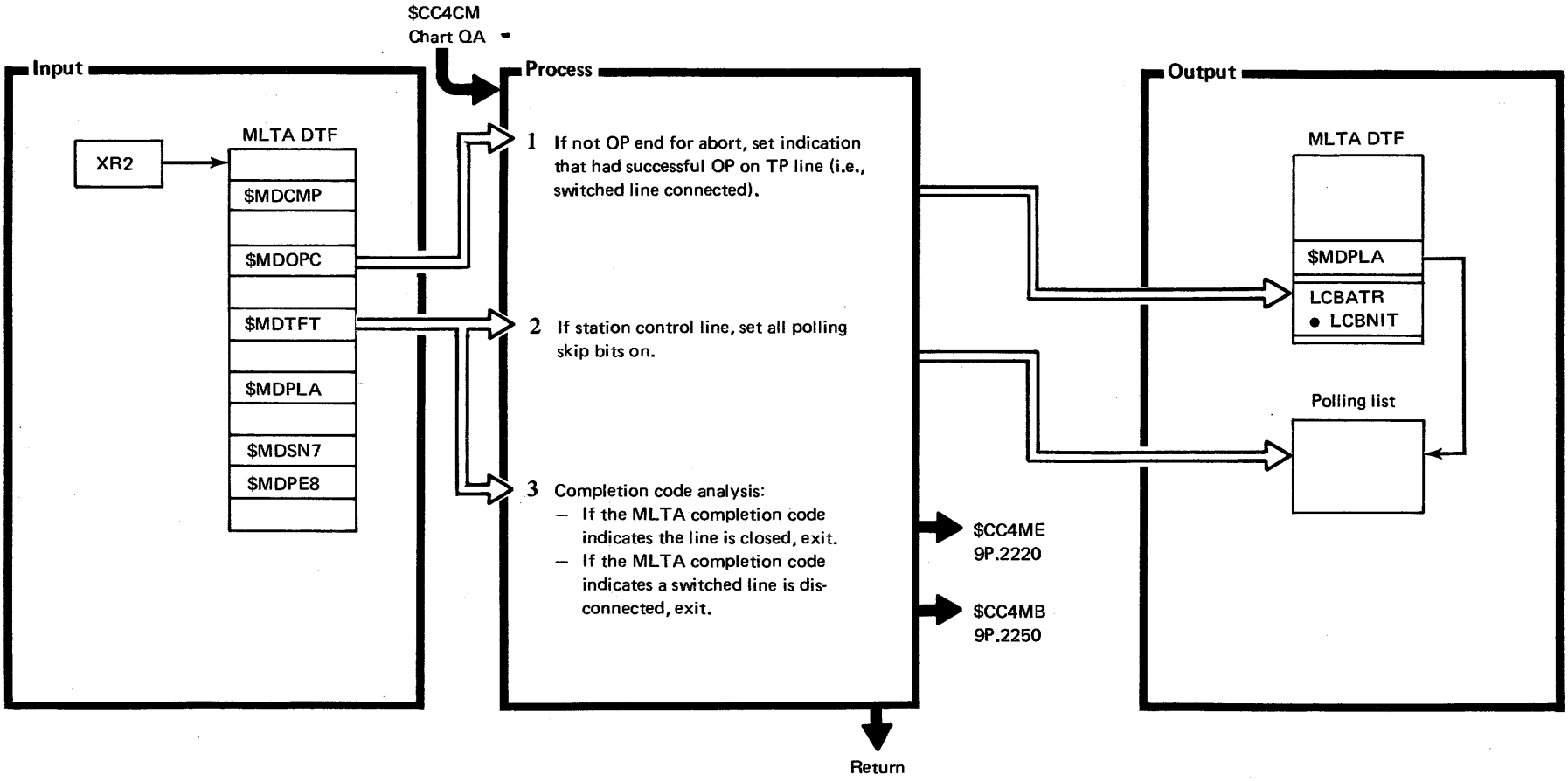


Diagram 9P.2280. $CC4SK (Models 8, 10, and 12 Only)

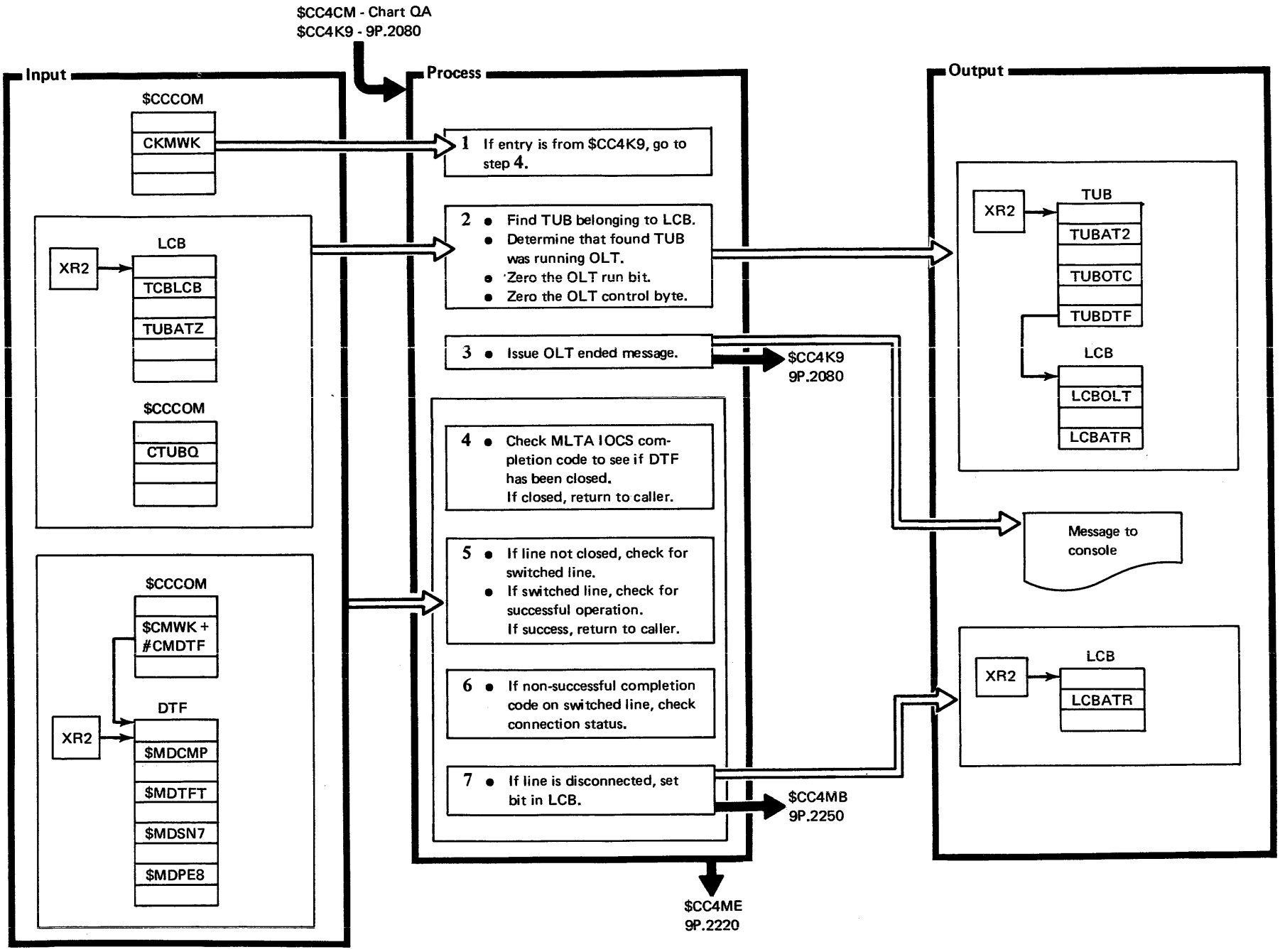


Diagram 9P.2290. $CC4T2

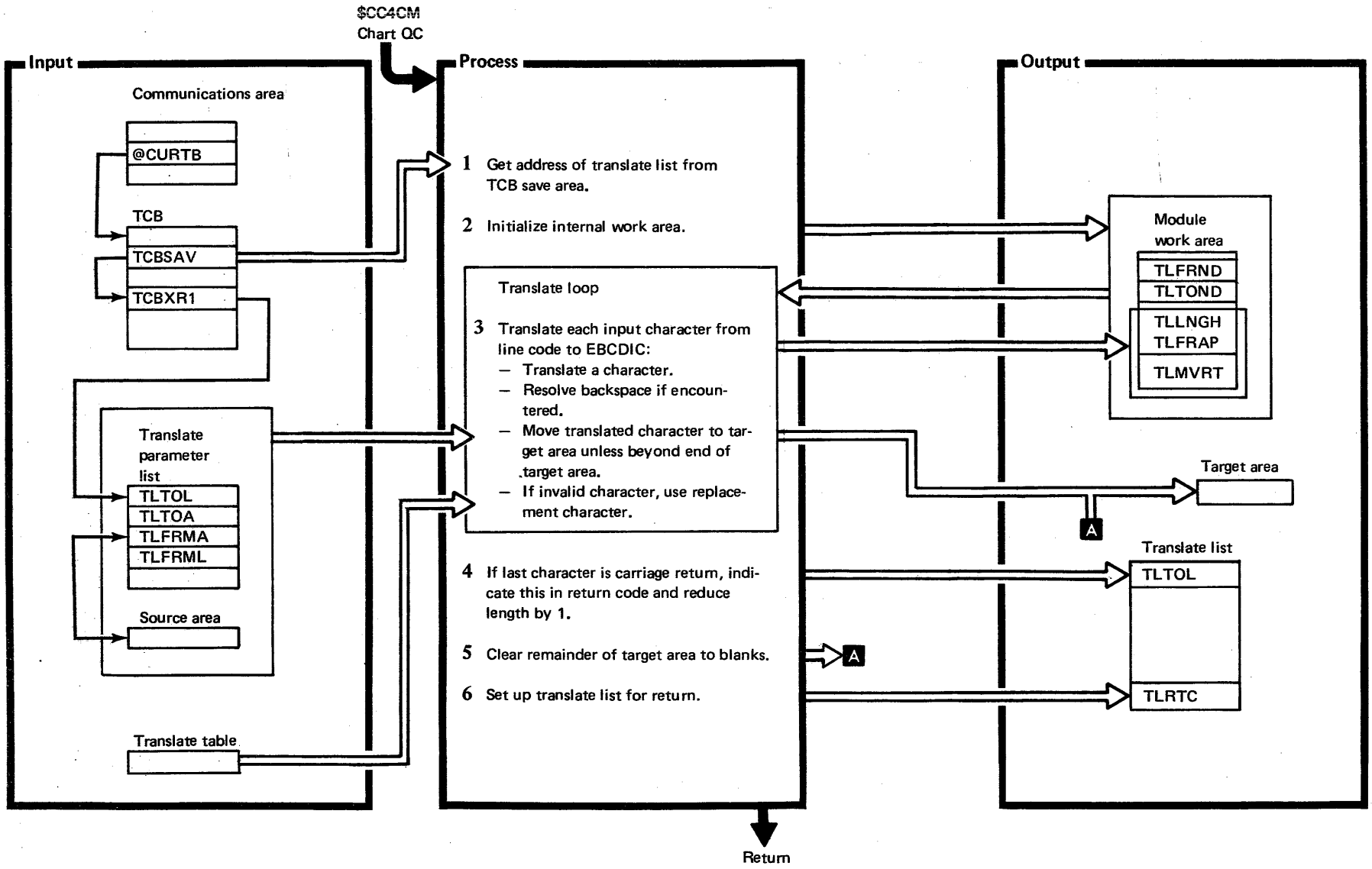


Diagram 9P.2300. $CC4Jx (Models 8, 10, and 12 Only)

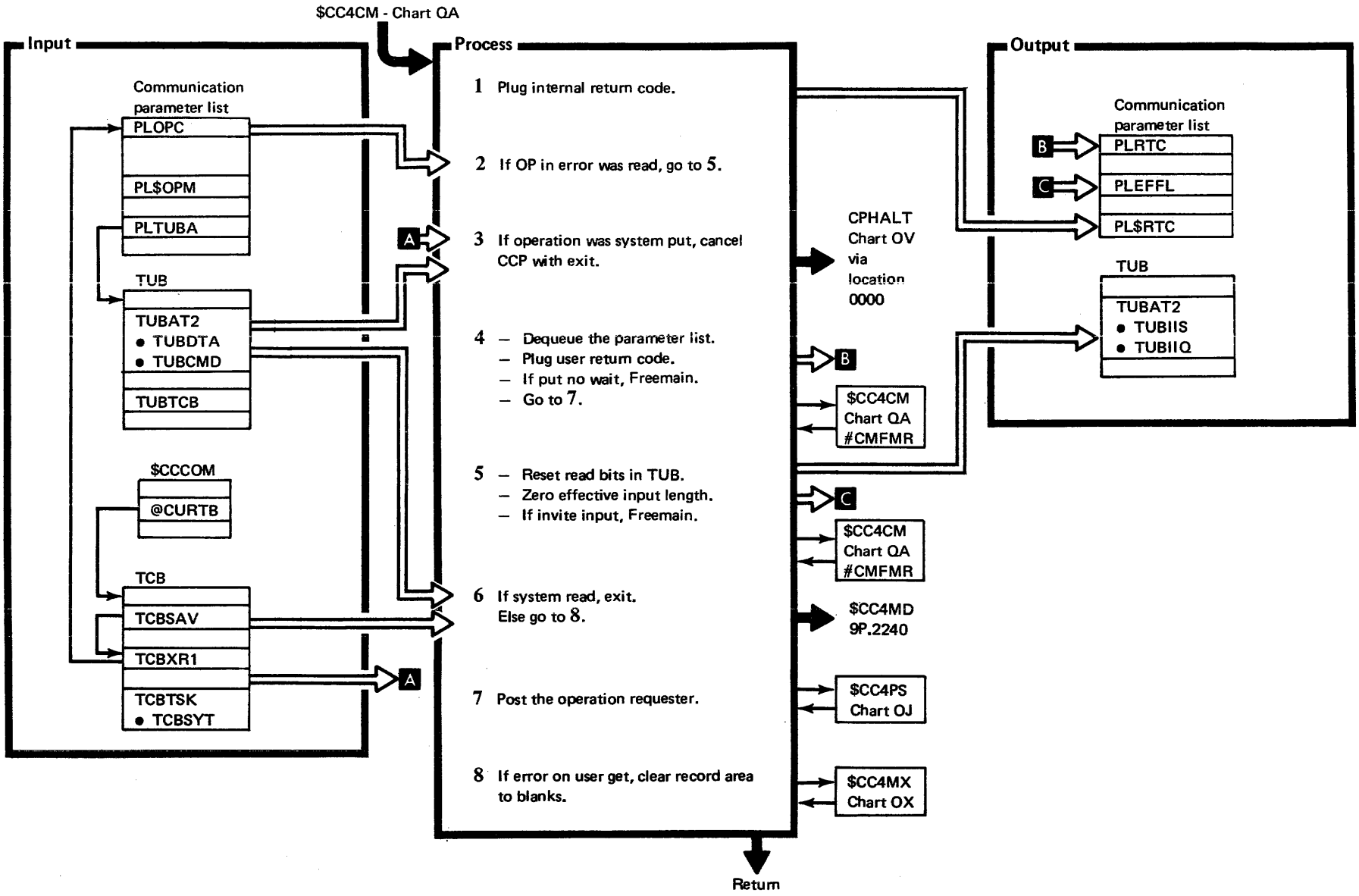


Diagram 9P.2310. $CC4WR

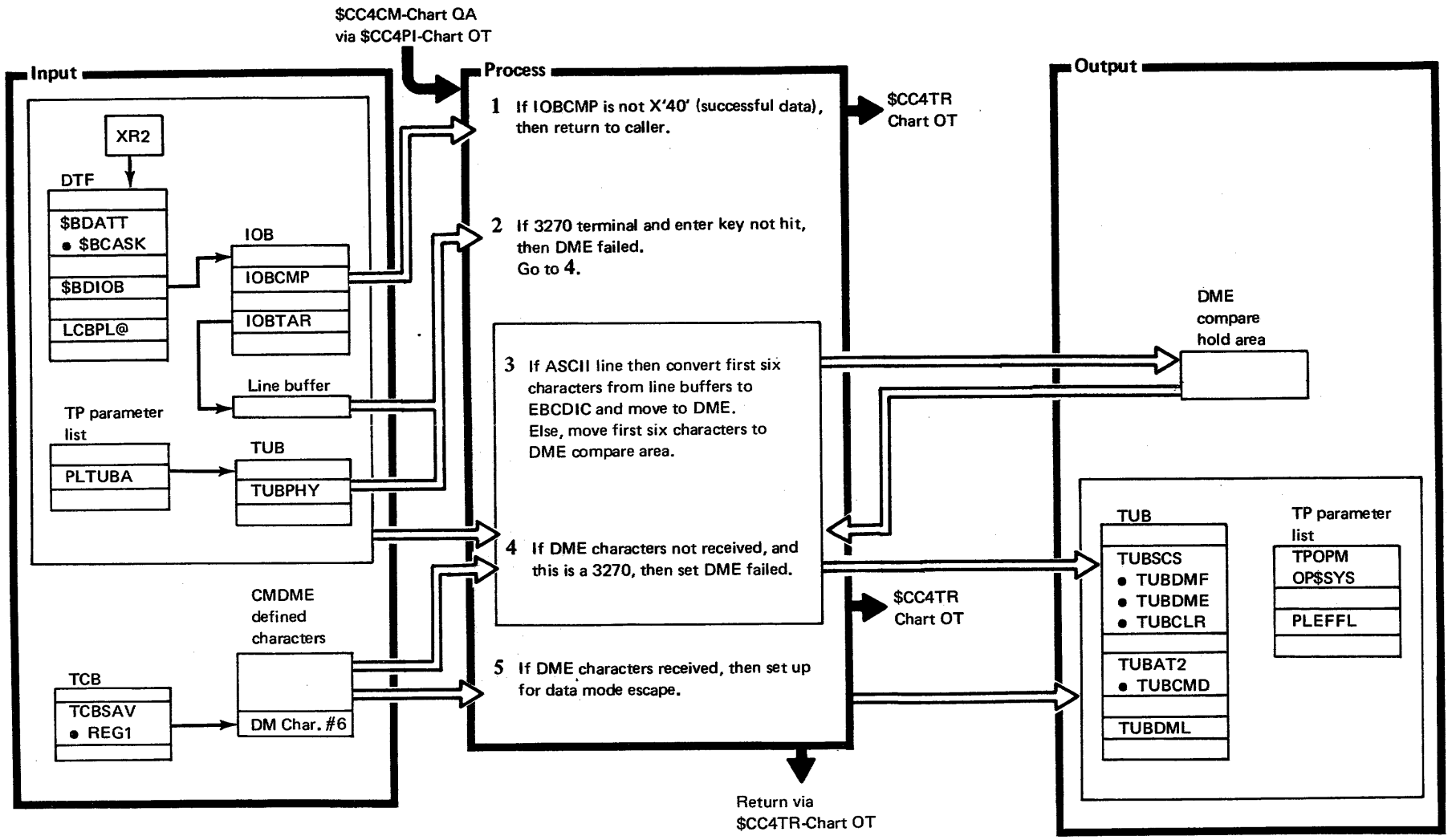


Diagram 9P.2320. $CC4BI

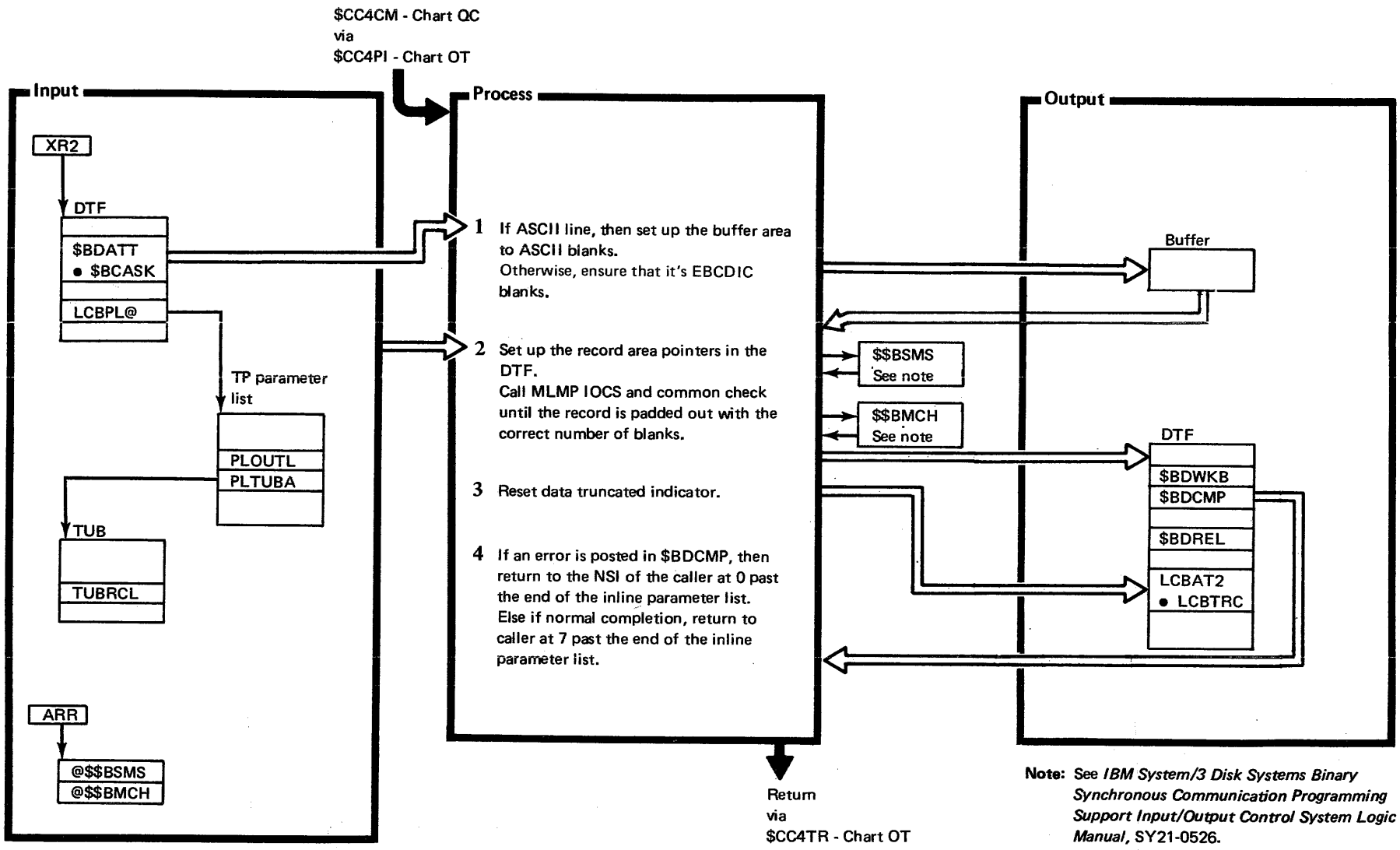


Diagram 9P.2321. $CC4BB

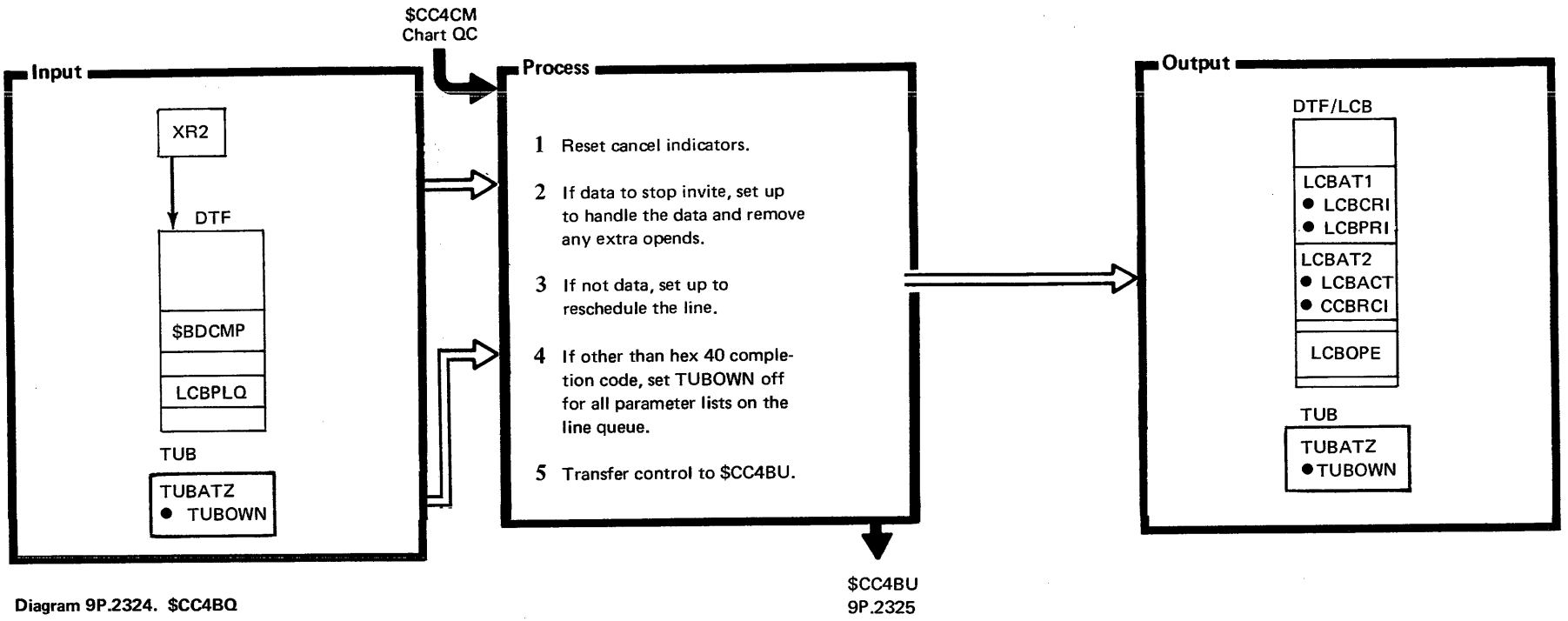
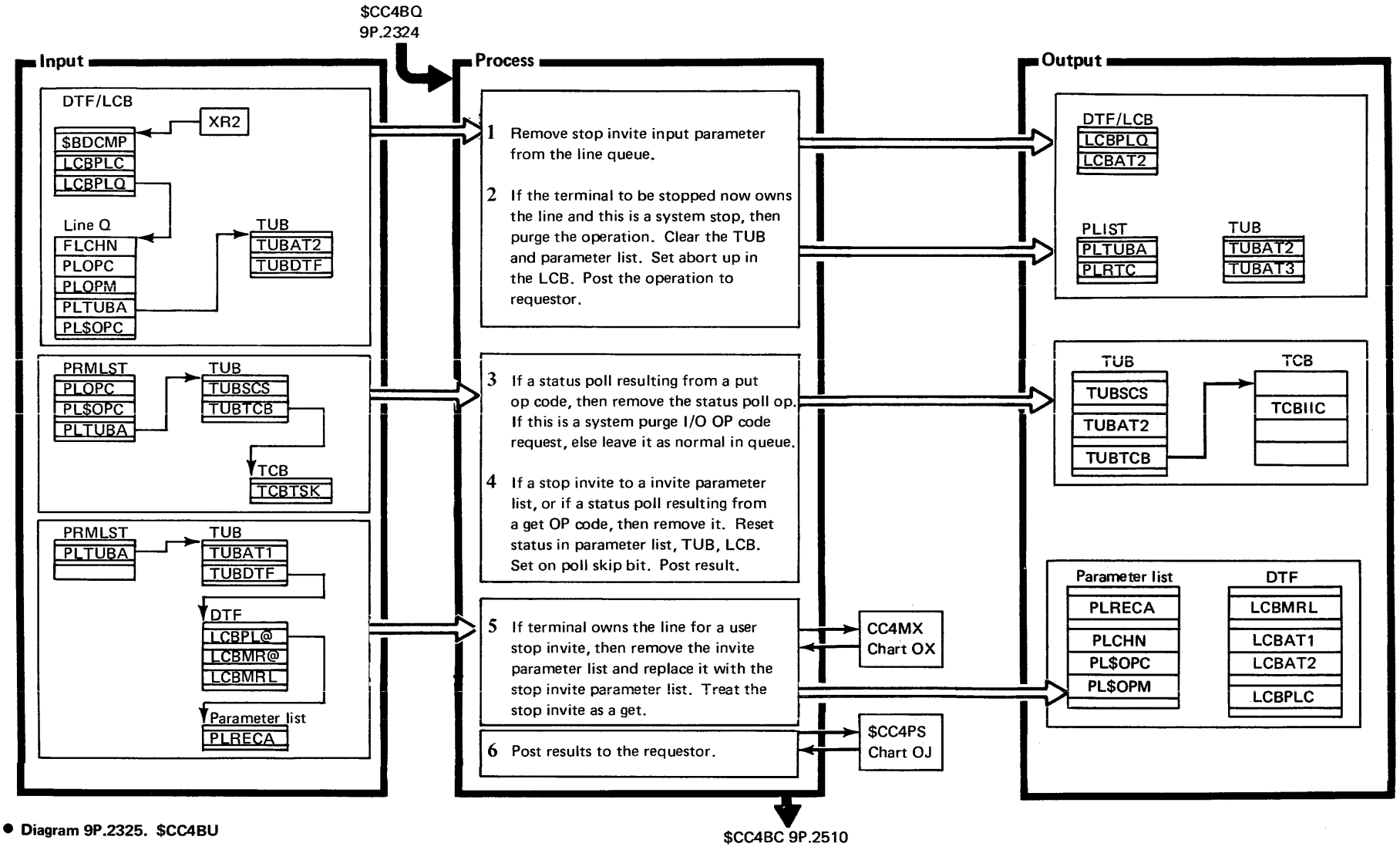


Diagram 9P.2324. $CC4BQ



● **Diagram 9P.2325. $CC4BU**

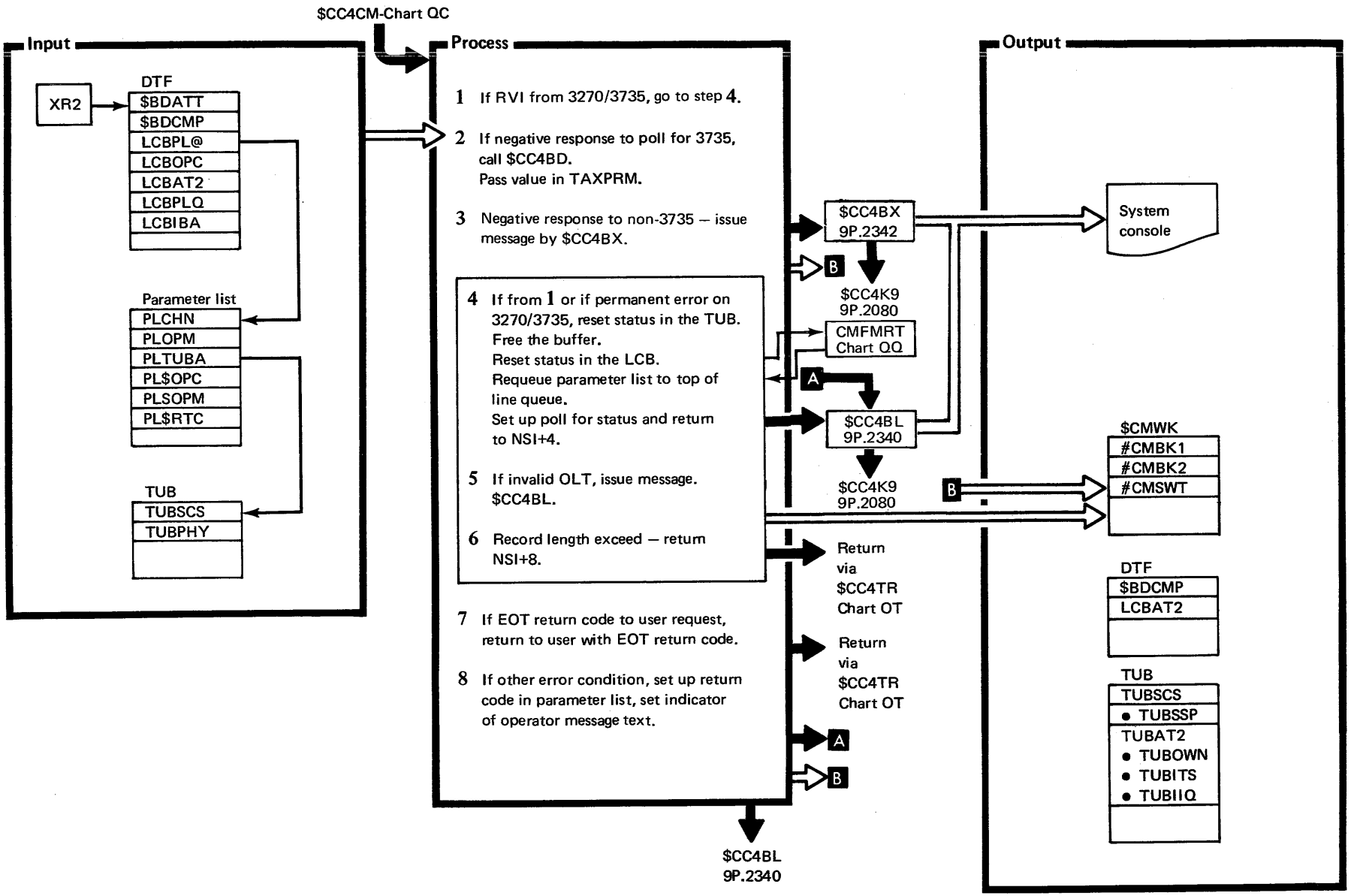


Diagram 9P.2330. $CC4BE

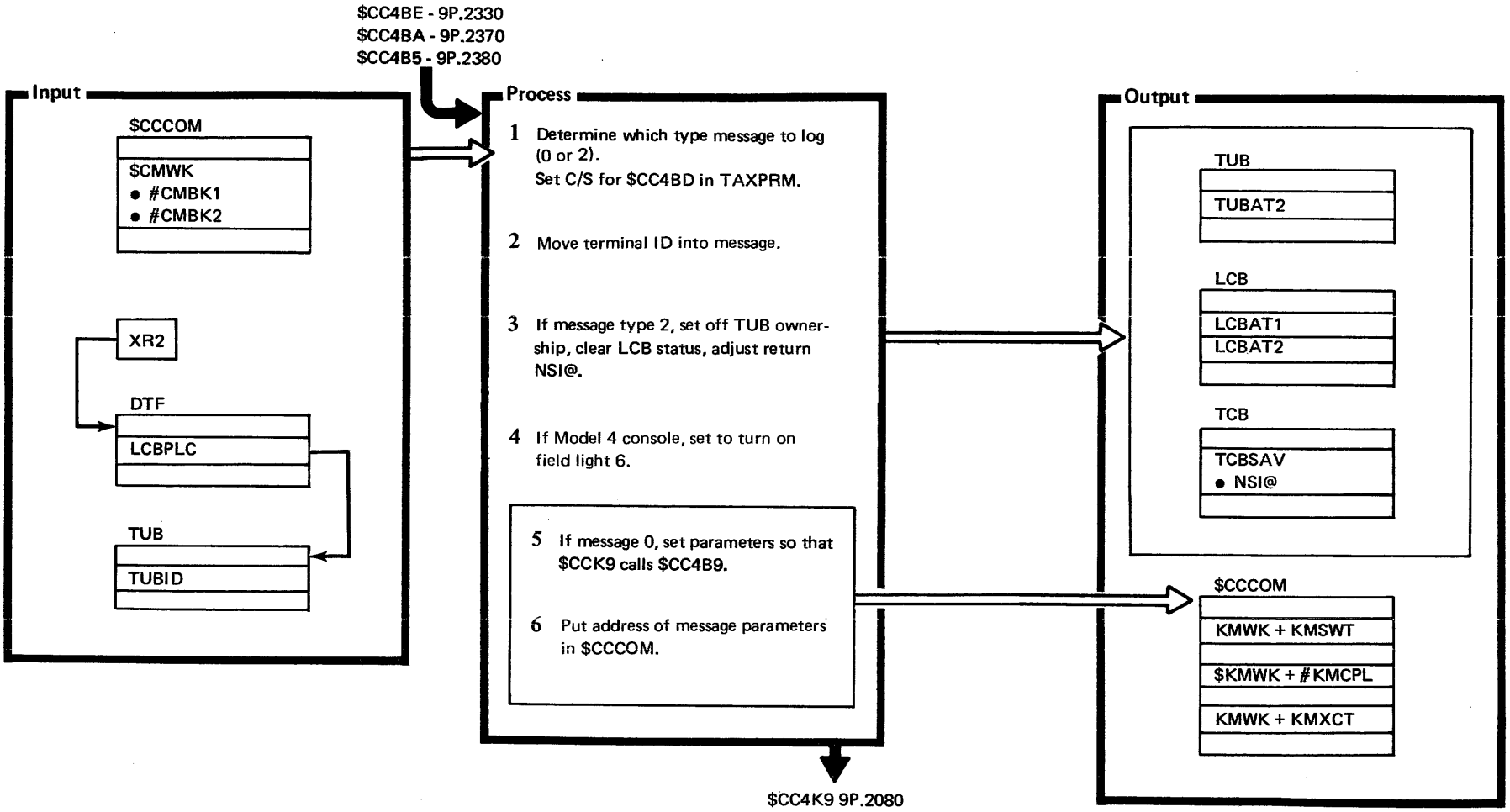


Diagram 9P.2340. $CC4BL

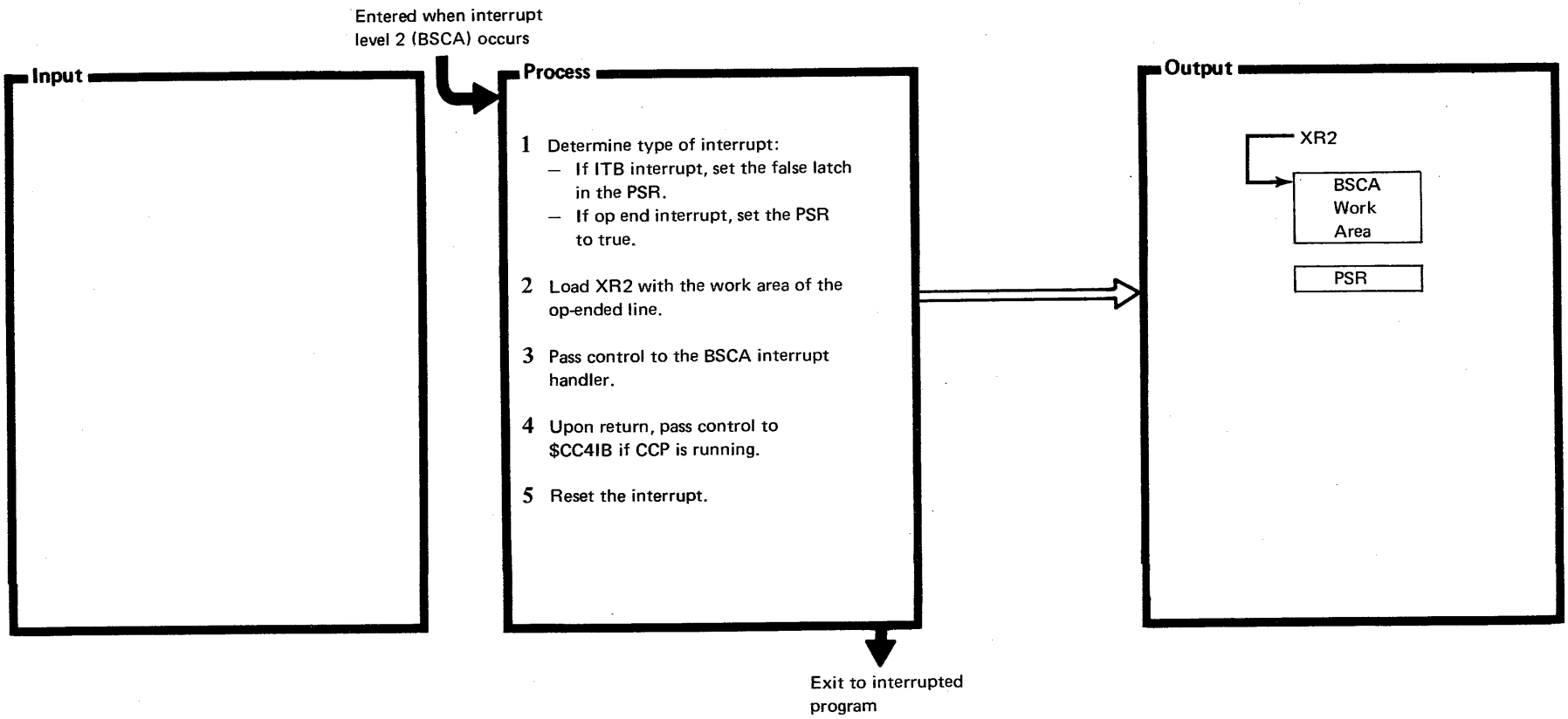


Diagram 9P.2345. First Level Interrupt Handler ($CC4BN)

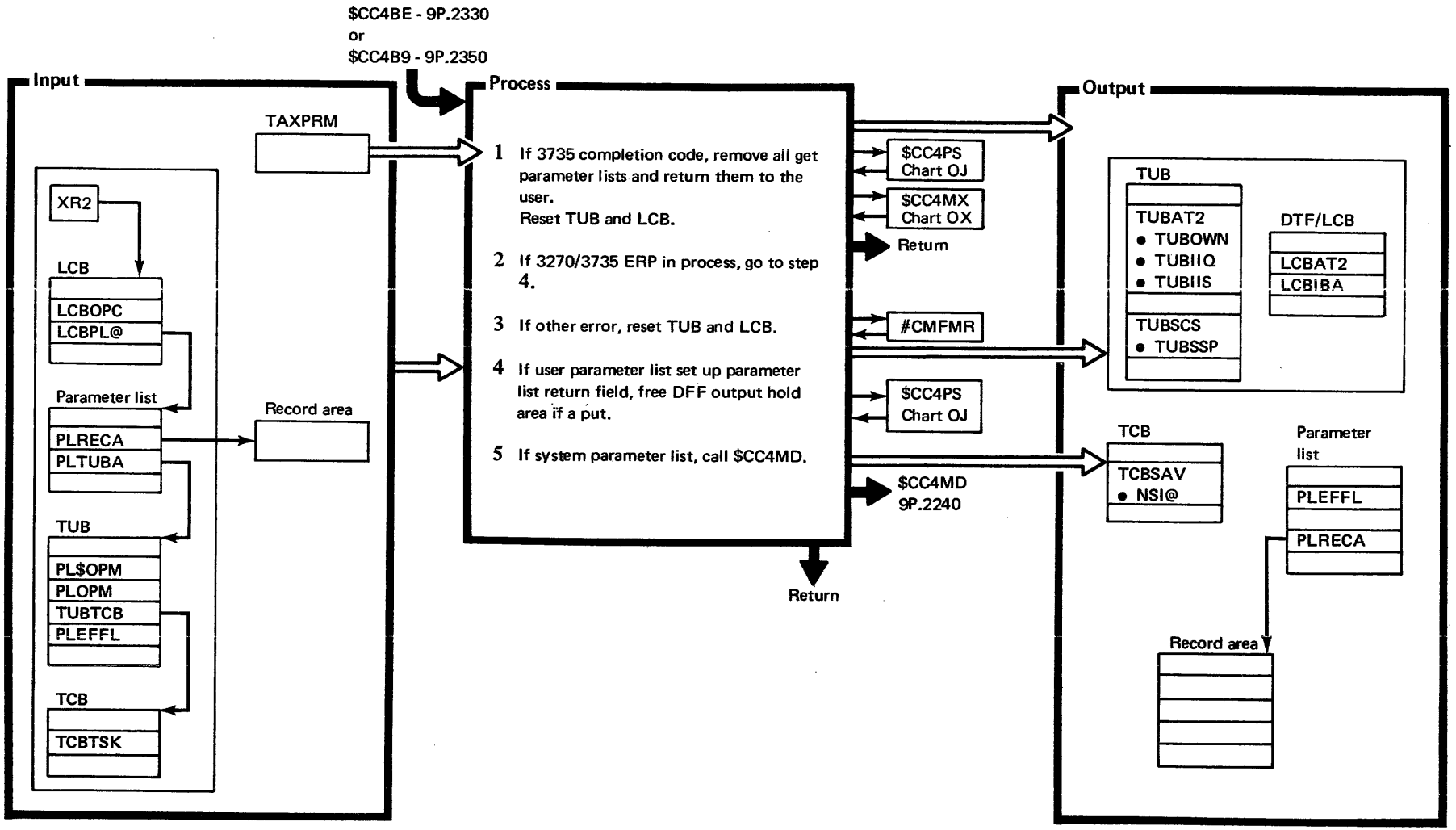


Diagram 9P.2341. $CC4BD

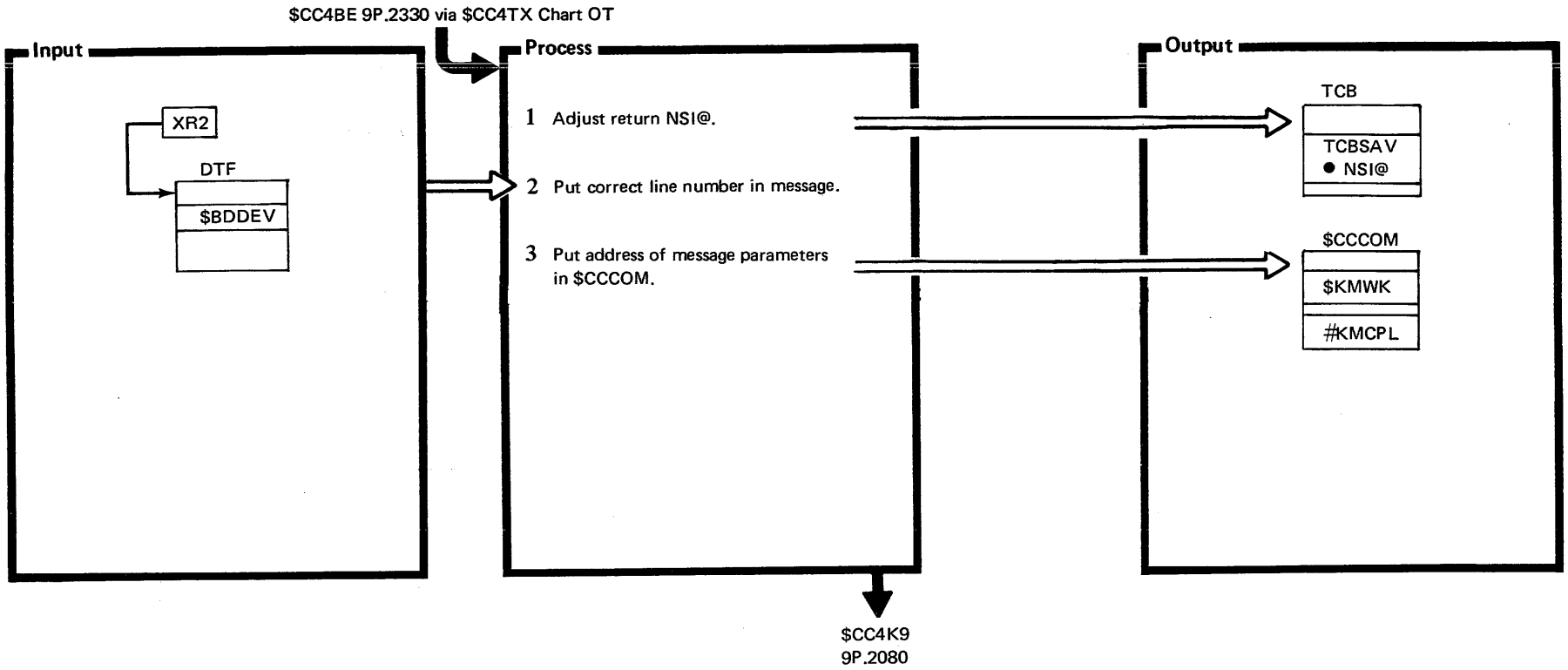


Diagram 9P.2342. $CC4BX

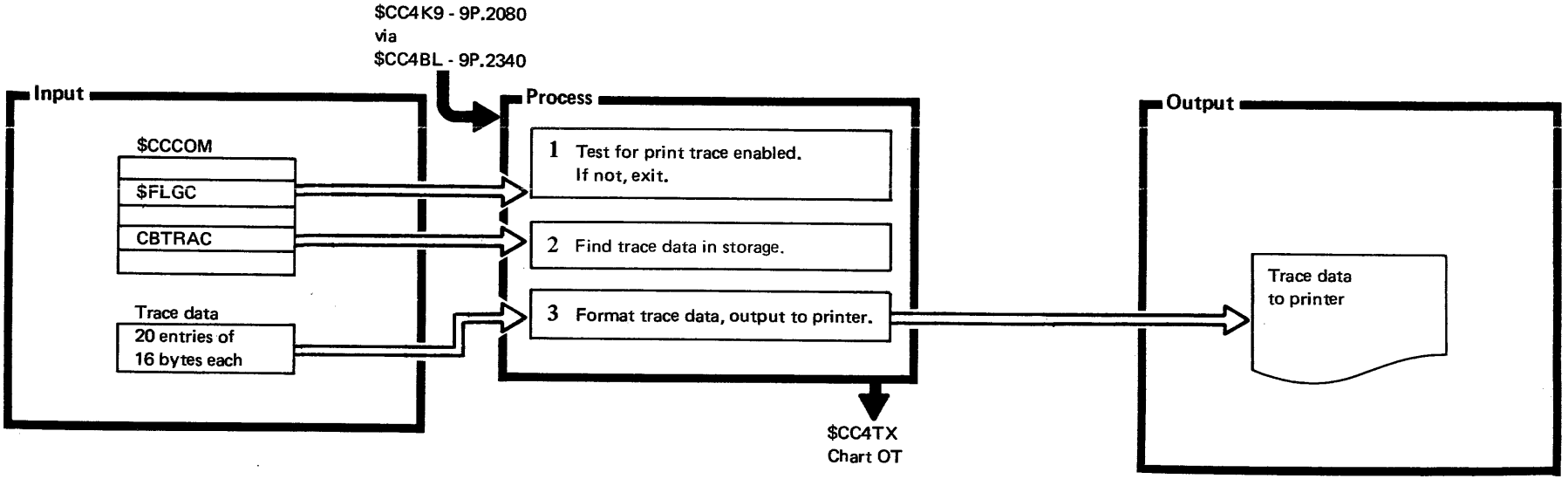


Diagram 9P.2350. $CC4B9

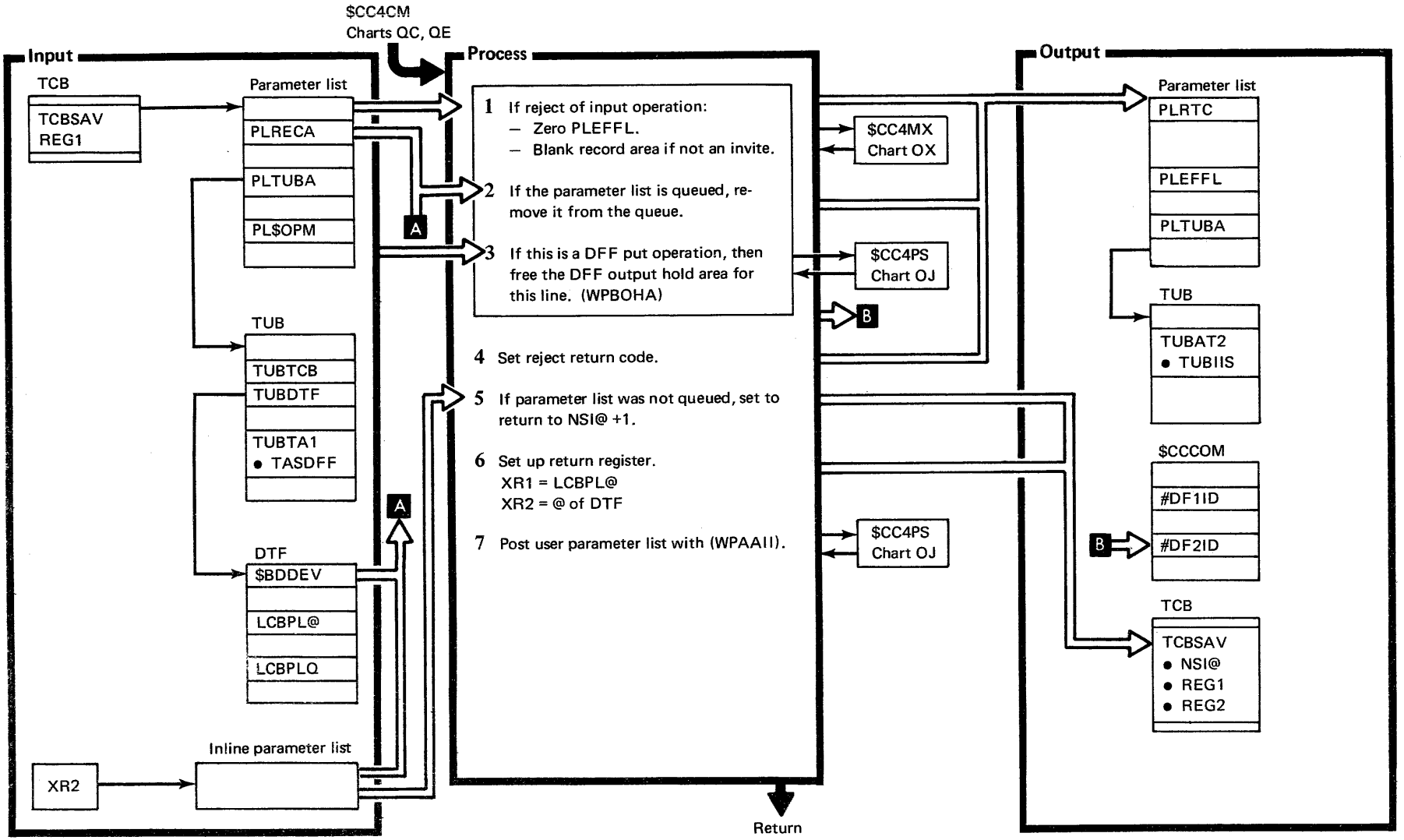


Diagram 9P.2360. $CC4BR

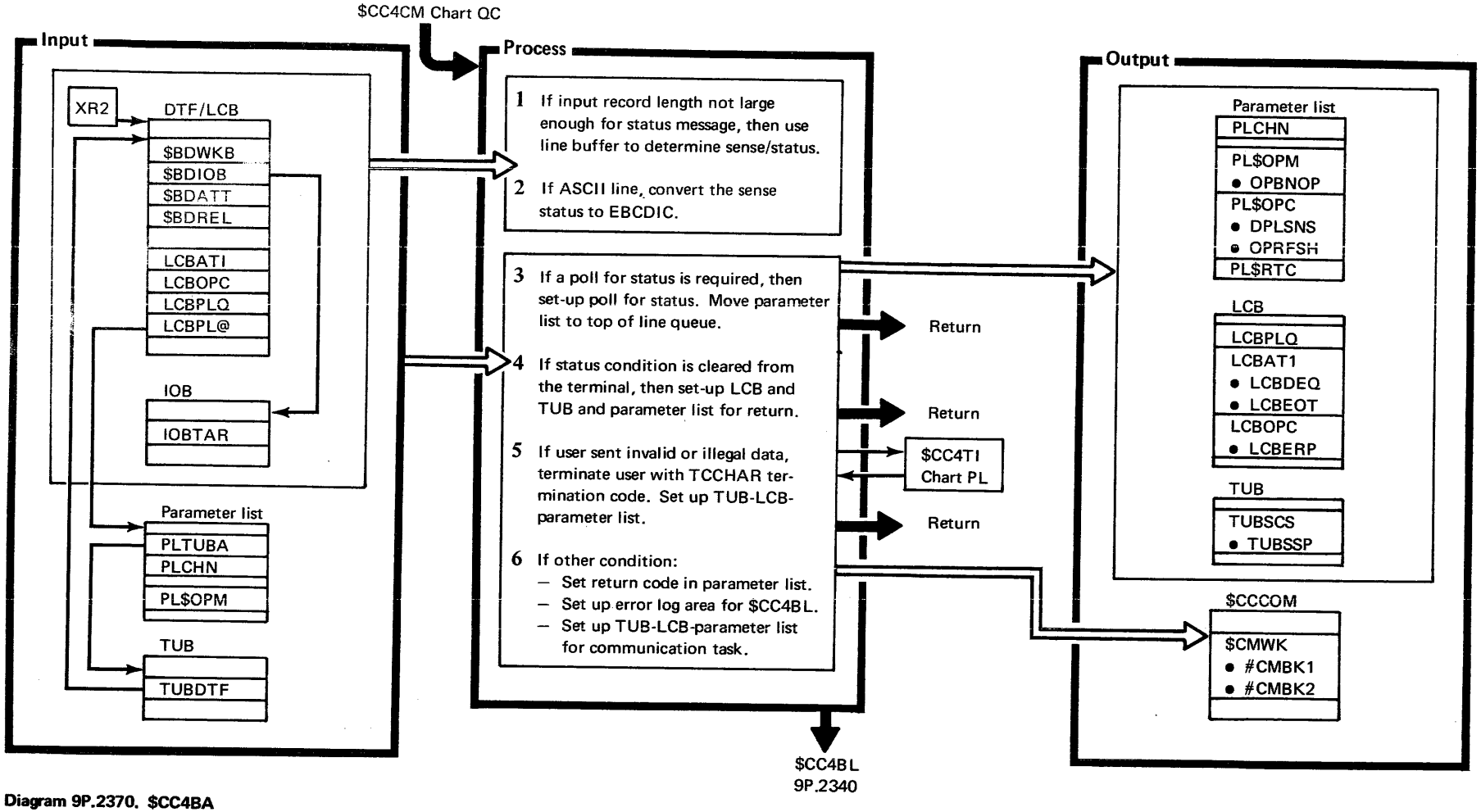


Diagram 9P.2370. $CC4BA

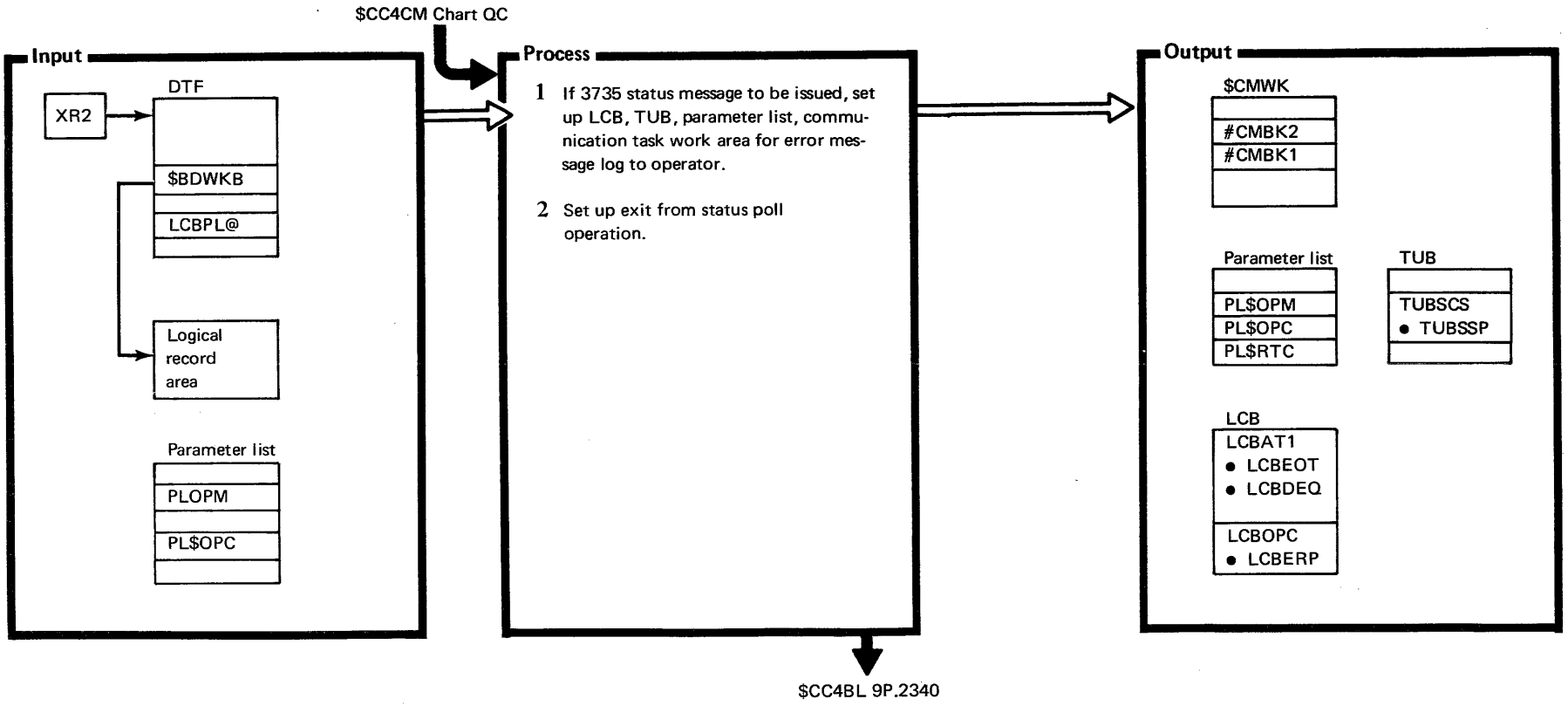


Diagram 9P.2380. $CC4B5

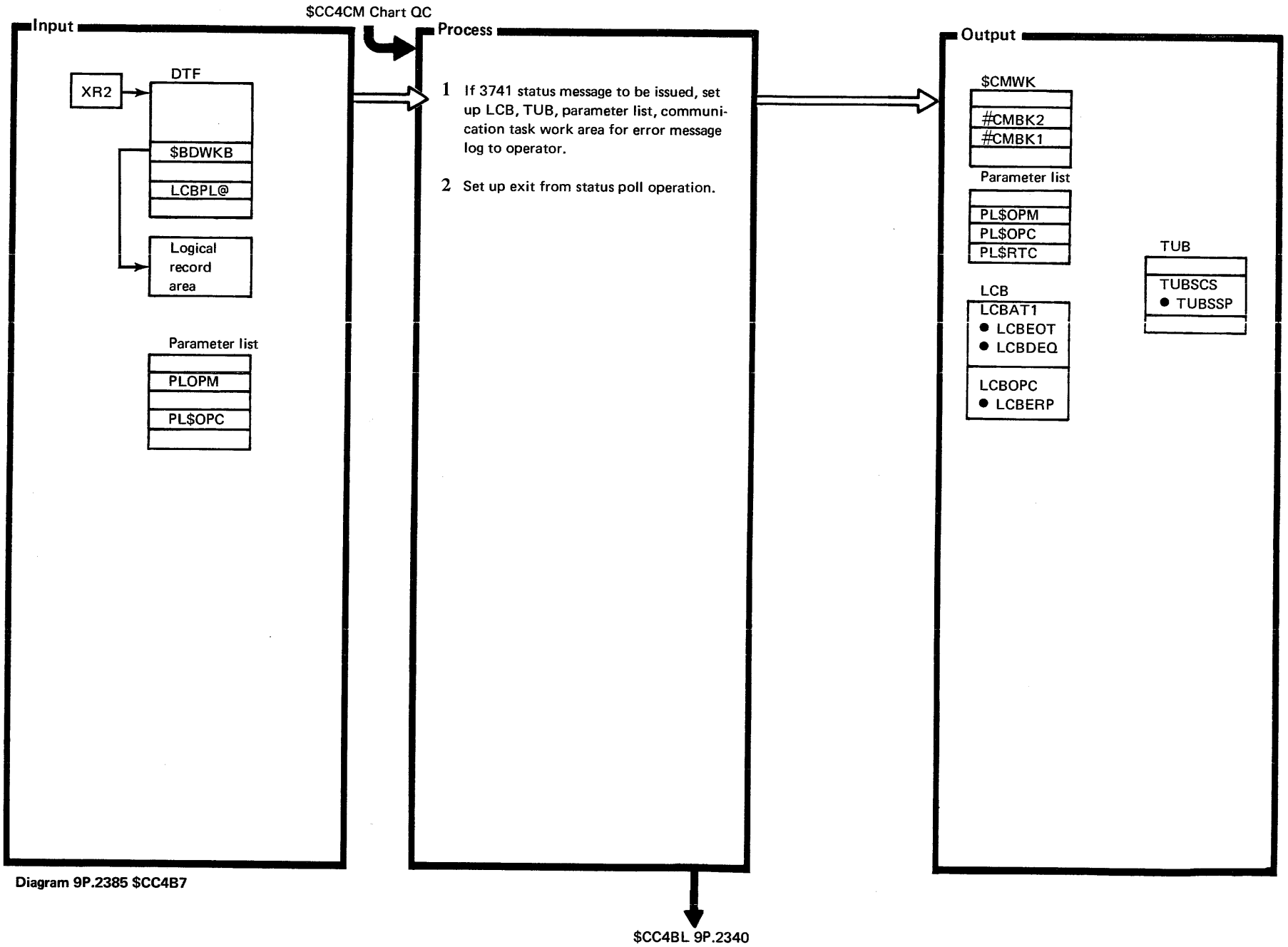


Diagram 9P.2385 $CC4B7

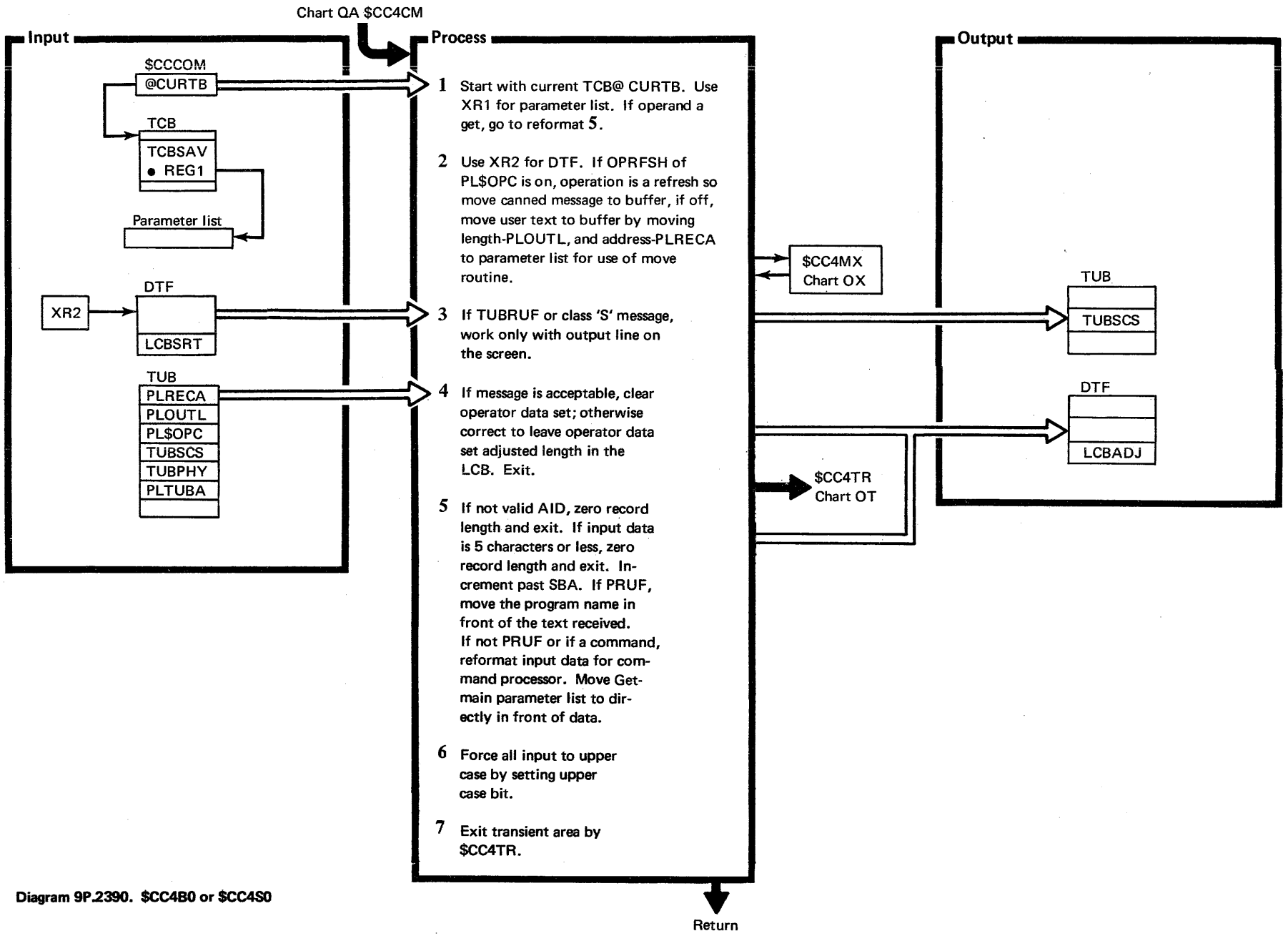


Diagram 9P.2390. $CC4B0 or $CC4S0

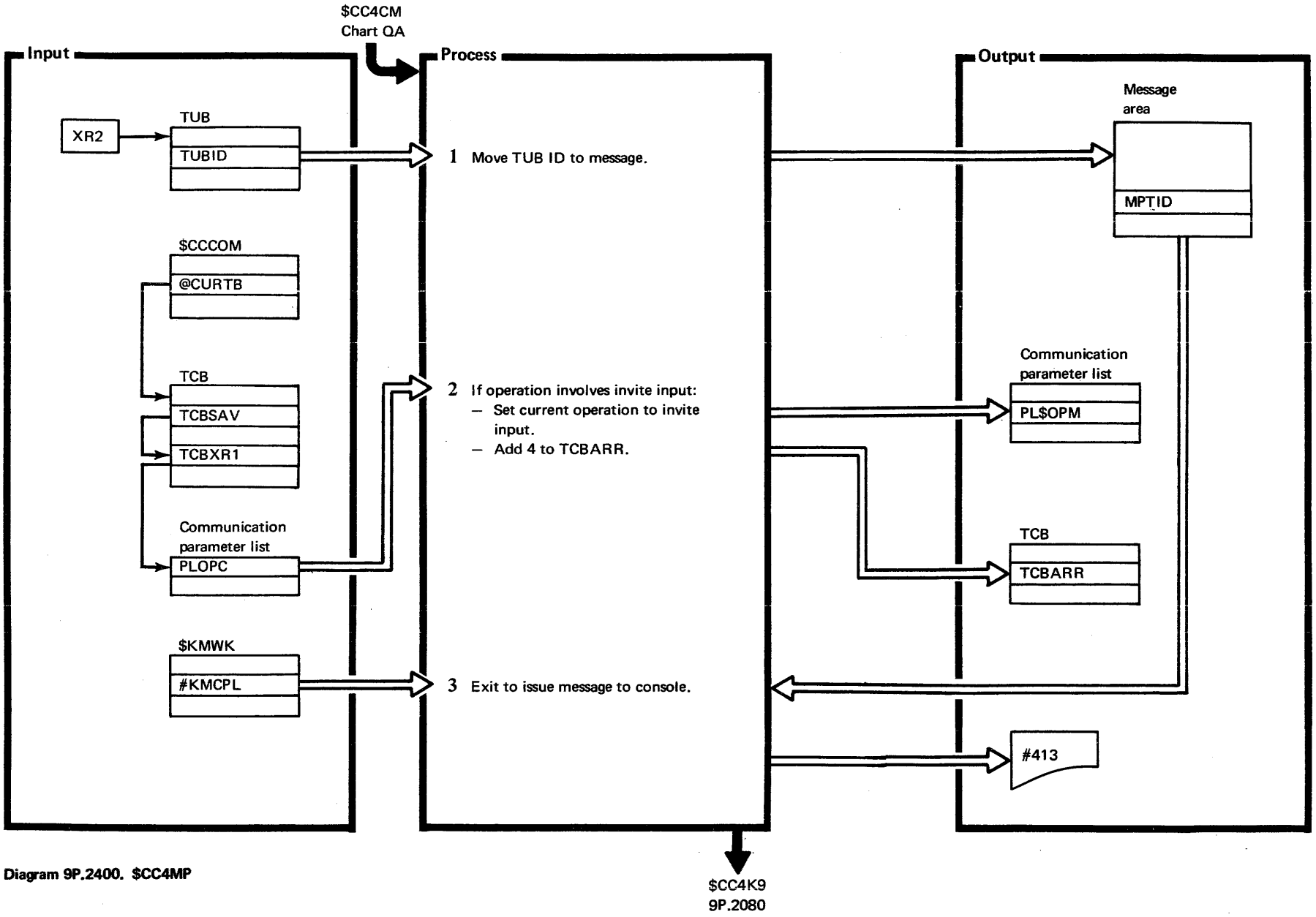


Diagram 9P.2400. $CC4MP

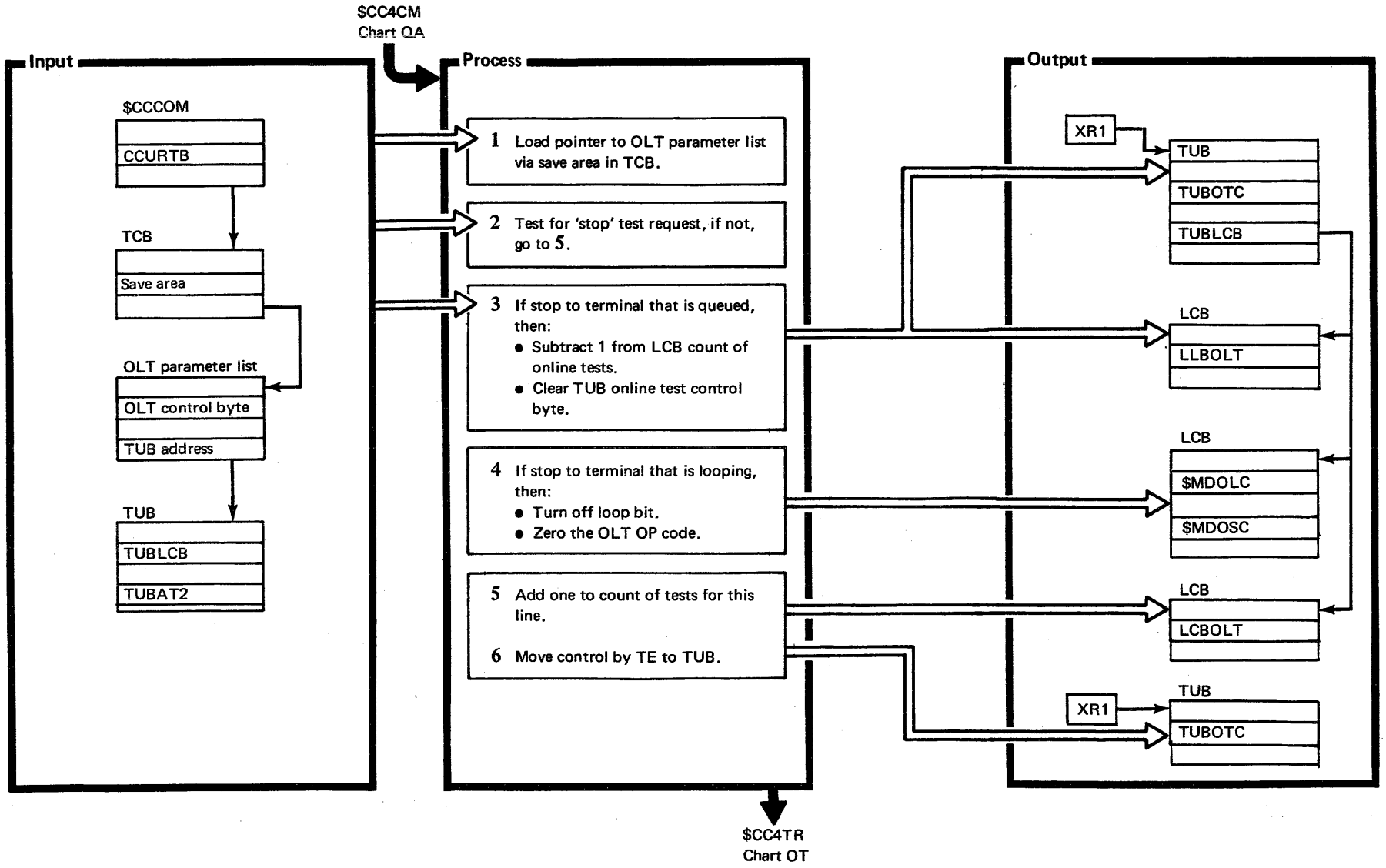
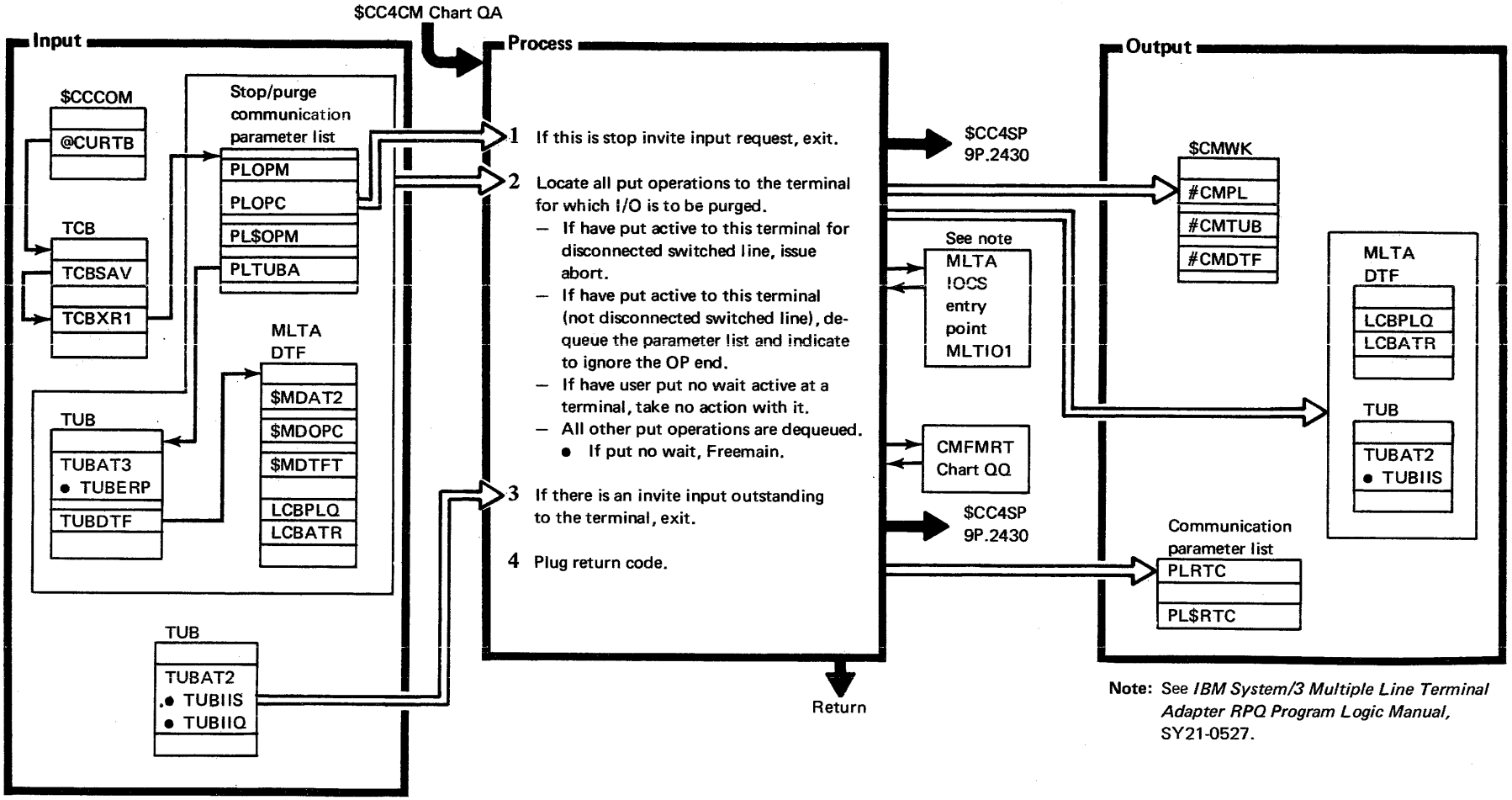


Diagram 9P.2410. $CC4T1



Note: See *IBM System/3 Multiple Line Terminal Adapter RPQ Program Logic Manual*, SY21-0527.

Diagram 9P.2420. $CC4PG

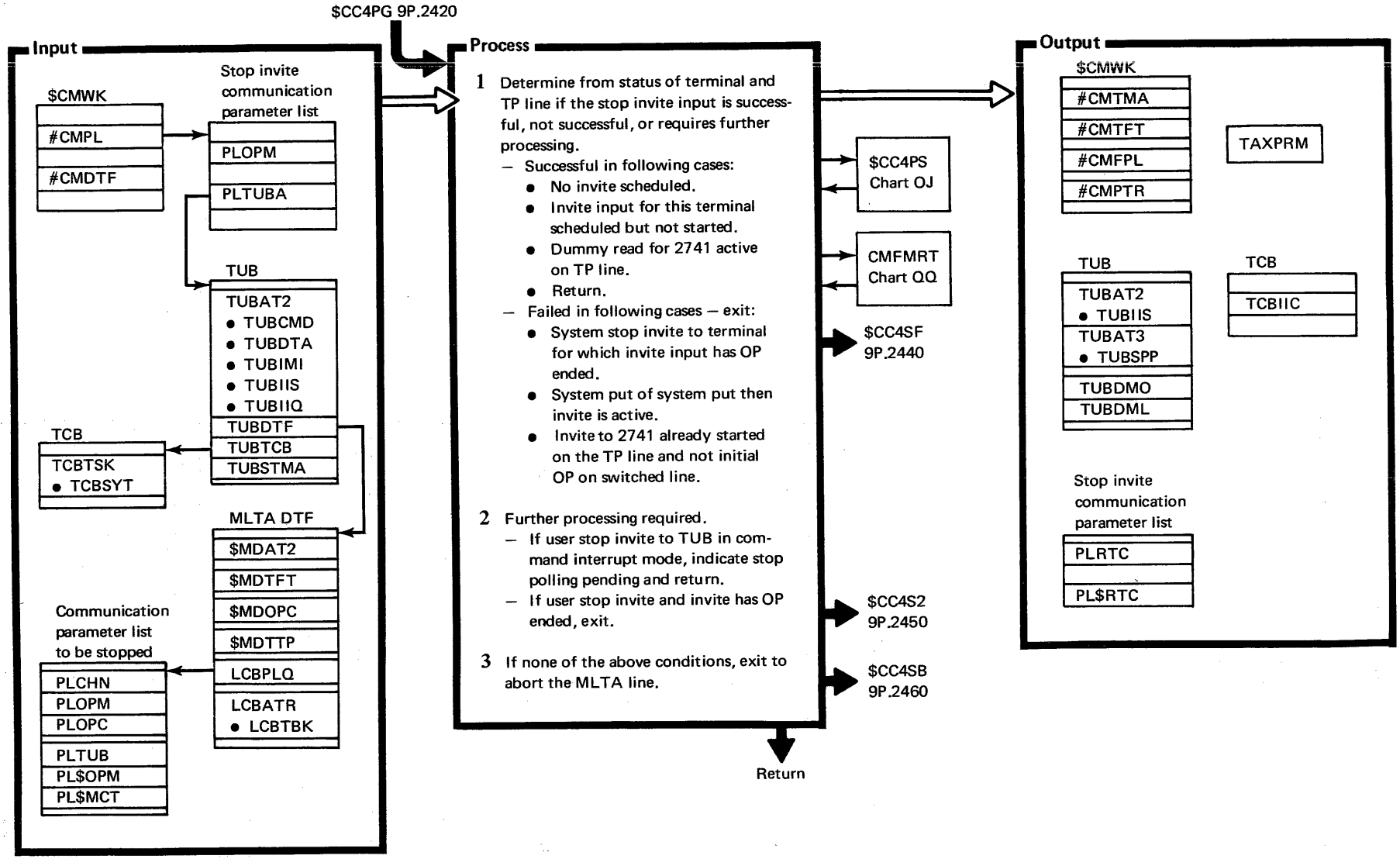


Diagram 9P.2430. $CC4SP (Models 8, 10, and 12 Only)

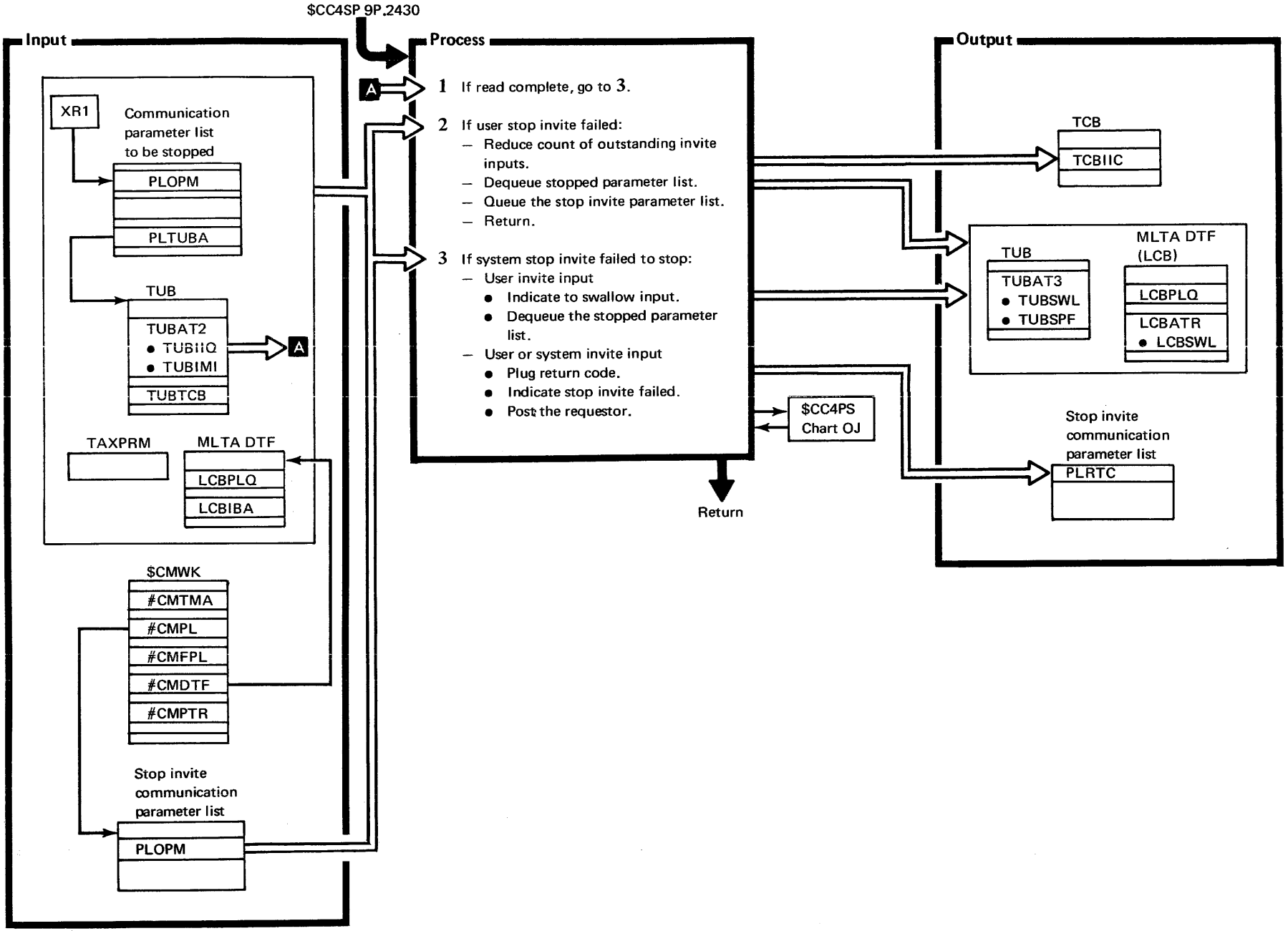


Diagram 9P.2440. $CC4SF

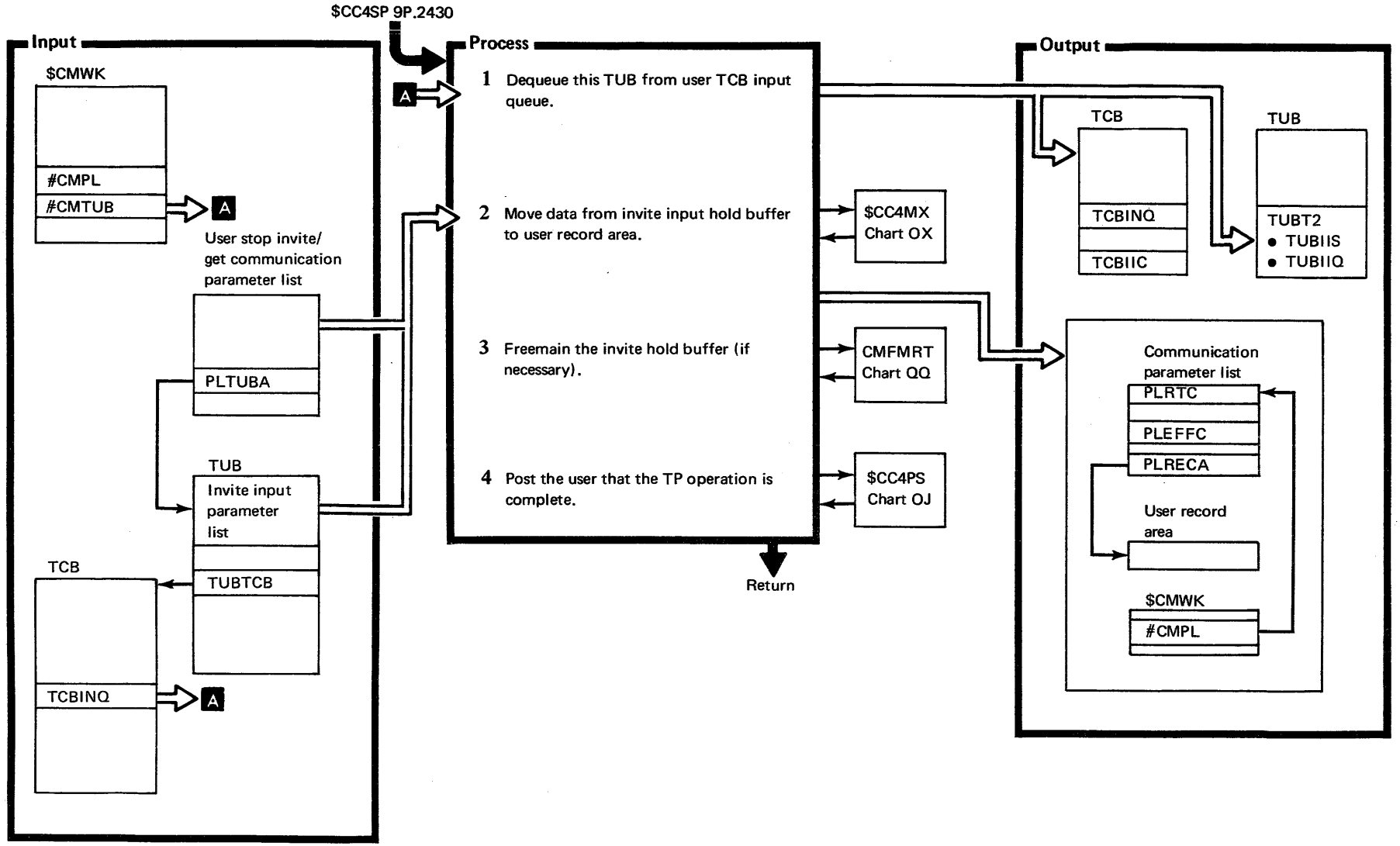
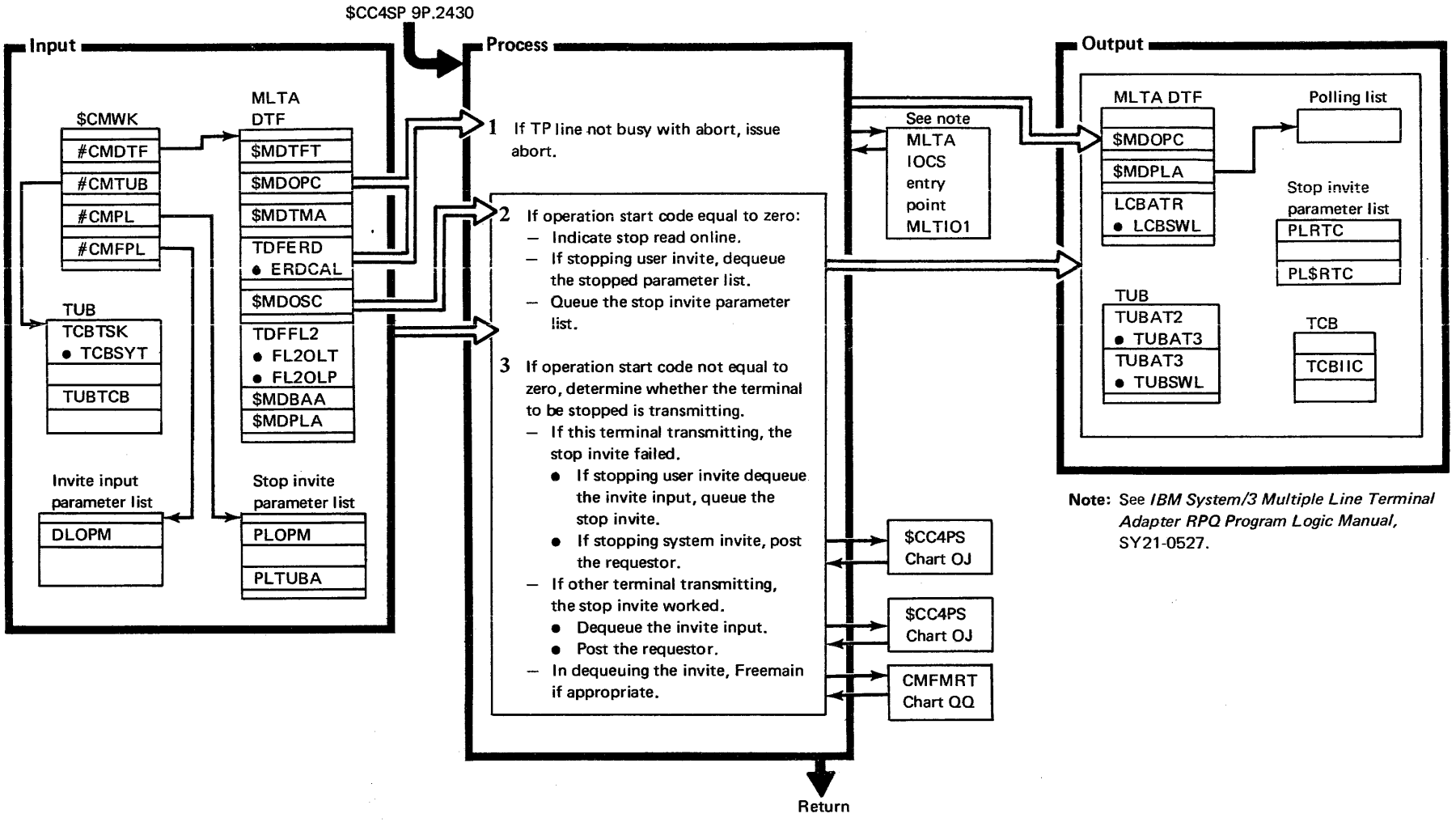


Diagram 9P.2450. $CC4S2



Note: See *IBM System/3 Multiple Line Terminal Adapter RPQ Program Logic Manual*, SY21-0527.

Diagram 9P.2460. $CC4SB (Models 8, 10, and 12 Only)

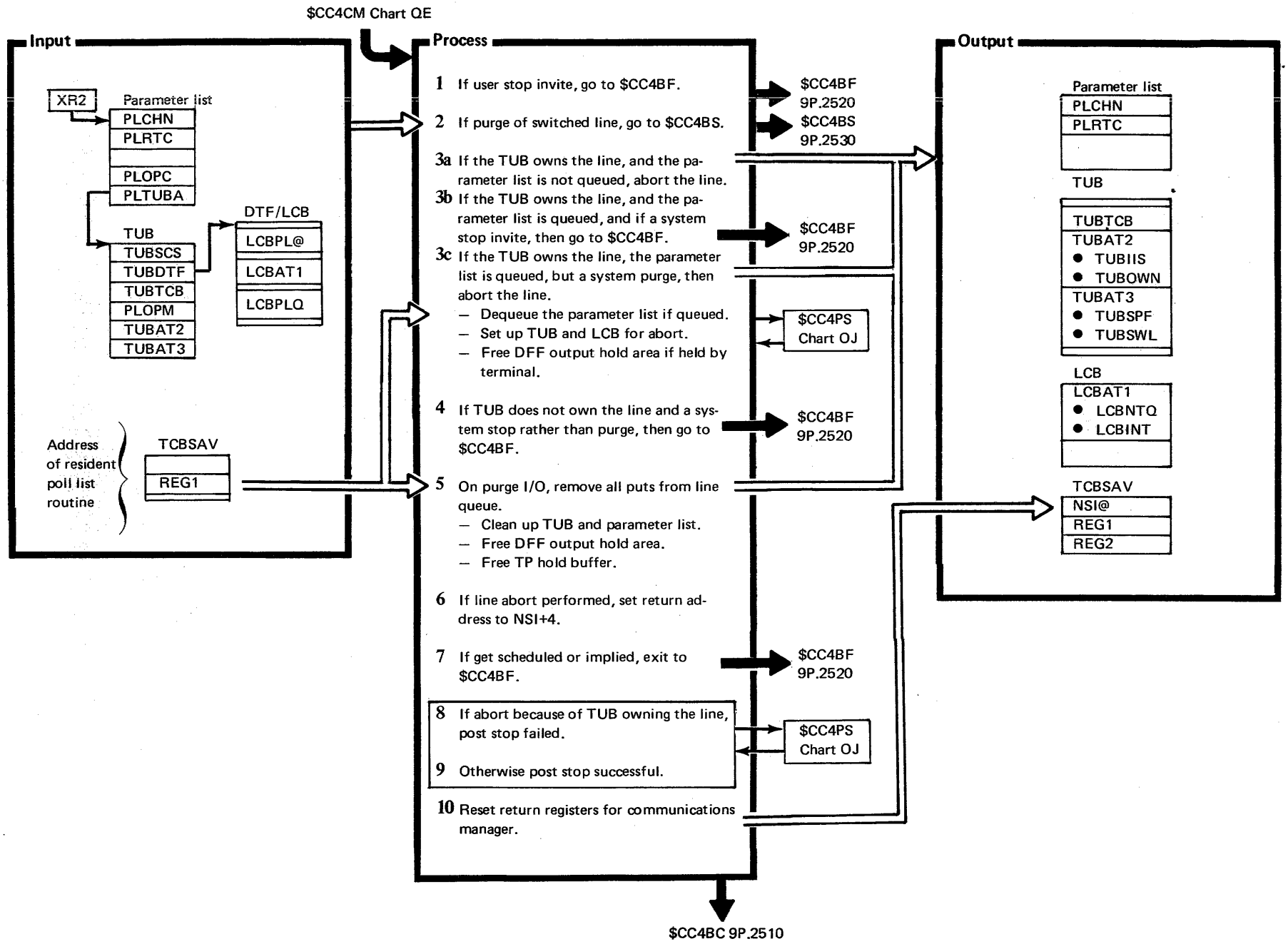


Diagram 9P.2500 $CC4BP

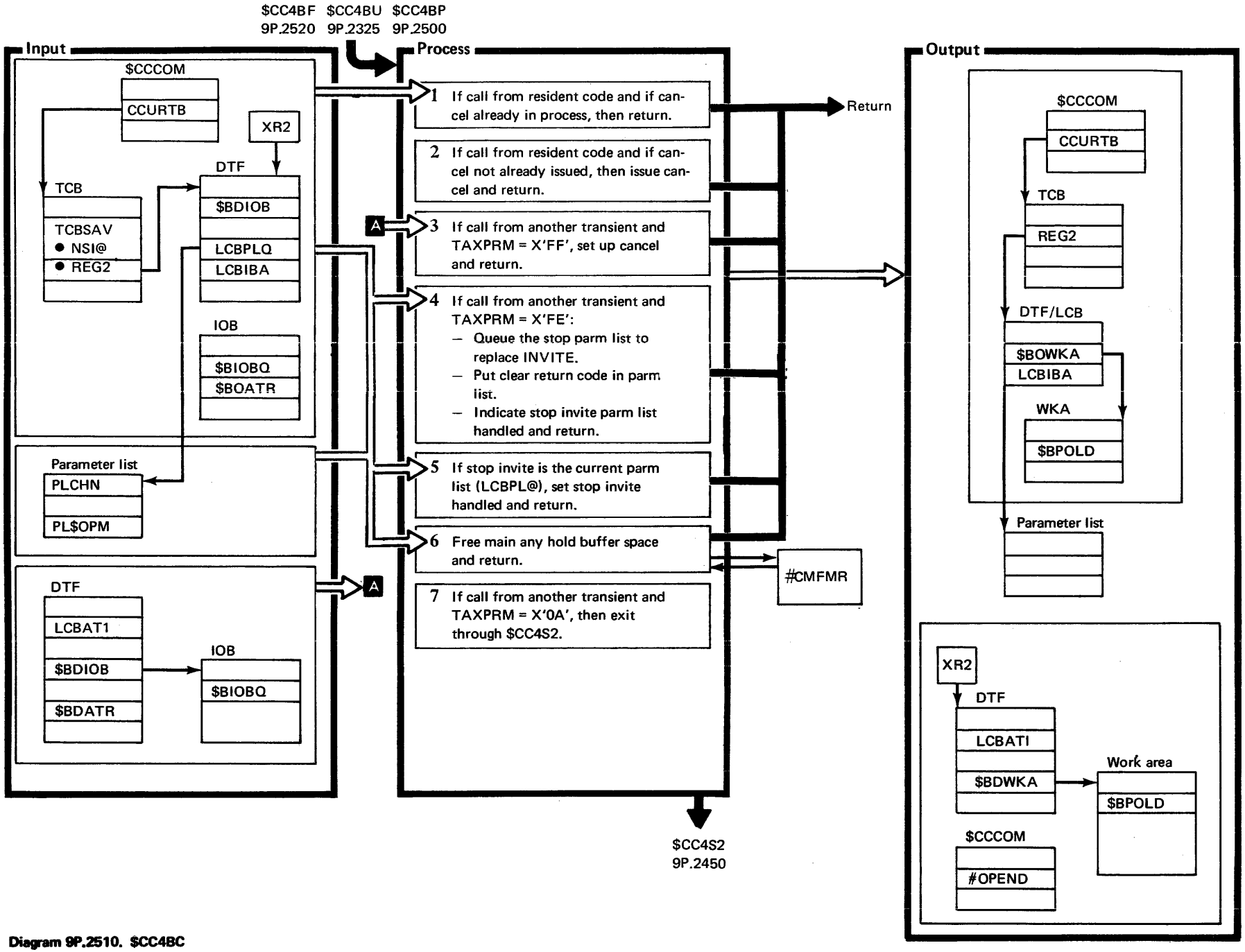


Diagram 9P.2510. $CC4BC

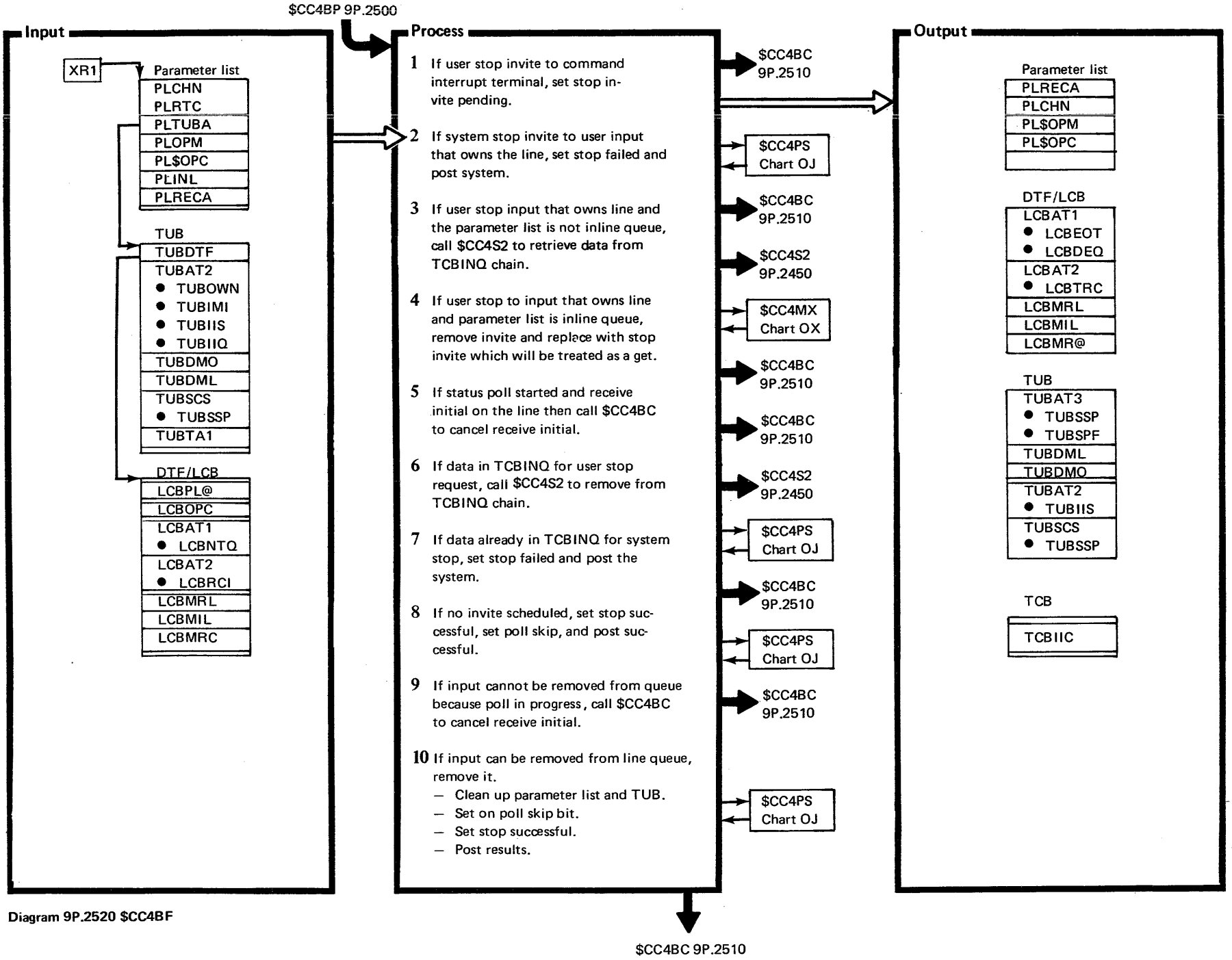


Diagram 9P.2520 $CC4BF

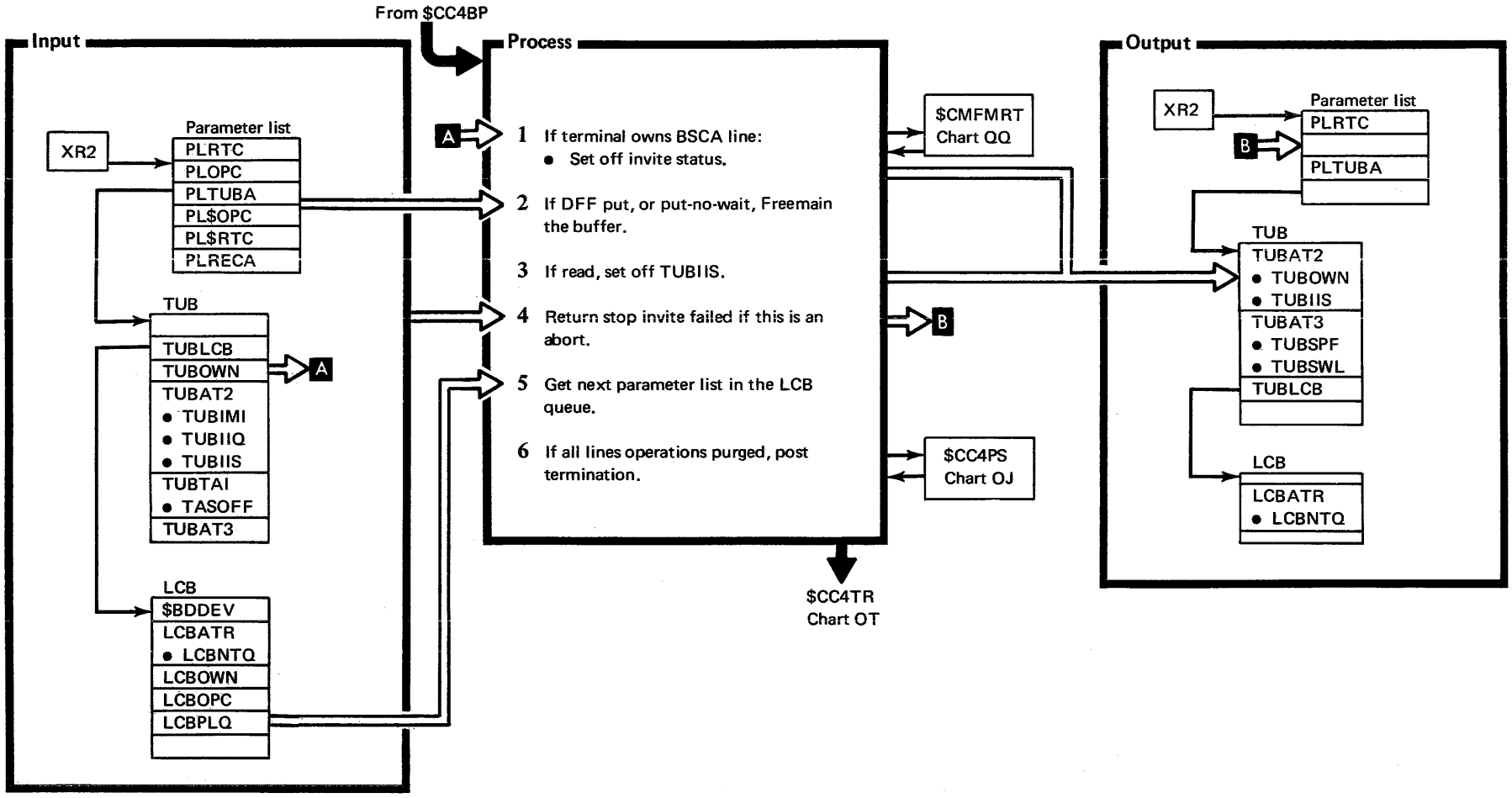
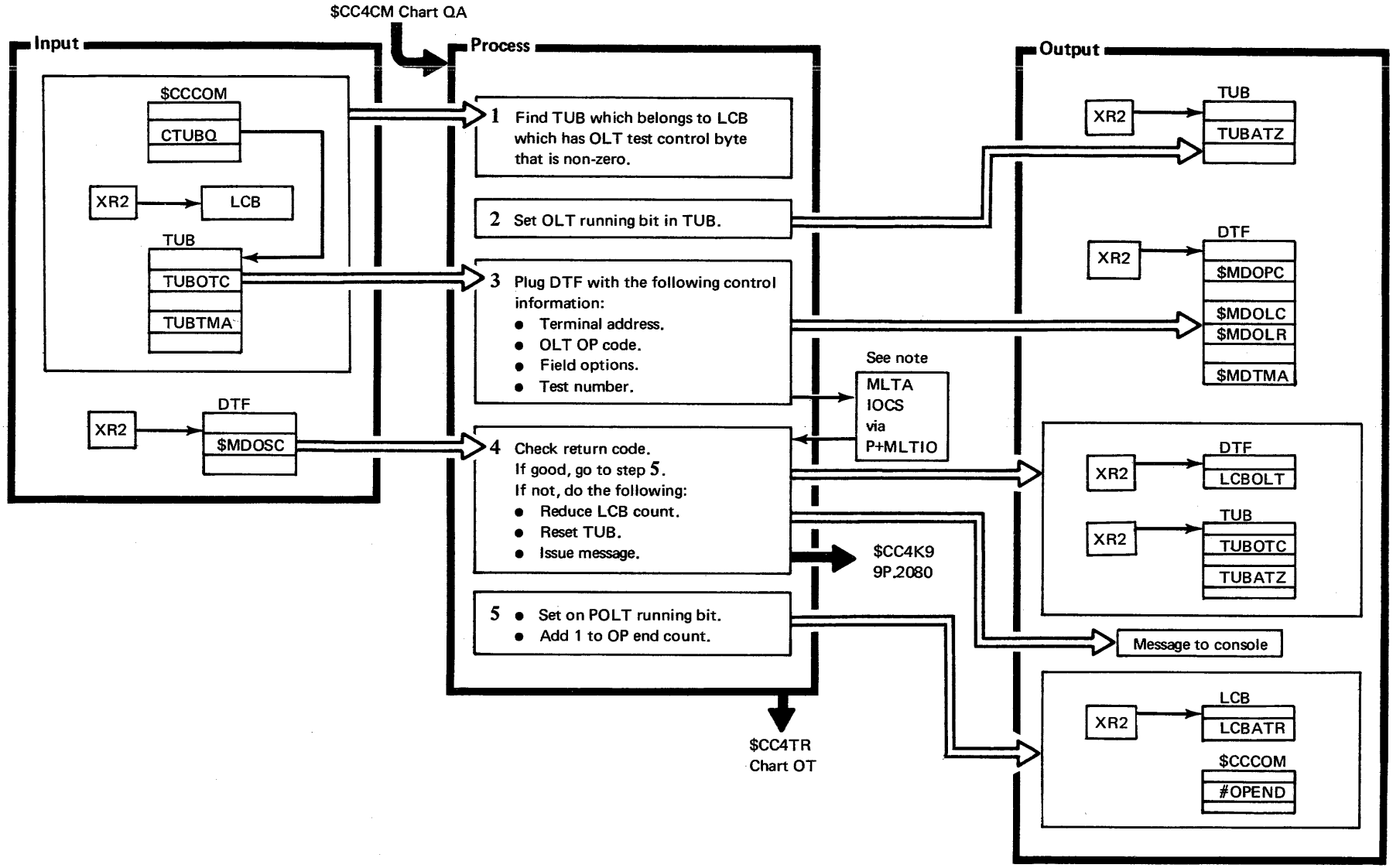


Diagram 9P.2530. $CC4BS



Note: See *IBM System/3 Multiple Line Terminal Adapter RPQ Program Logic Manual*, SY21-0527.

Diagram 9P.2600. $CC4MT (Models 8, 10, and 12 Only)

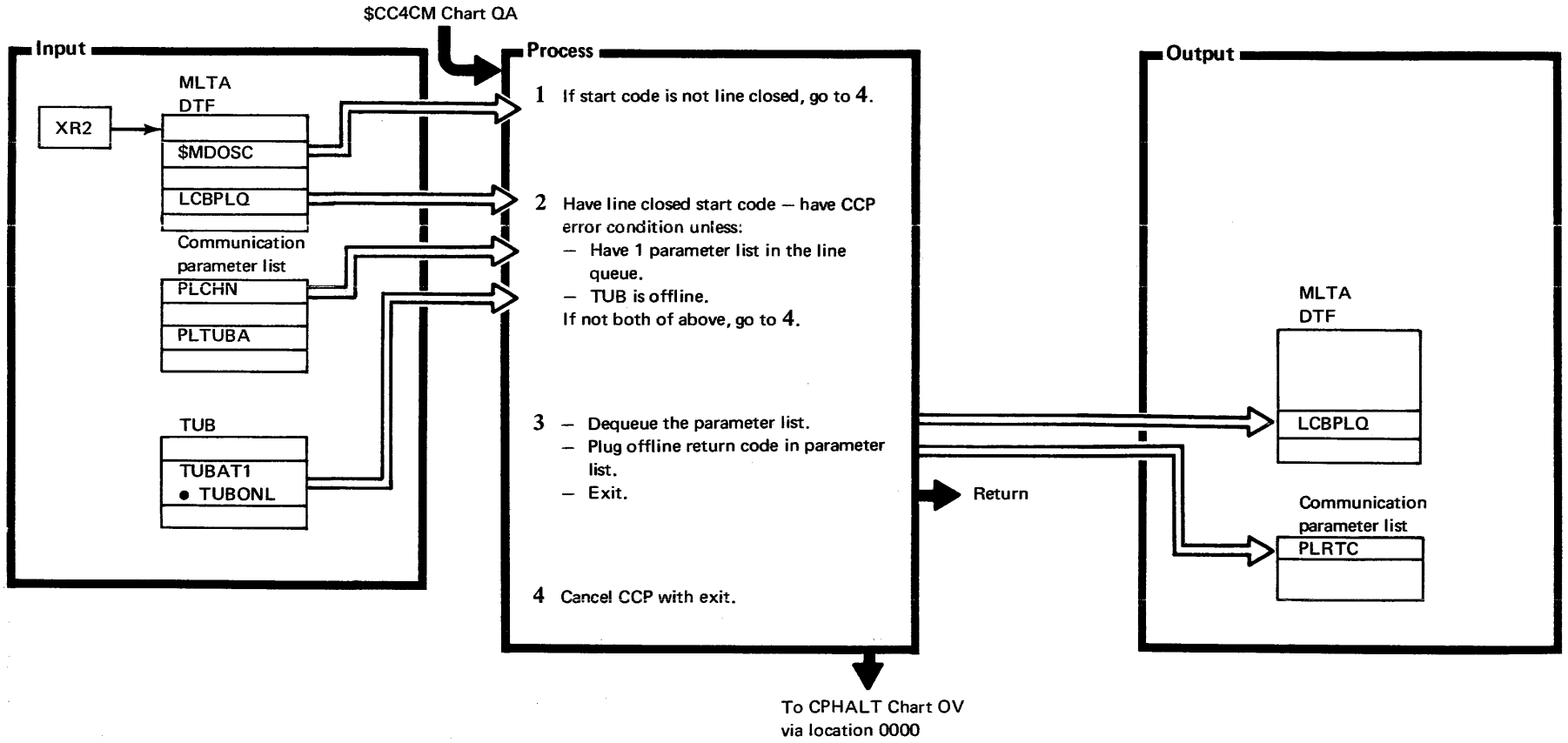


Diagram 9P.2610. $CC4SC (Models 8, 10, and 12 Only)

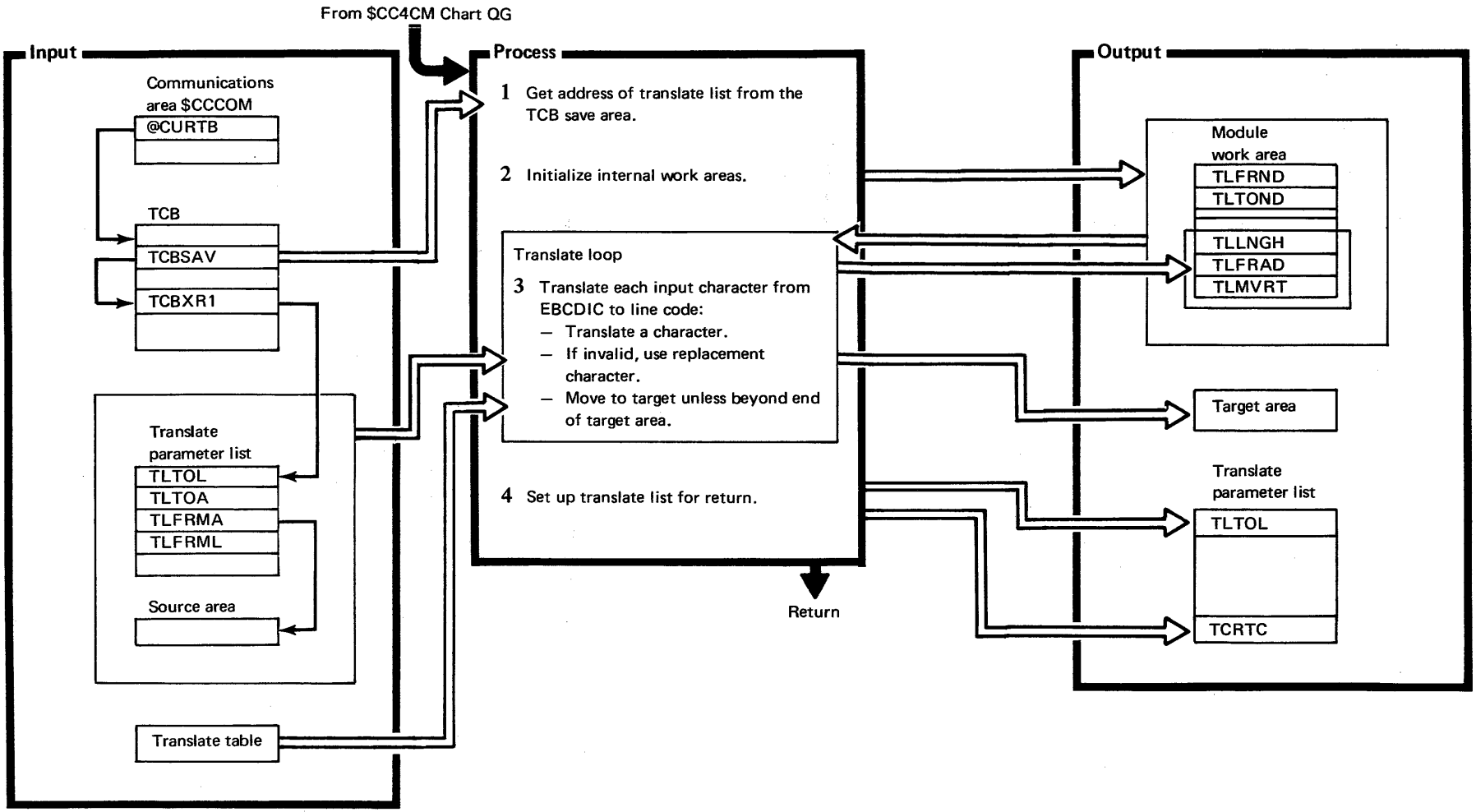
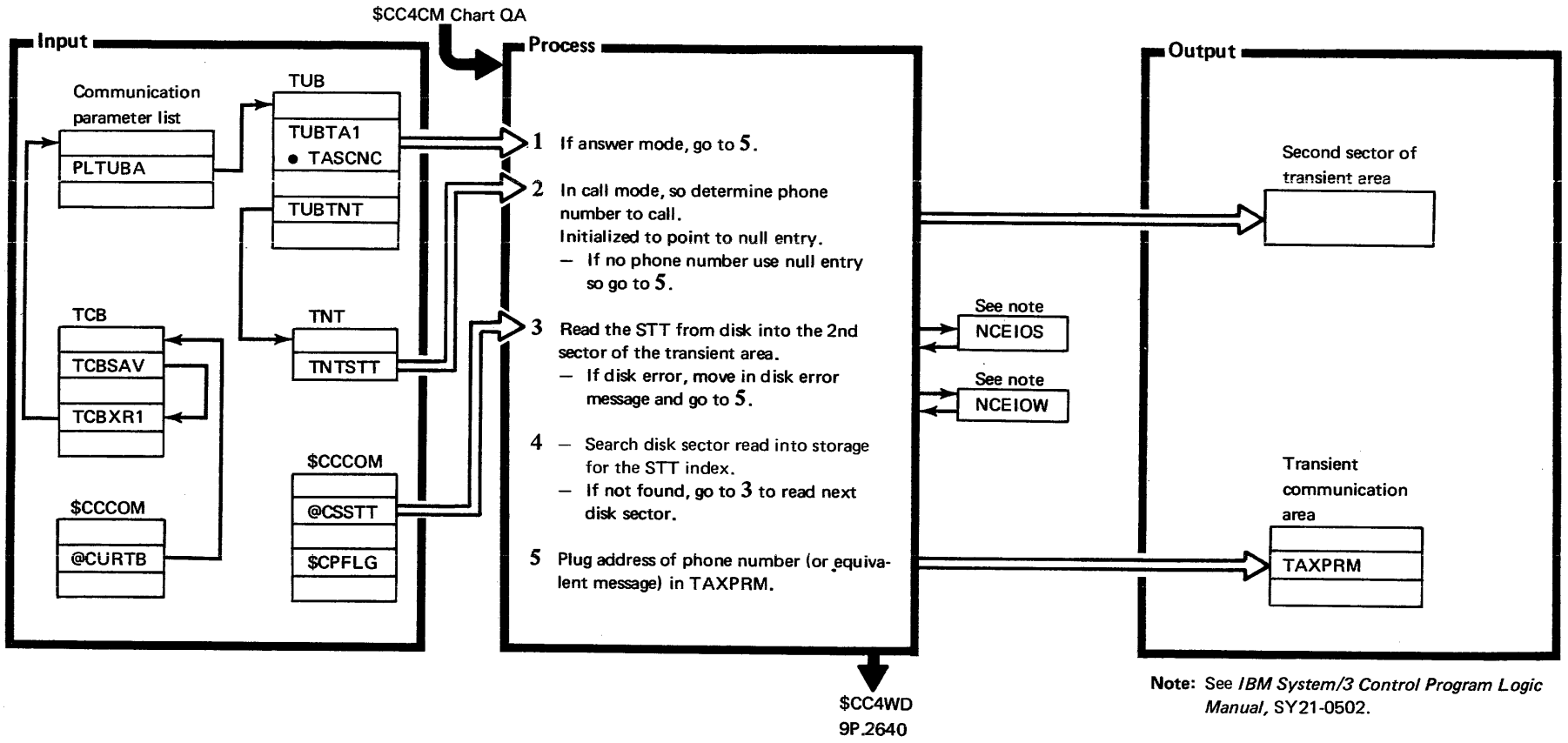


Diagram 9P.2620. $CC4Jx (Models 8, 10, and 12 Only)



Note: See *IBM System/3 Control Program Logic Manual,* SY21-0502.

Diagram 9P.2630. $CC4WC

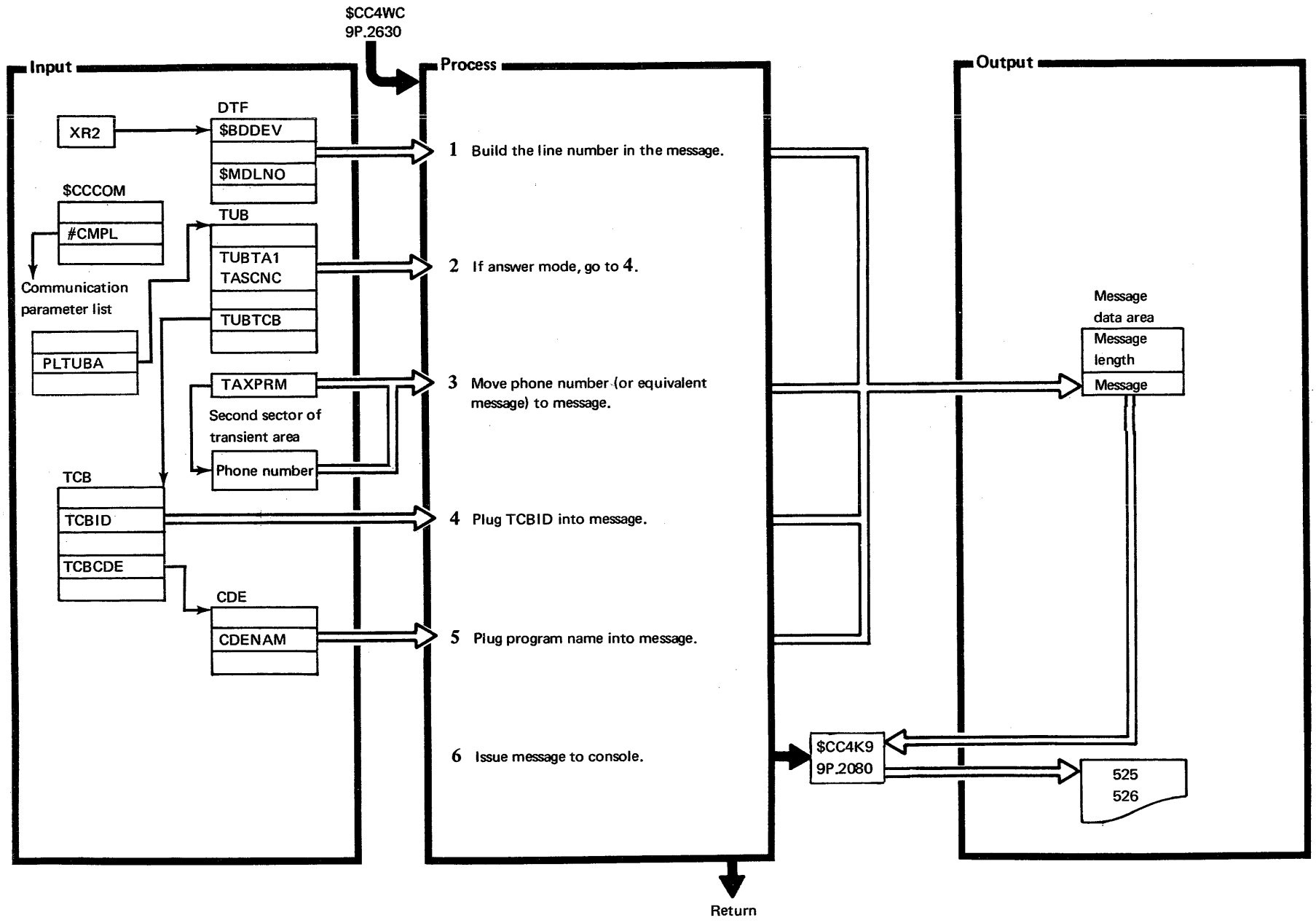


Diagram 9P.2640. $CC4WD

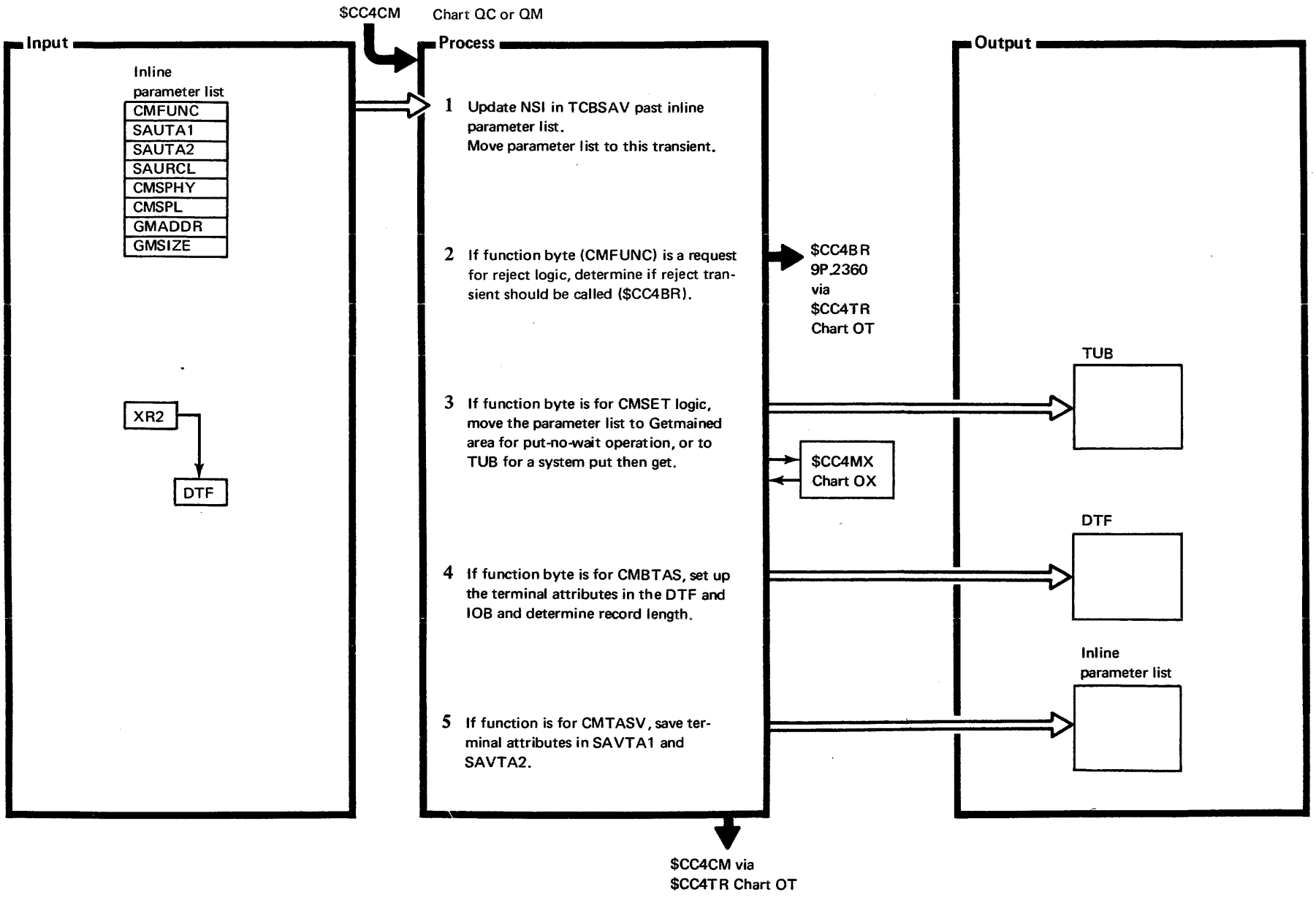


Diagram 9P.2710. $CC4B1

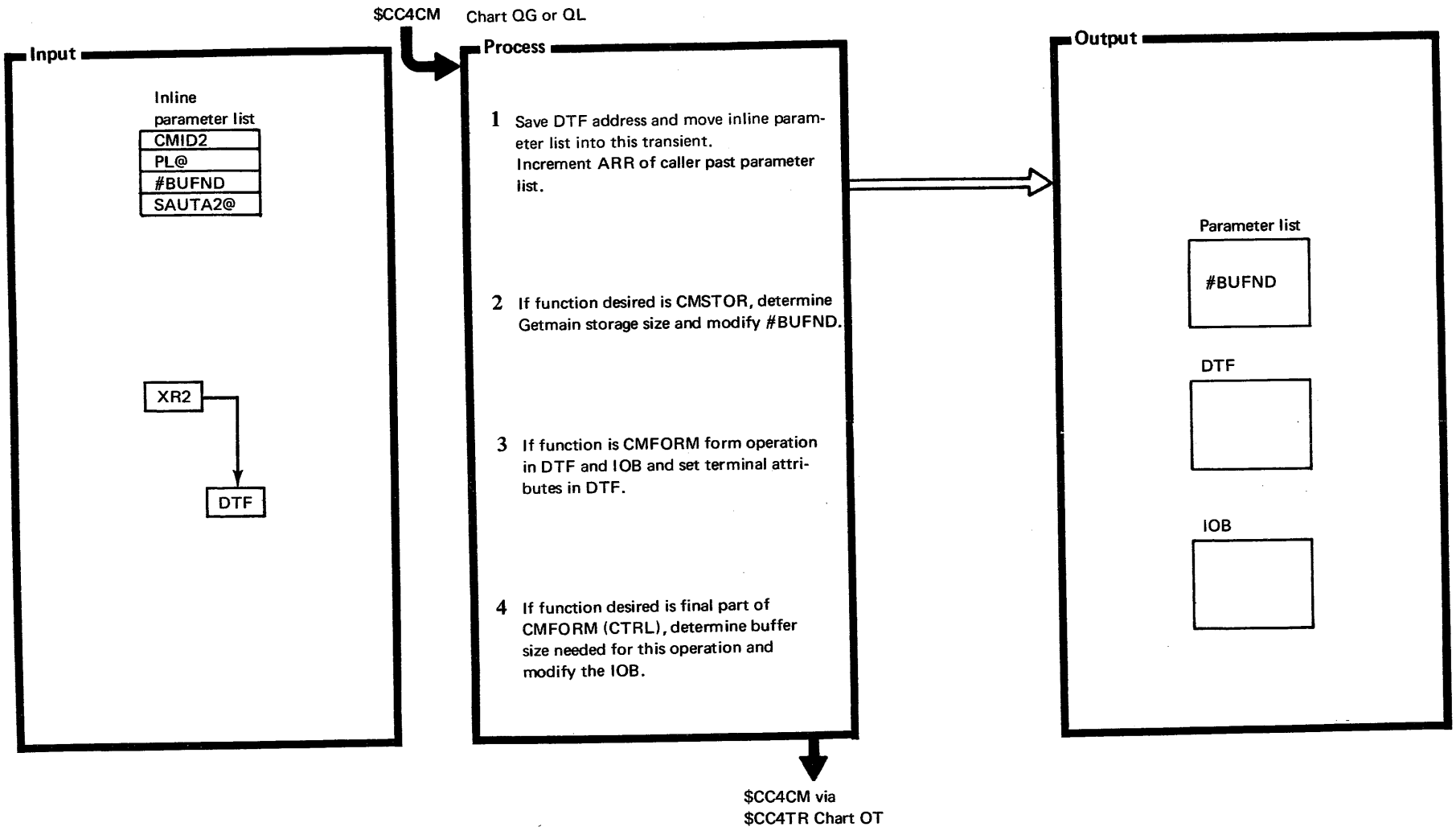


Diagram 9P.2720. $CC4B2

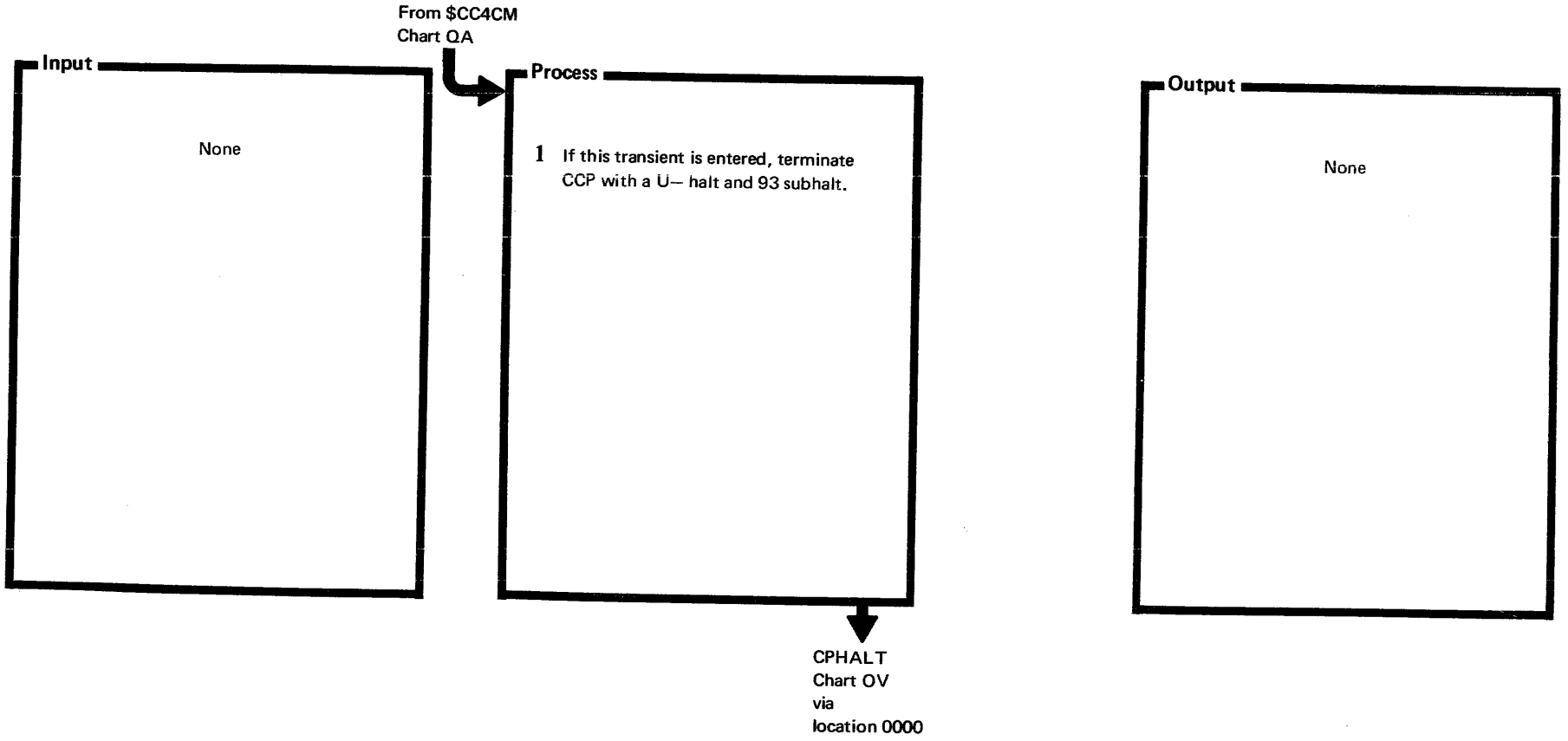


Diagram 9P.2730. $CC4B3

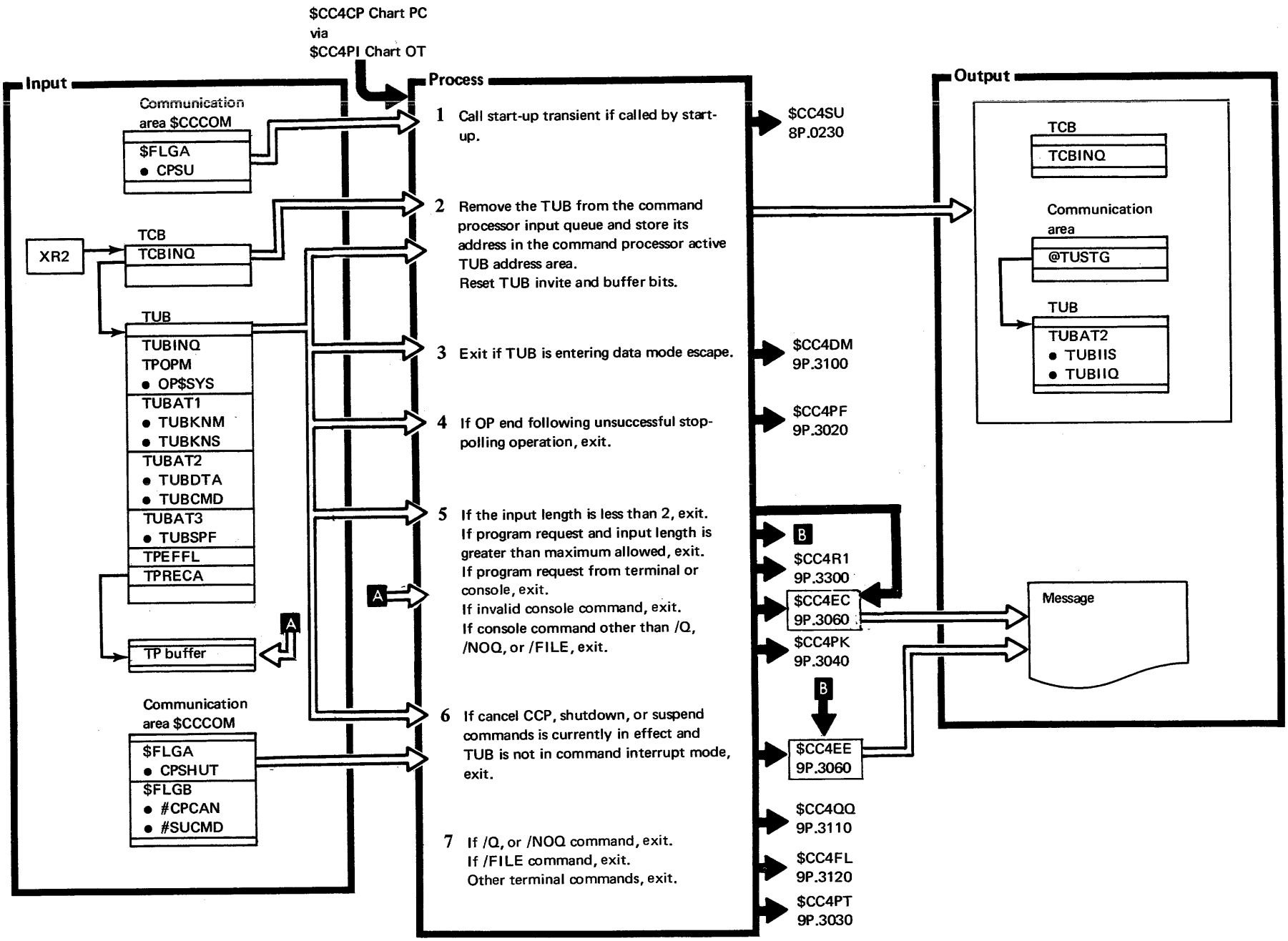


Diagram 9P.3010. $CC4PC

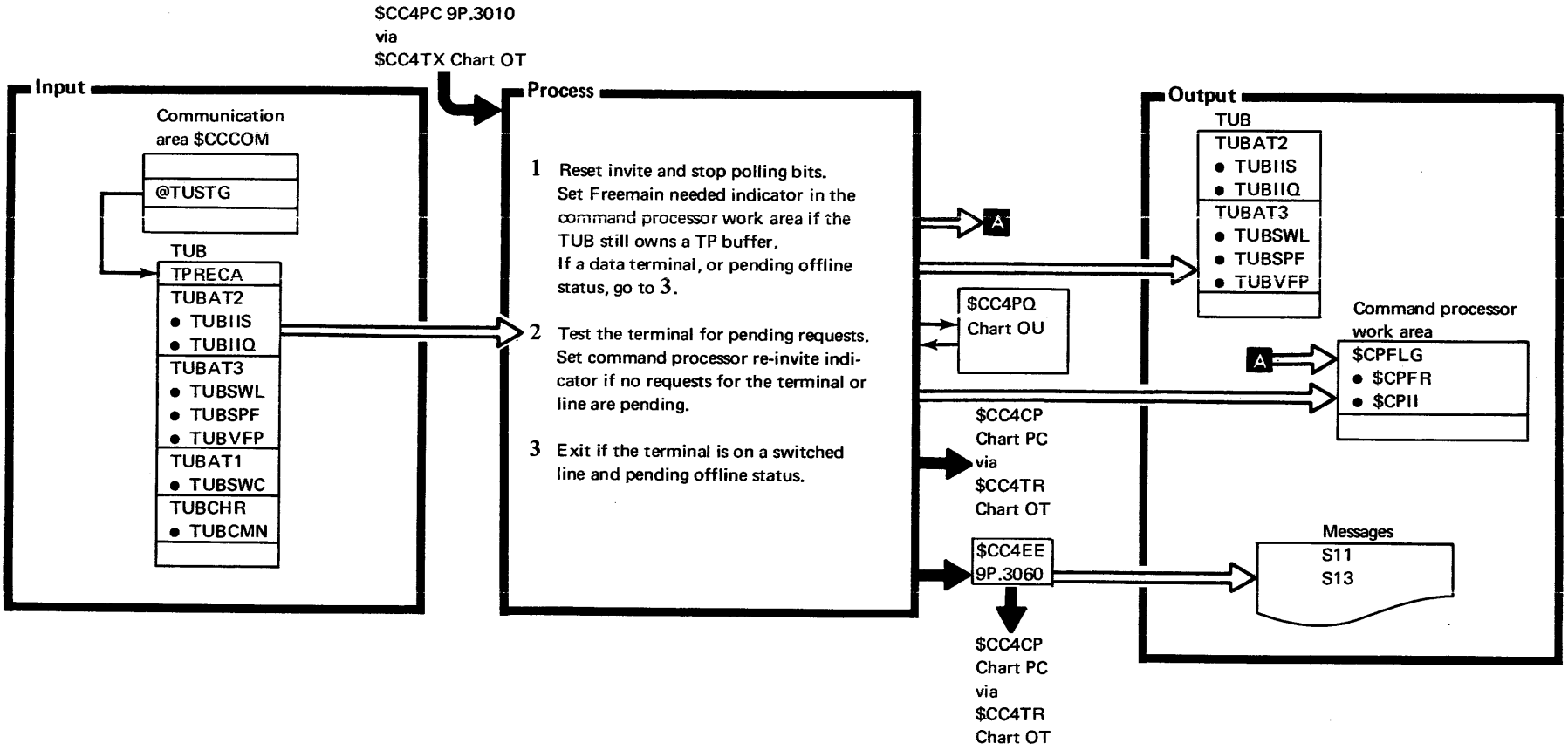


Diagram 9P.3020. $CC4PF

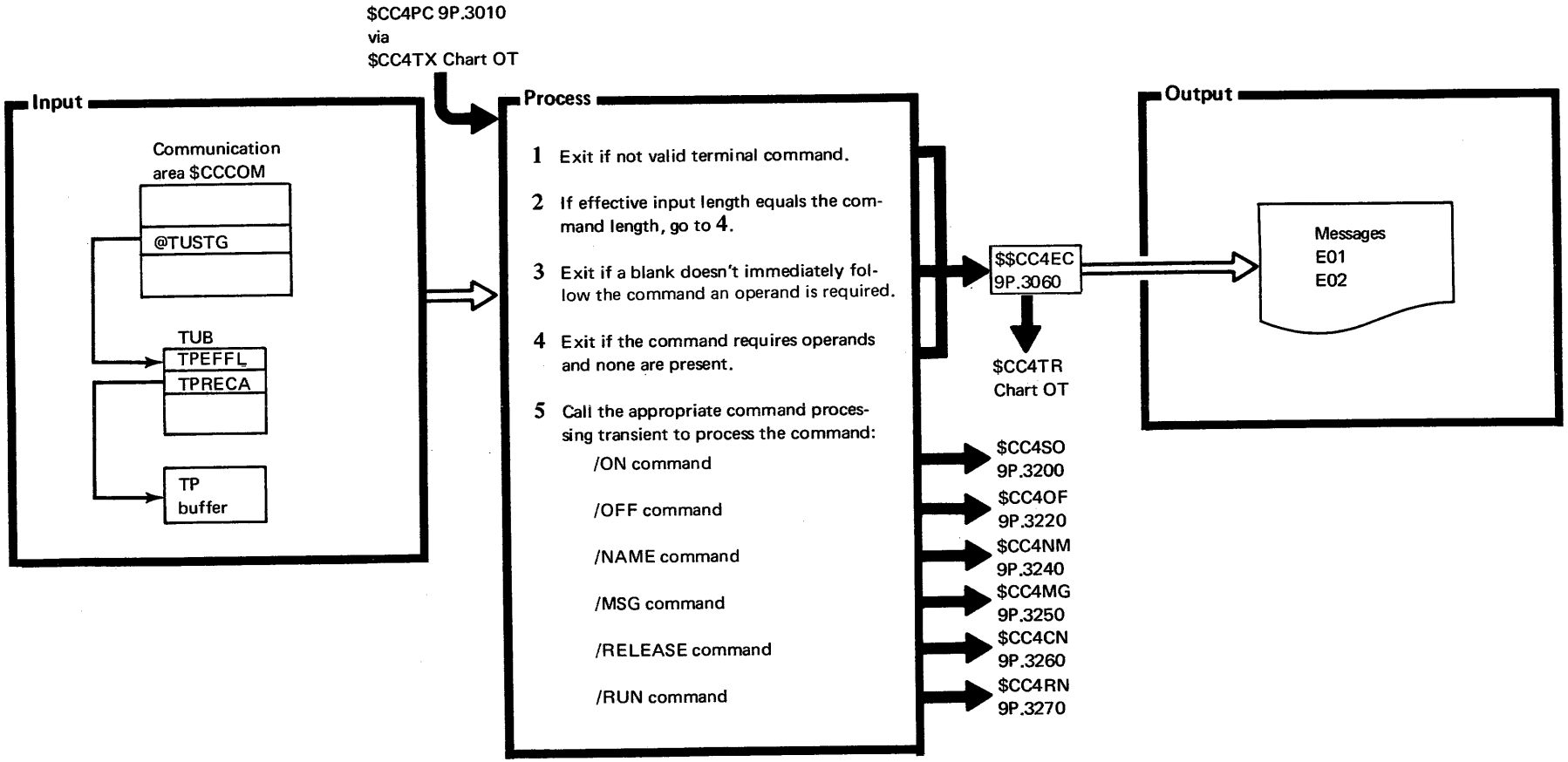
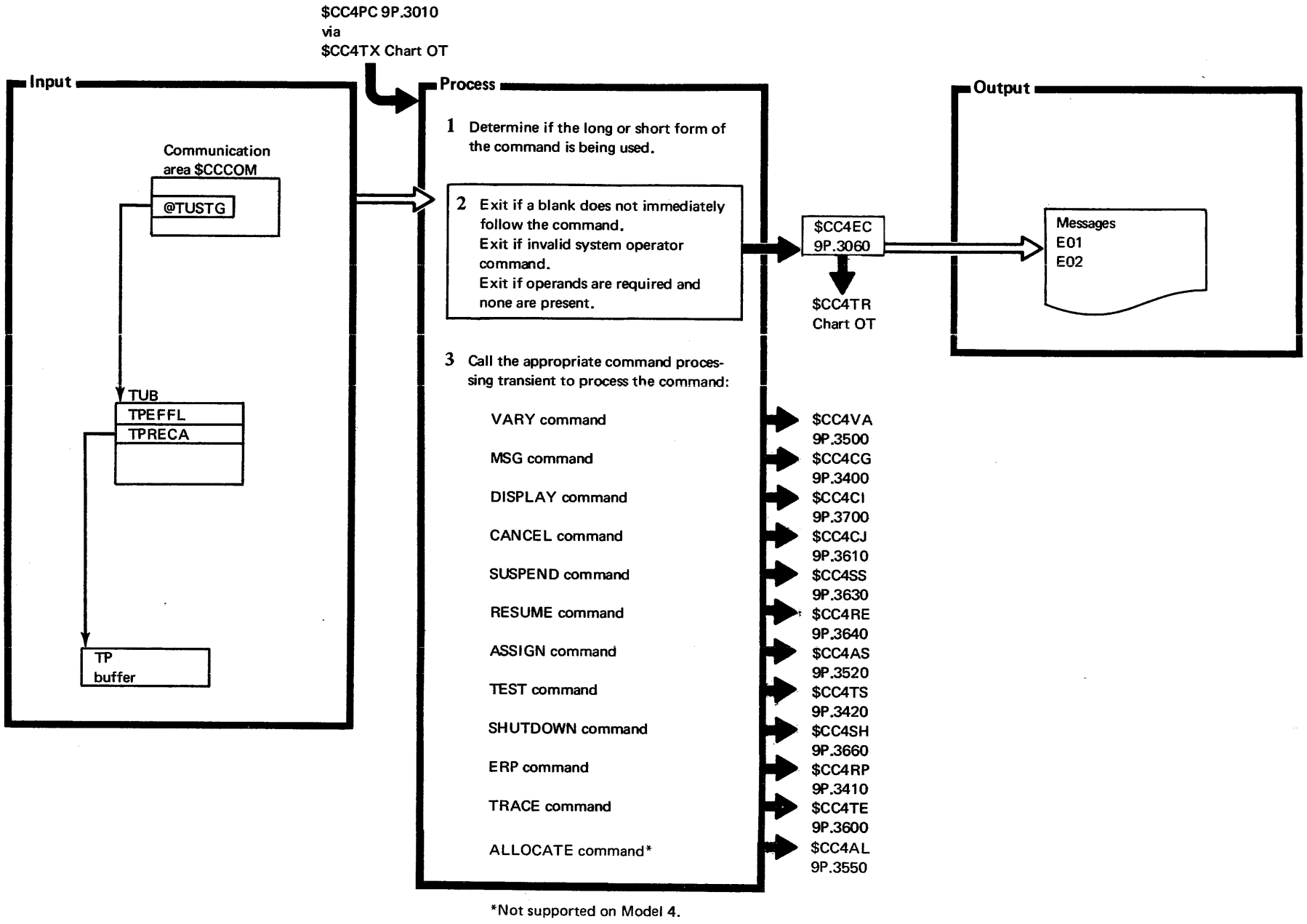


Diagram 9P.3030. $CC4PT



*Not supported on Model 4.

Diagram 9P.3040. $CC4PK (Models 8, 10, and 12)

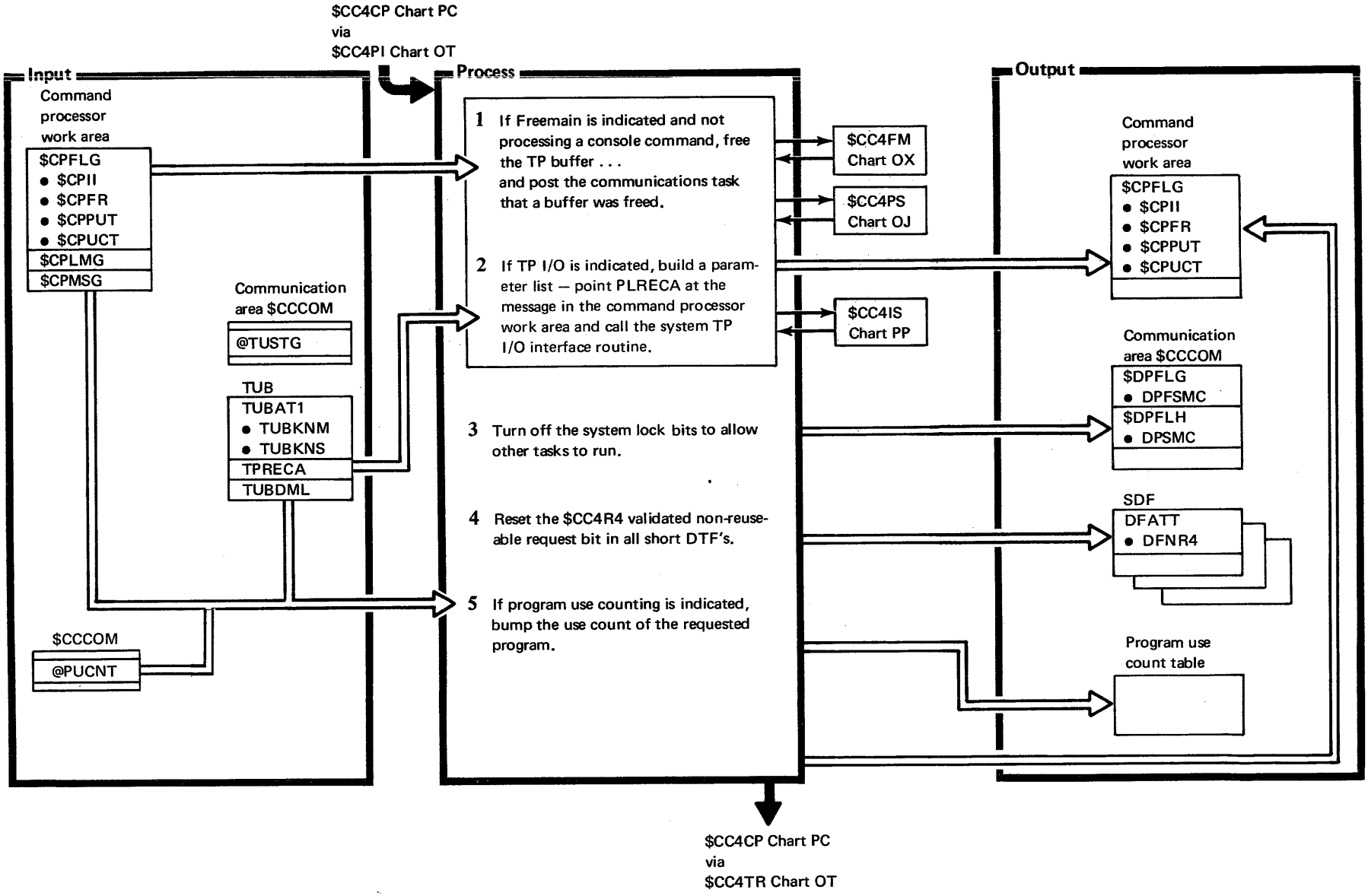
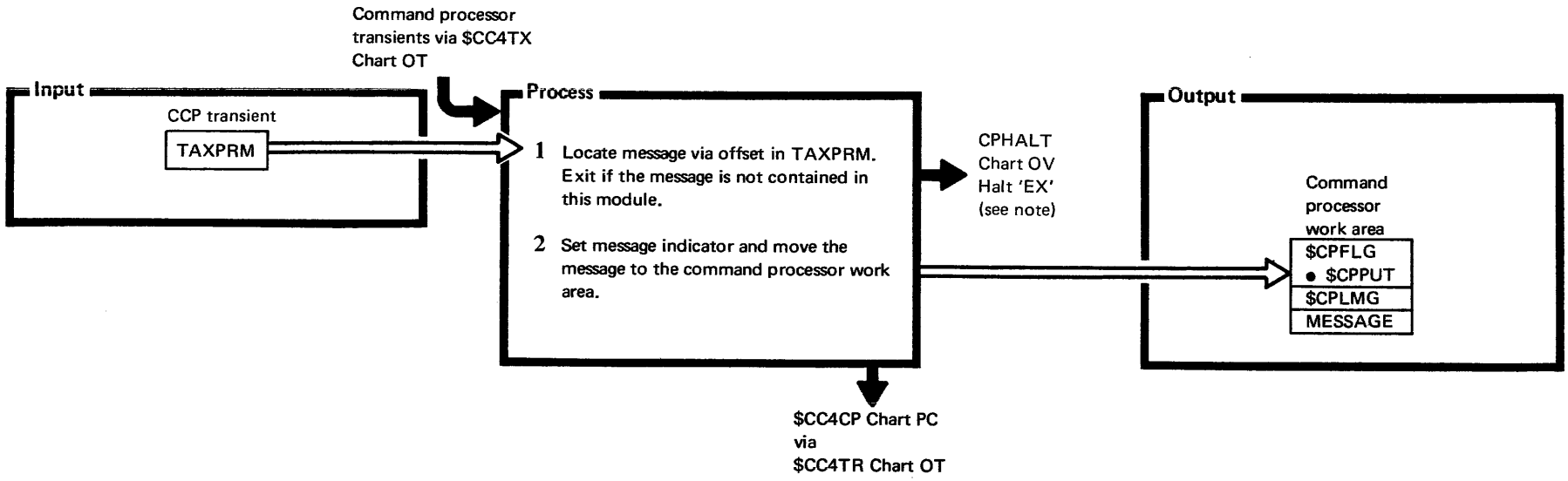


Diagram 9P.3050. $CC4PR



Note: On Model 4, the halts are converted to halt indicators. These Model 4 halt indicators are listed in Appendix C of this manual.

Diagram 9P.3060. $CC4Ex

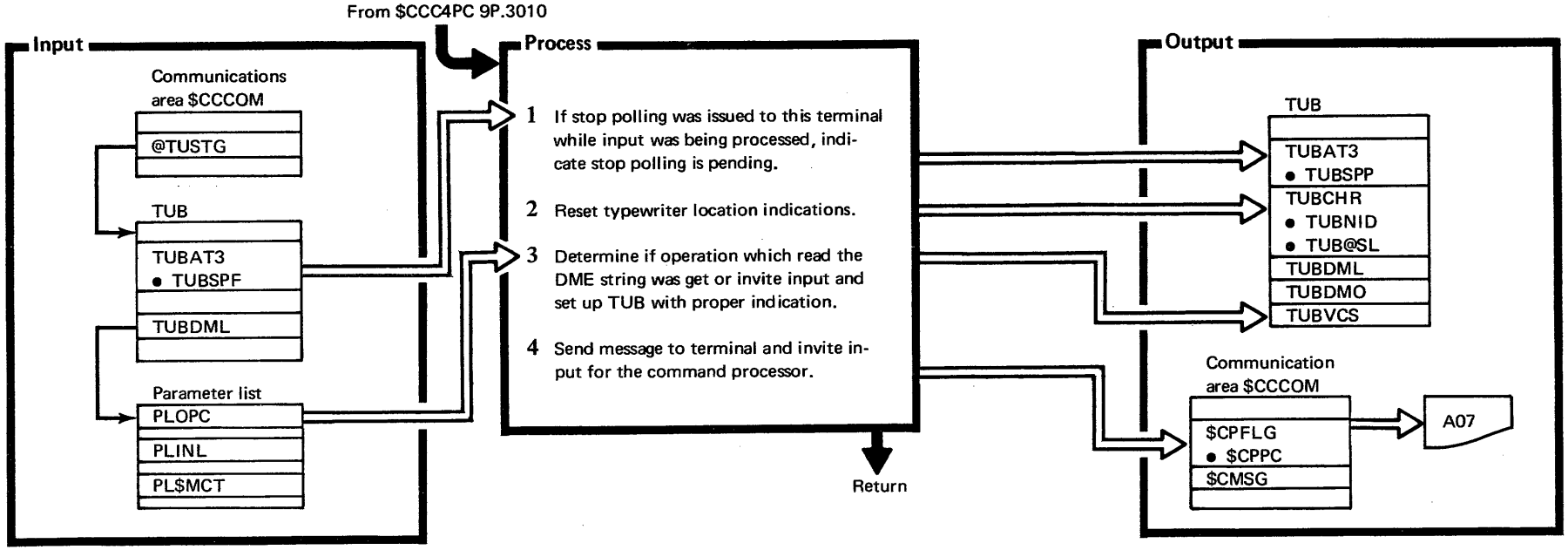


Diagram 9P.3100. $CC4DM

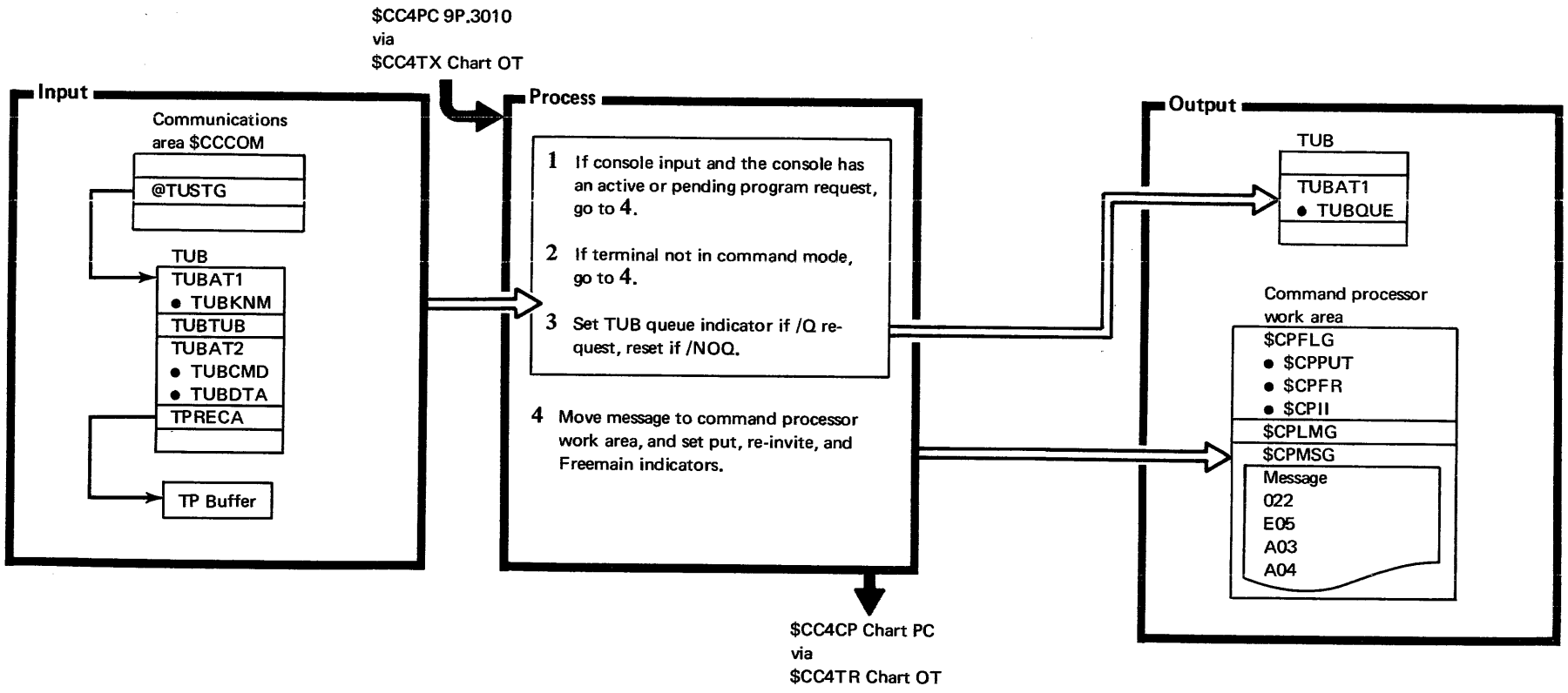


Diagram 9P.3110. $CC4QQ

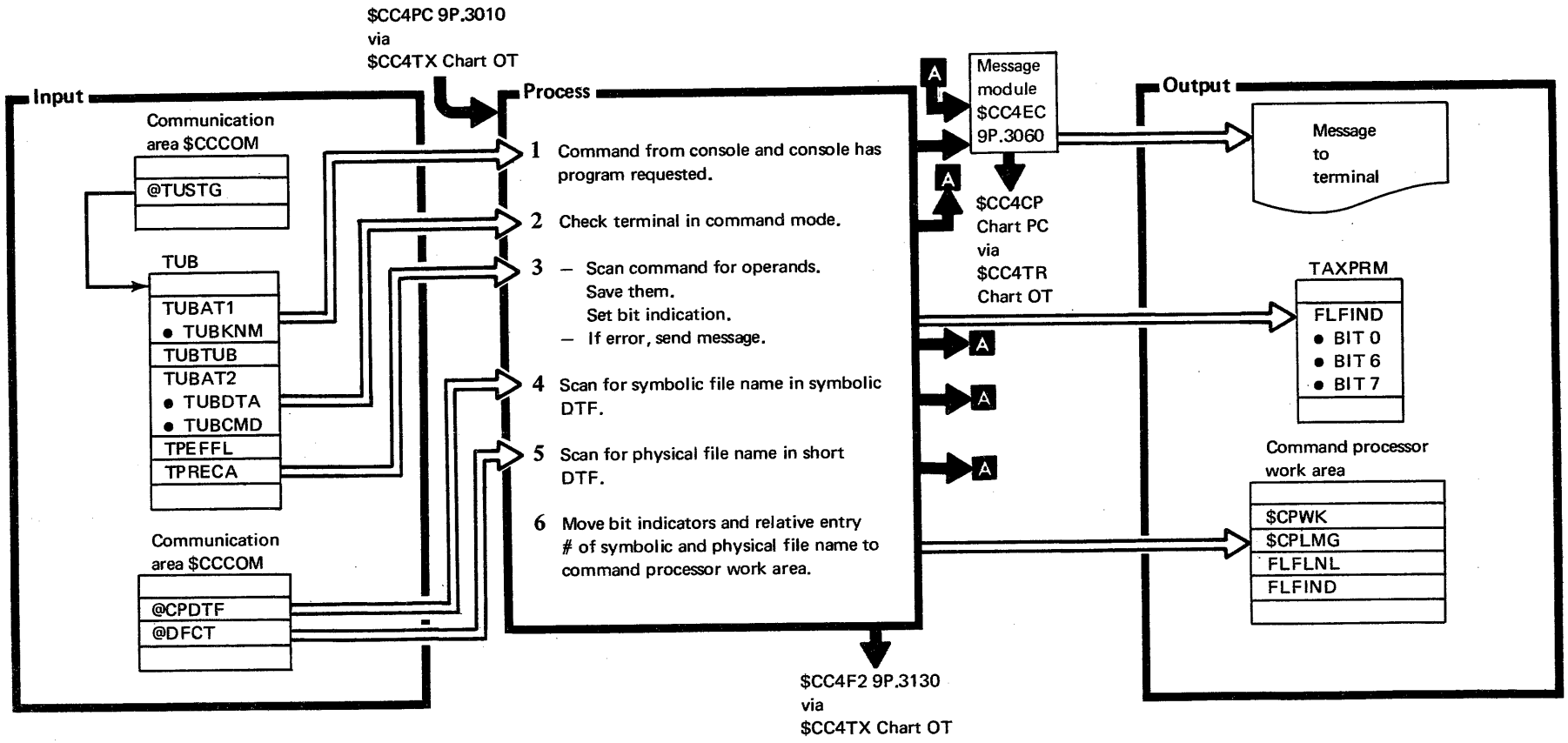


Diagram 9P.3120. $CC4FL

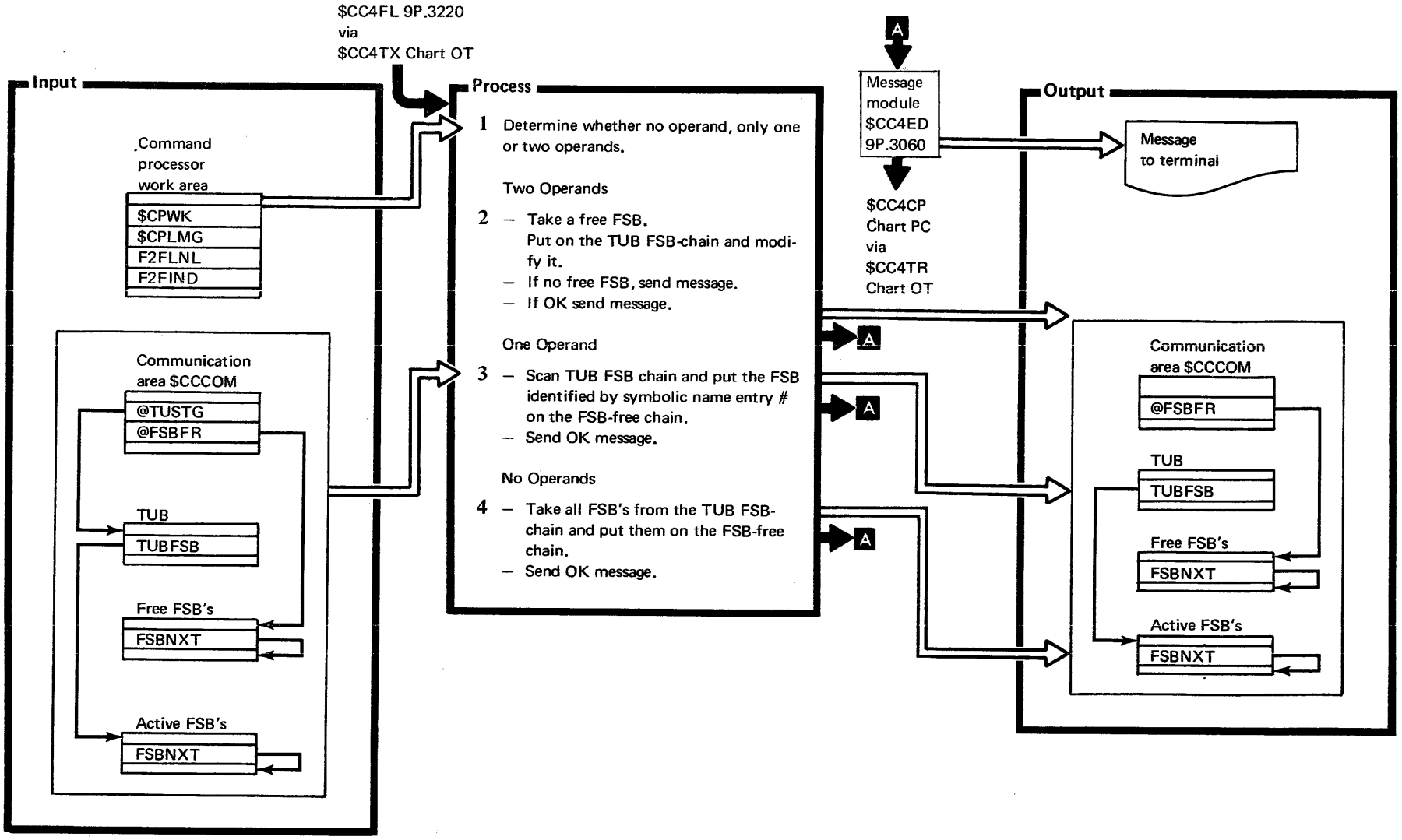


Diagram 9P.3130. $CC4F2

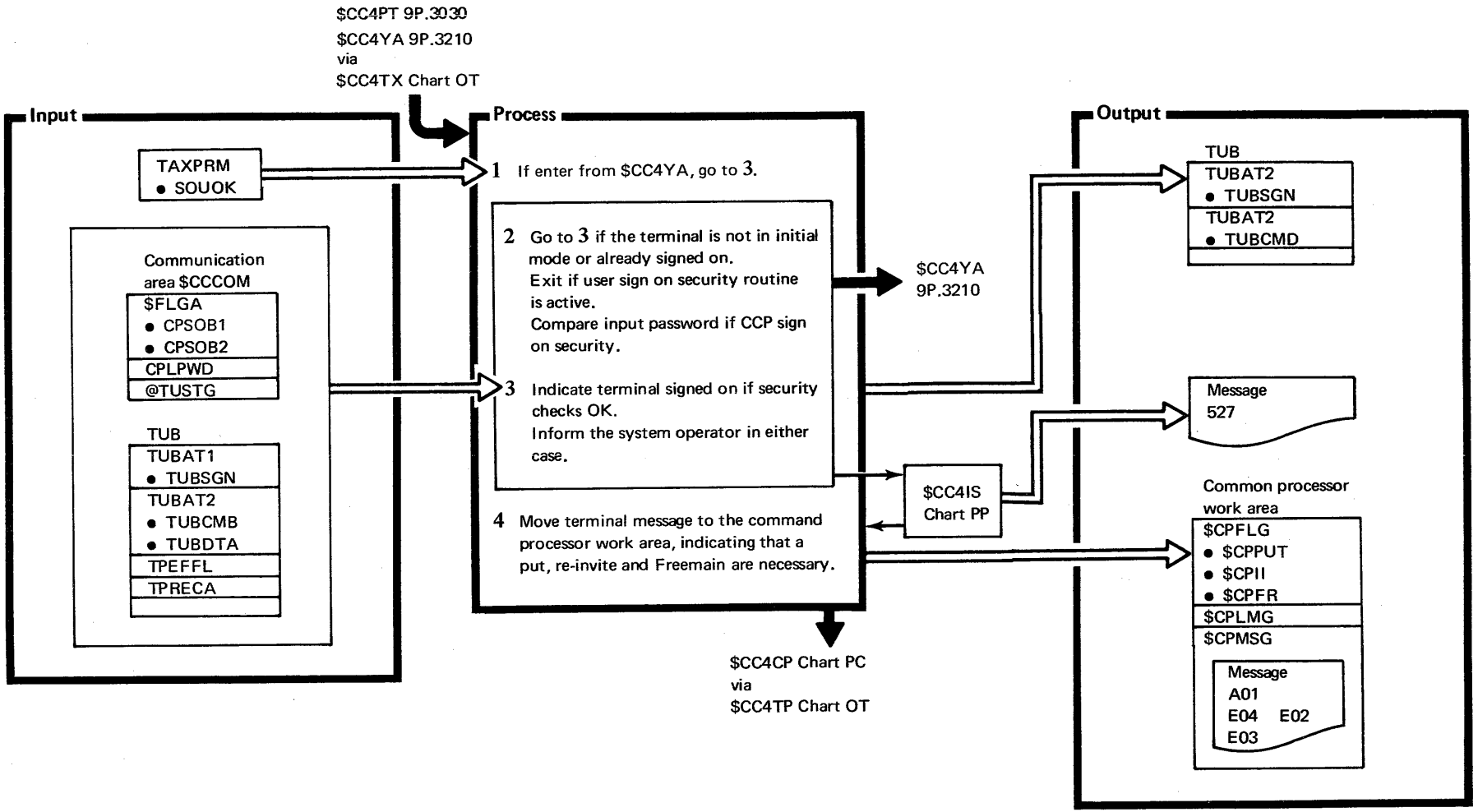


Diagram 9P.3200. $CC4SO

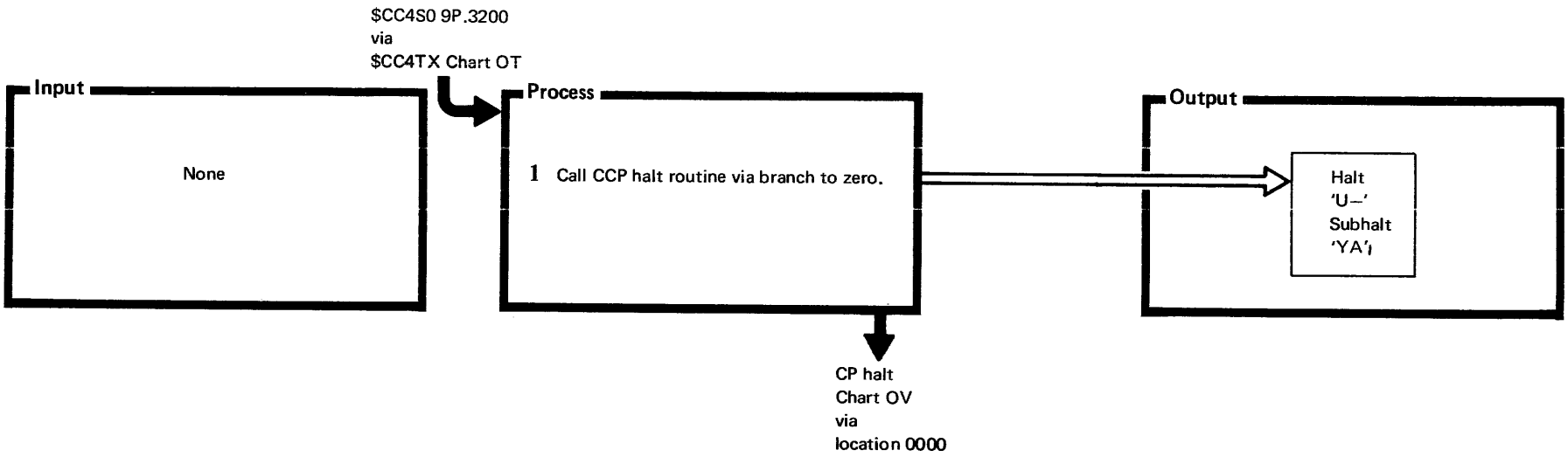


Diagram 9P.3210. $CC4YA

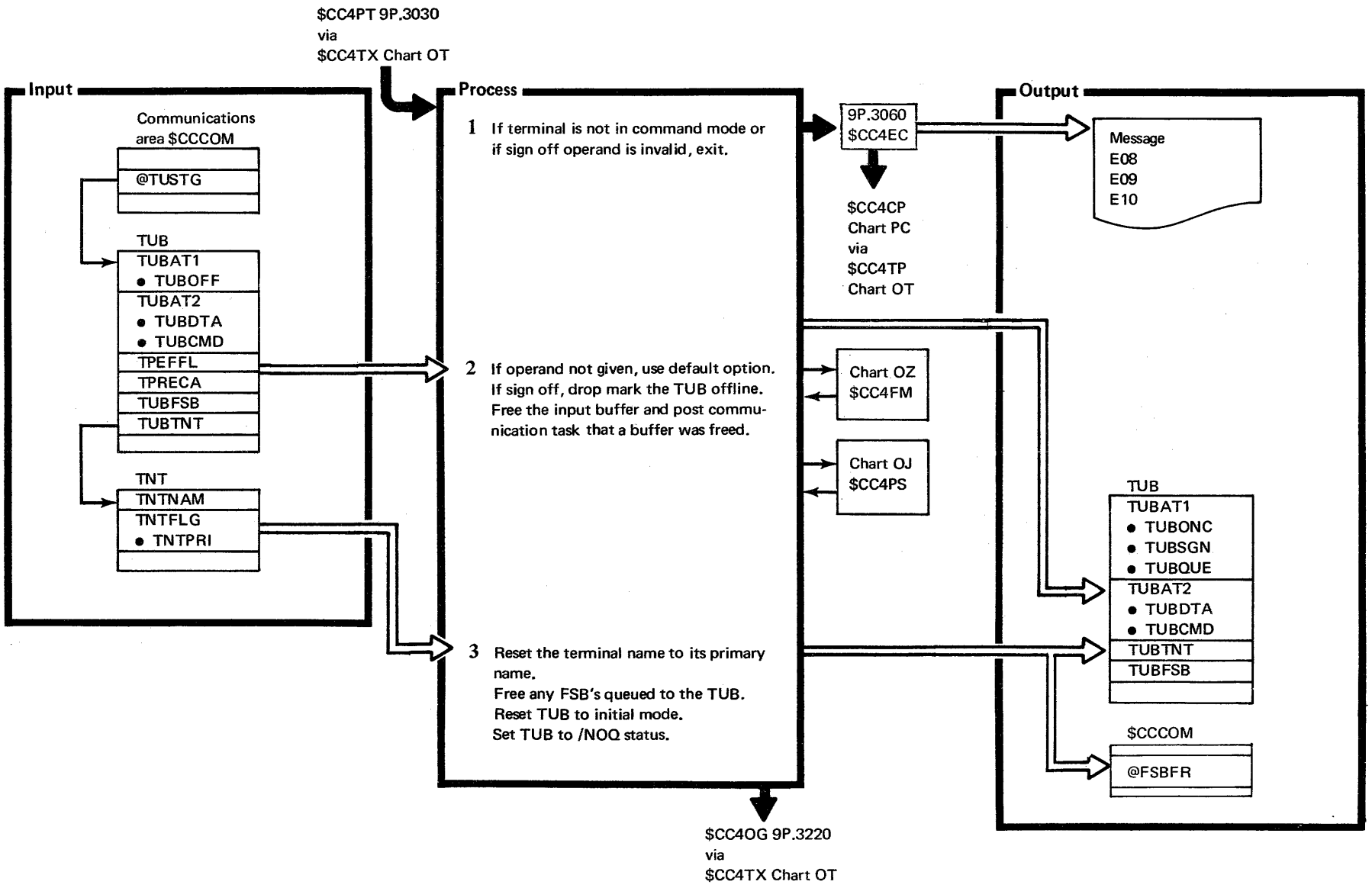


Diagram 9P.3220. $CC4OF

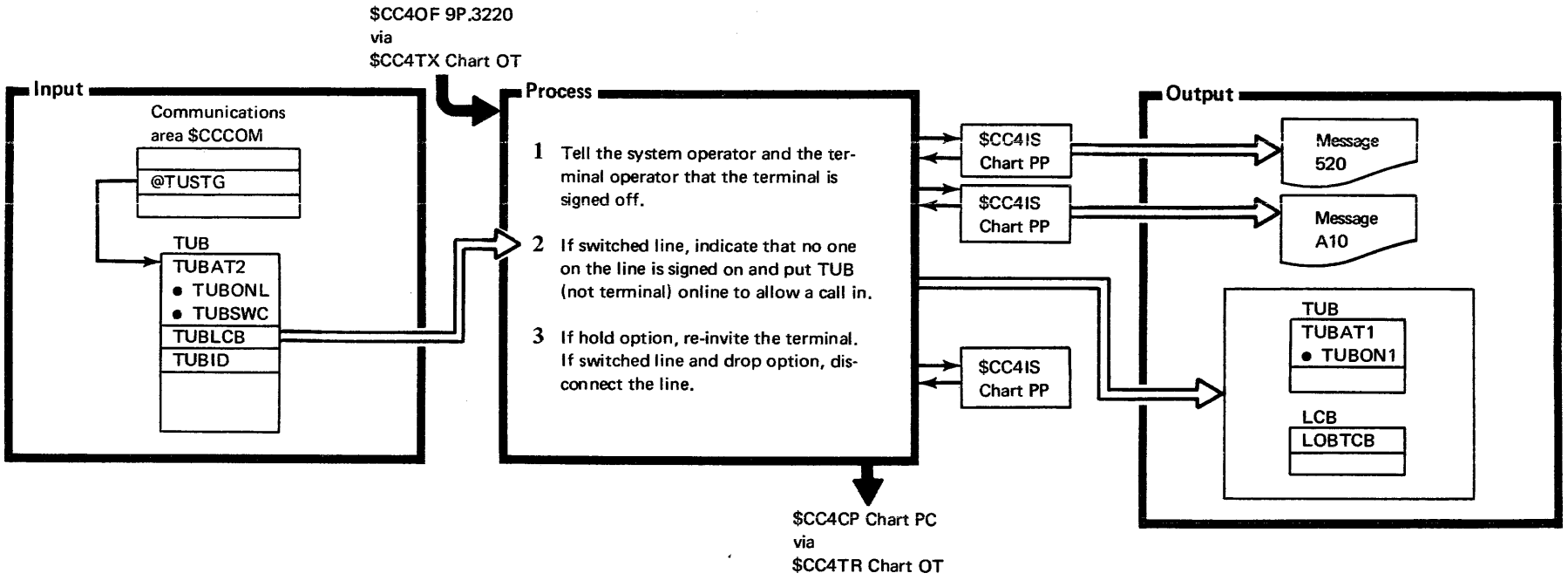


Diagram 9P.3230. $CC4OG

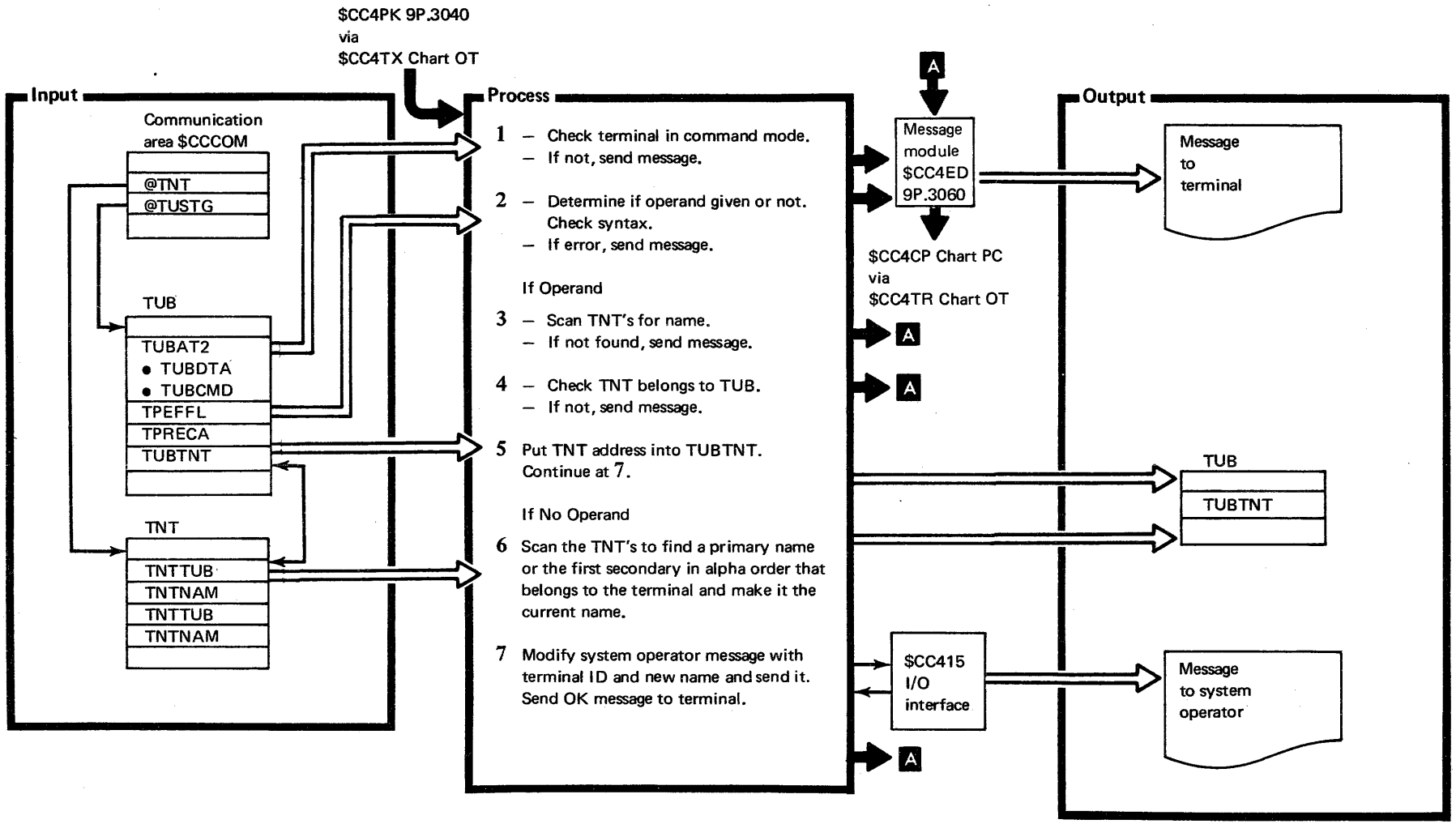


Diagram 9P.3240. $CC4NM

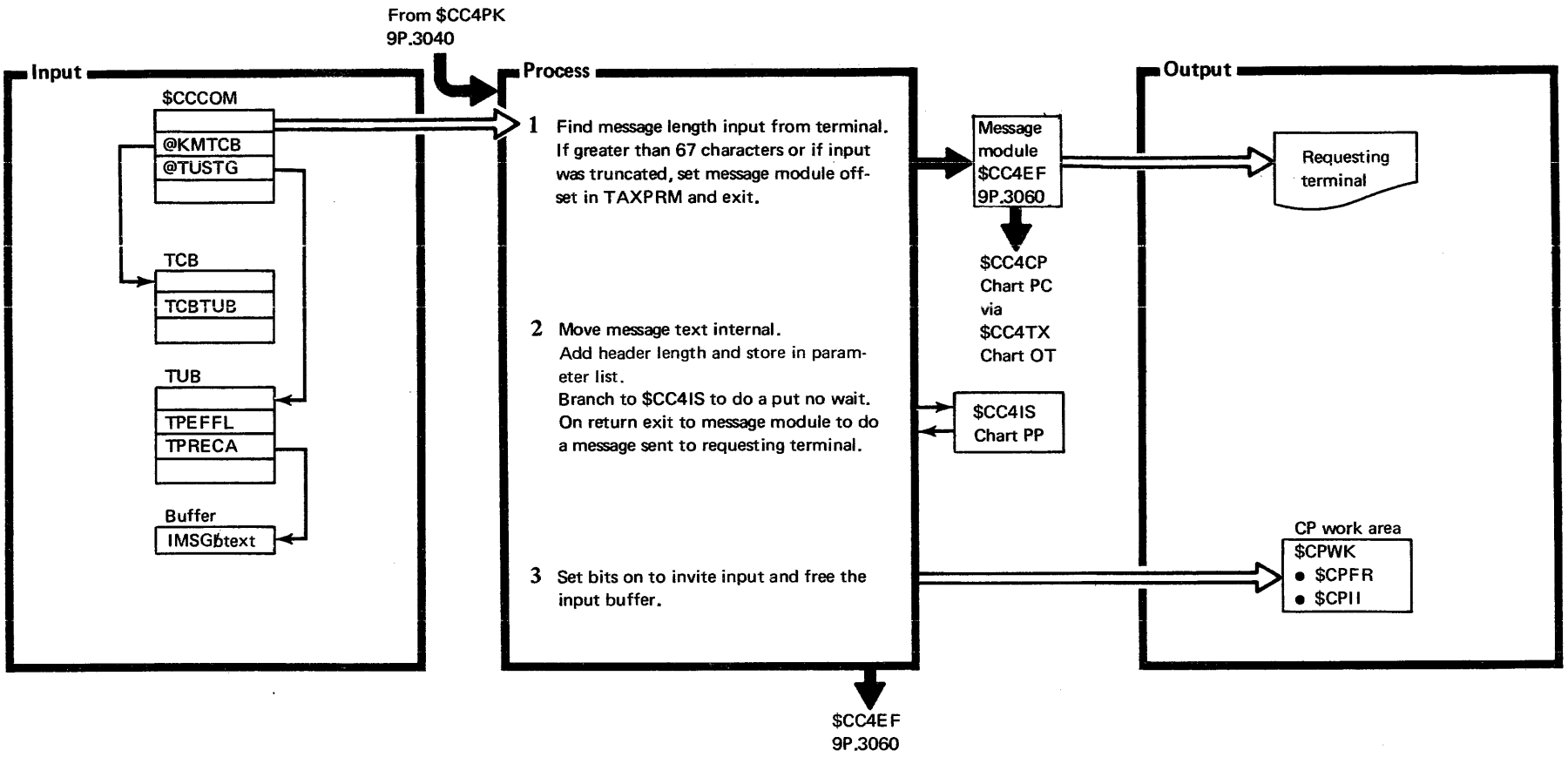


Diagram 9P.3250. $CC4MG

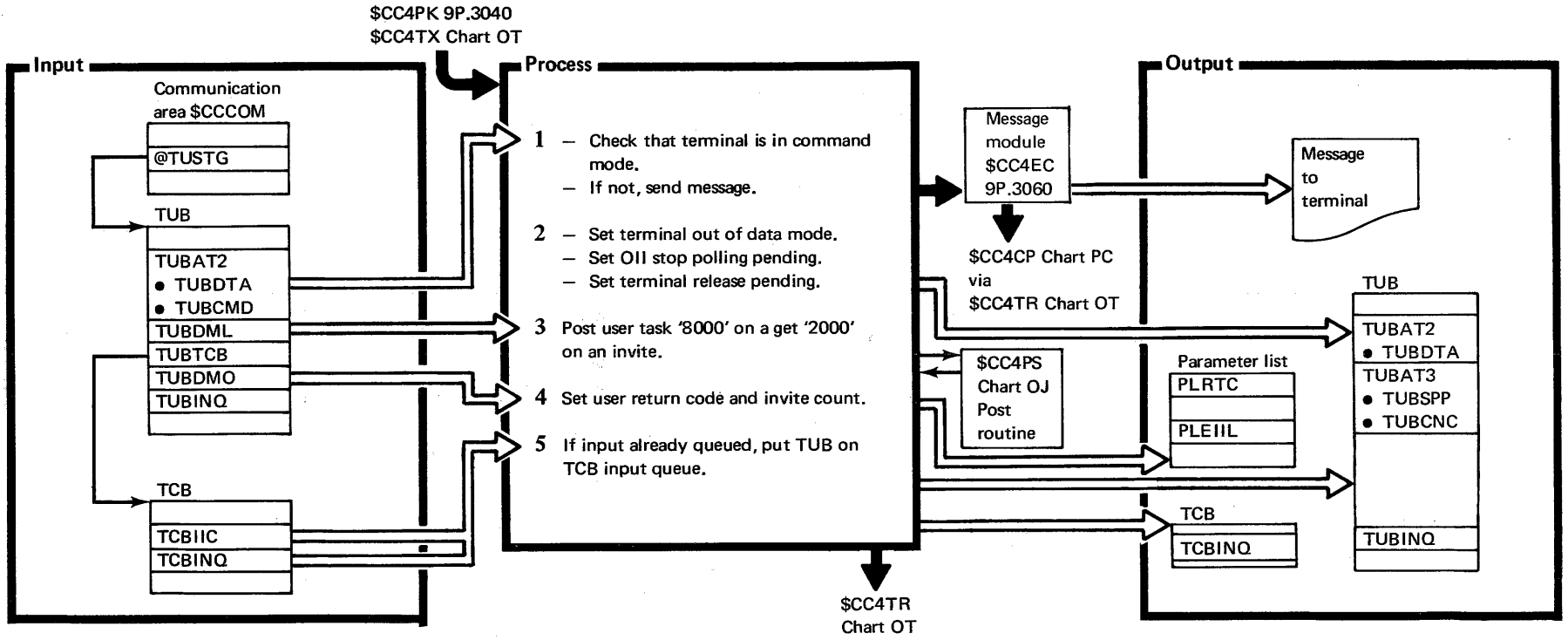


Diagram 9P.3260. $CC4CN

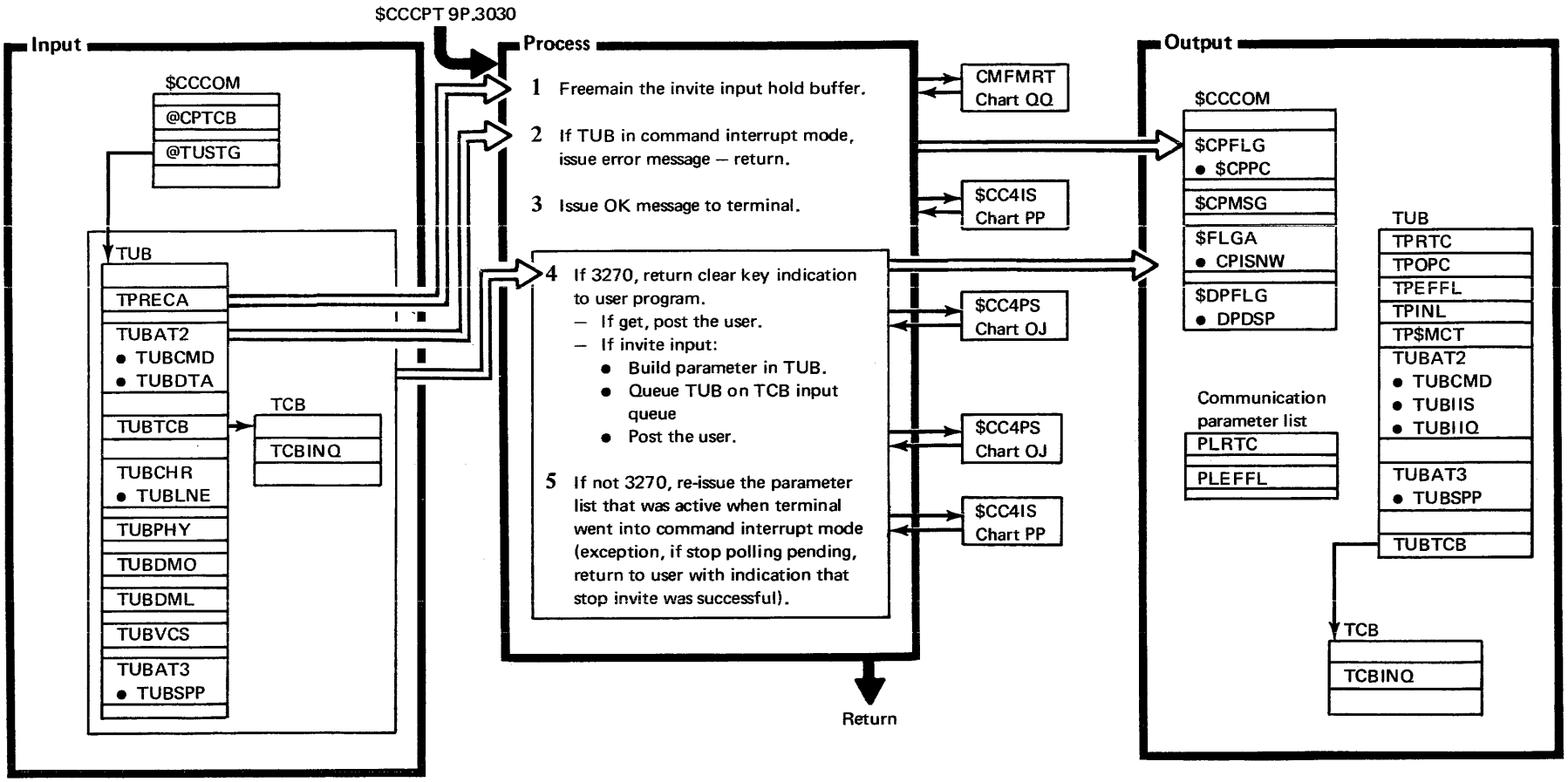


Diagram 9P.3270. $CC4RN

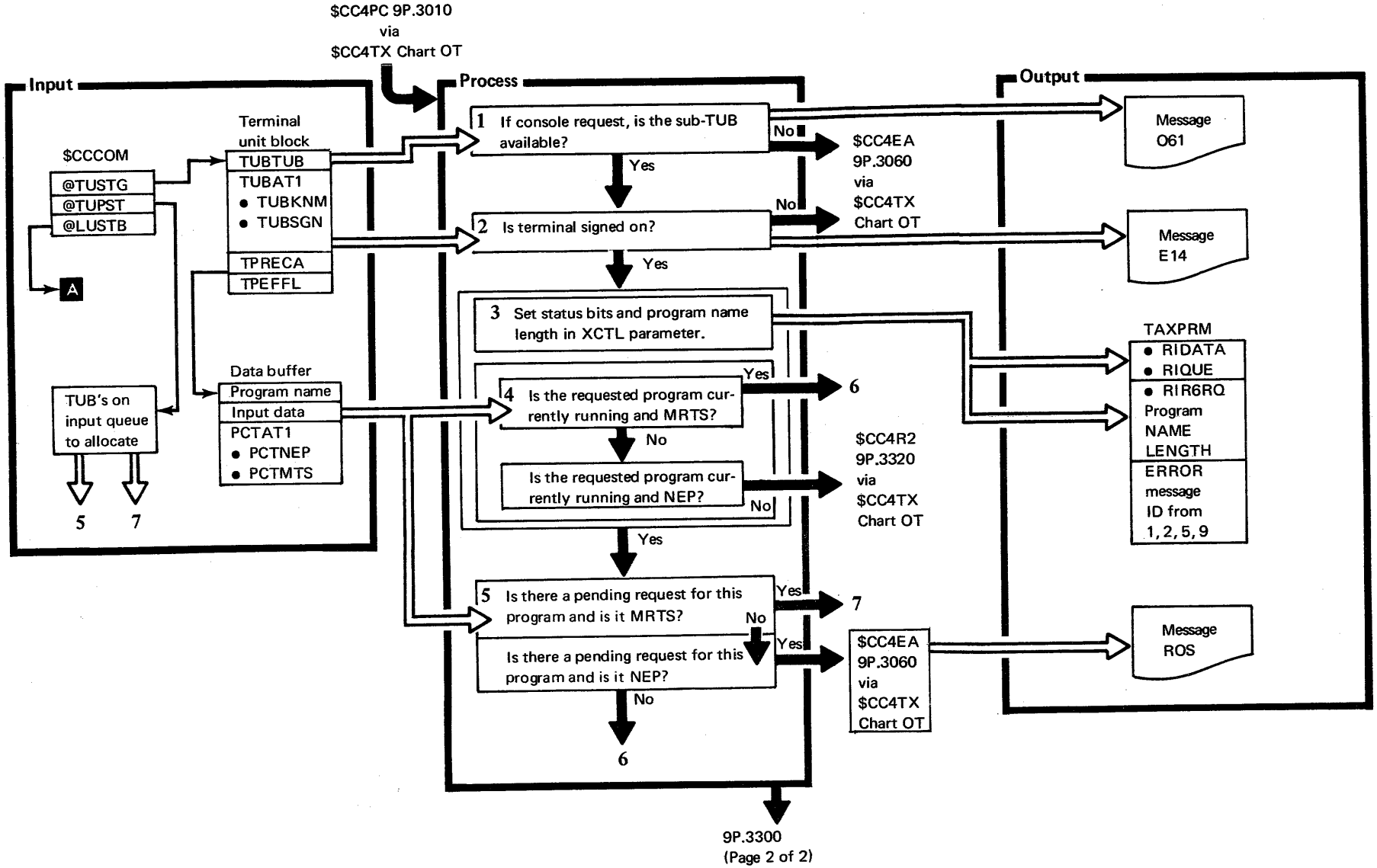


Diagram 9P.3300 (Part 1 of 2). $CC4R1

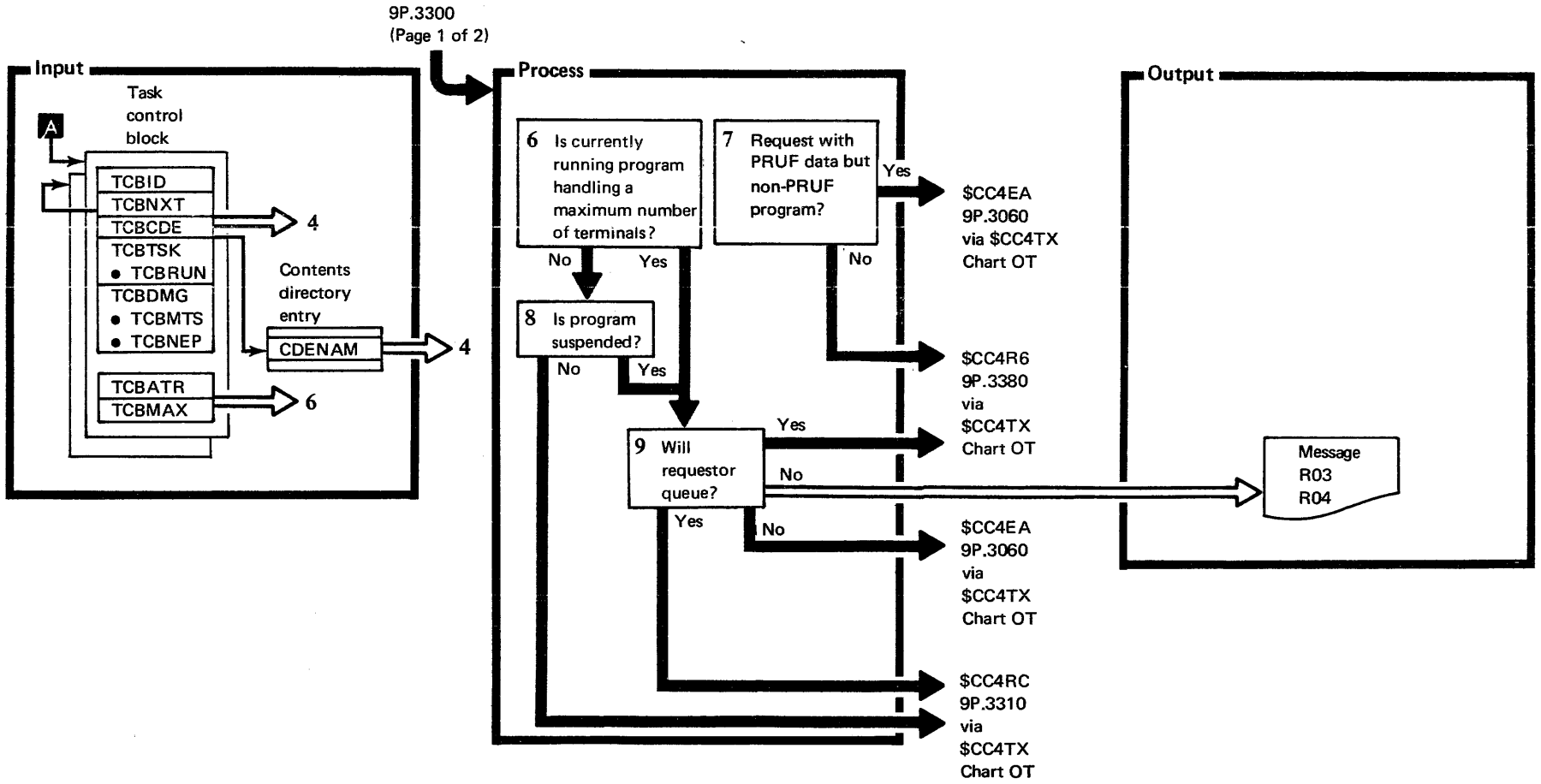


Diagram 9P.3300 (Part 2 of 2). $CC4R1

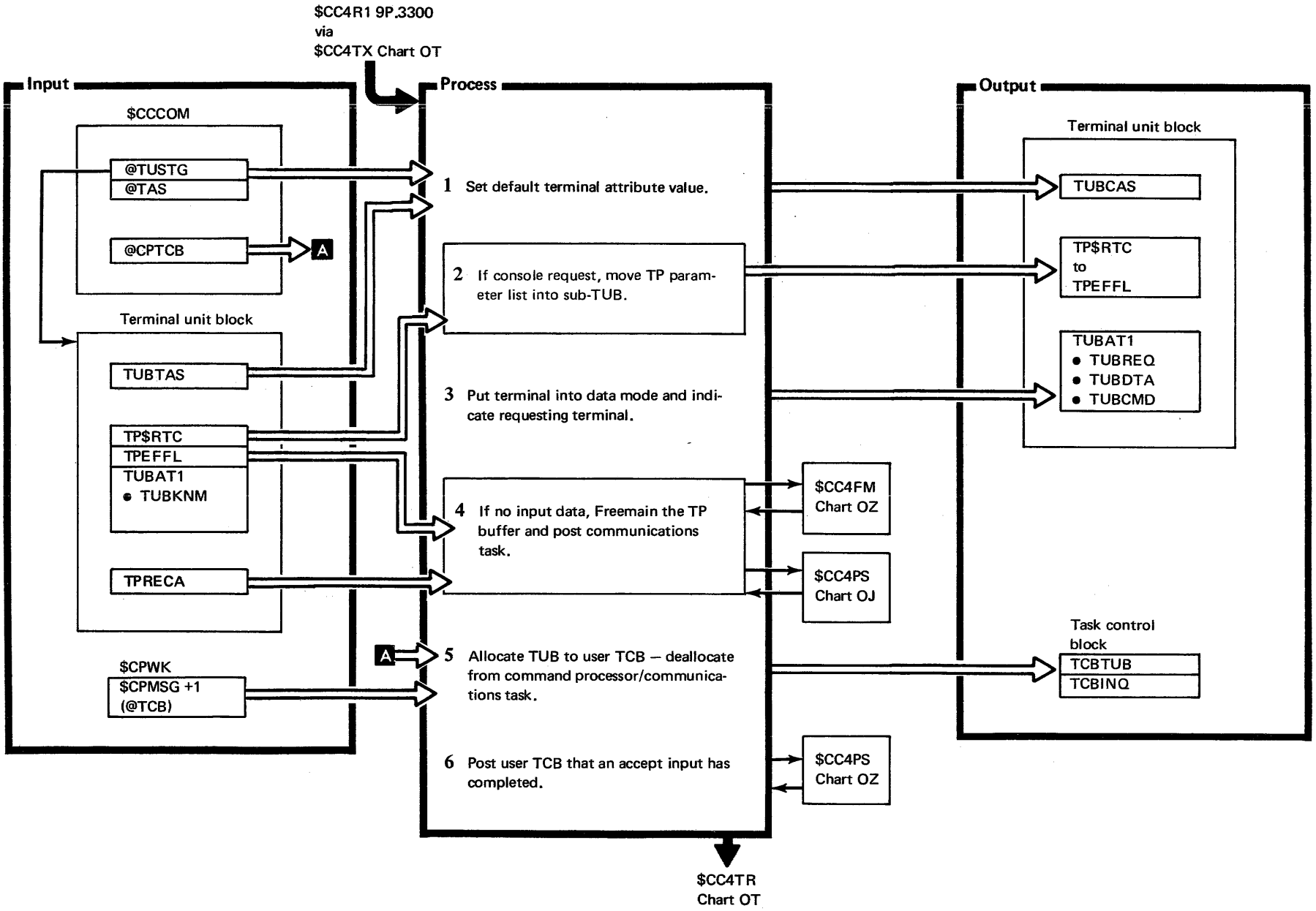
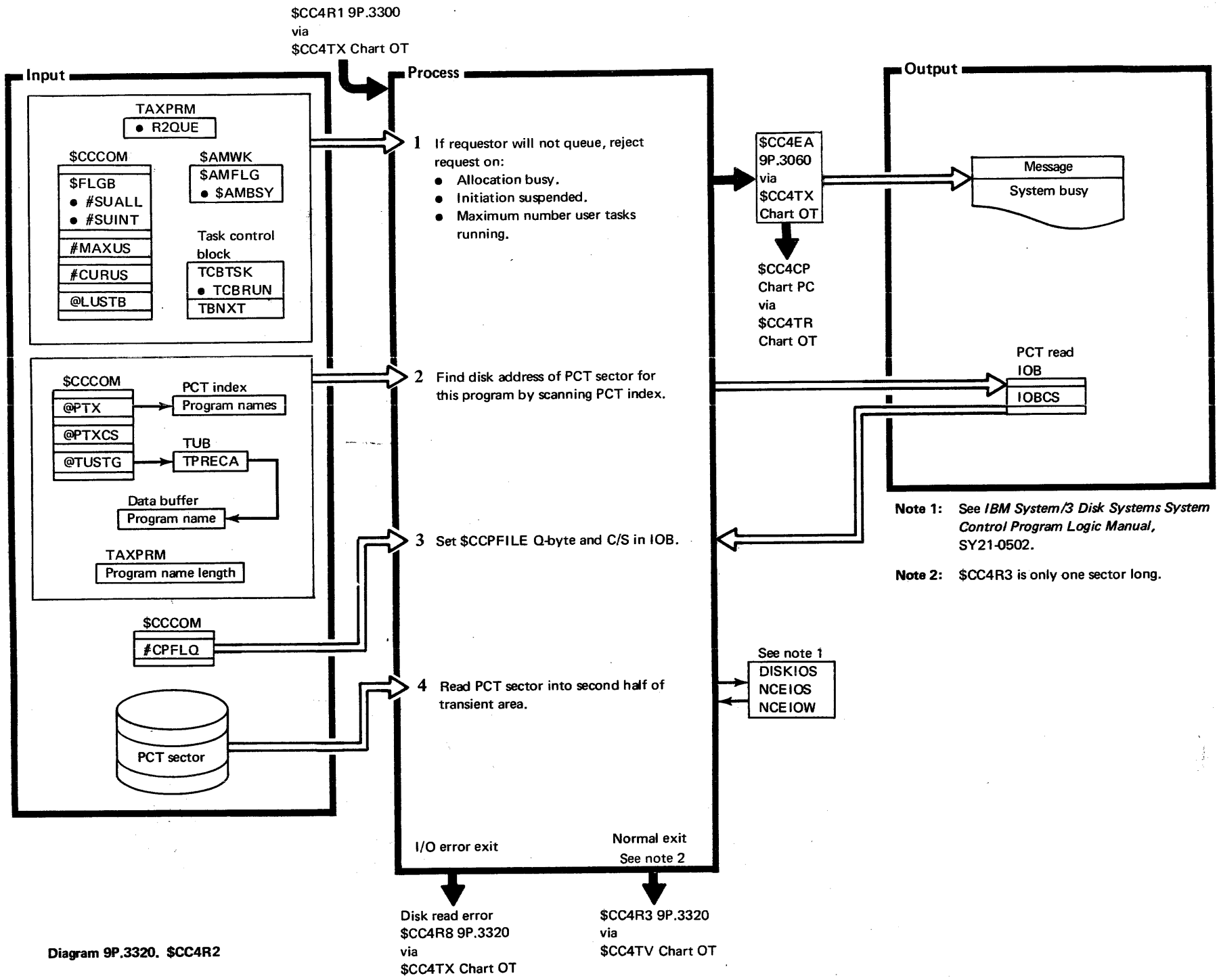


Diagram 9P.3310. $CC4RC



Note 1: See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.

Note 2: $CC4R3 is only one sector long.

Diagram 9P.3320. $CC4R2

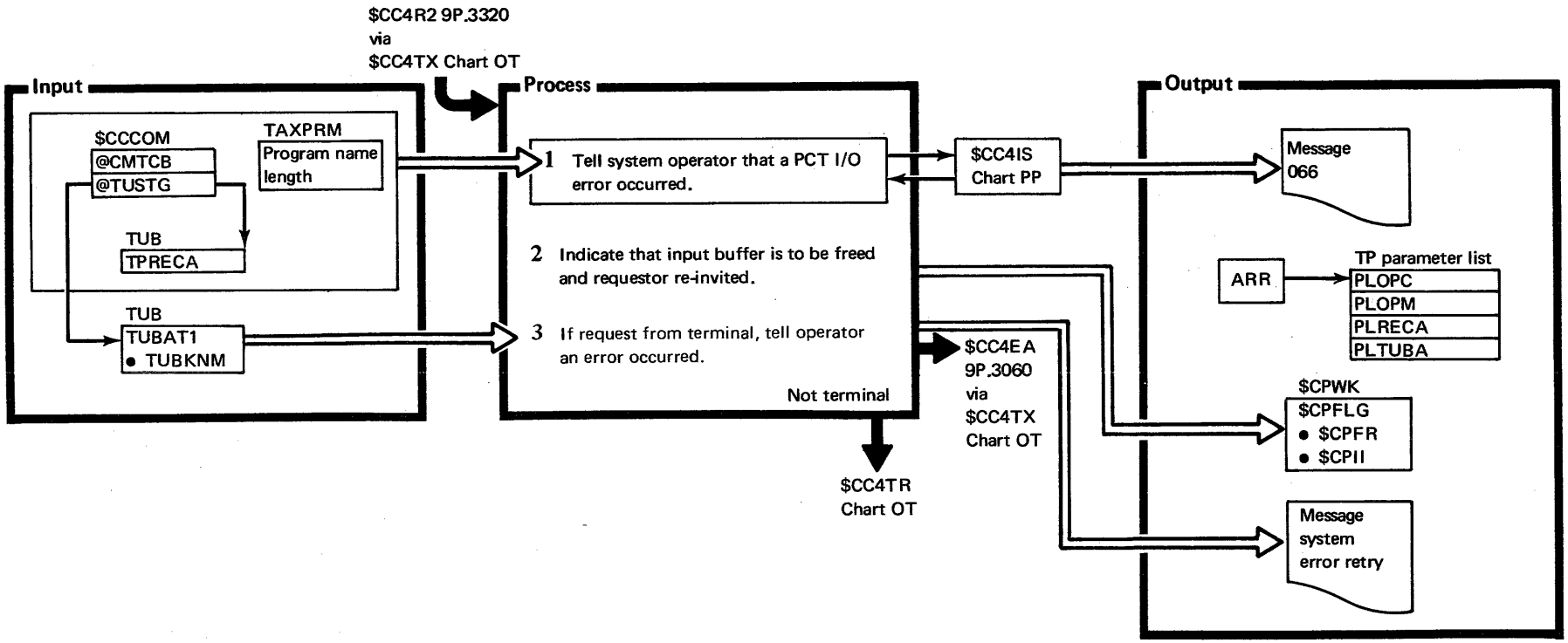
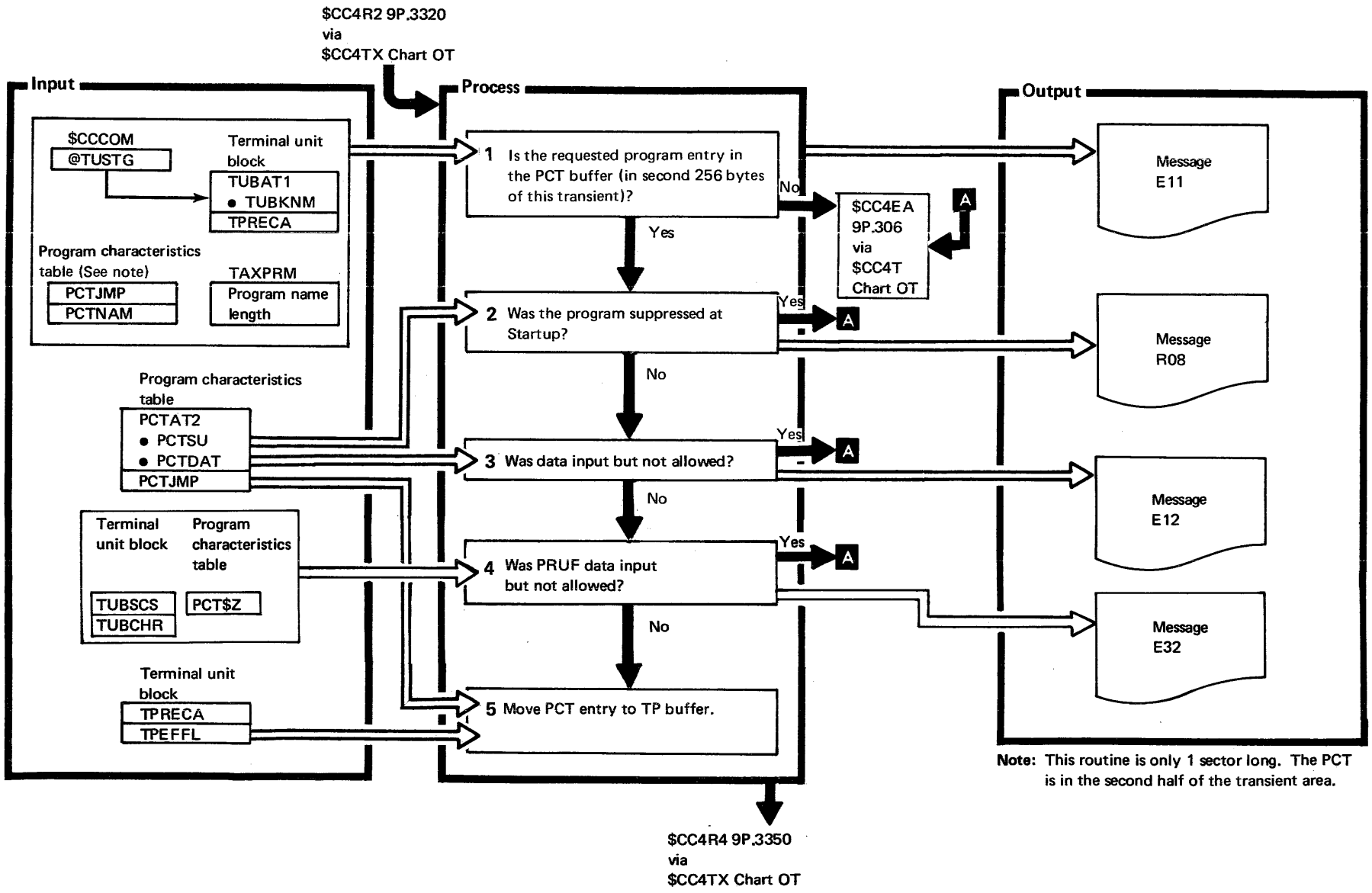


Diagram 9P.3330. $CC4R8



Note: This routine is only 1 sector long. The PCT is in the second half of the transient area.

Diagram 9P.3340. $CC4R3

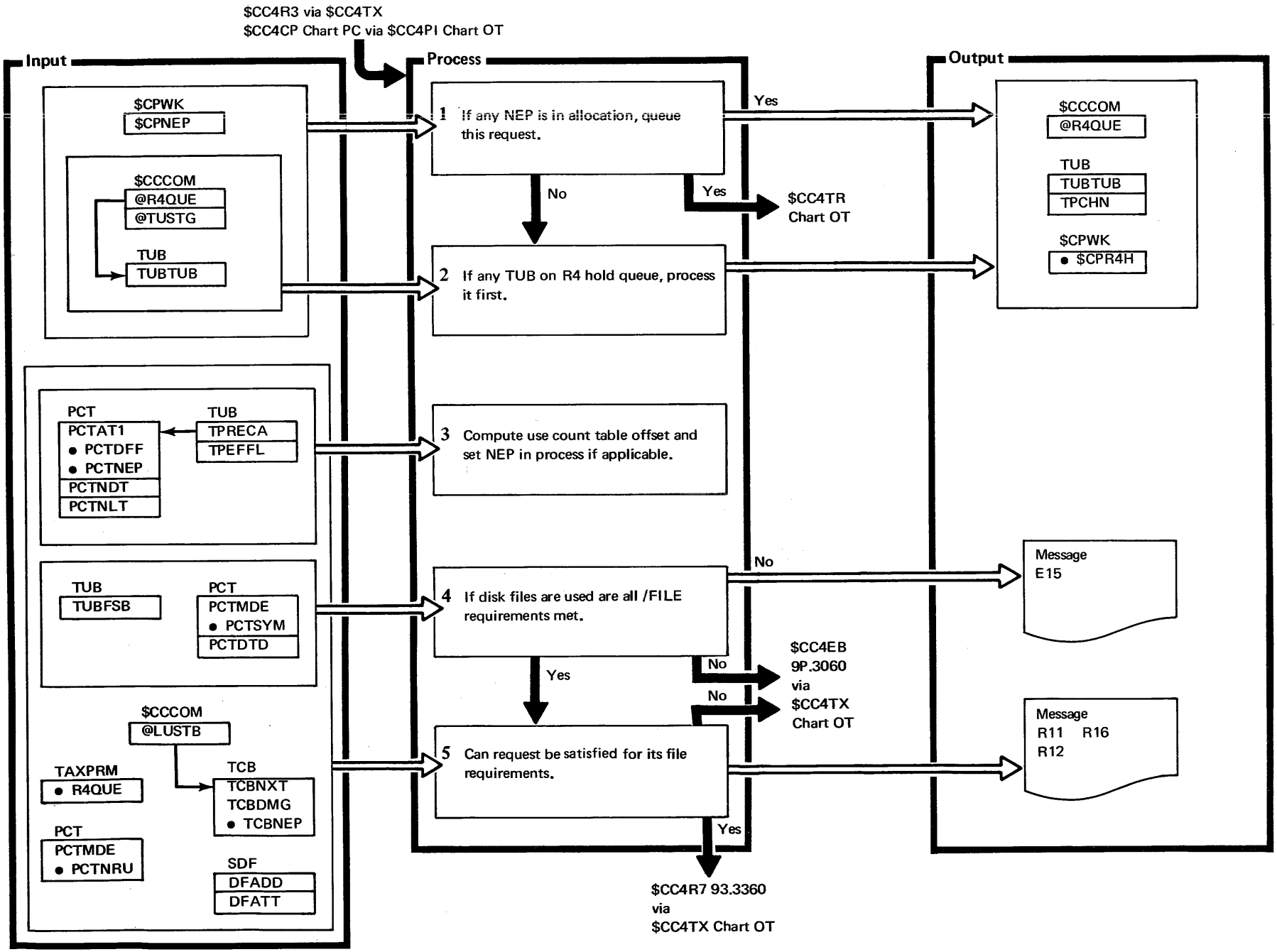
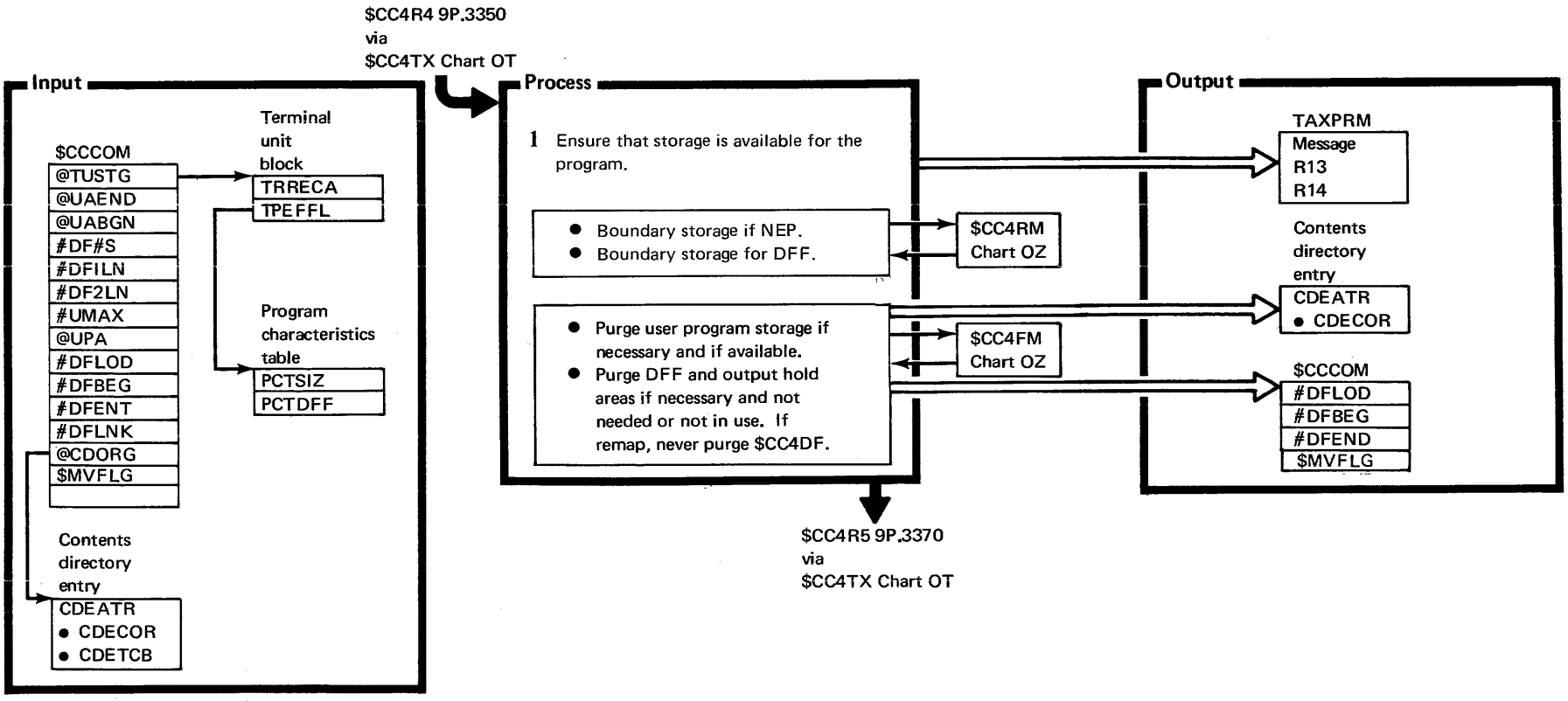


Diagram 9P.3350. $CC4R4



• **Diagram 9P.3360. $CC4R7**

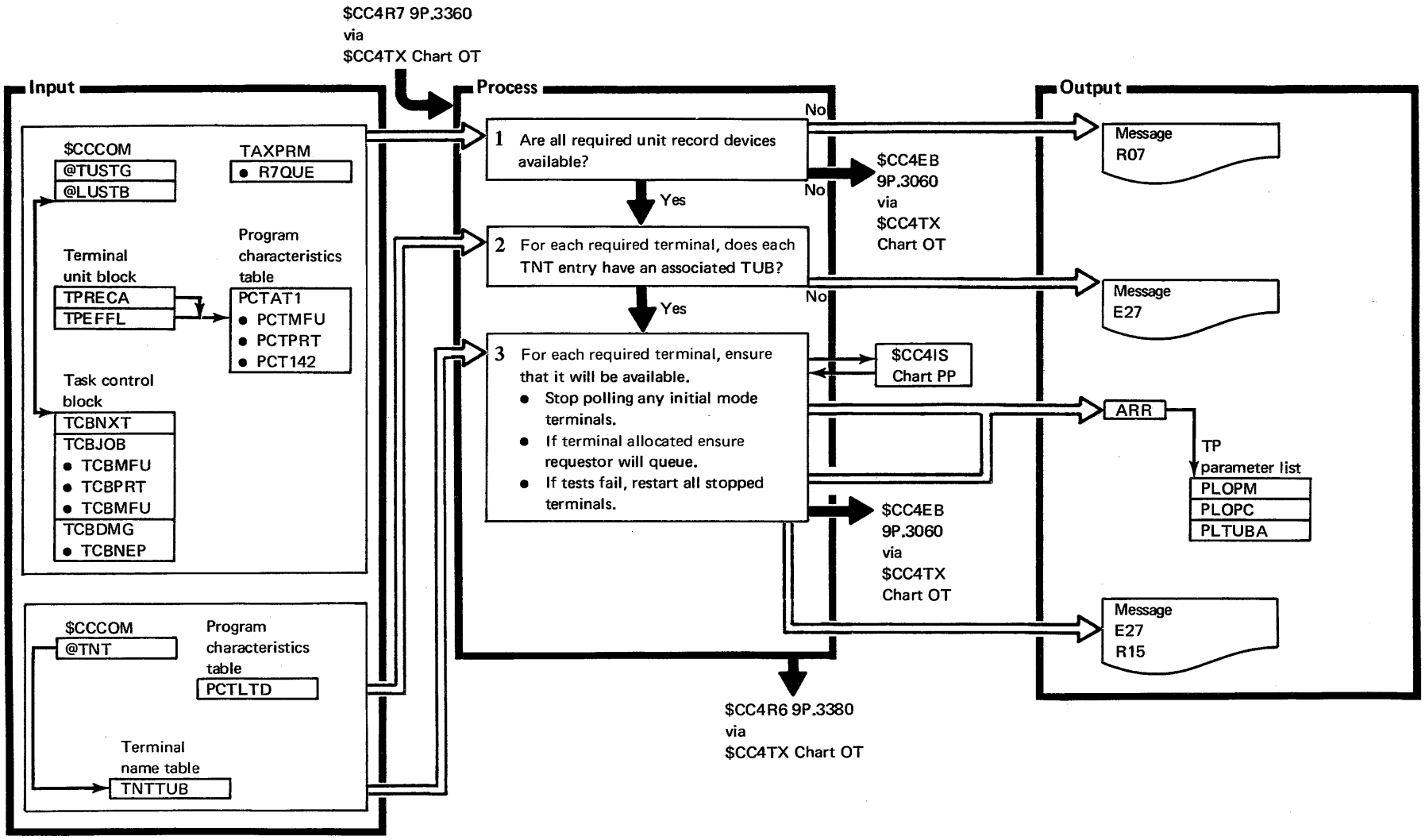


Diagram 9P.3370. $CC4R5

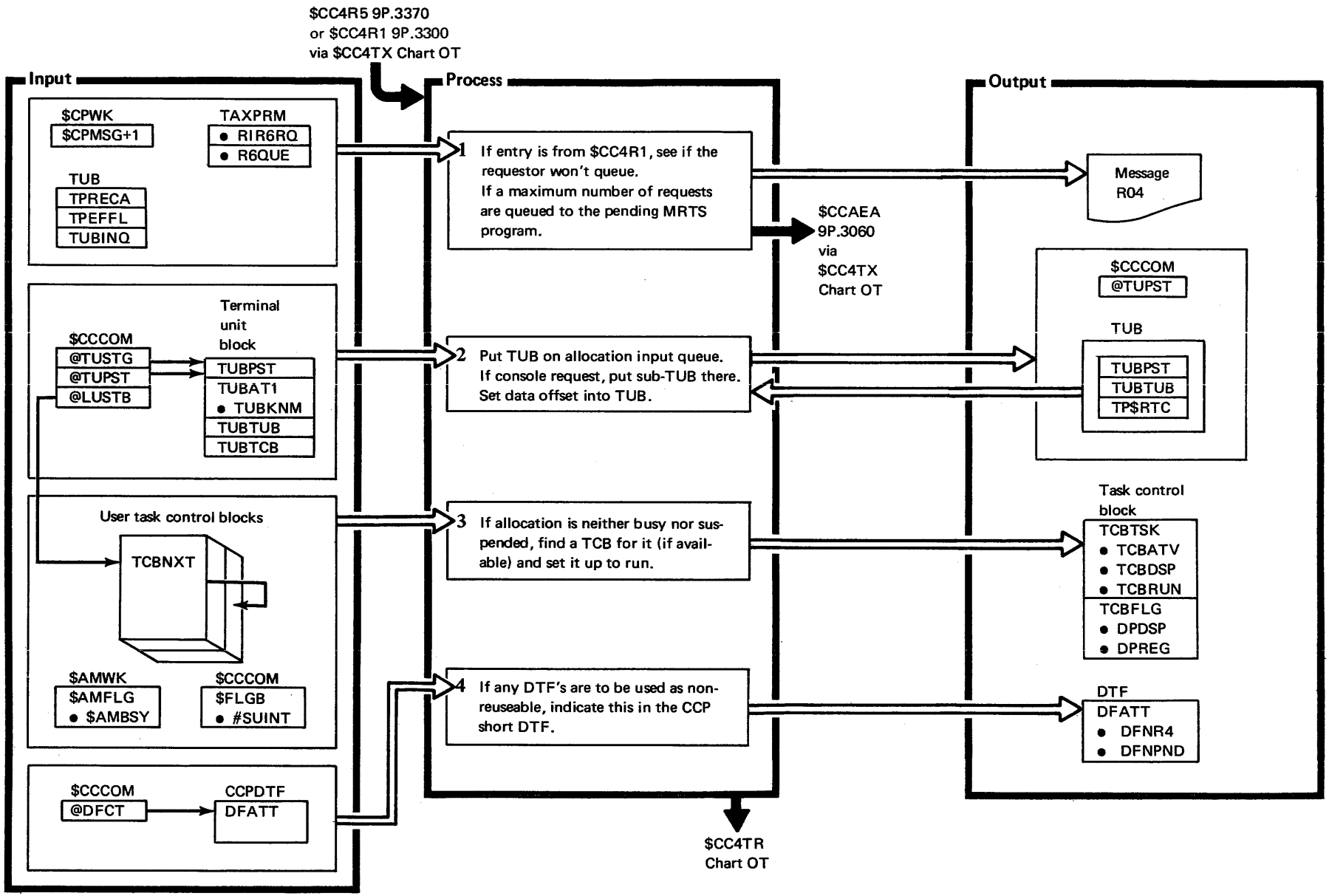


Diagram 9P.3380. $CC4R6

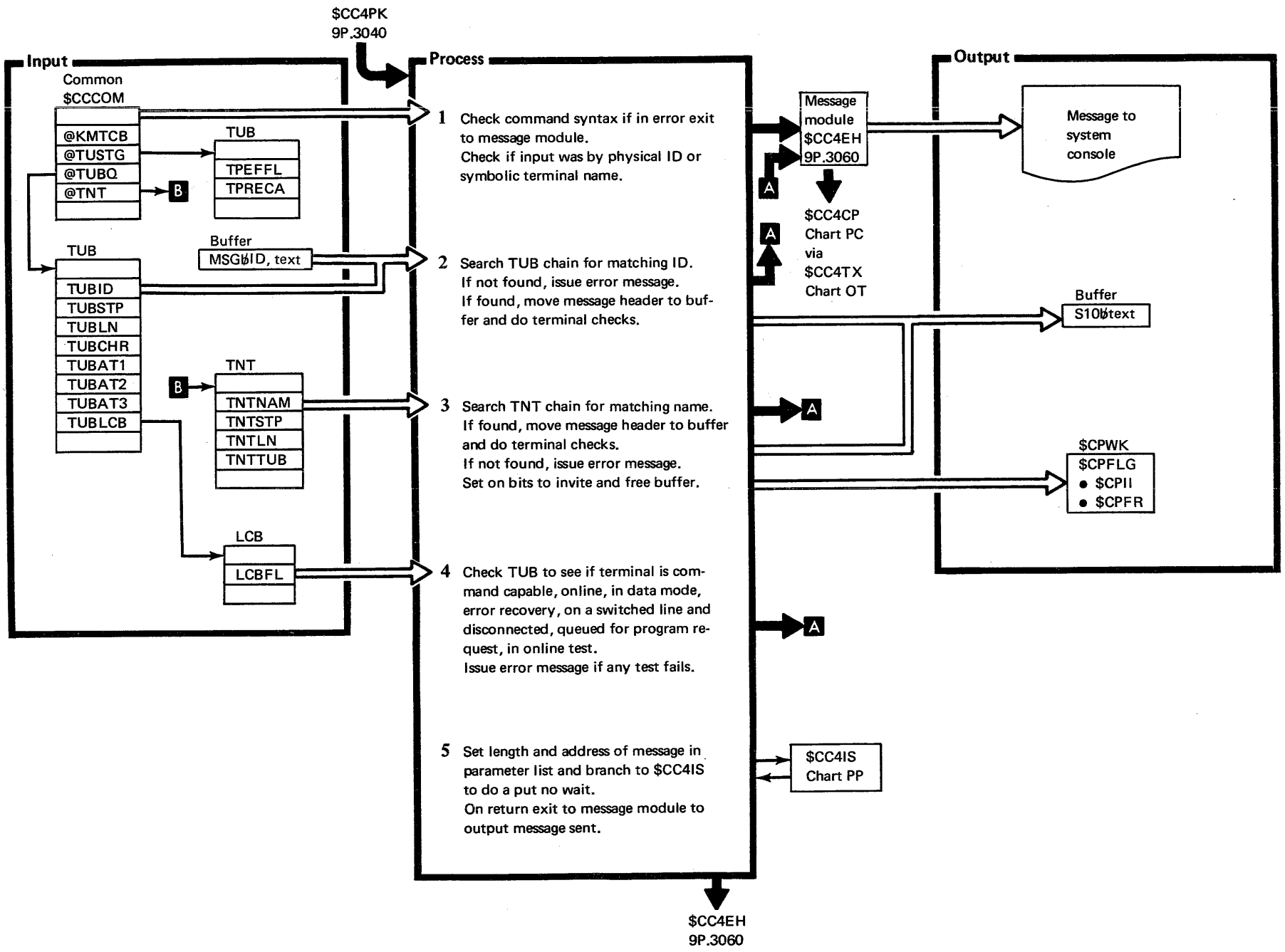


Diagram 9P.3400. $CC4CG

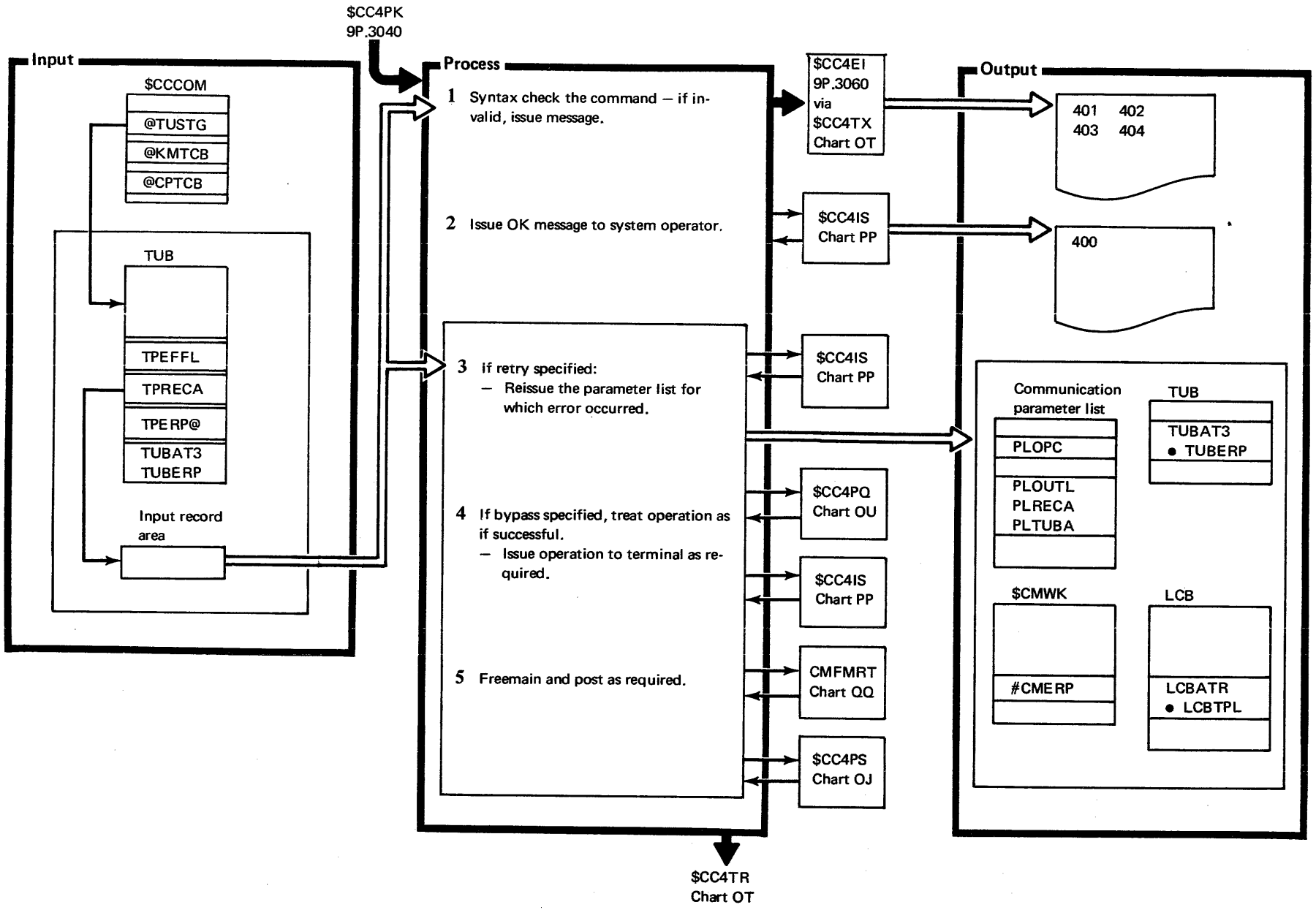
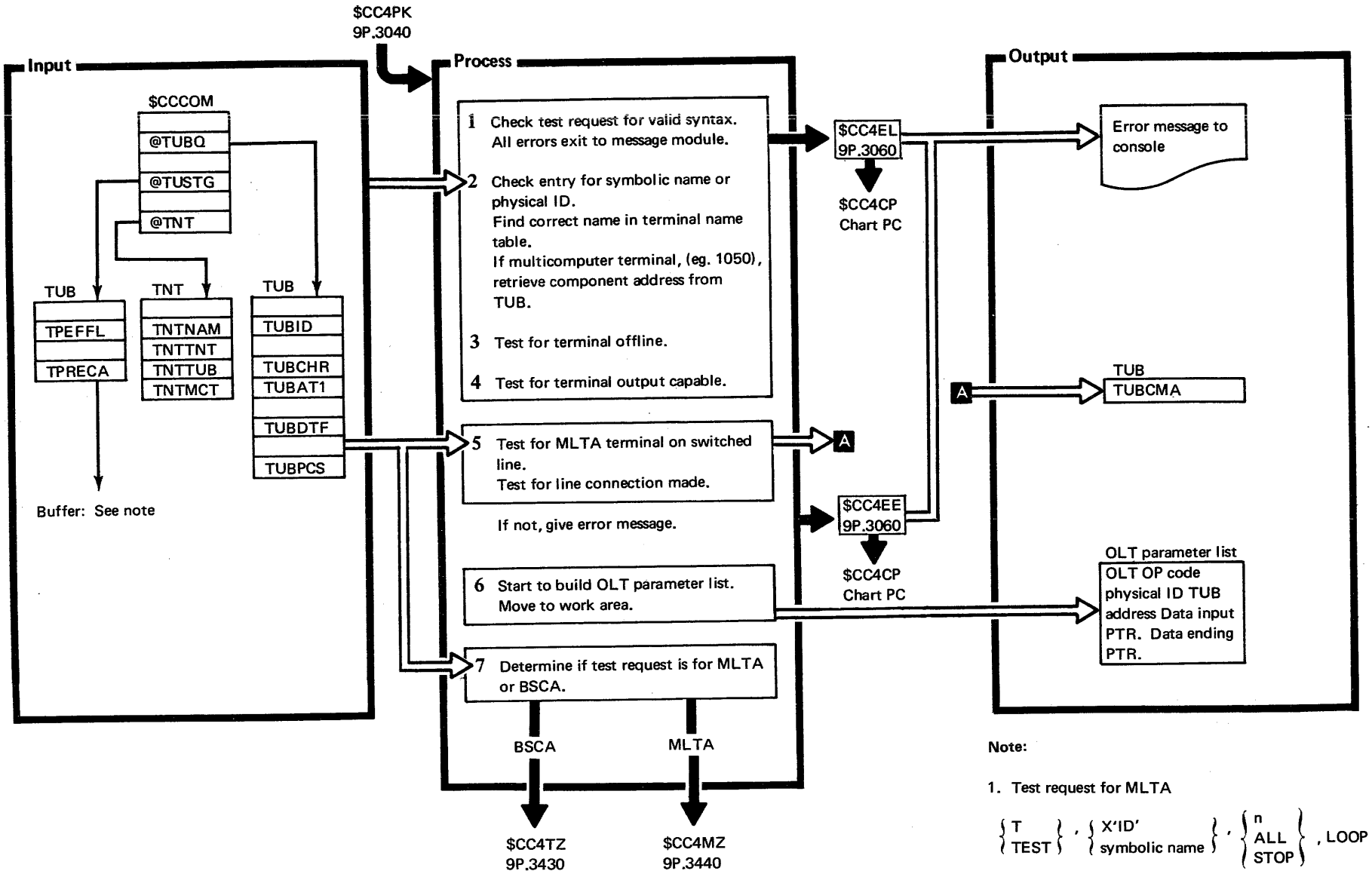


Diagram 9P.3410. $CC4RP



Note:

1. Test request for MLTA

$$\left\{\begin{matrix}T\\TEST\end{matrix}\right\},\left\{\begin{matrix}X'ID'\\symbolic\ name\end{matrix}\right\},\left\{\begin{matrix}n\\ALL\\STOP\end{matrix}\right\},LOOP$$

2. Test request for BSCA

$$\left\{\begin{matrix}T\\TEST\end{matrix}\right\},\left\{\begin{matrix}X'ID'\\symbolic\ name\end{matrix}\right\},n,cc,[msg]$$

Diagram 9P.3420. $CC4TS

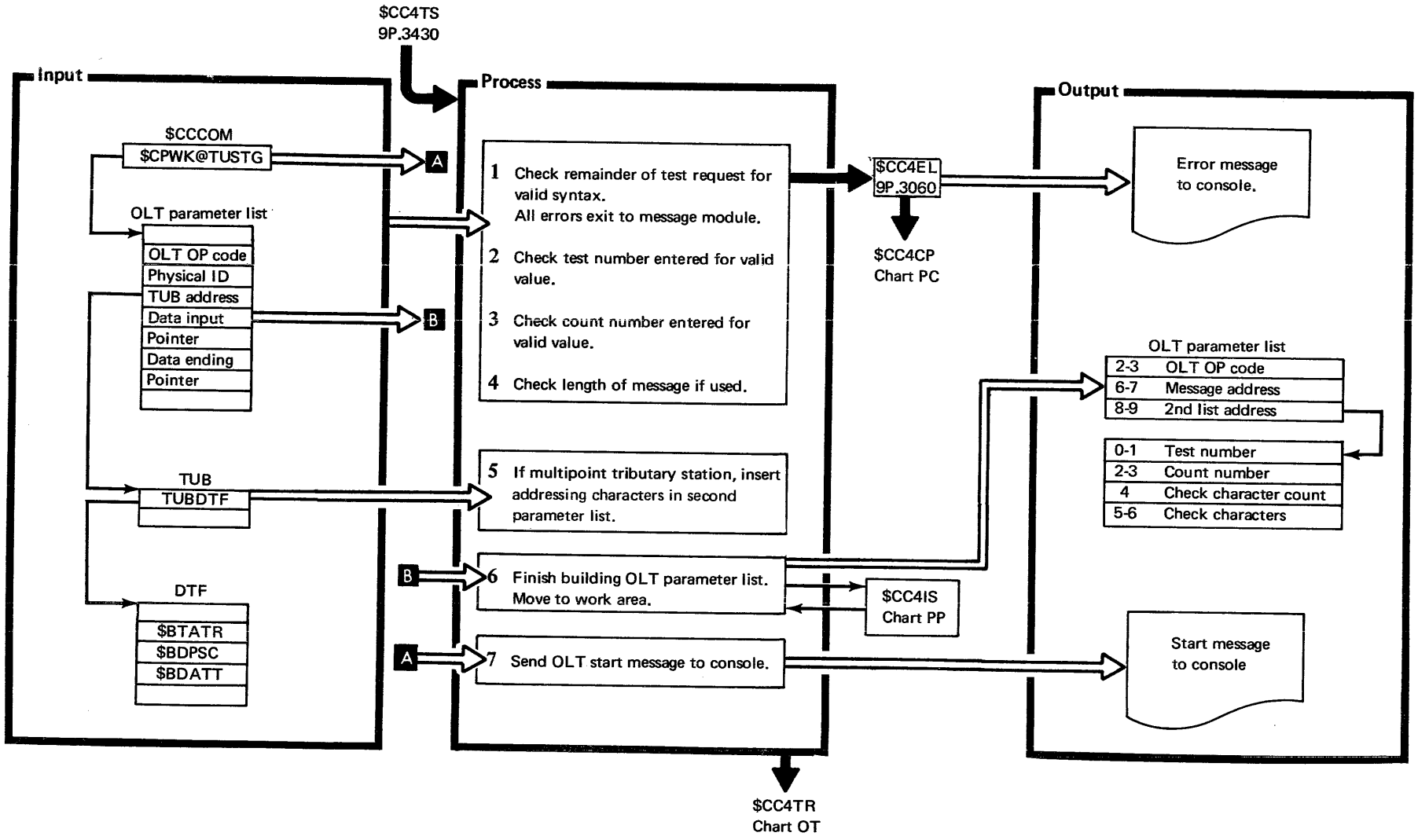


Diagram 9P.3430. $CC4TZ

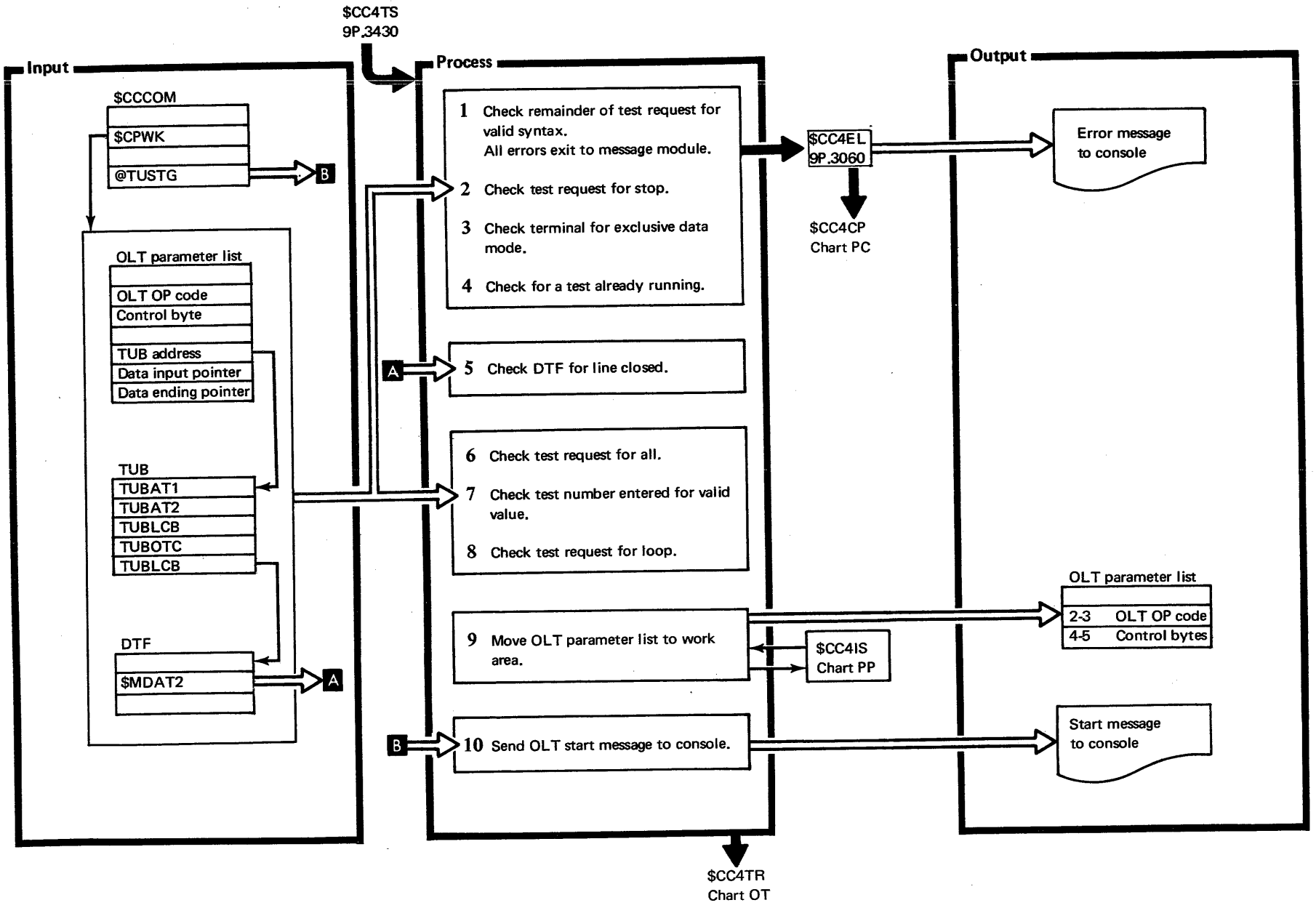


Diagram 9P.3440. $CC4MZ (Models 8, 10, and 12 Only)

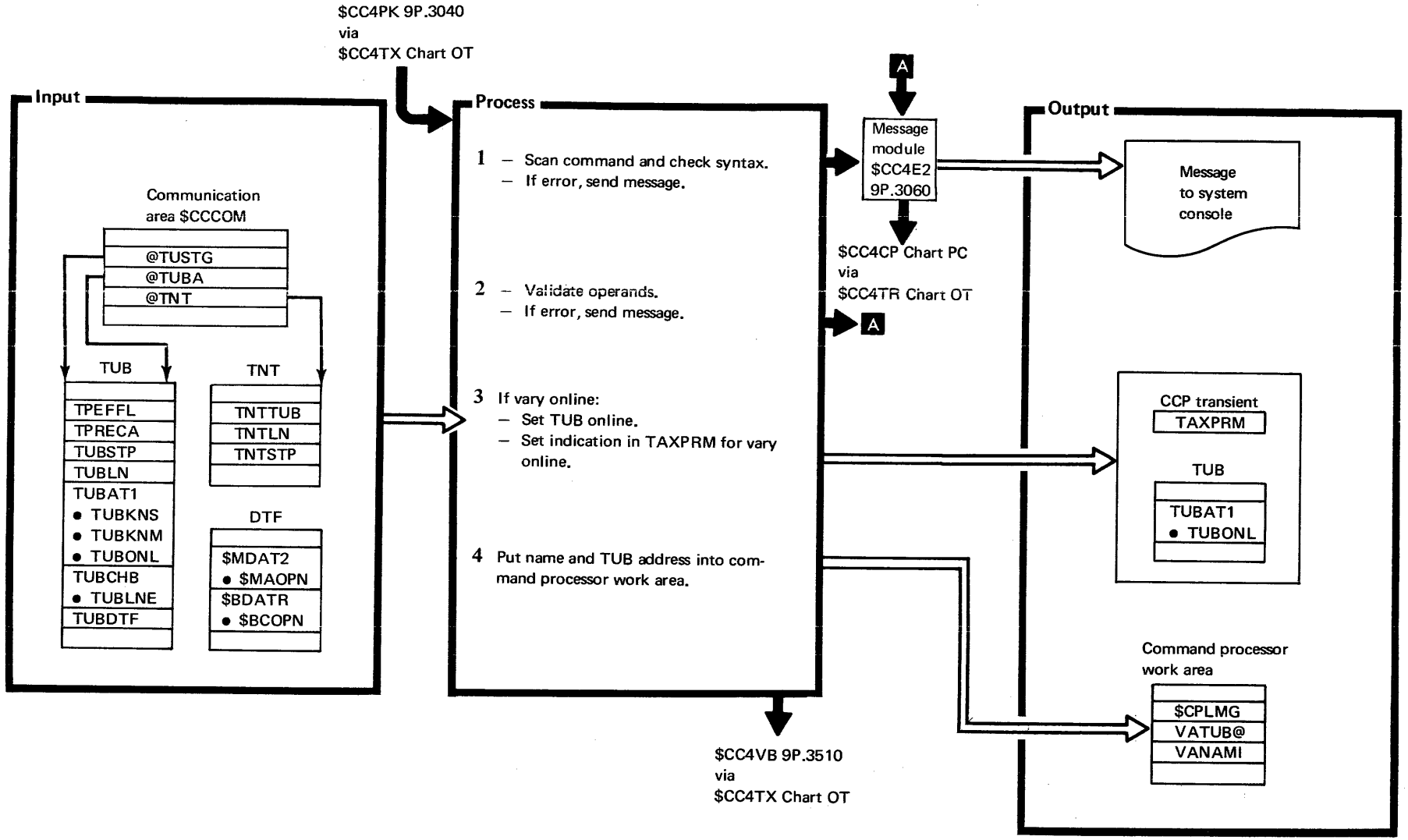


Diagram 9P.3500. $CC4VA

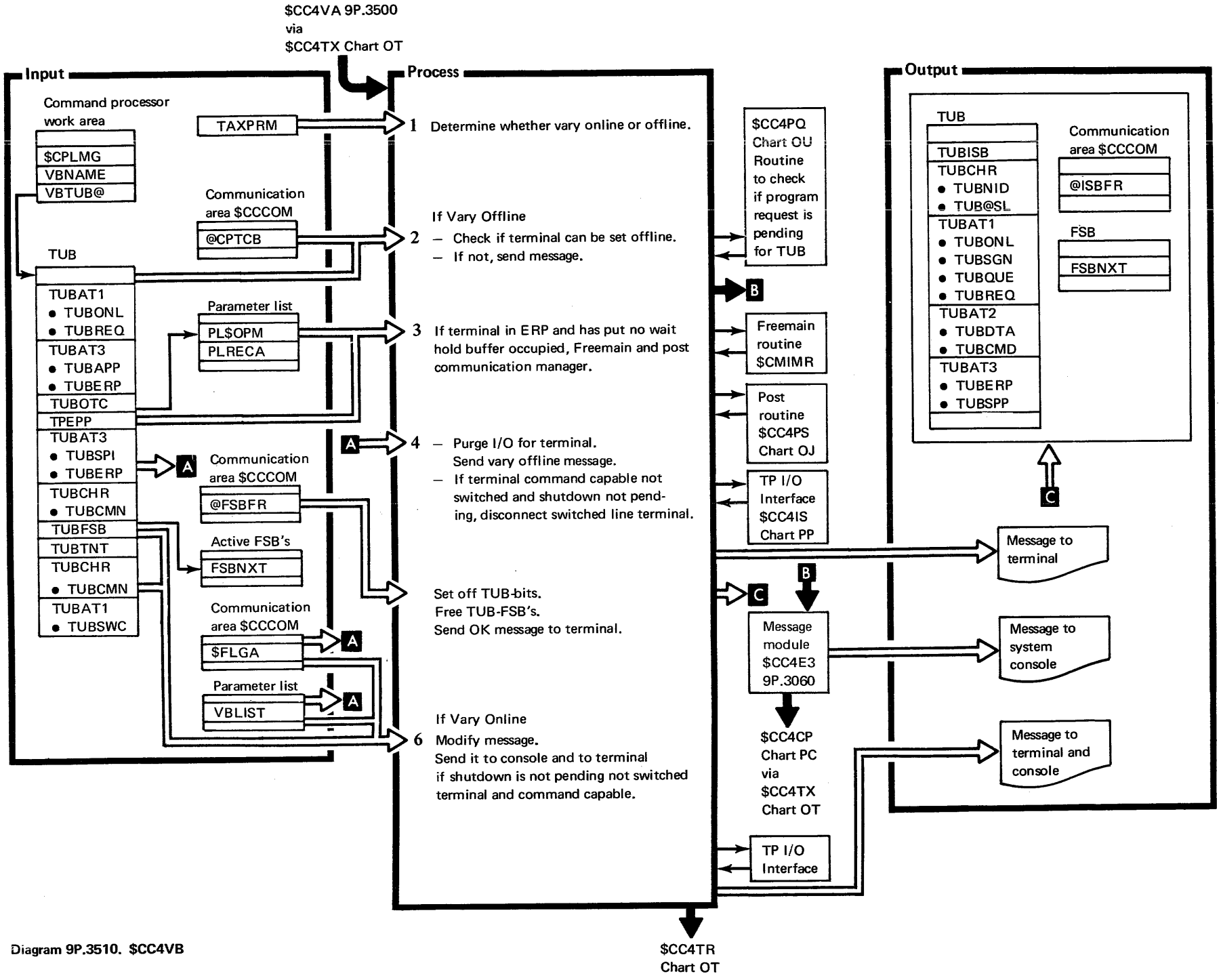


Diagram 9P.3510. $CC4VB

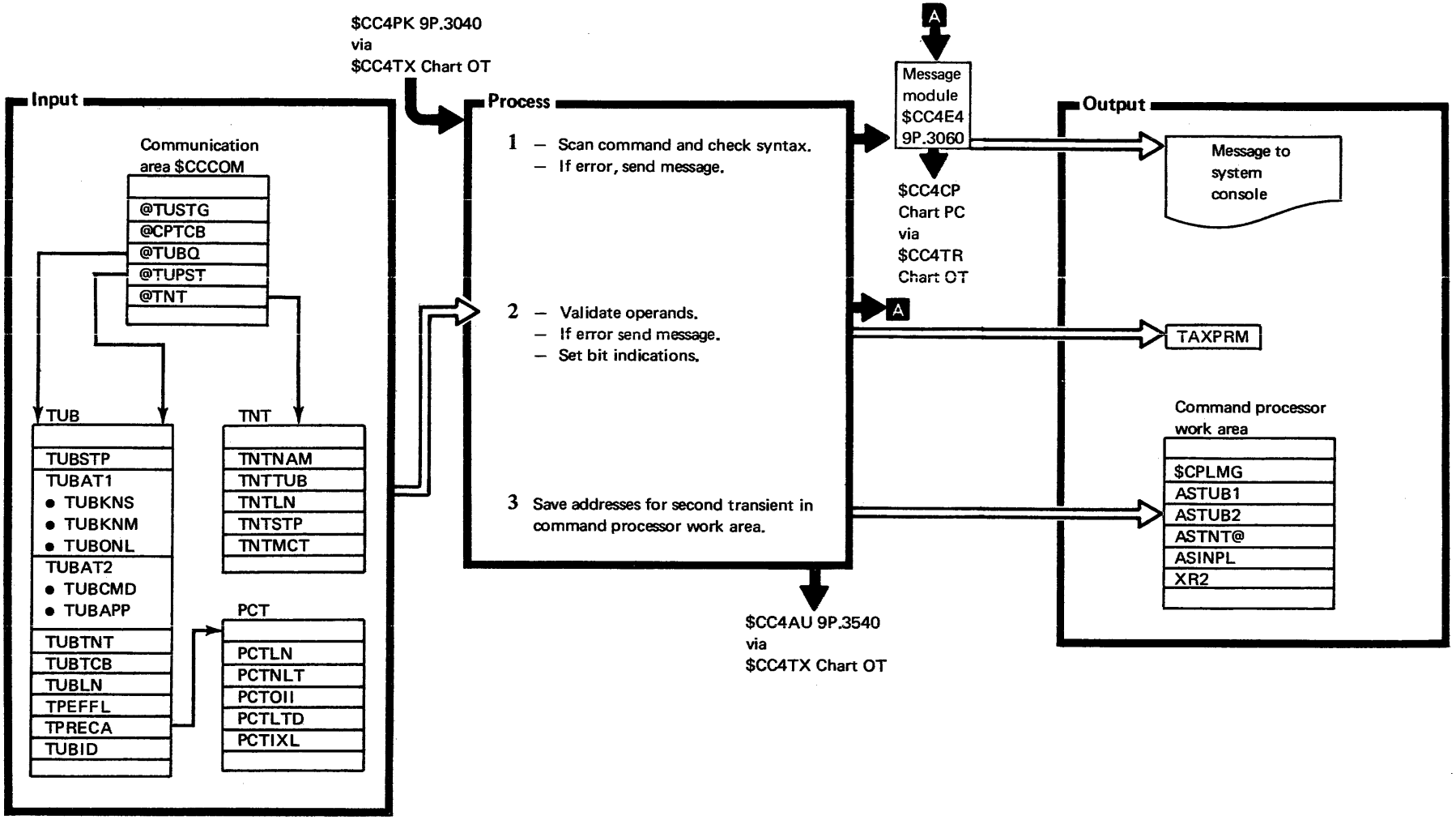


Diagram 9P.3520. $CC4AS

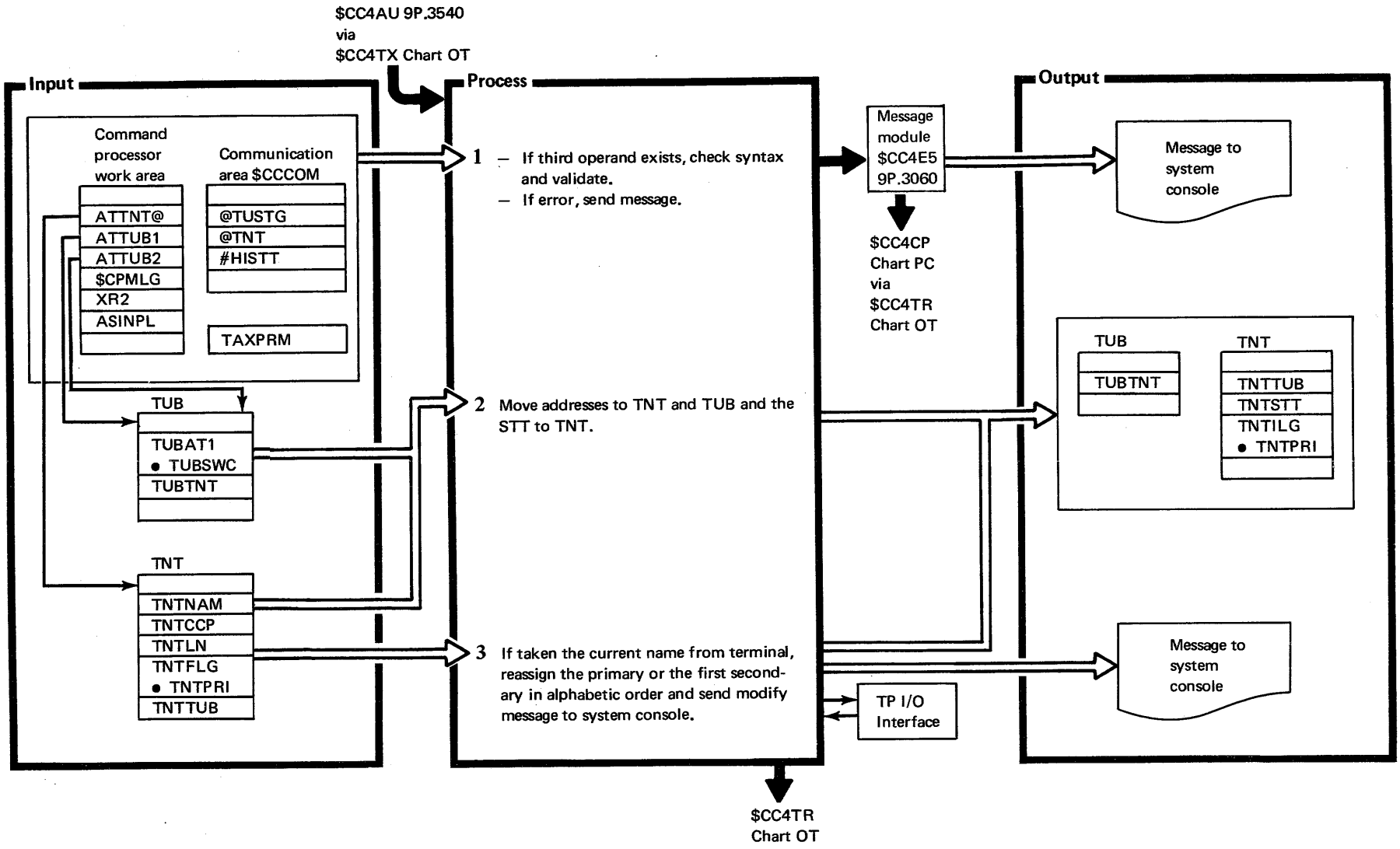


Diagram 9P.3530. $CC4AT

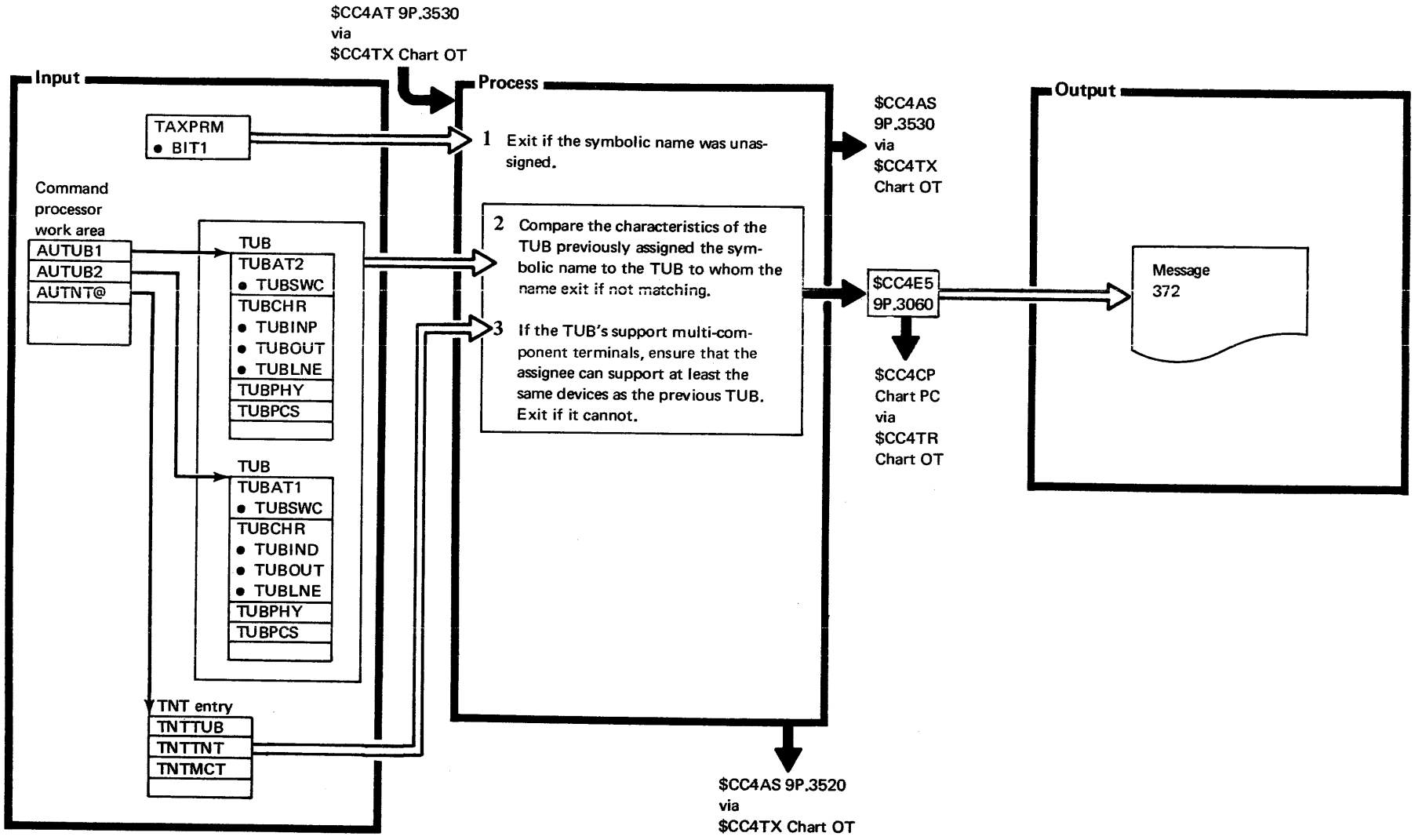


Diagram 9P.3540. $CC4AU

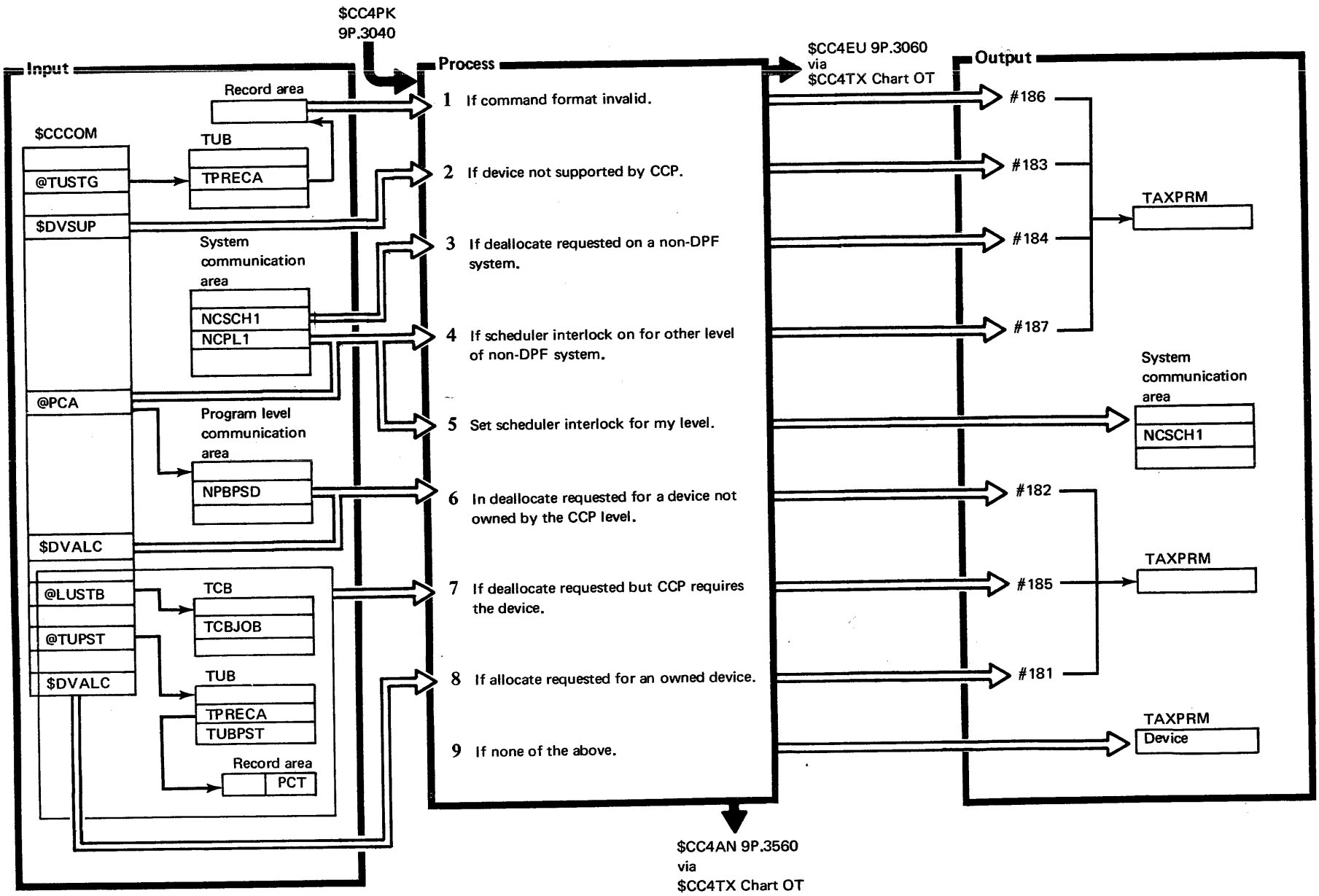
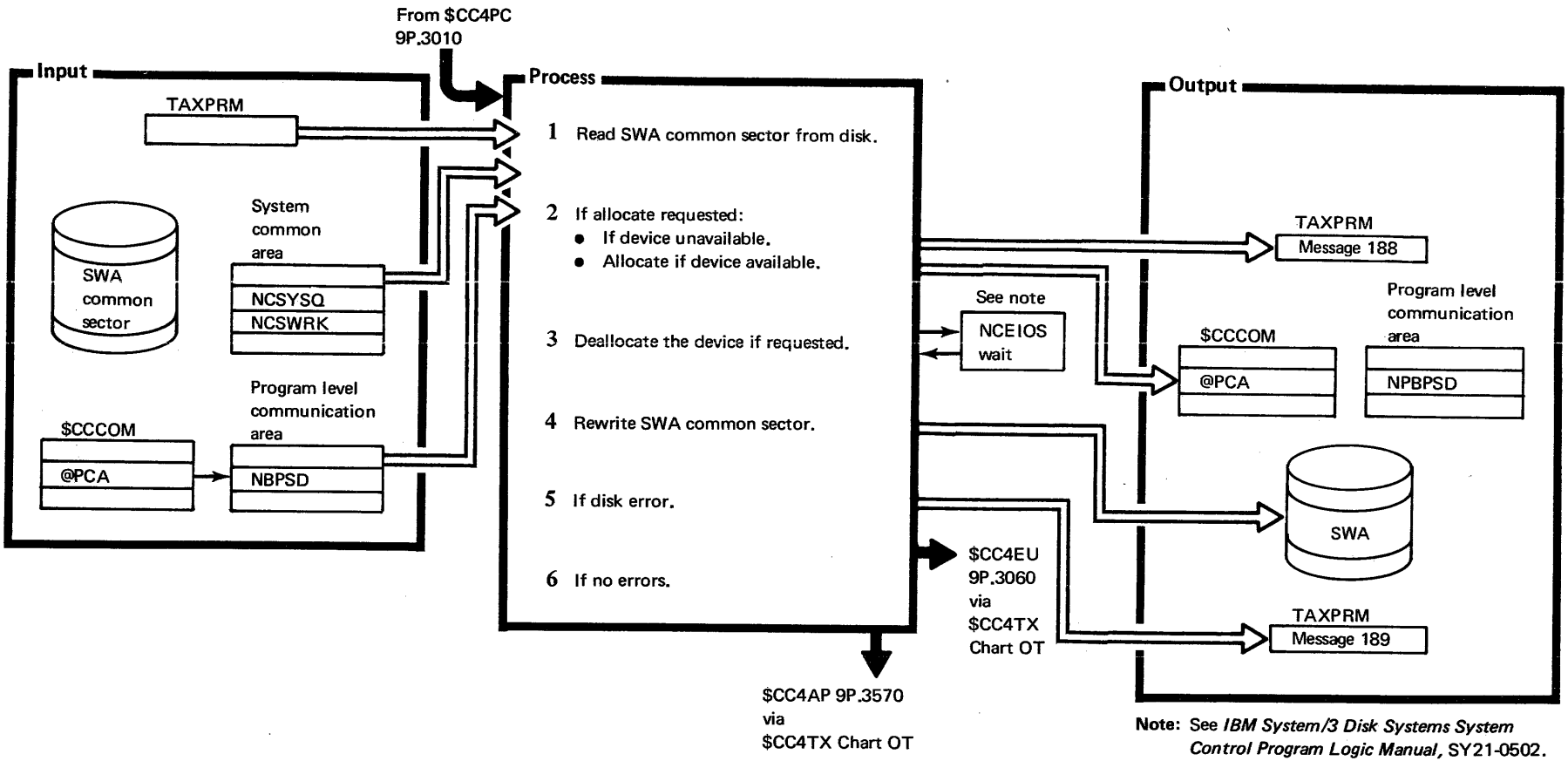
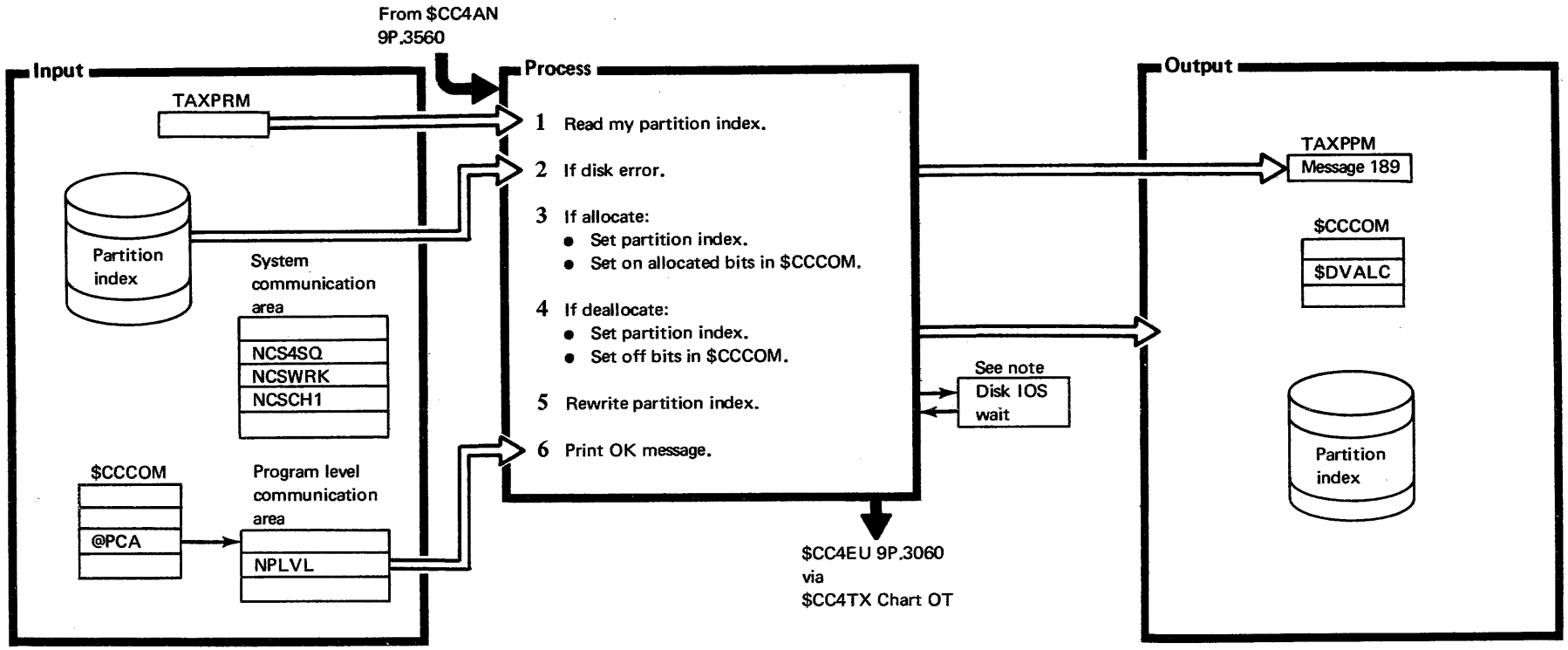


Diagram 9P.3550. $CC4AL



Note: See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.

Diagram 9P.3560. $CC4AN



Note: See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.

Diagram 9P.3570. $CC4AP

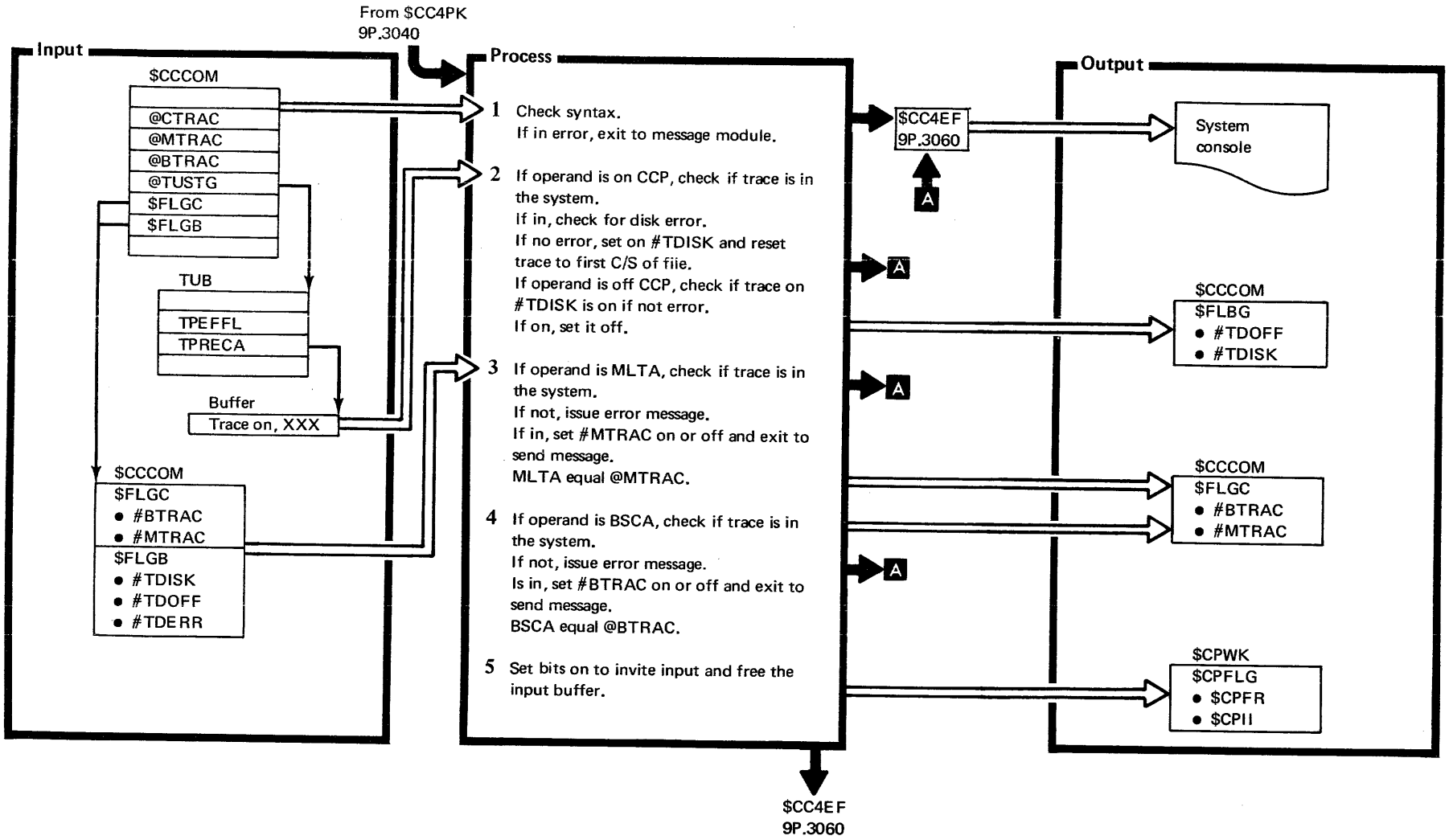


Diagram 9P.3600. $CC4TE

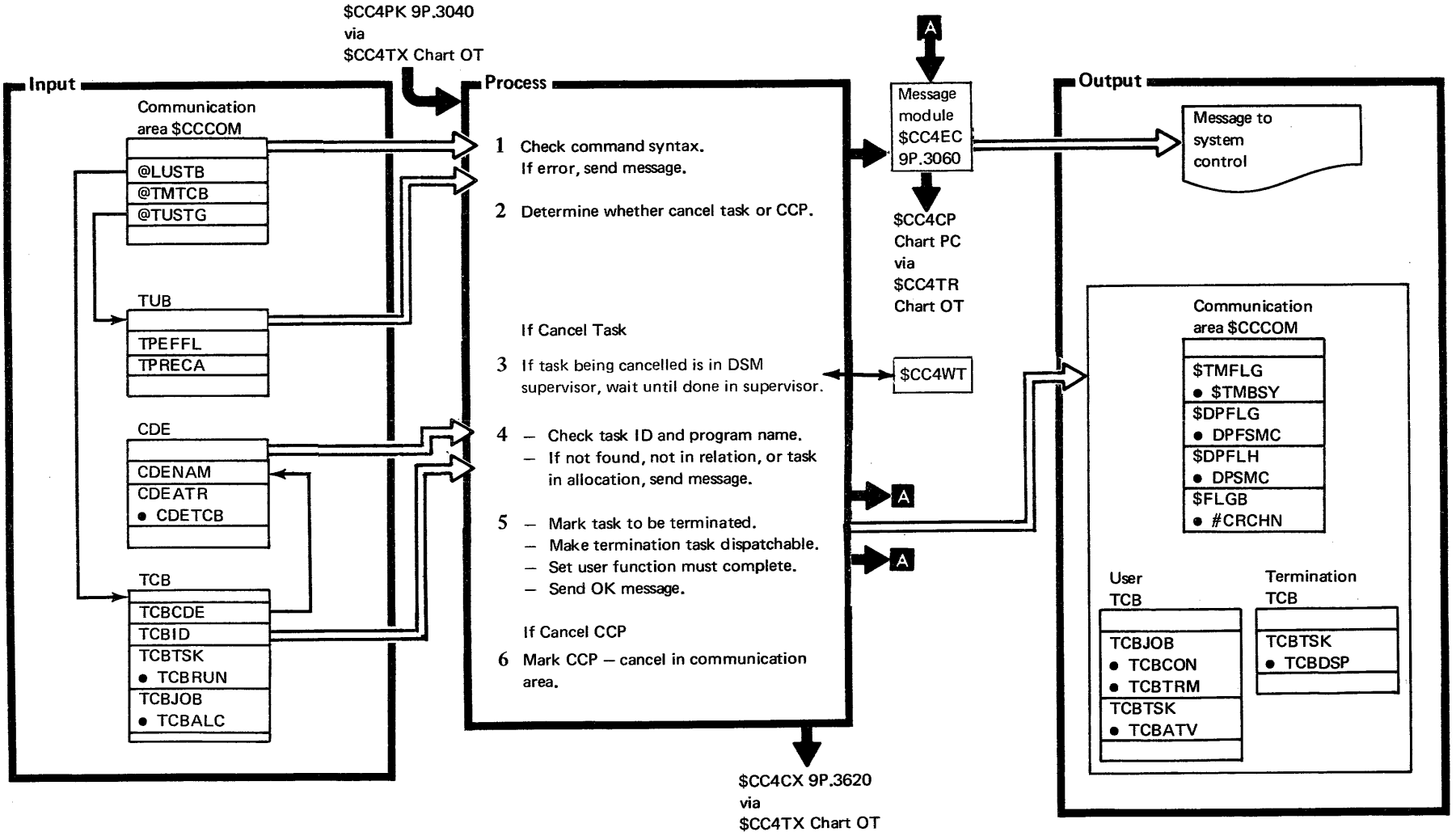


Diagram 9P.3610. $CC4CJ

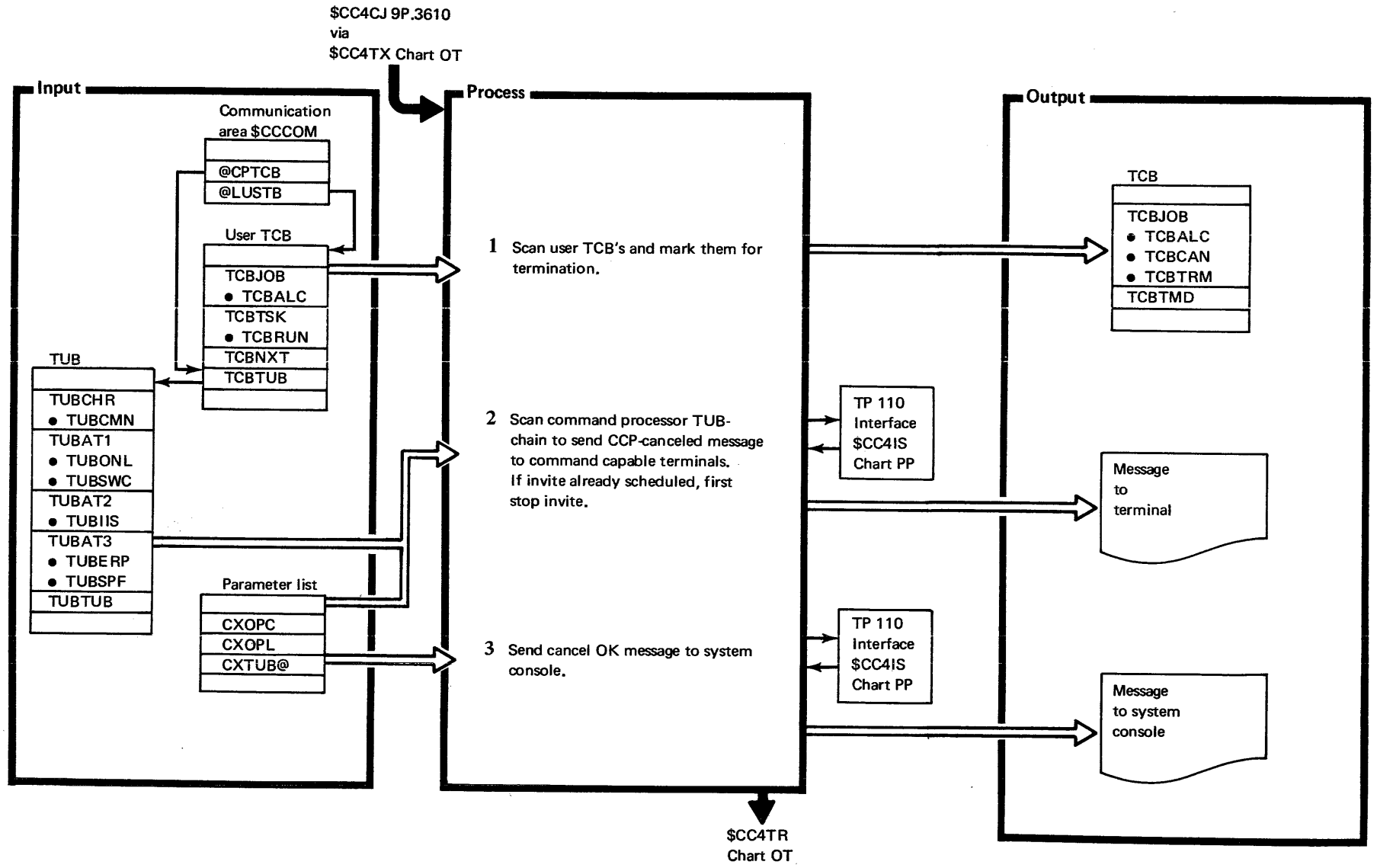


Diagram 9P.3620. $CC4CX

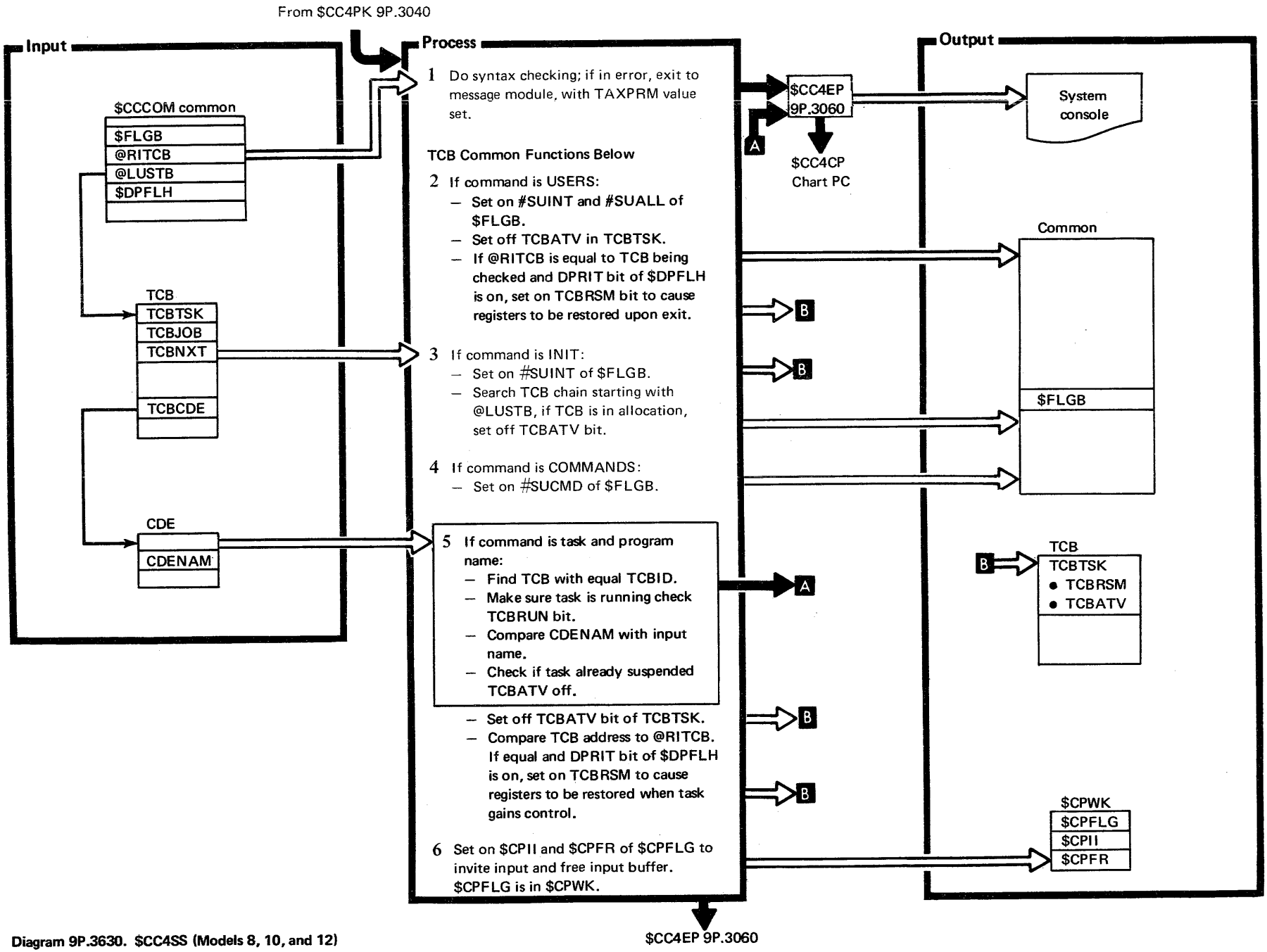


Diagram 9P.3630. $CC4SS (Models 8, 10, and 12)

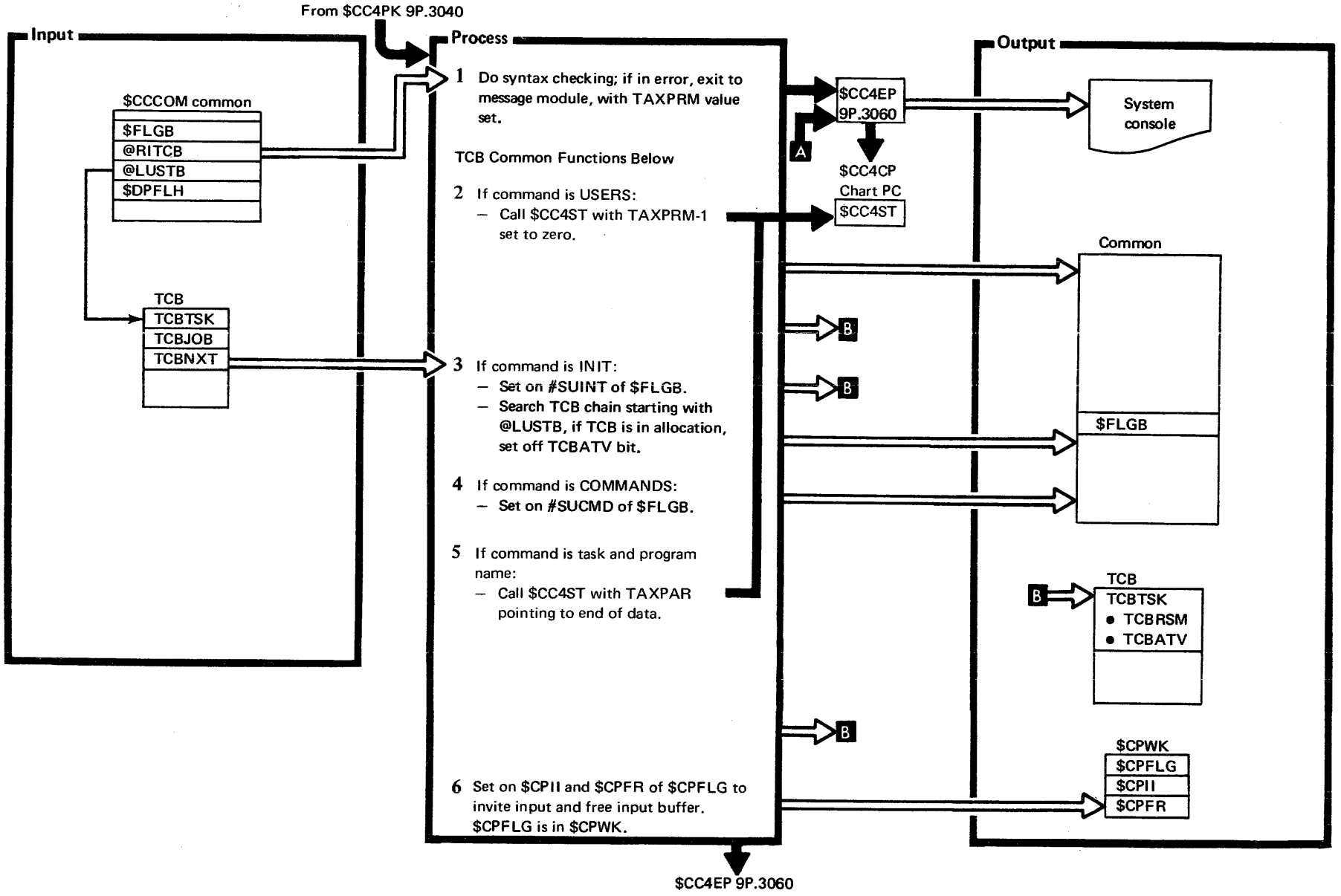


Diagram 9P.3635. $CC4SS (Model 4)

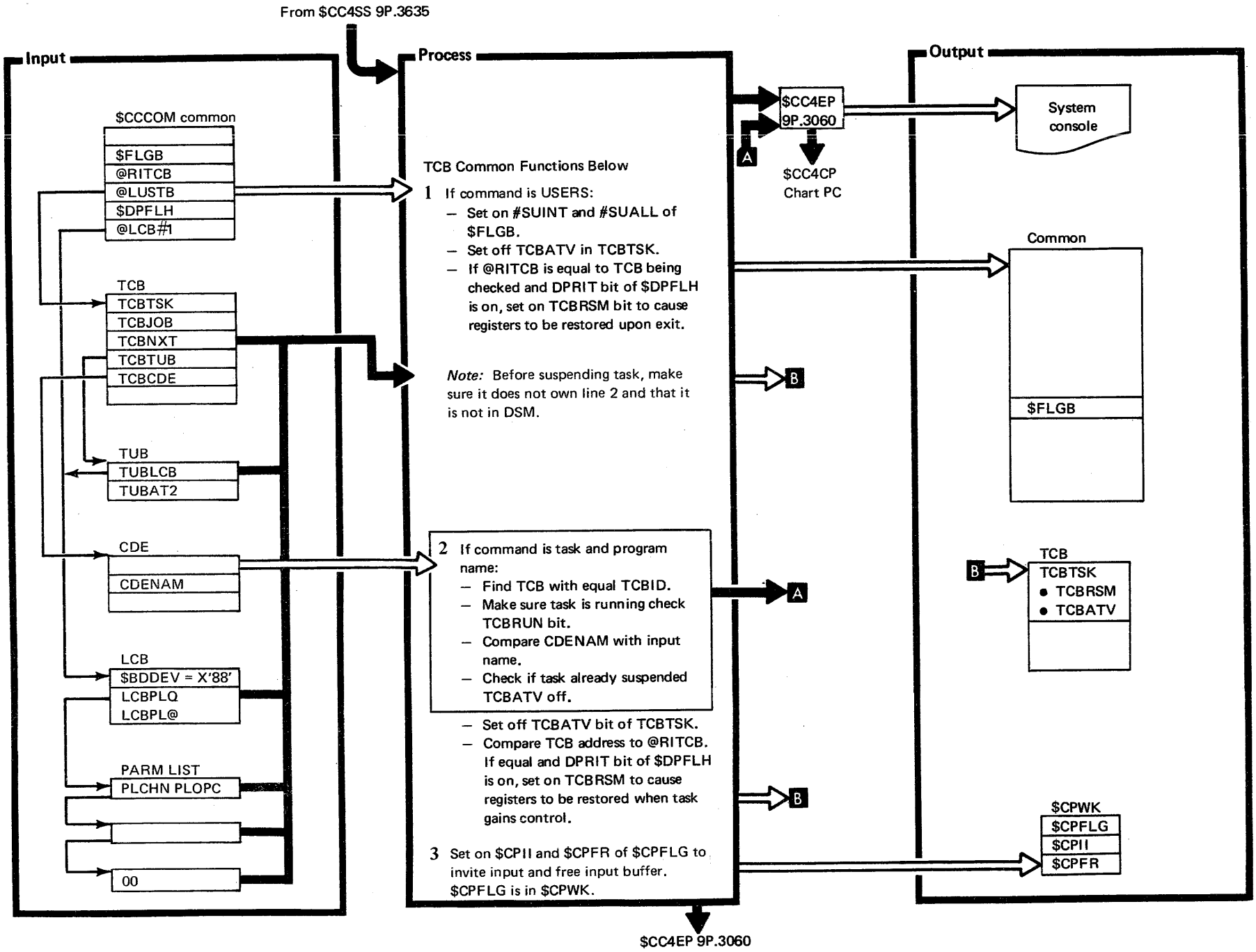


Diagram 9P.3640. $CC4ST (Model 4)

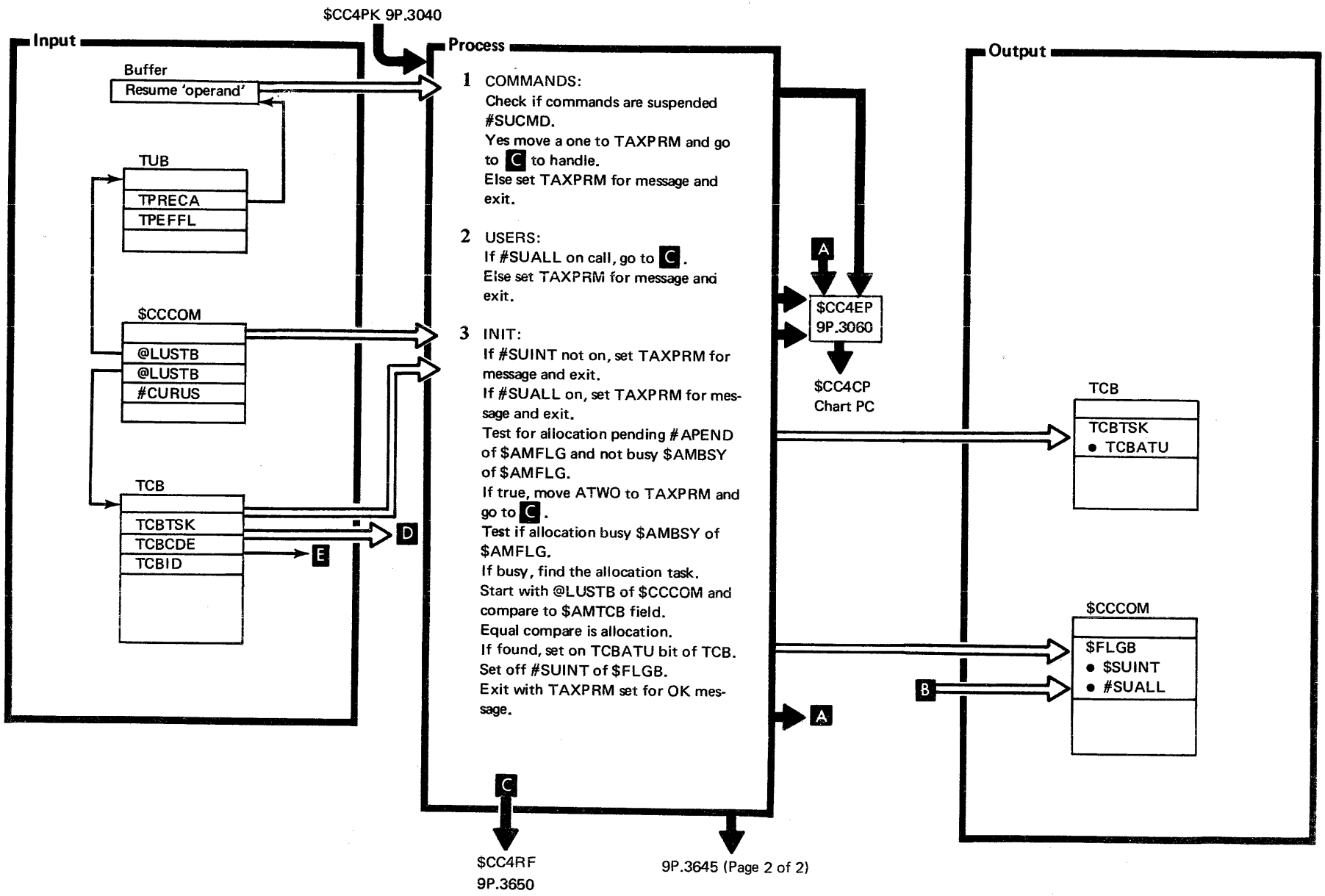


Diagram 9P.3645 (Part 1 of 2). $CC4RE

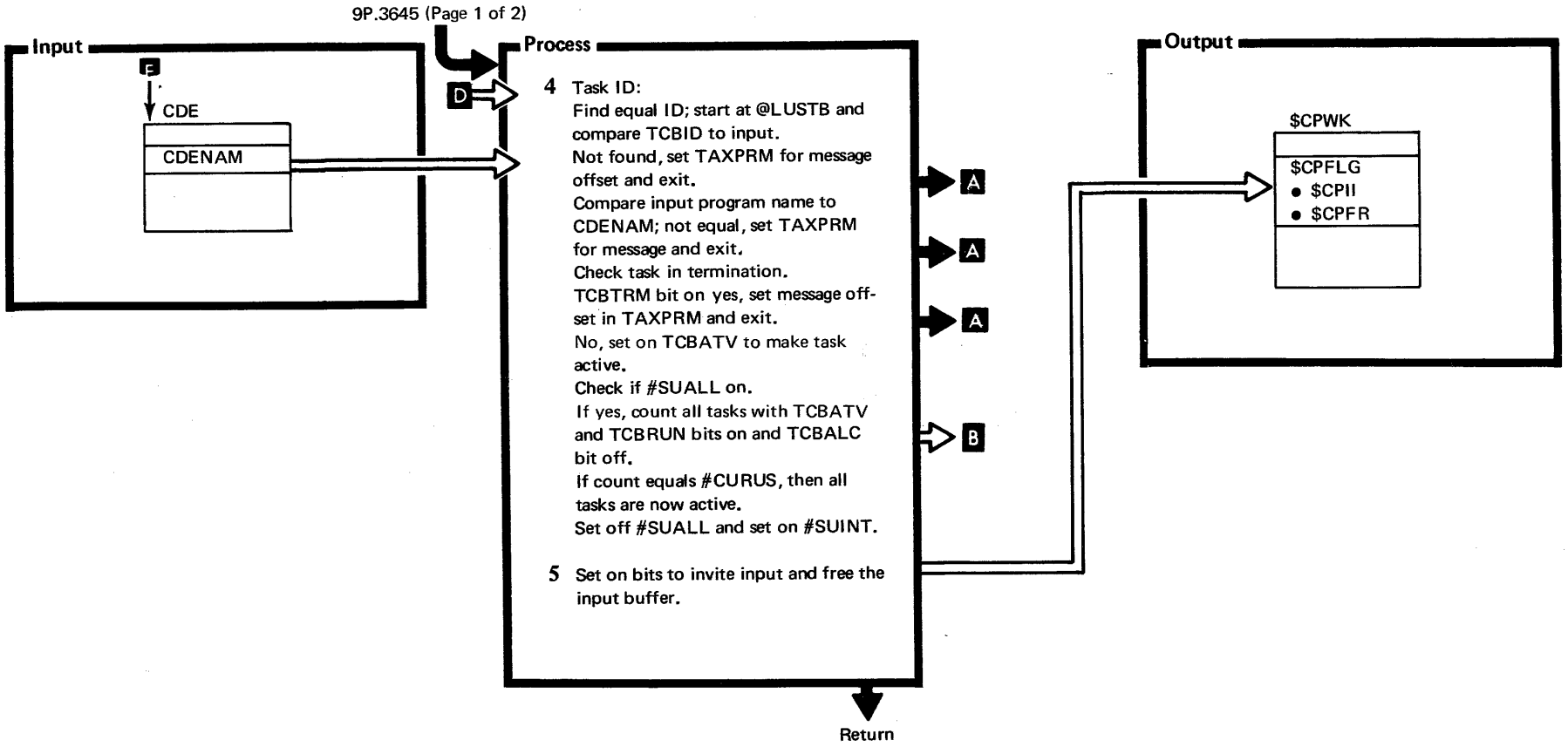


Diagram 9P.3645 (Part 2 of 2). $CC4RE

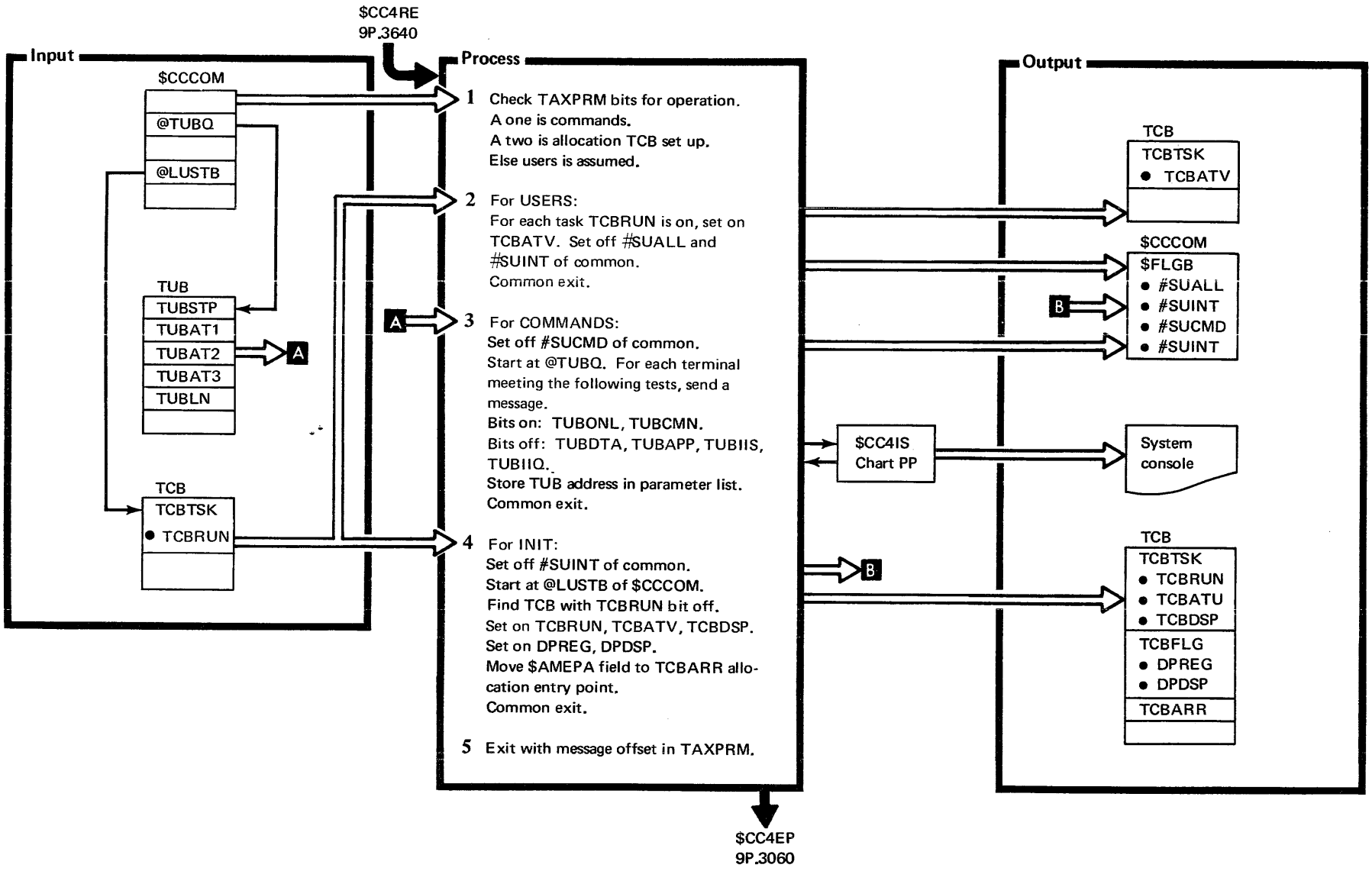


Diagram 9P.3650. $CC4RF

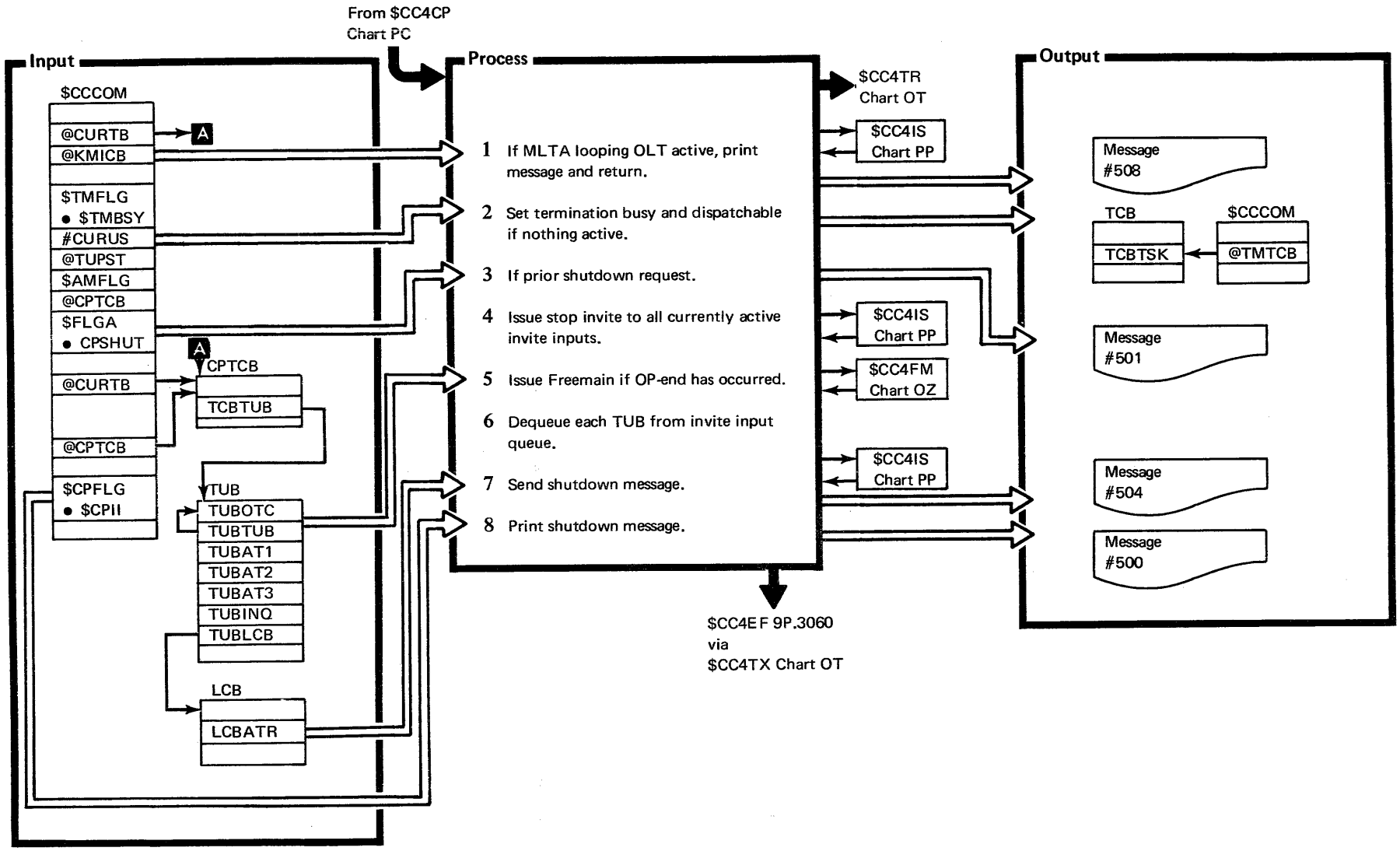


Diagram 9P.3660. $CC4SH

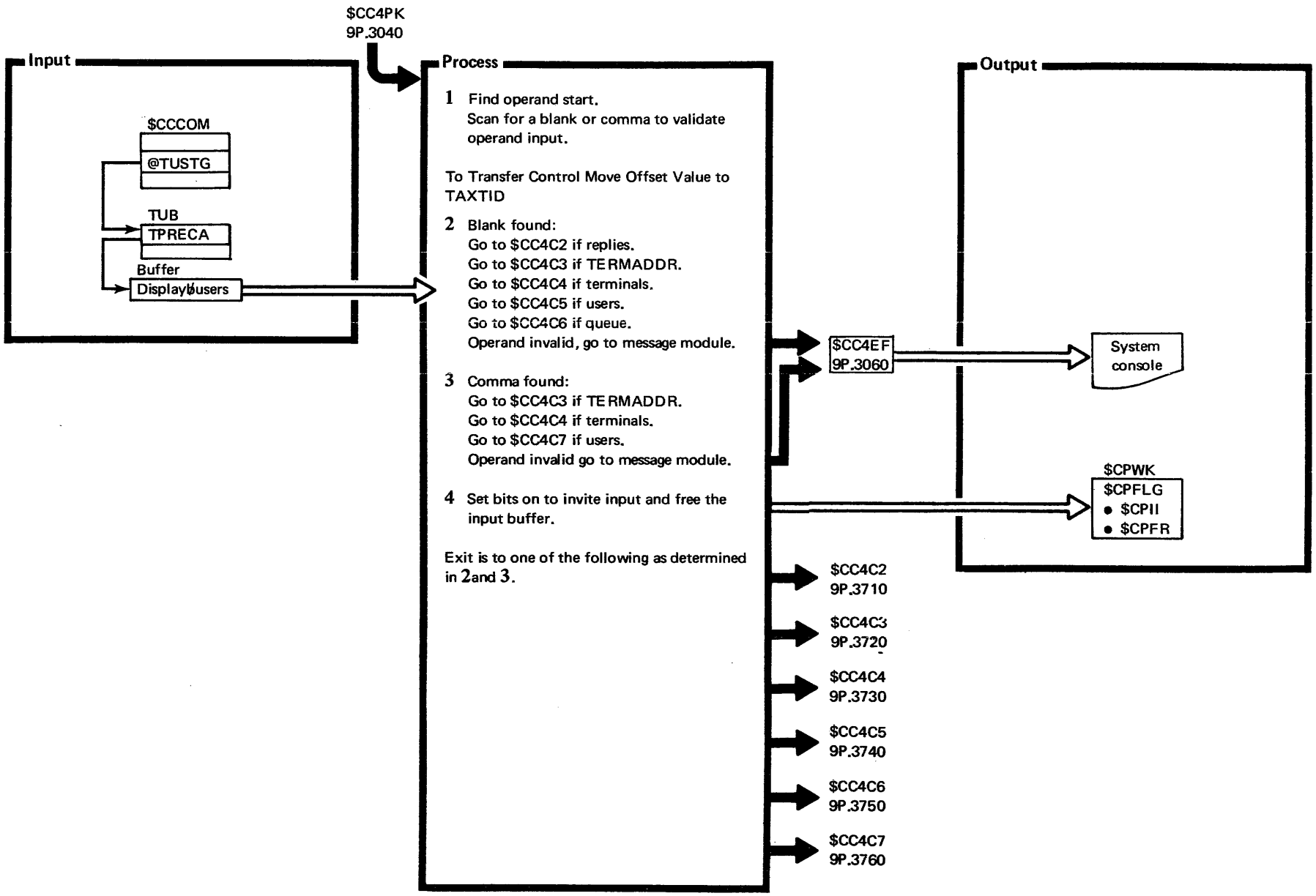


Diagram 9P.3700. $CC4C1

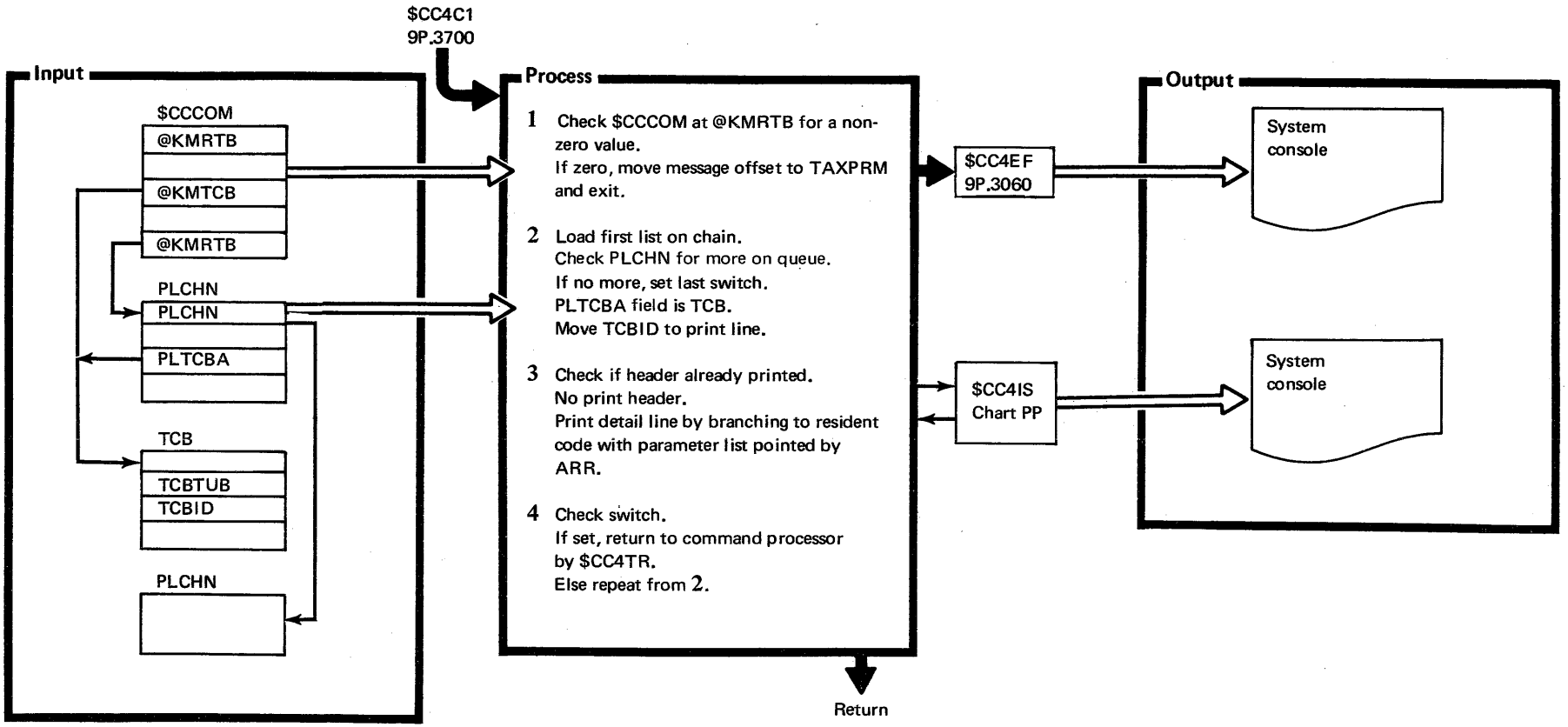


Diagram 9P.3710. $CC4C2 (Models 8, 10, 12)

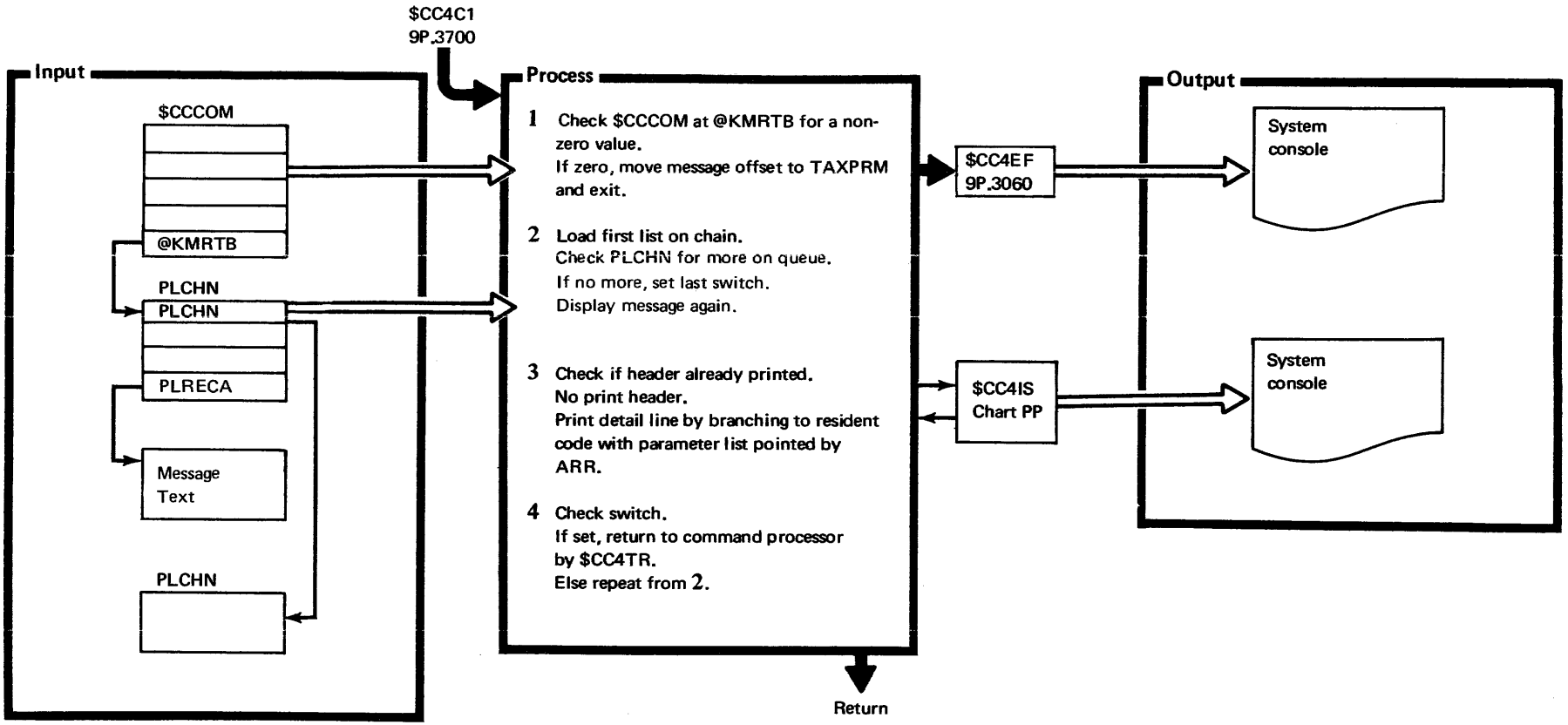


Diagram 9P.3715. $CC4C2 (Model 4)

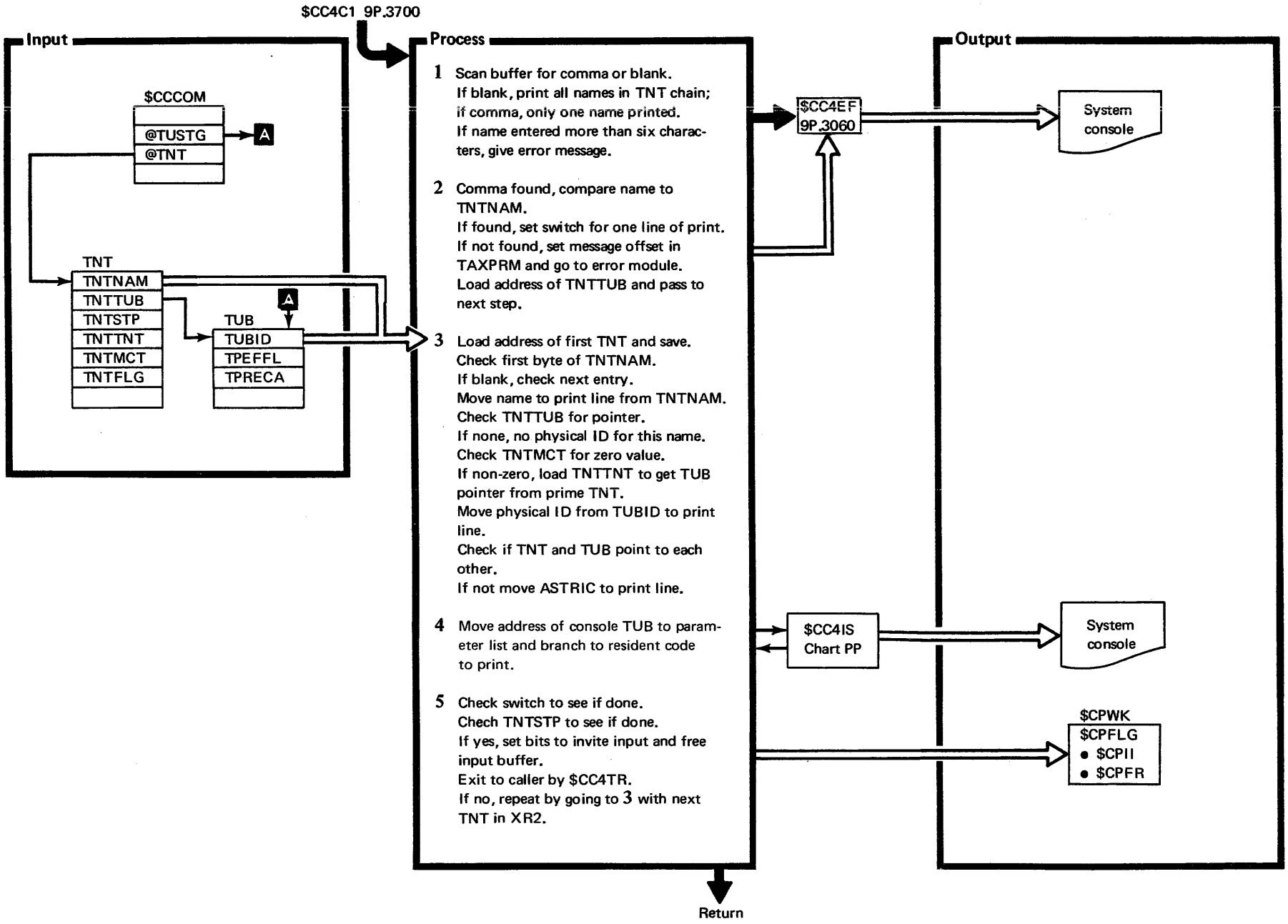


Diagram 9P.3720. $CC4C3

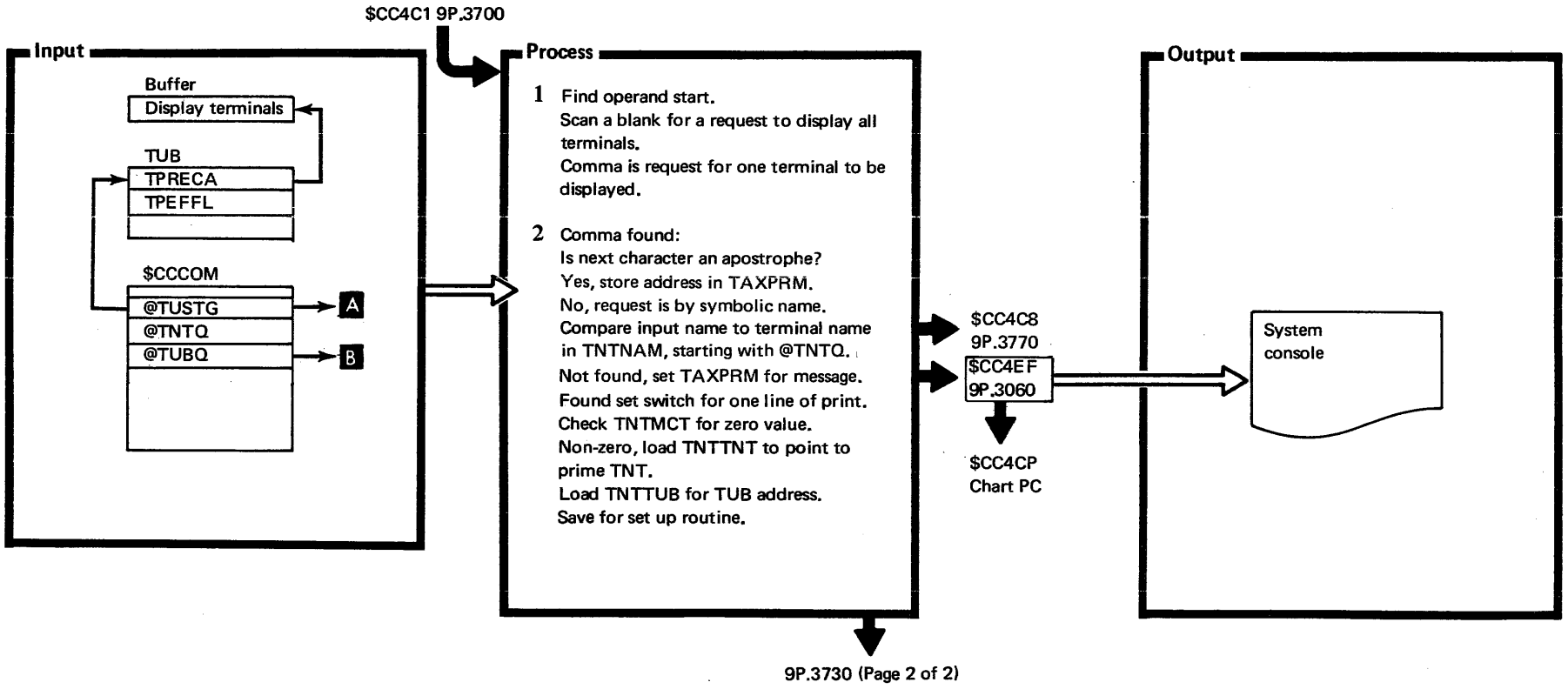


Diagram 9P.3730 (Part 1 of 2). $CC4C4

9P.3730 (Page 1 of 2)

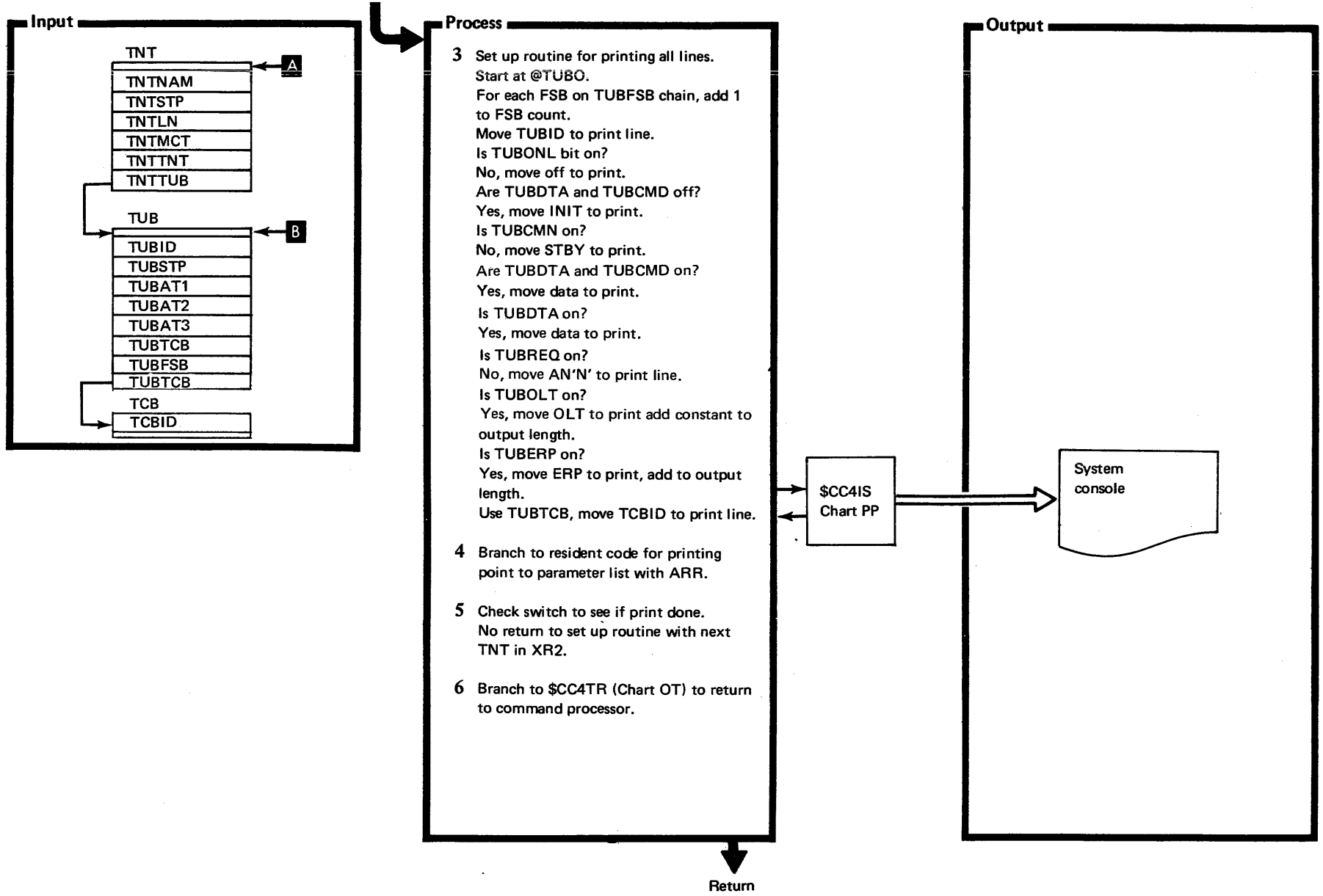


Diagram 9P.3730 (Part 2 of 2). $CC4C4

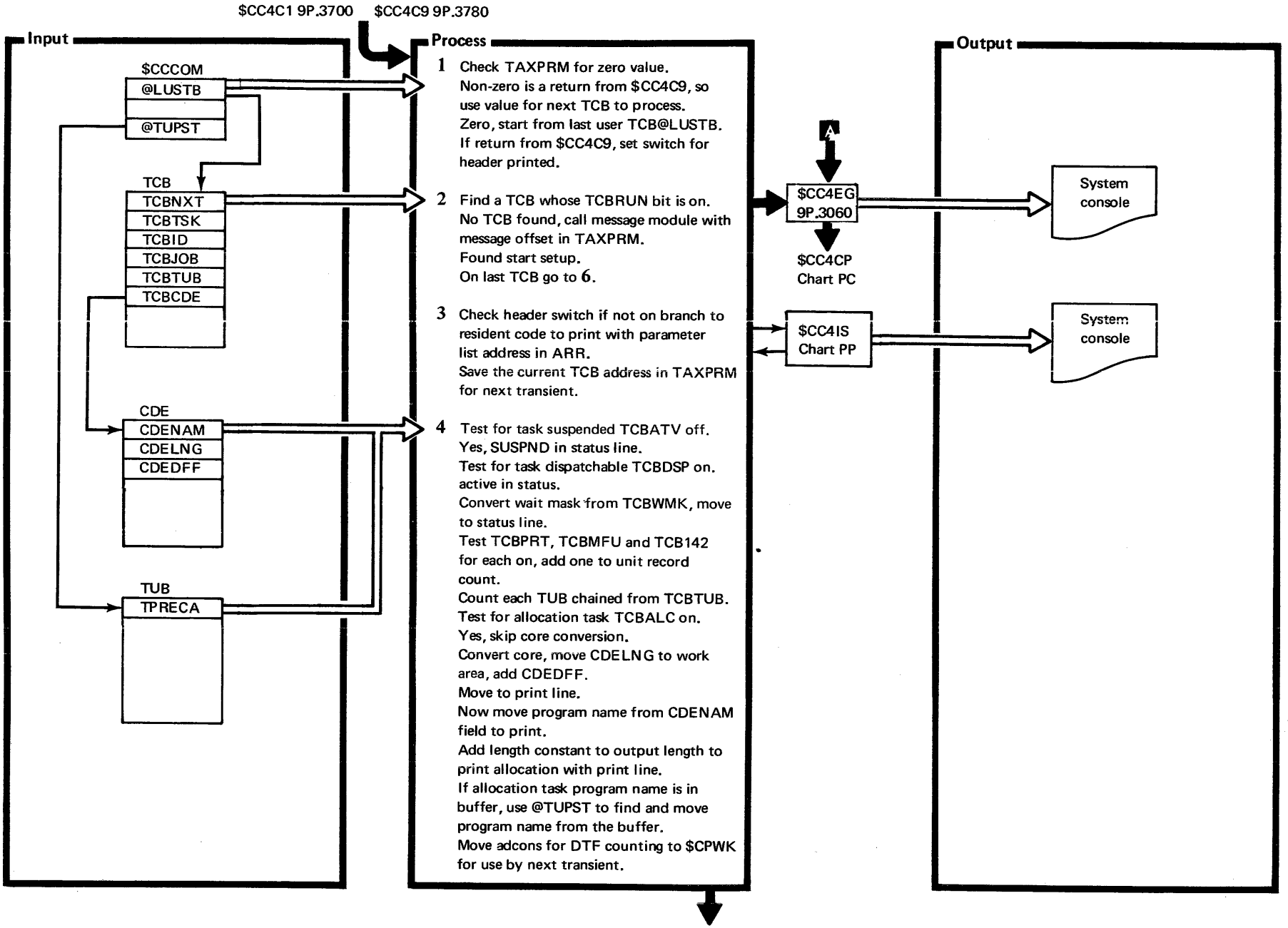


Diagram 9P.3740 (Part 1 of 2). $CC4C5

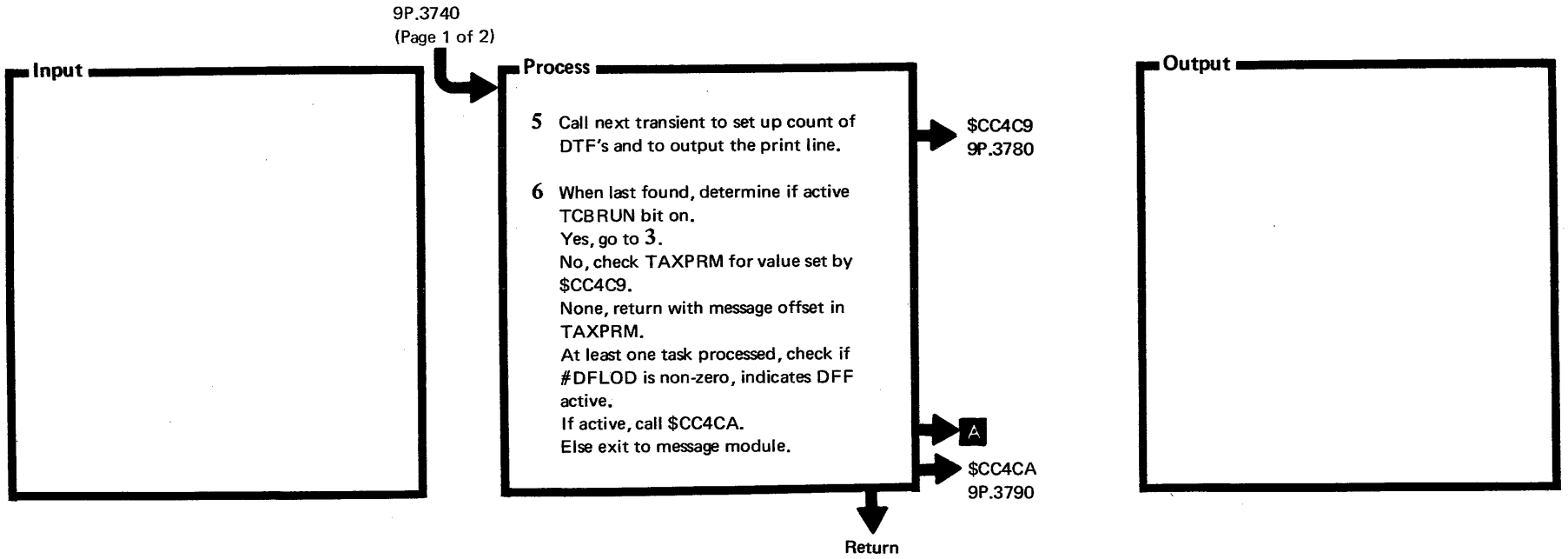


Diagram 9P.3740 (Part 2 of 2). $CC4C5

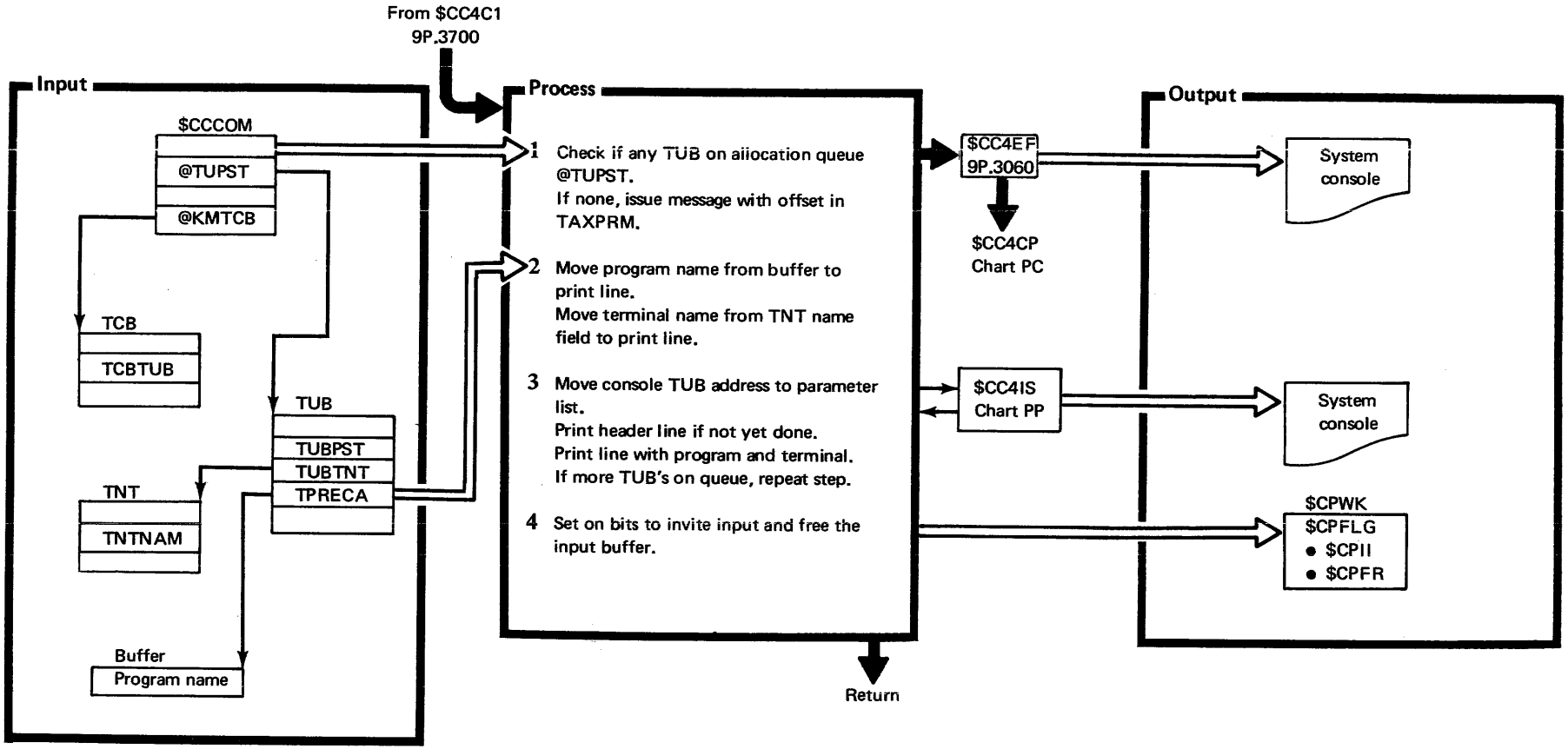


Diagram 9P.3750. $CC4C6

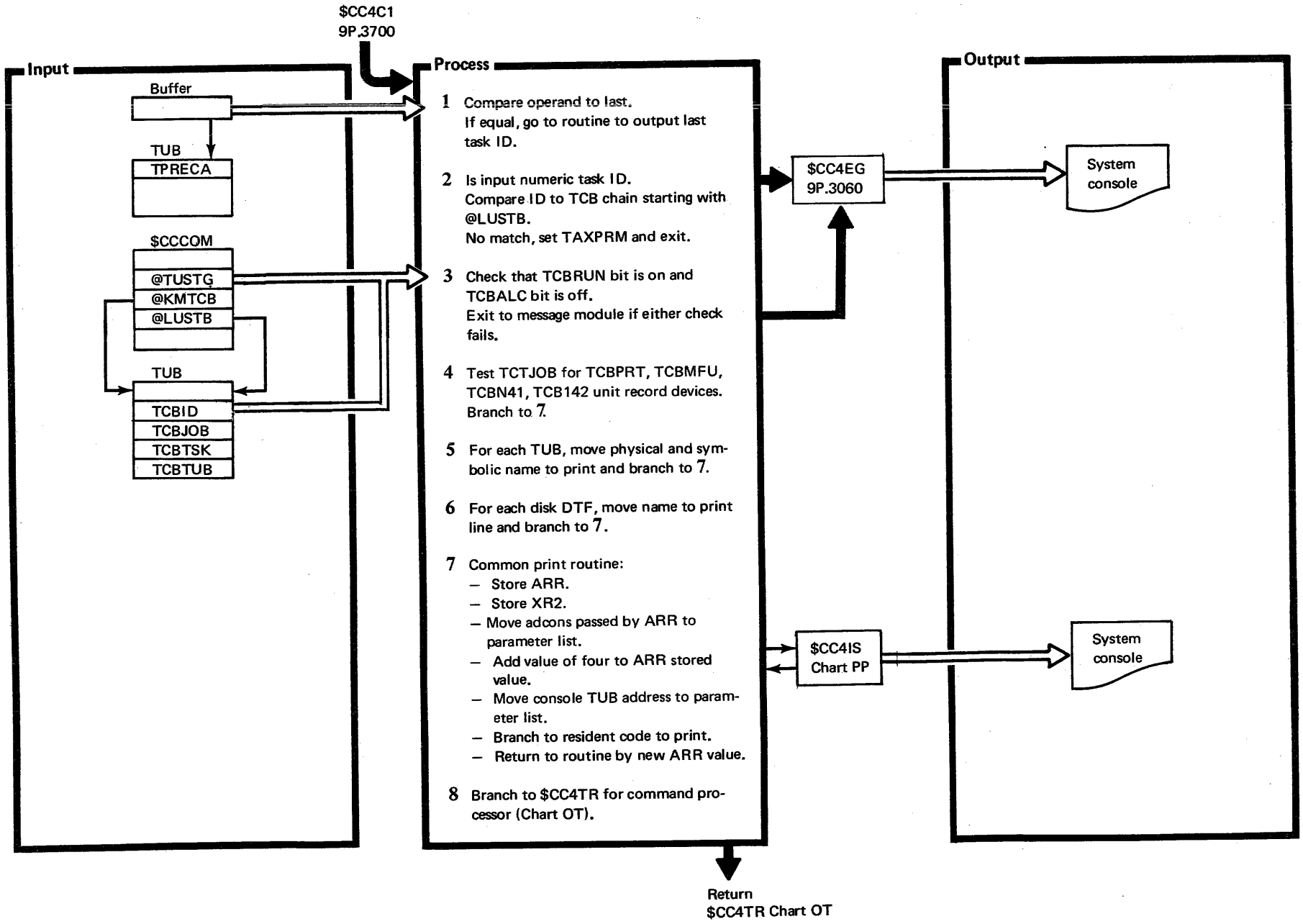


Diagram 9P.3760. $CC4C7

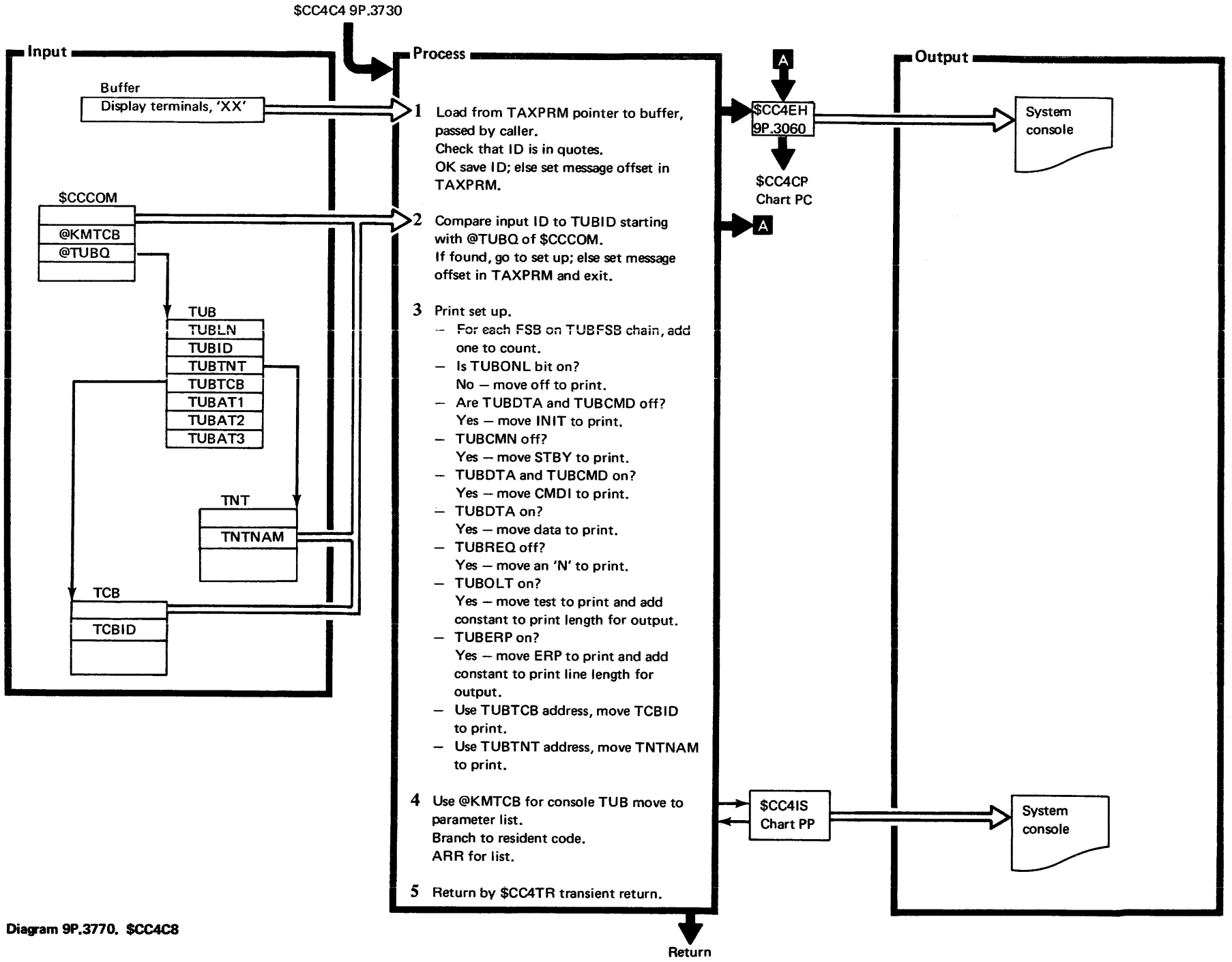


Diagram 9P.3770. $CC4C8

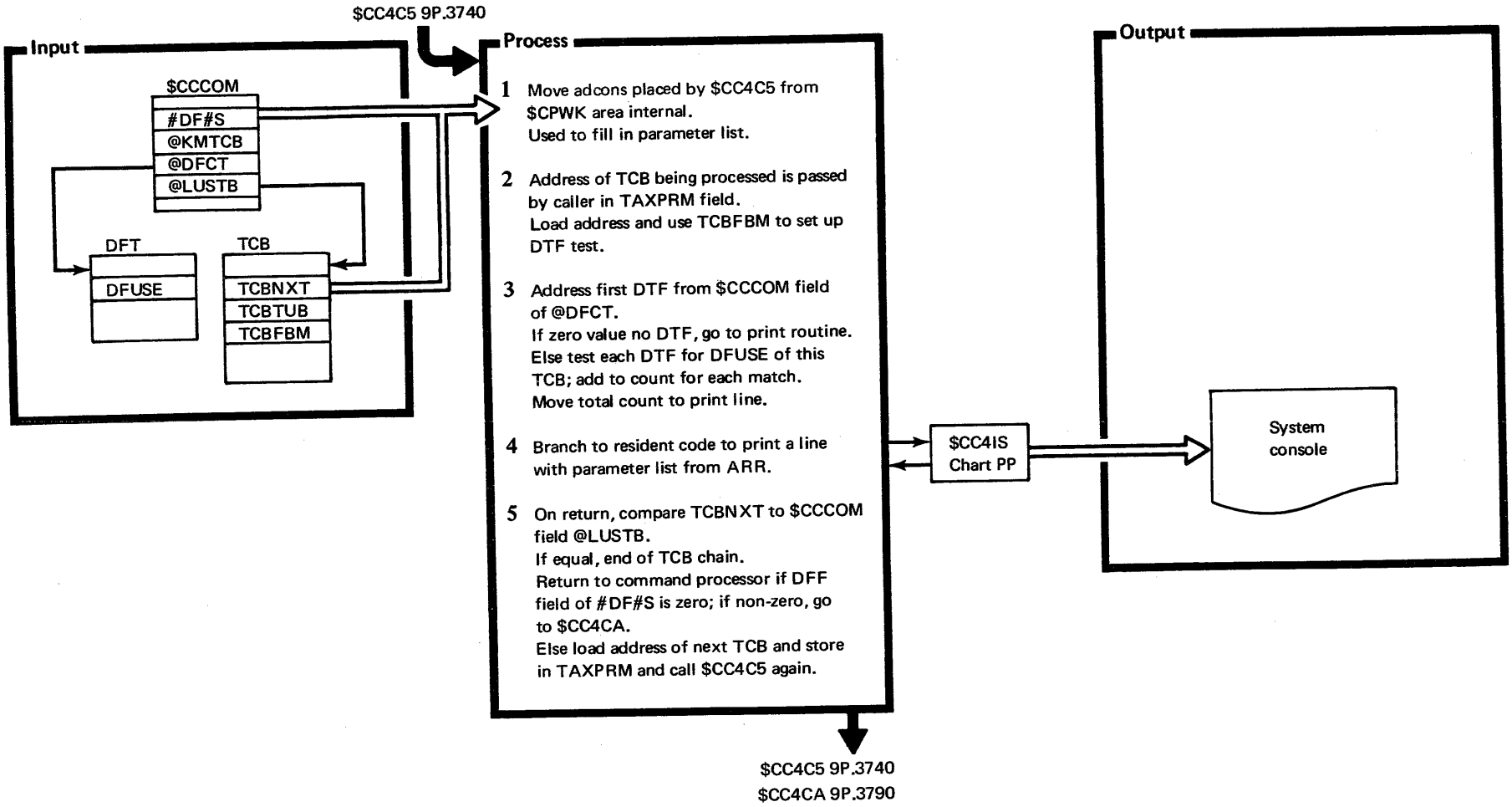


Diagram 9P.3780. $CC4C9

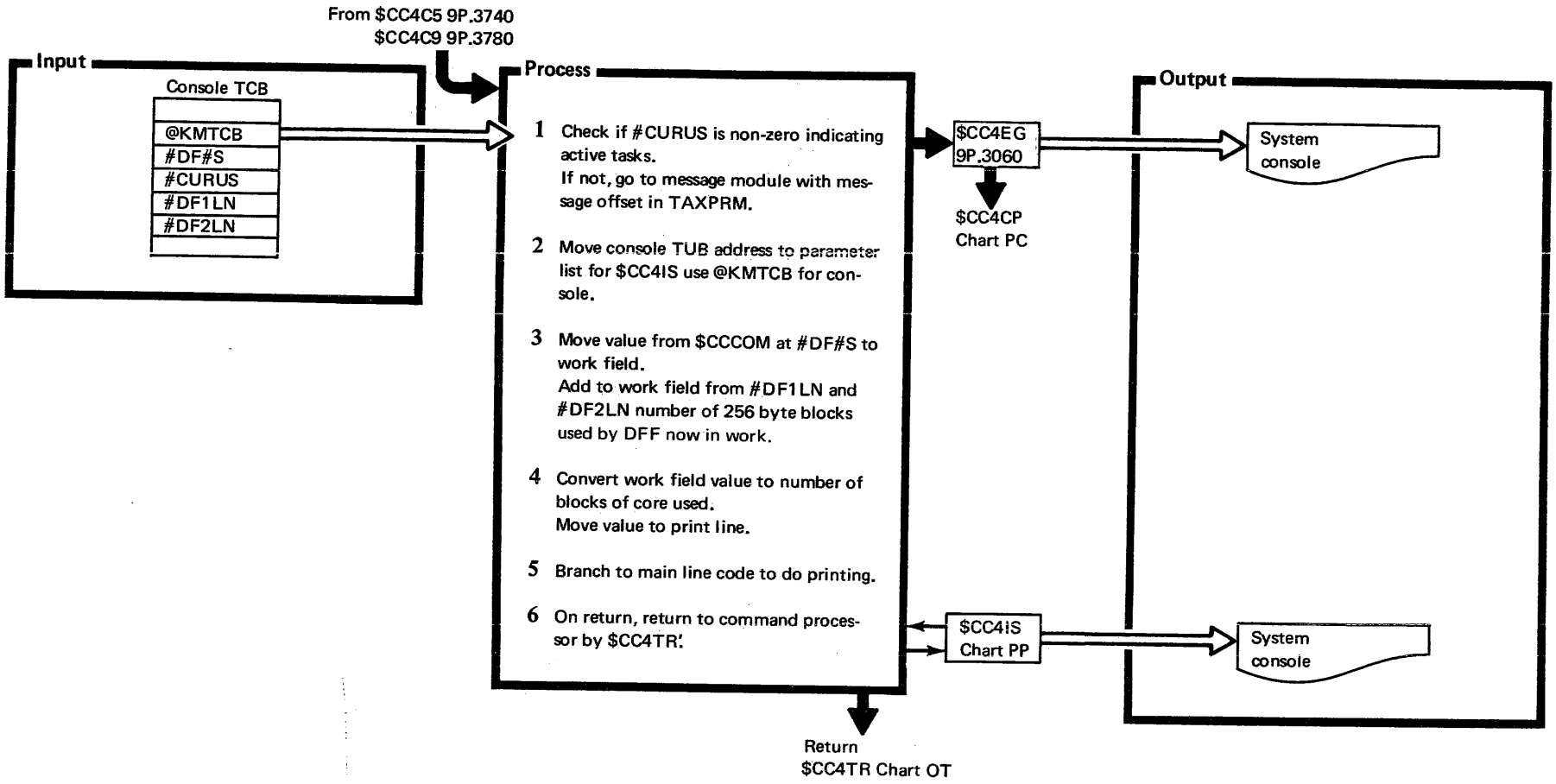


Diagram 9P.3790. $CC4CA

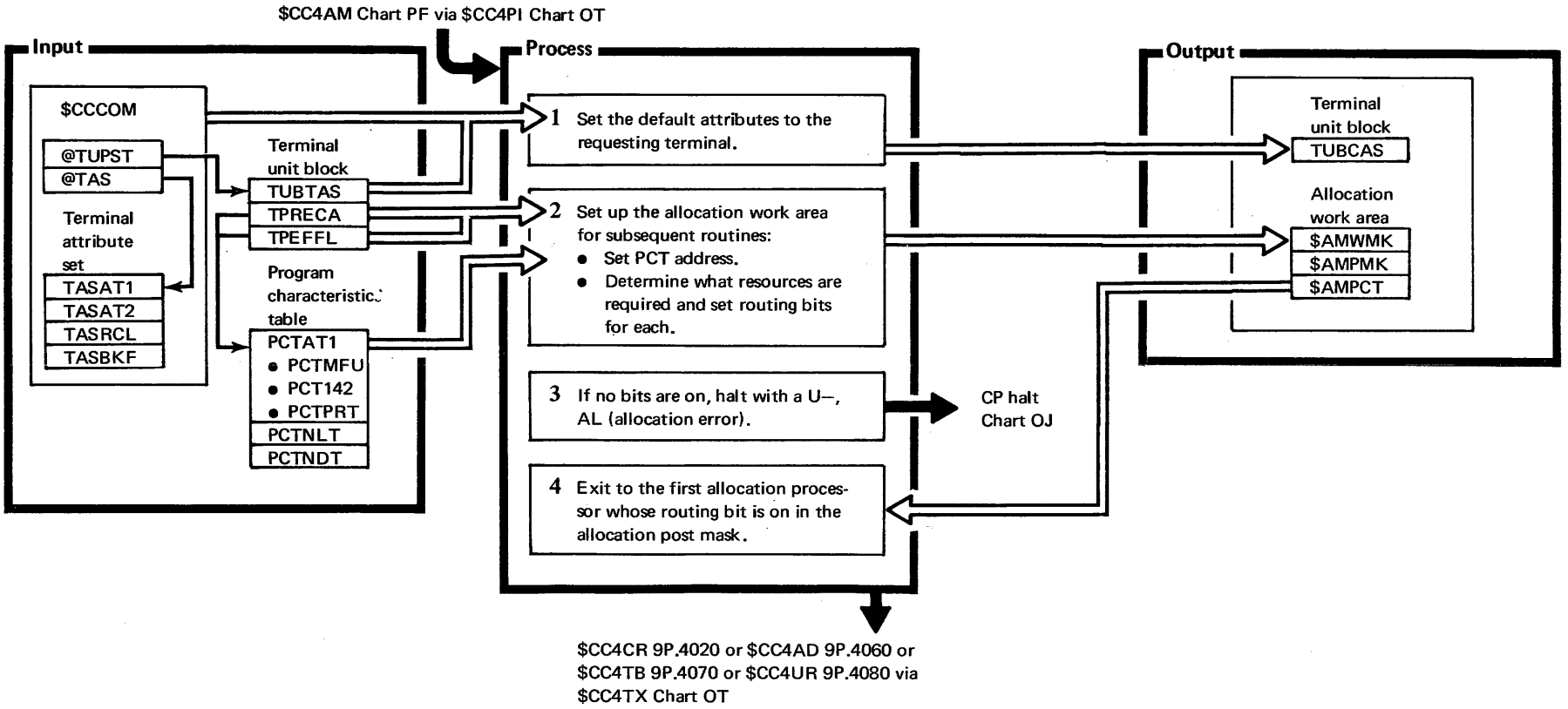
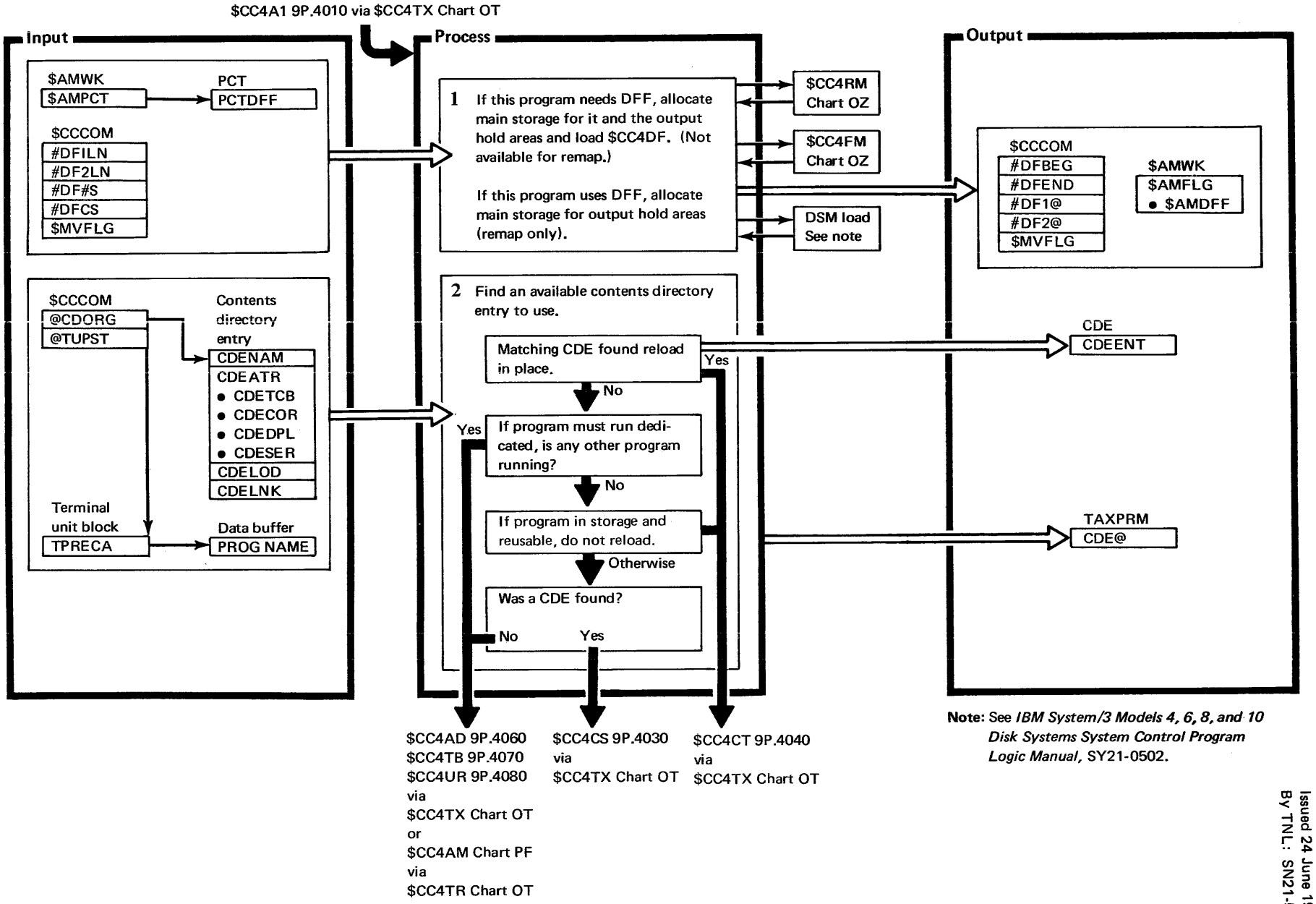


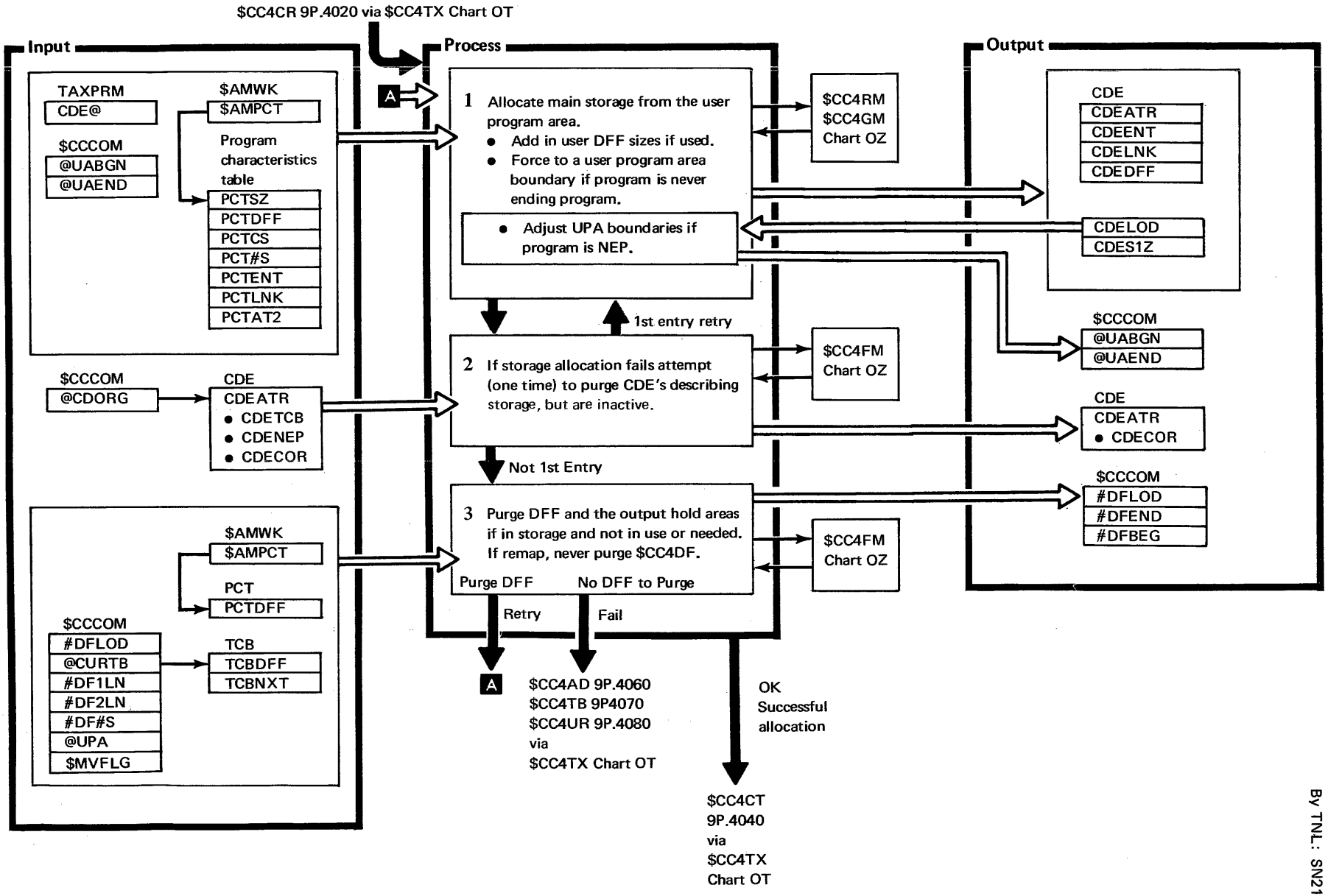
Diagram 9P.4010. $CC4A1



Note: See *IBM System/3 Models 4, 6, 8, and 10 Disk Systems System Control Program Logic Manual*, SY21-0502.

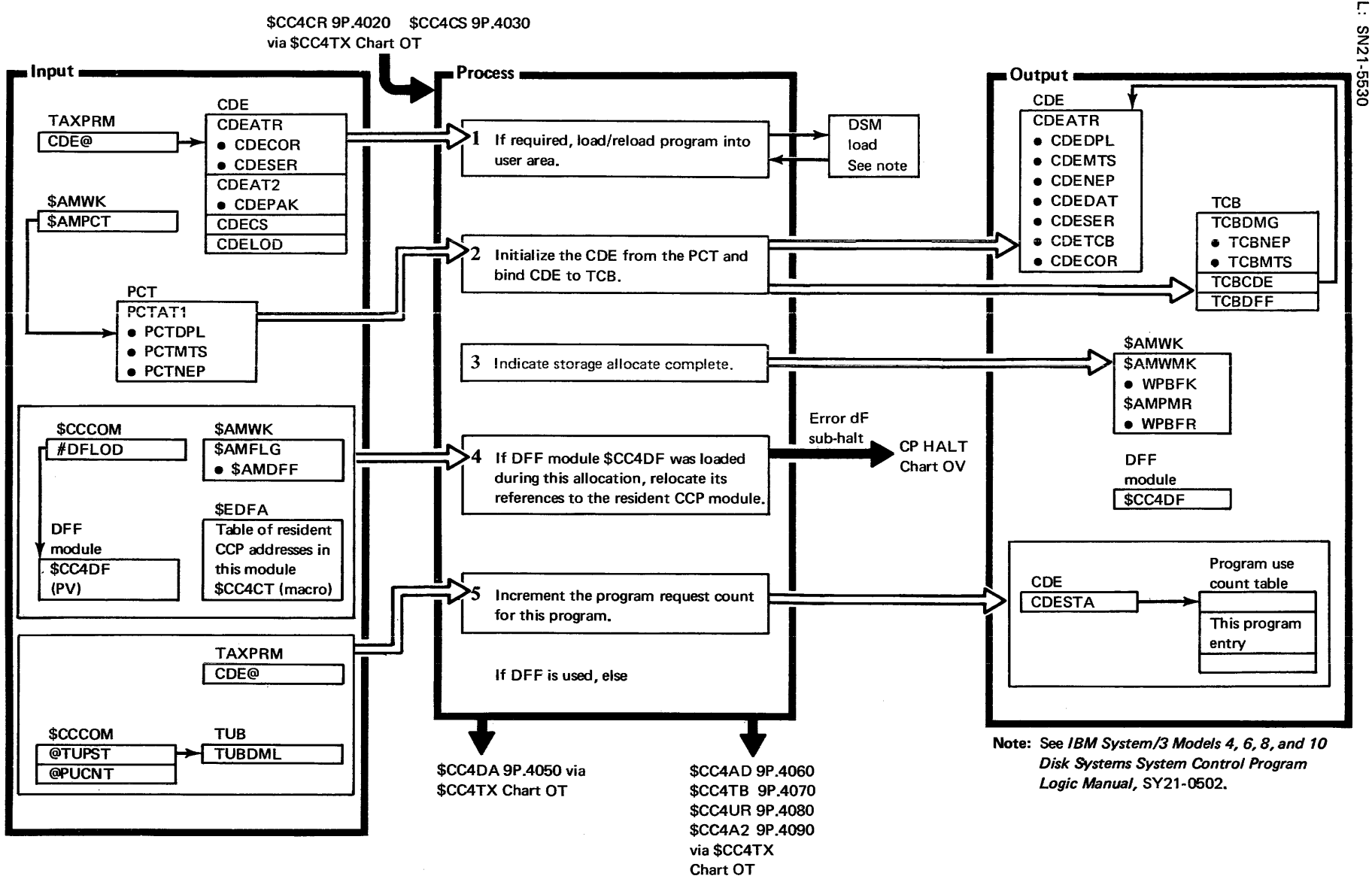
Page of SY21-0531-2
Issued 24 June 1977
By TNL: SN21-5530

• **Diagram 9P.4020. $CC4CR**



● **Diagram 9P.4030. $CC4CS**

Page of SY21-0531-2
Issued 24 June 1977
By TNL: SN21-5530



Note: See *IBM System/3 Models 4, 6, 8, and 10 Disk Systems System Control Program Logic Manual*, SY21-0502.

Diagram 9P.4040. $CC4CT

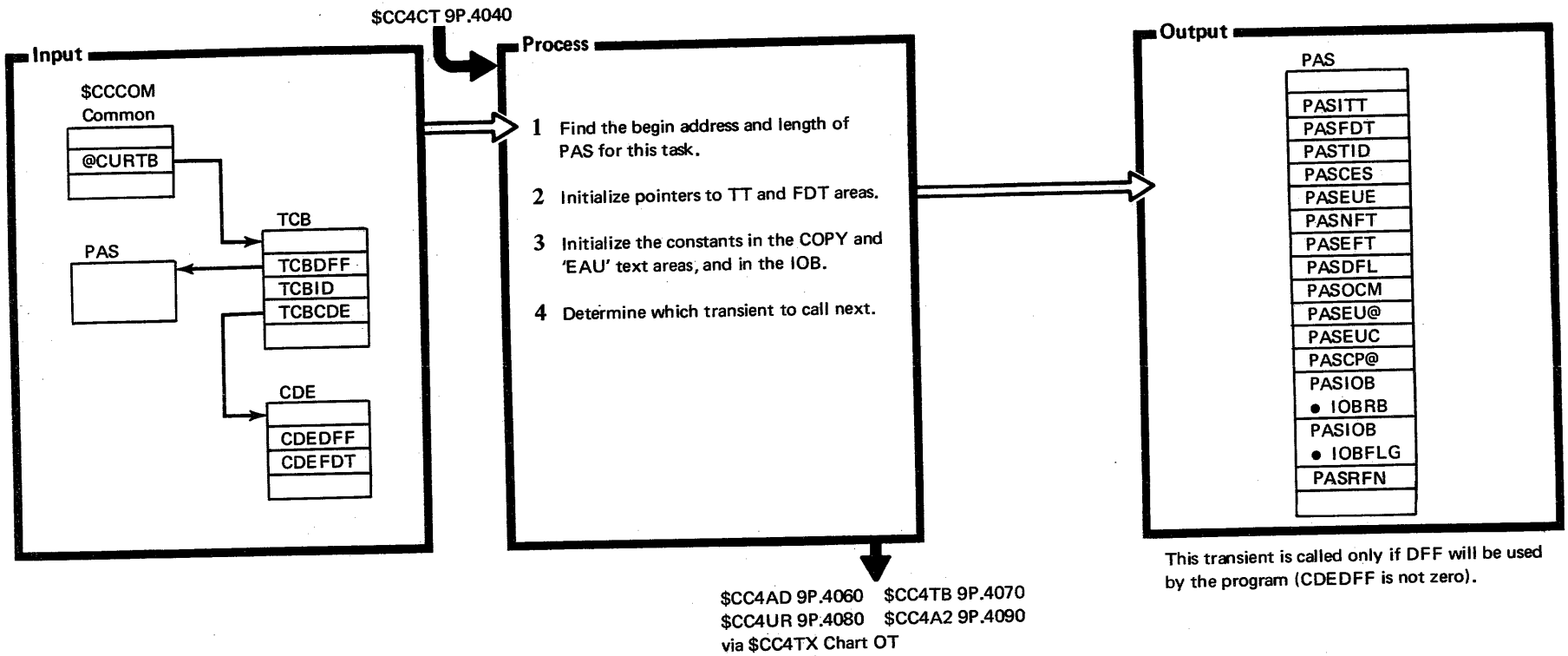


Diagram 9P.4050. $CC4DA

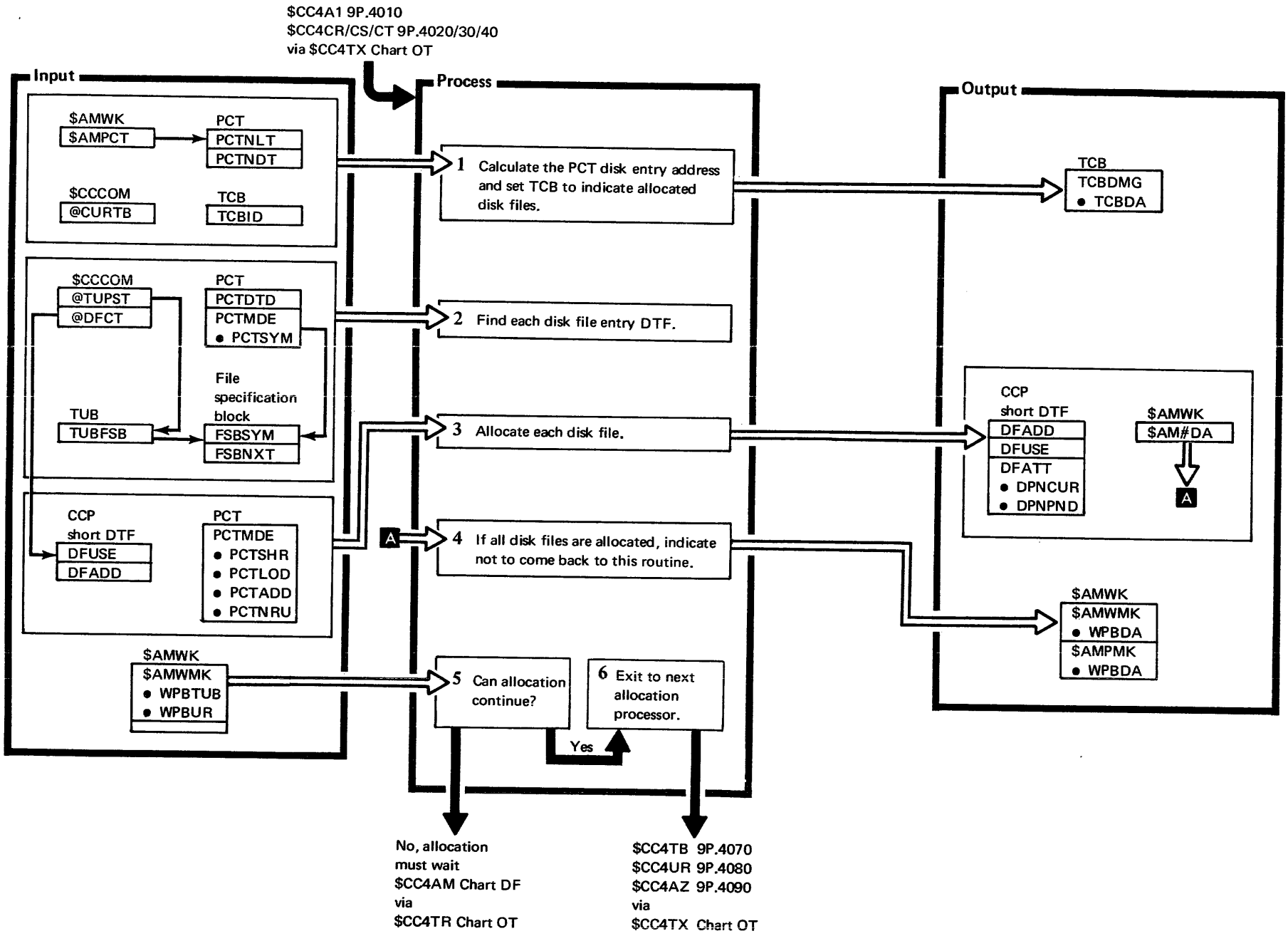


Diagram 9P.4060. $CC4AD

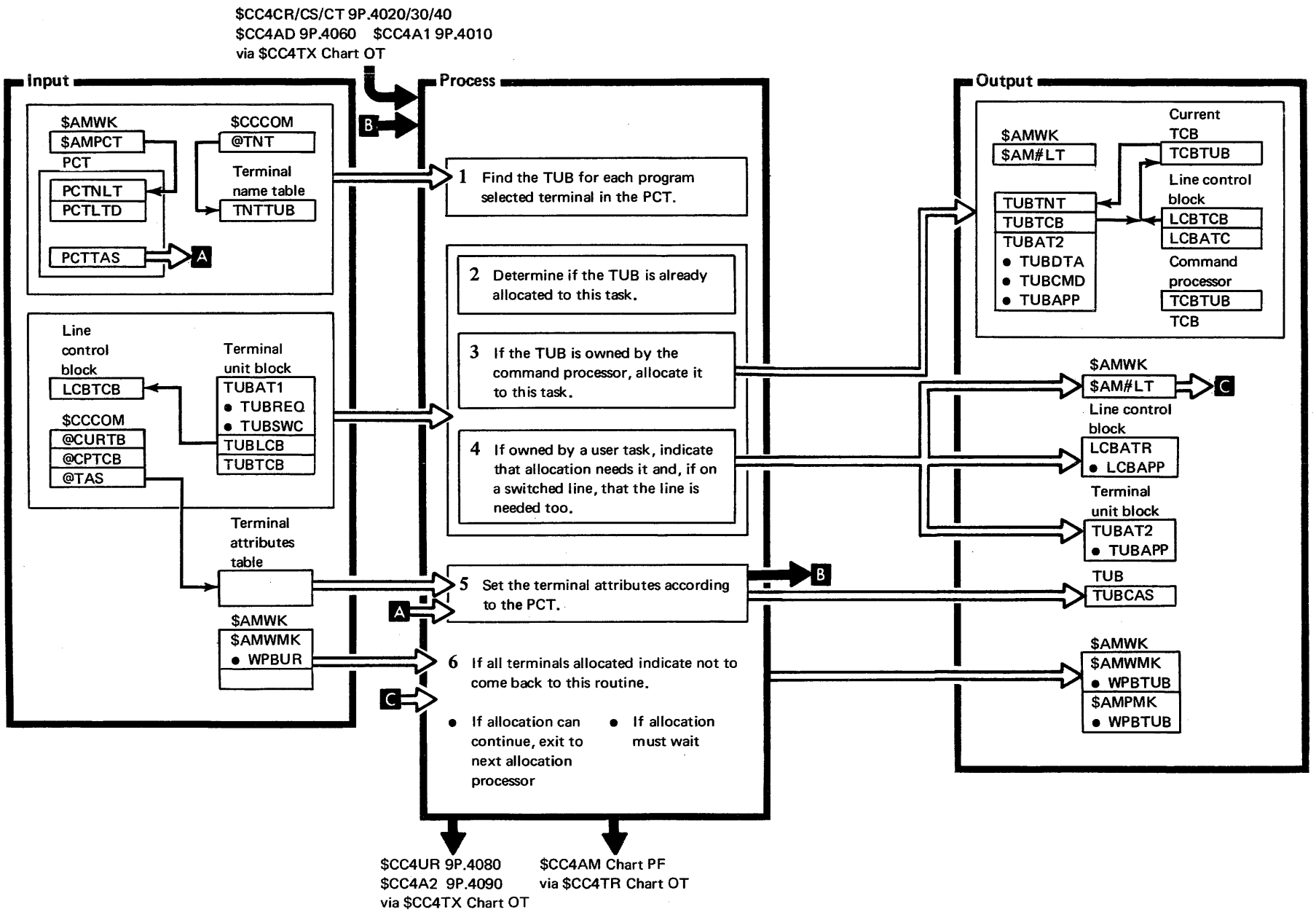


Diagram 9P.4070. $CC4TB

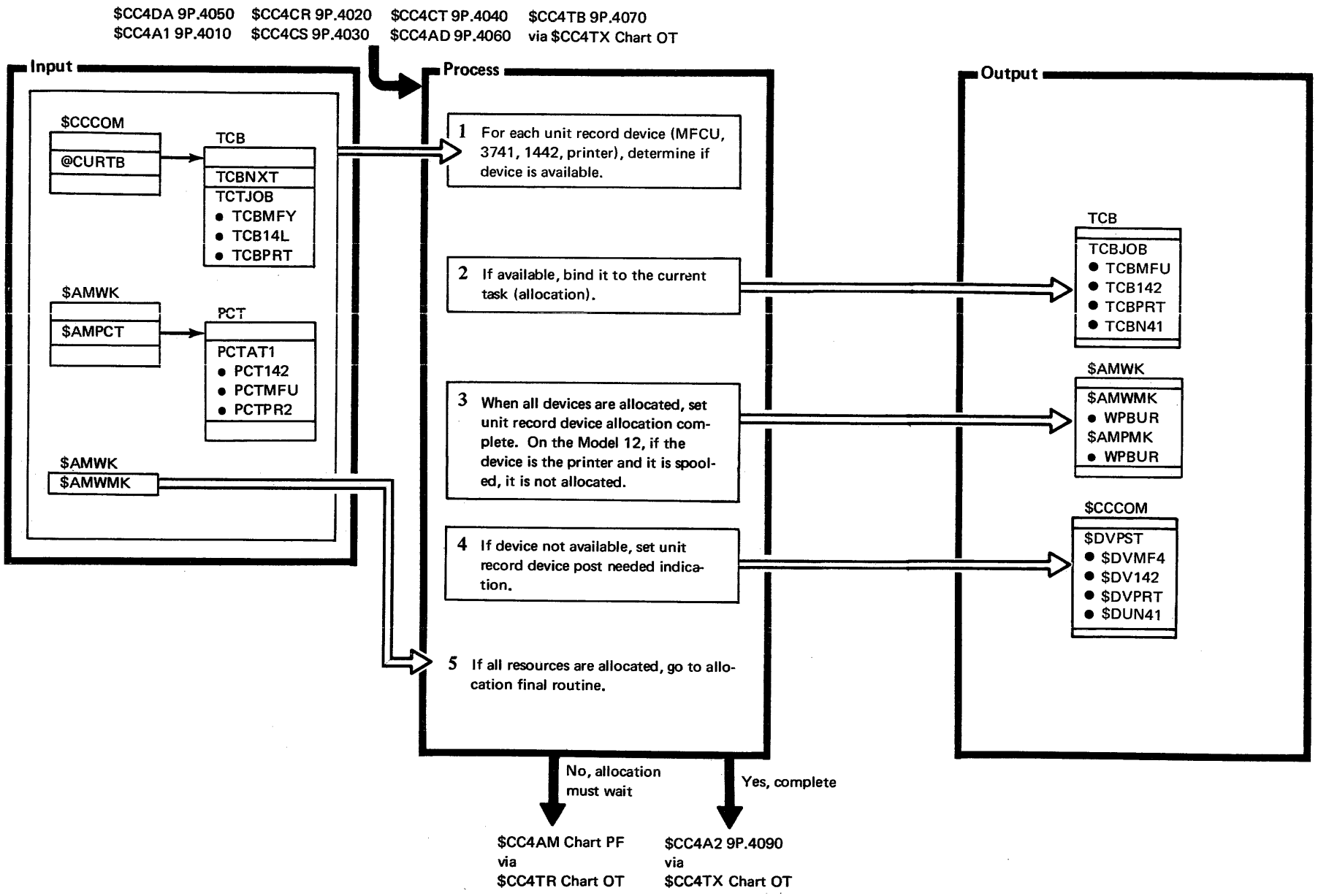


Diagram 9P.4080. $CC4UR

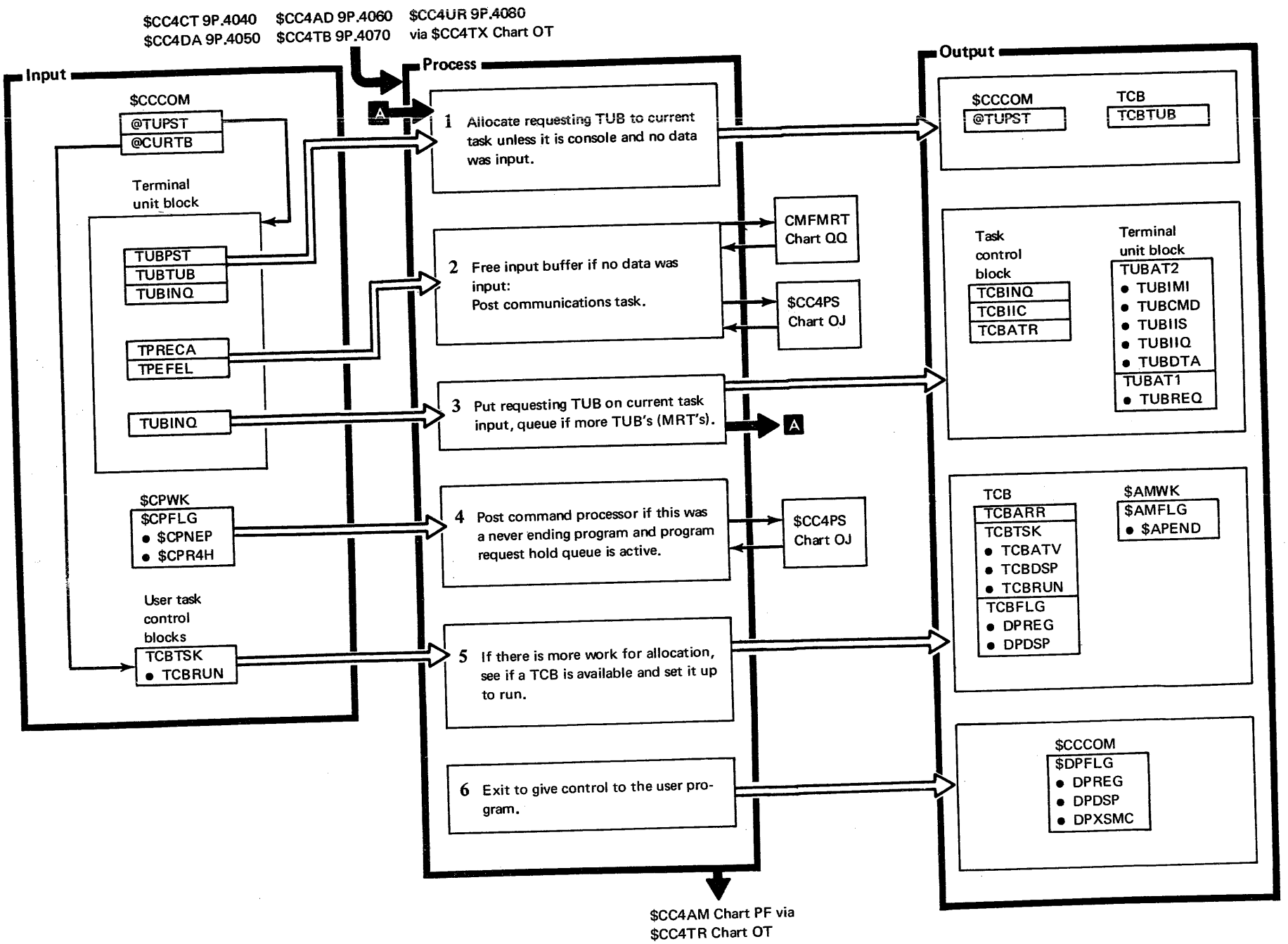


Diagram 9P.4090. $CC4A2

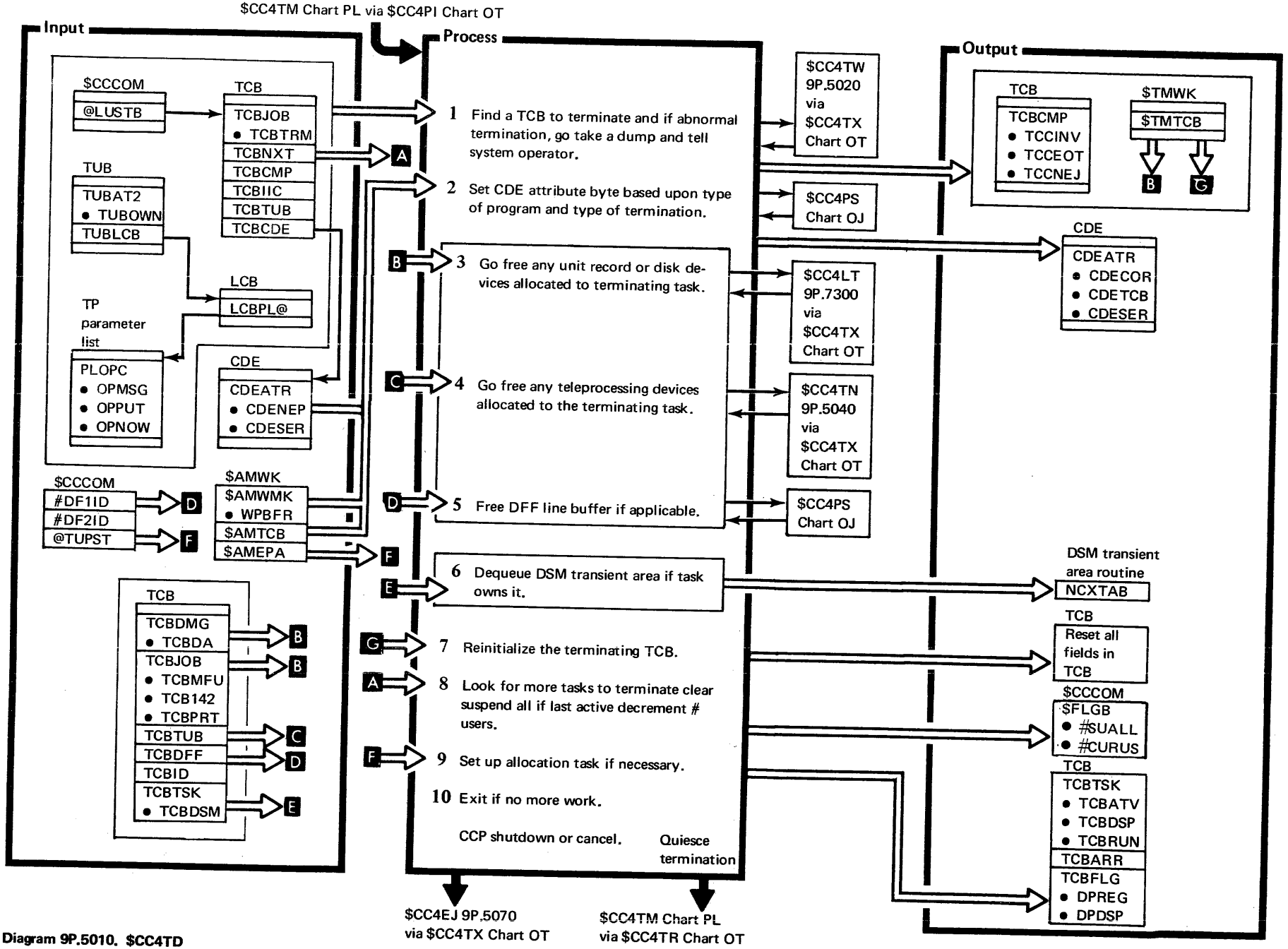


Diagram 9P.5010. $CC4TD

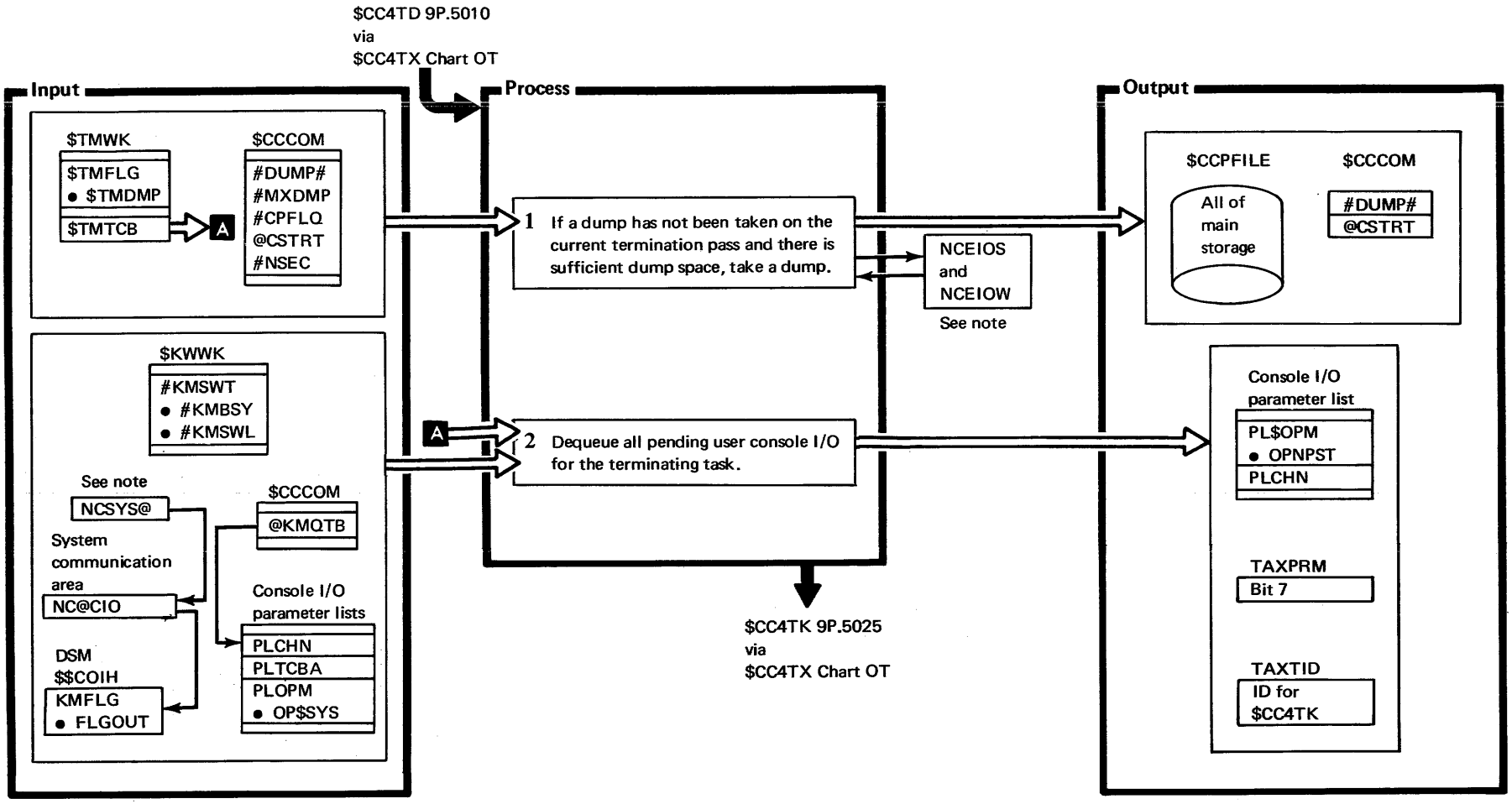
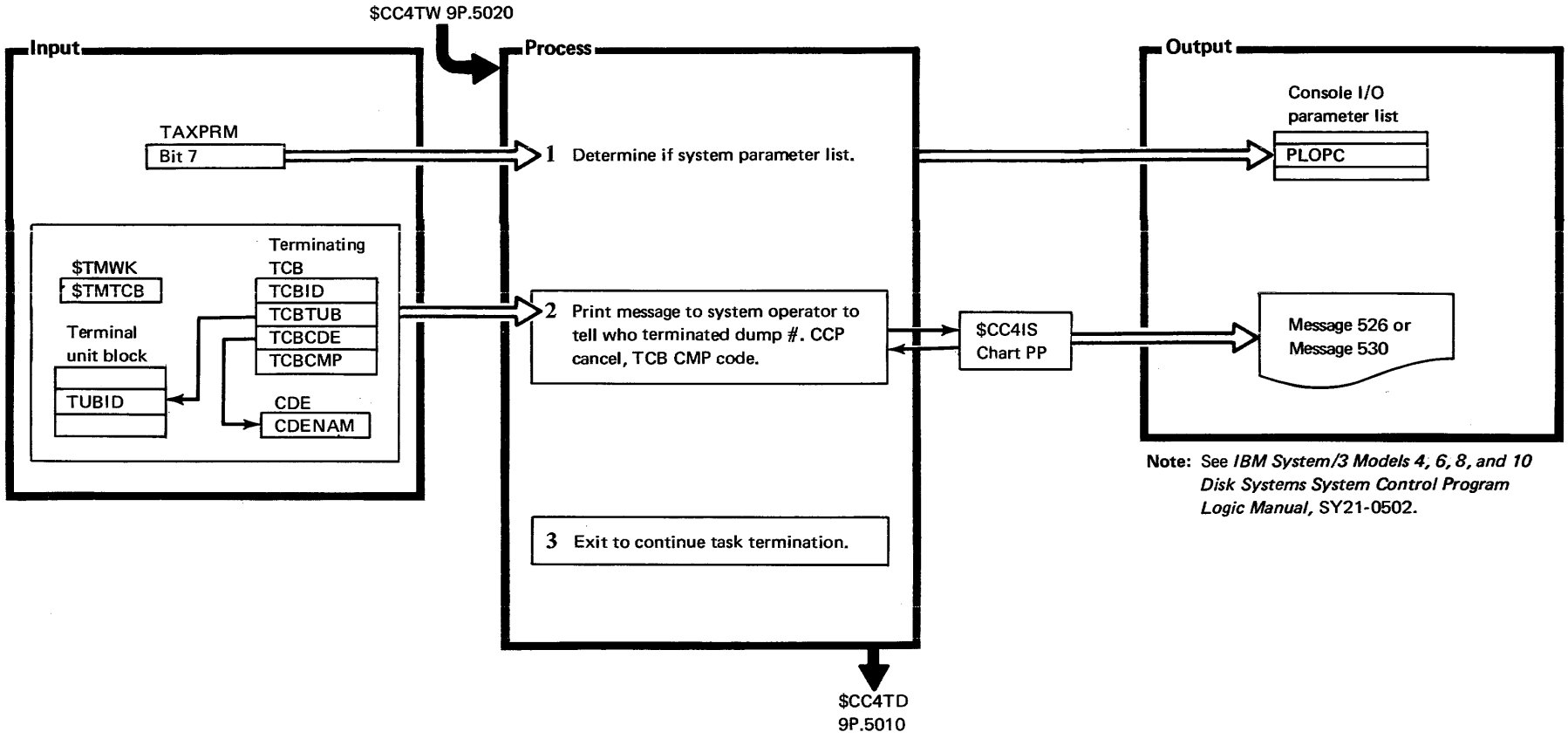


Diagram 9P.5020. $CC4TW

Page of SY21-0531-2
Issued 24 June 1977
By TNL: SN21-5530

Page of SY21-0531-2
Issued 24 June 1977
By TNL: SN21-5530



Note: See *IBM System/3 Models 4, 6, 8, and 10 Disk Systems System Control Program Logic Manual*, SY21-0502.

● **Diagram 9P.5025. $CC4TK**

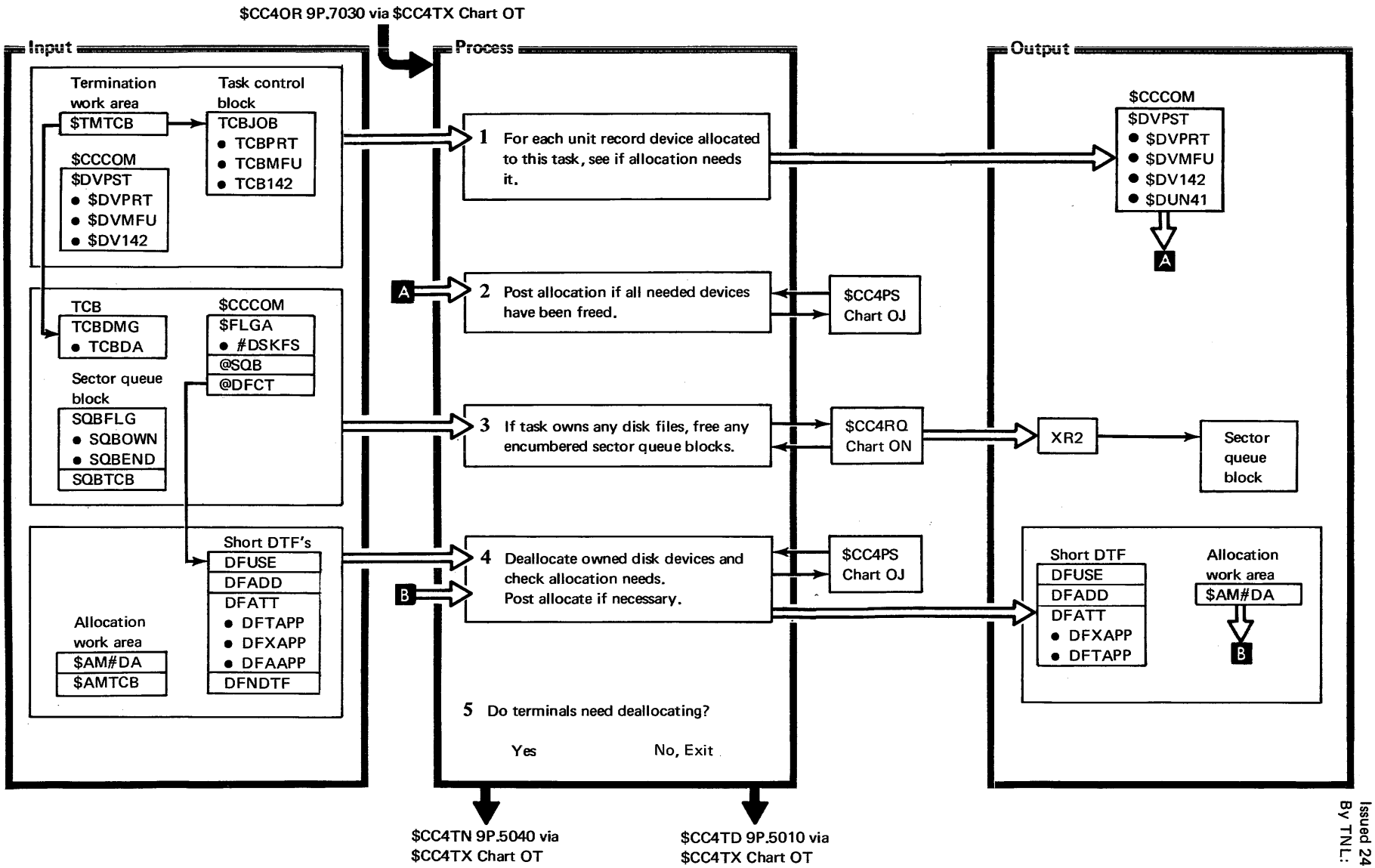


Diagram 9P.5030. $CC4TF

Page of SY21-0531-2
Issued 24 June 1977
By TNL: SN21-5530

This page intentionally left blank.

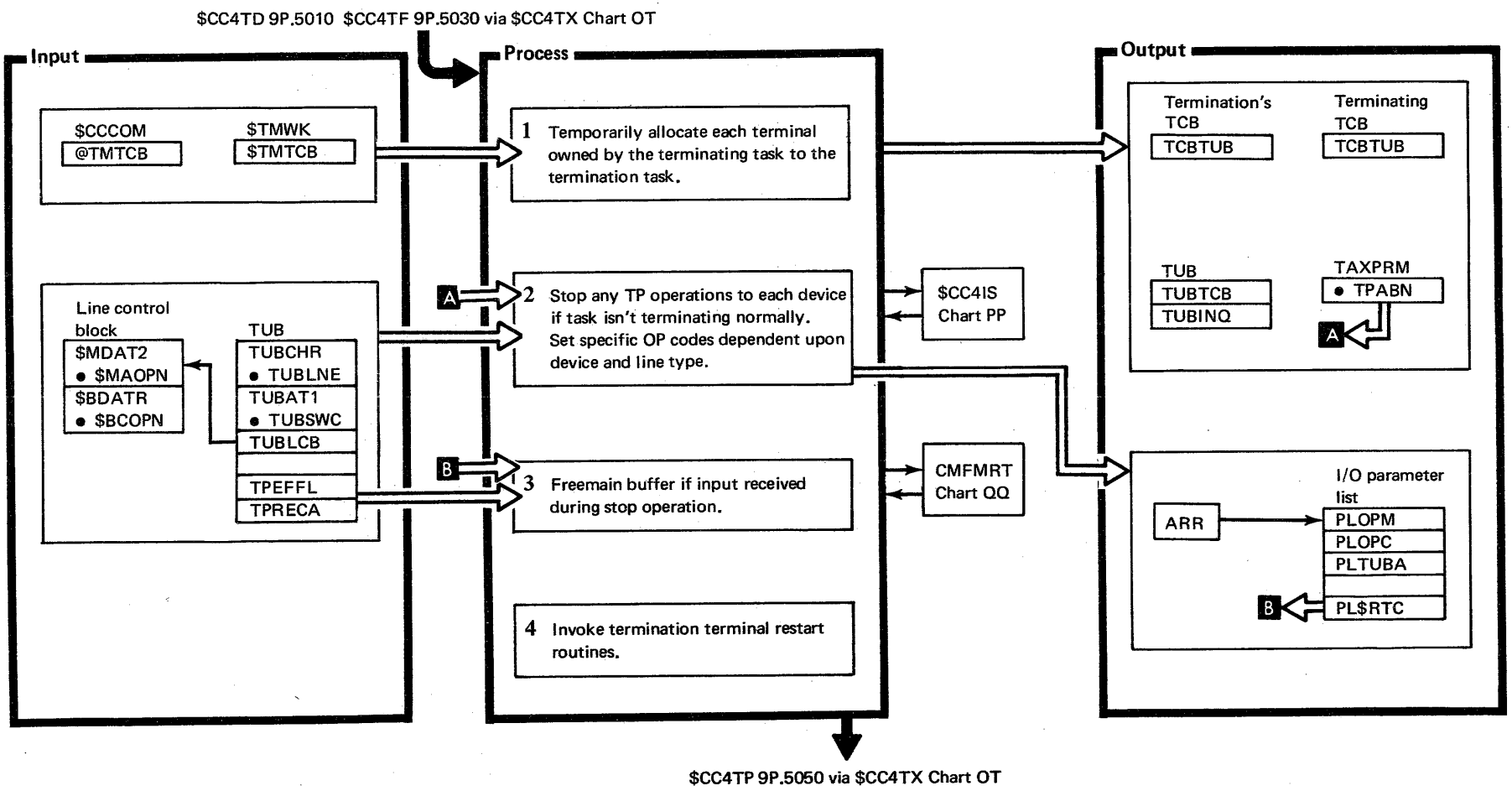


Diagram 9P.5040. $CC4TN

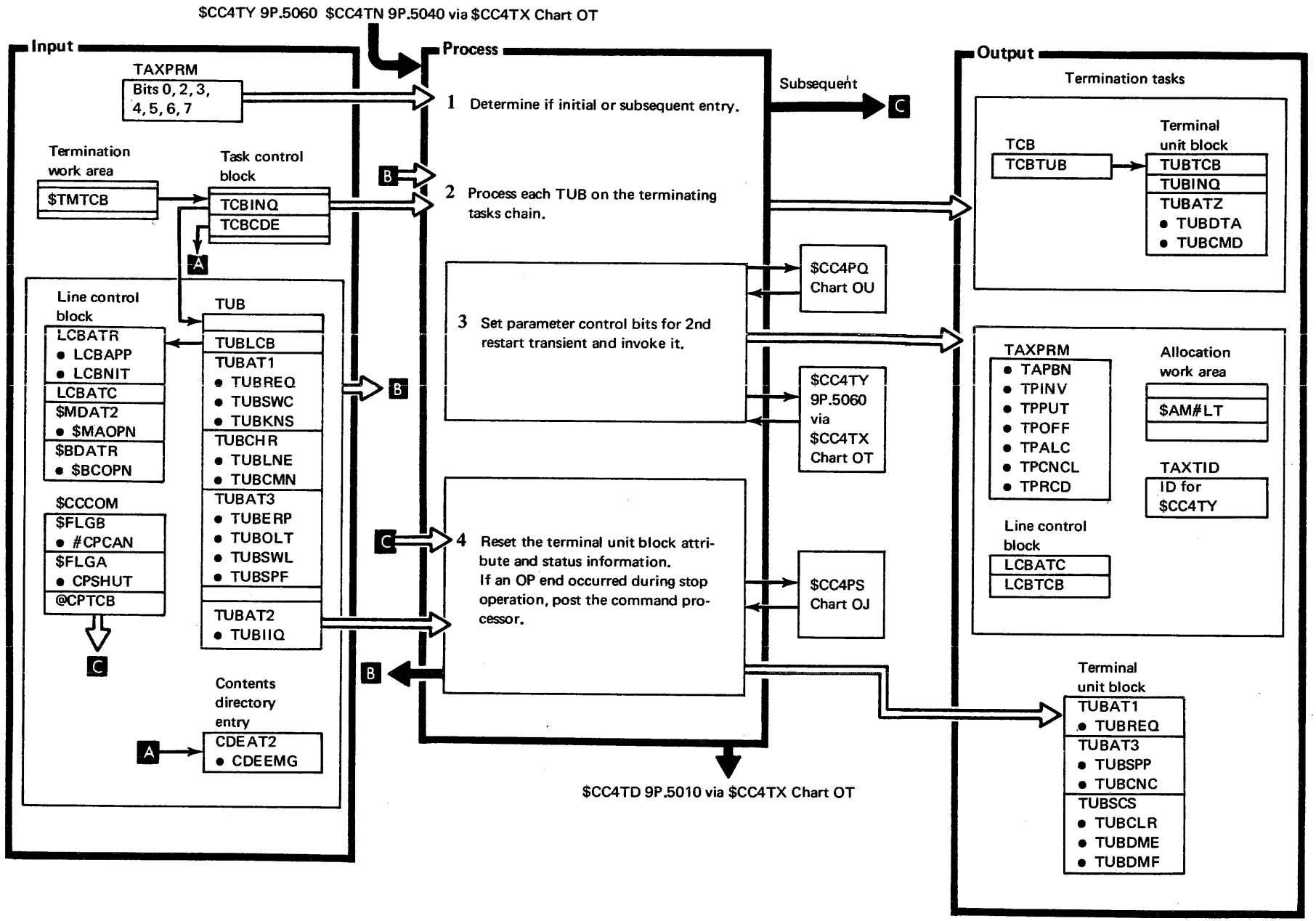


Diagram 9P.5050. $CC4TP

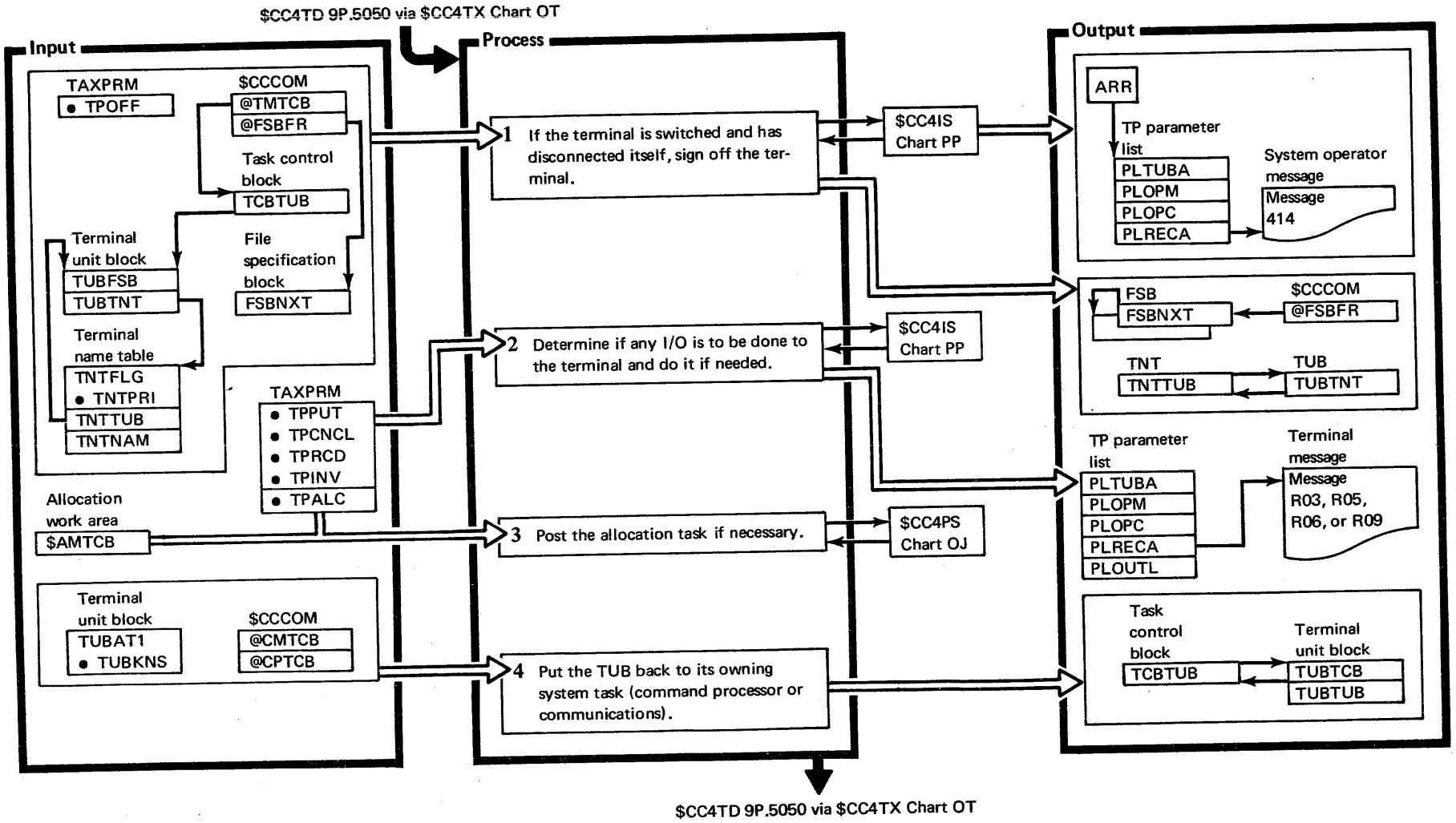
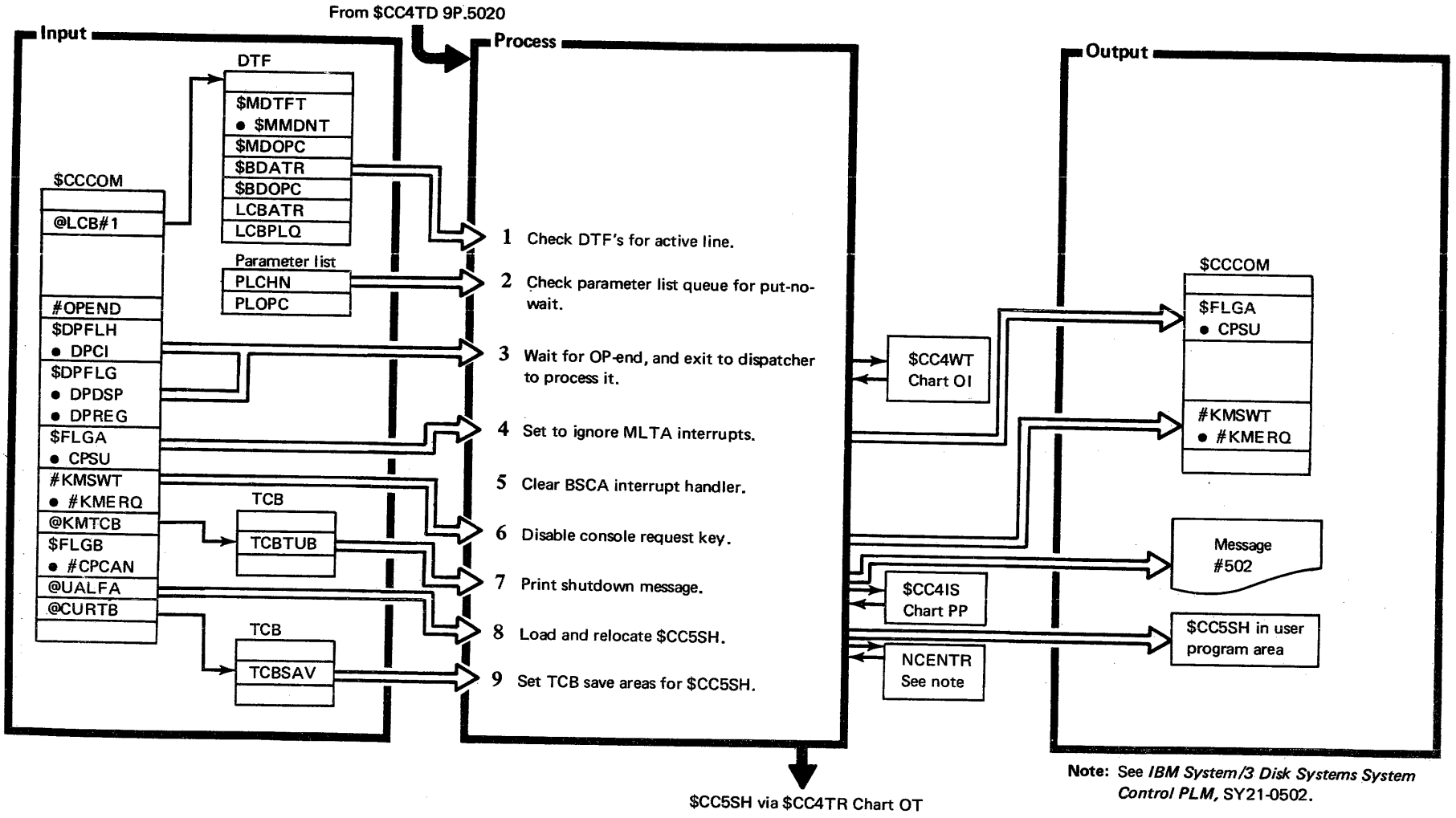


Diagram 9P.5060. $CC4TY



Note: See *IBM System/3 Disk Systems System Control PLM,* SY21-0502.

Diagram 9P.5070. $CC4EJ

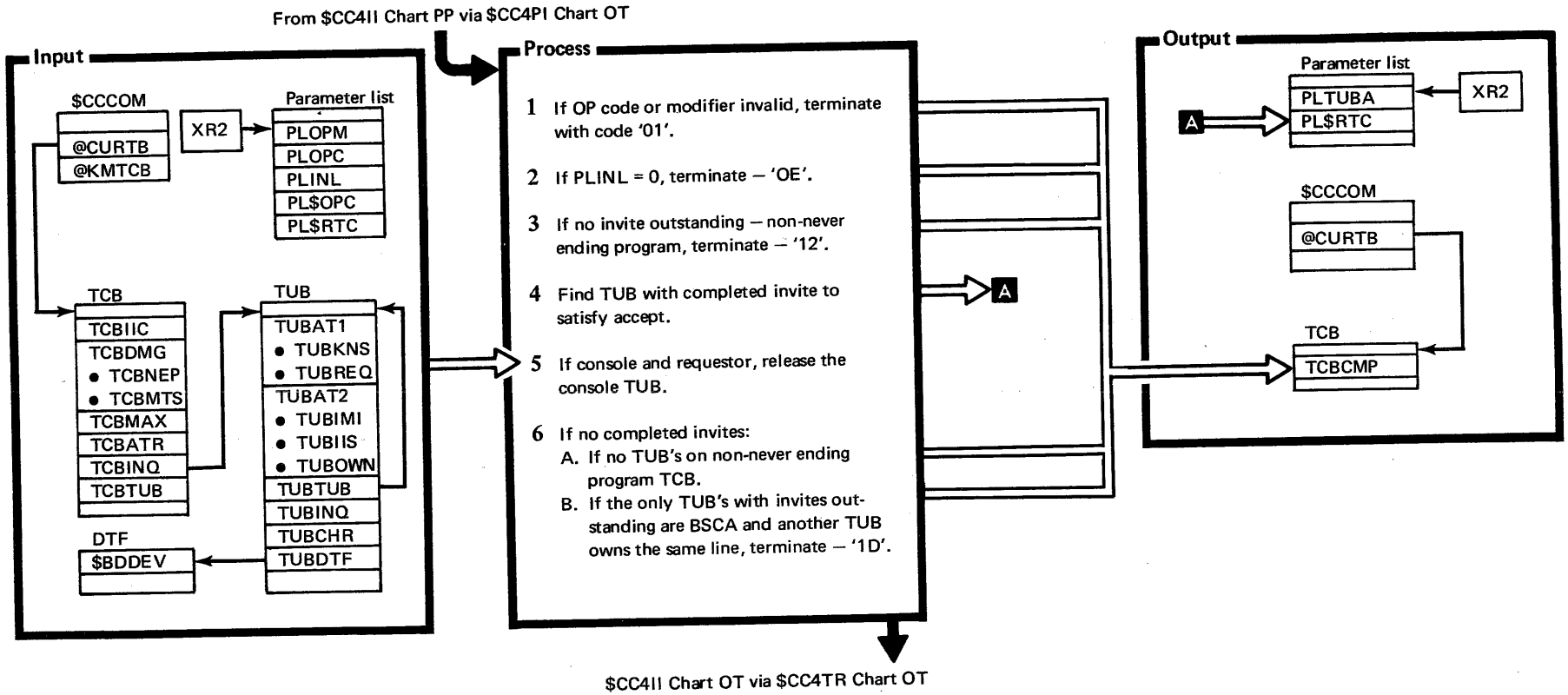


Diagram 9P.6010. $CC4AB

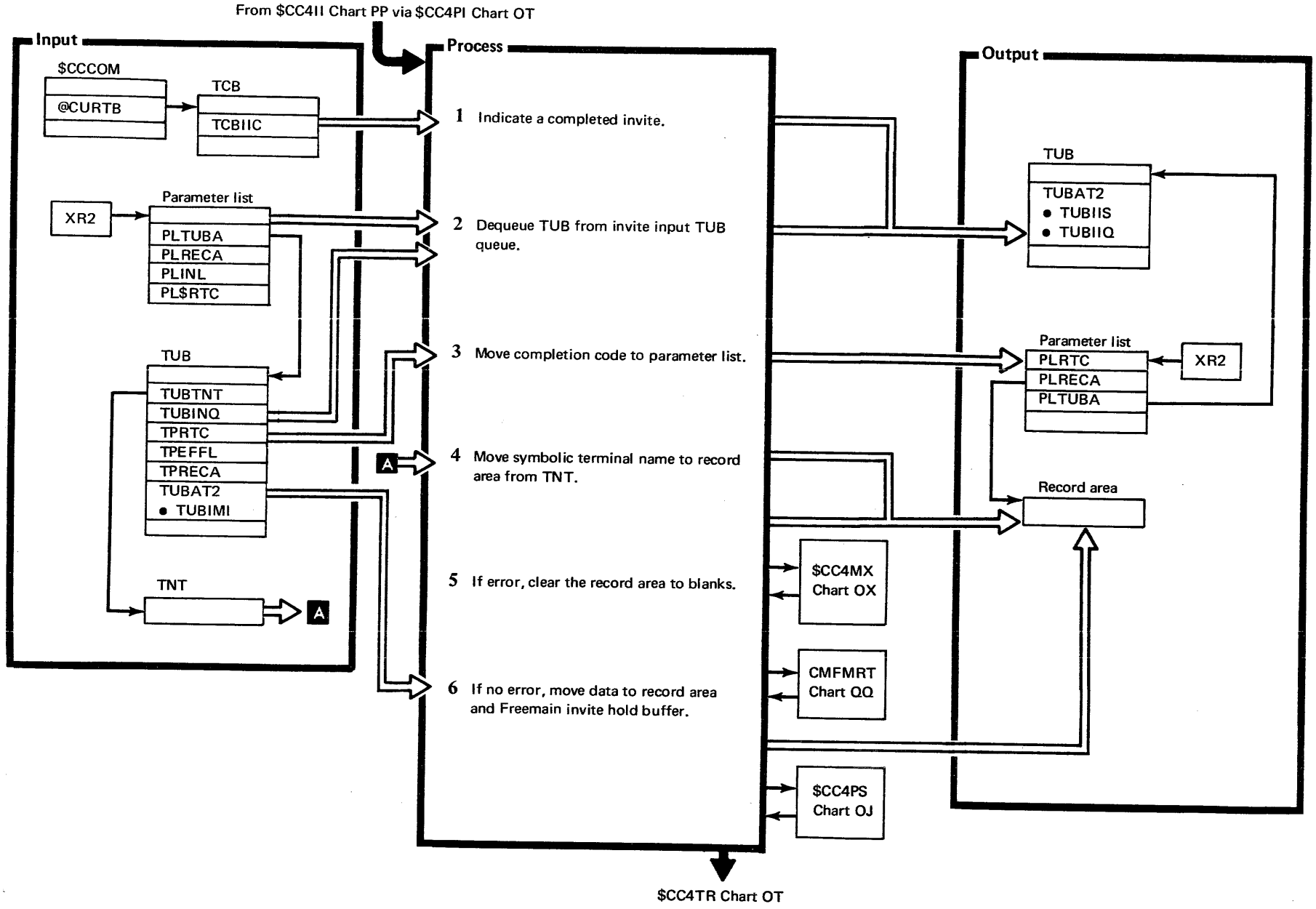


Diagram 9P.6020. $CC4AC

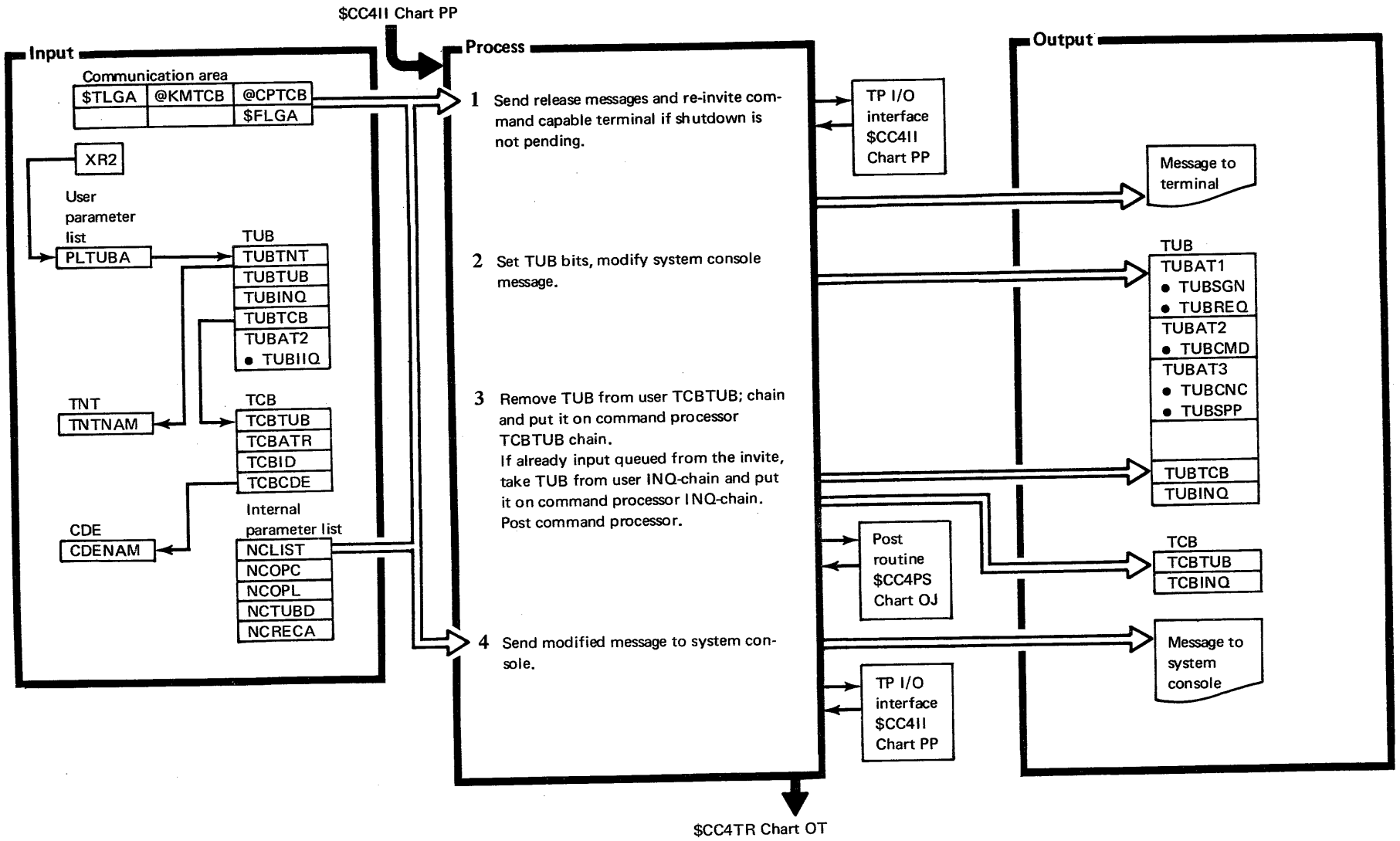


Diagram 9P.6030. $CC4NC

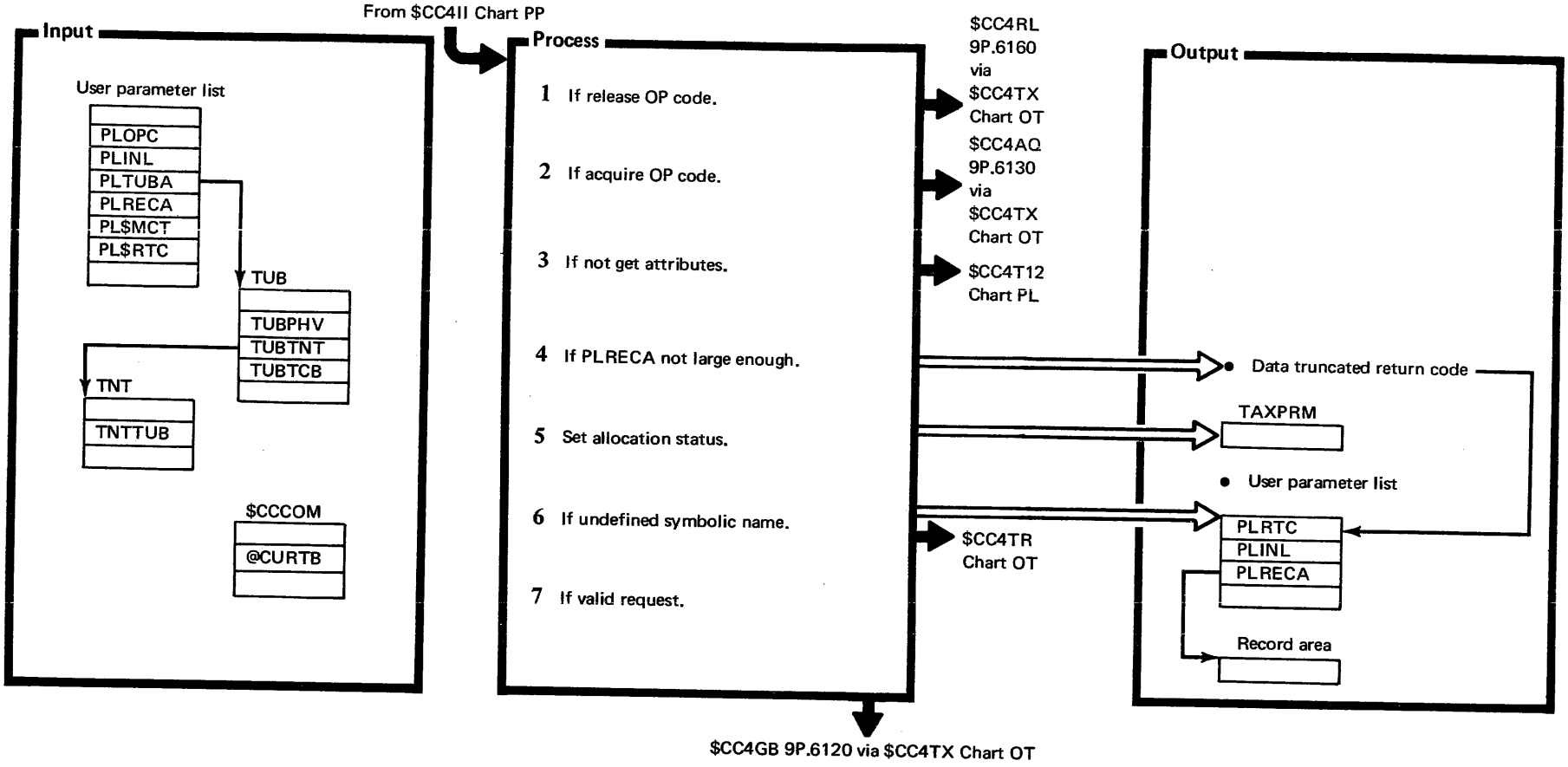


Diagram 9P.6100. $CC4GA

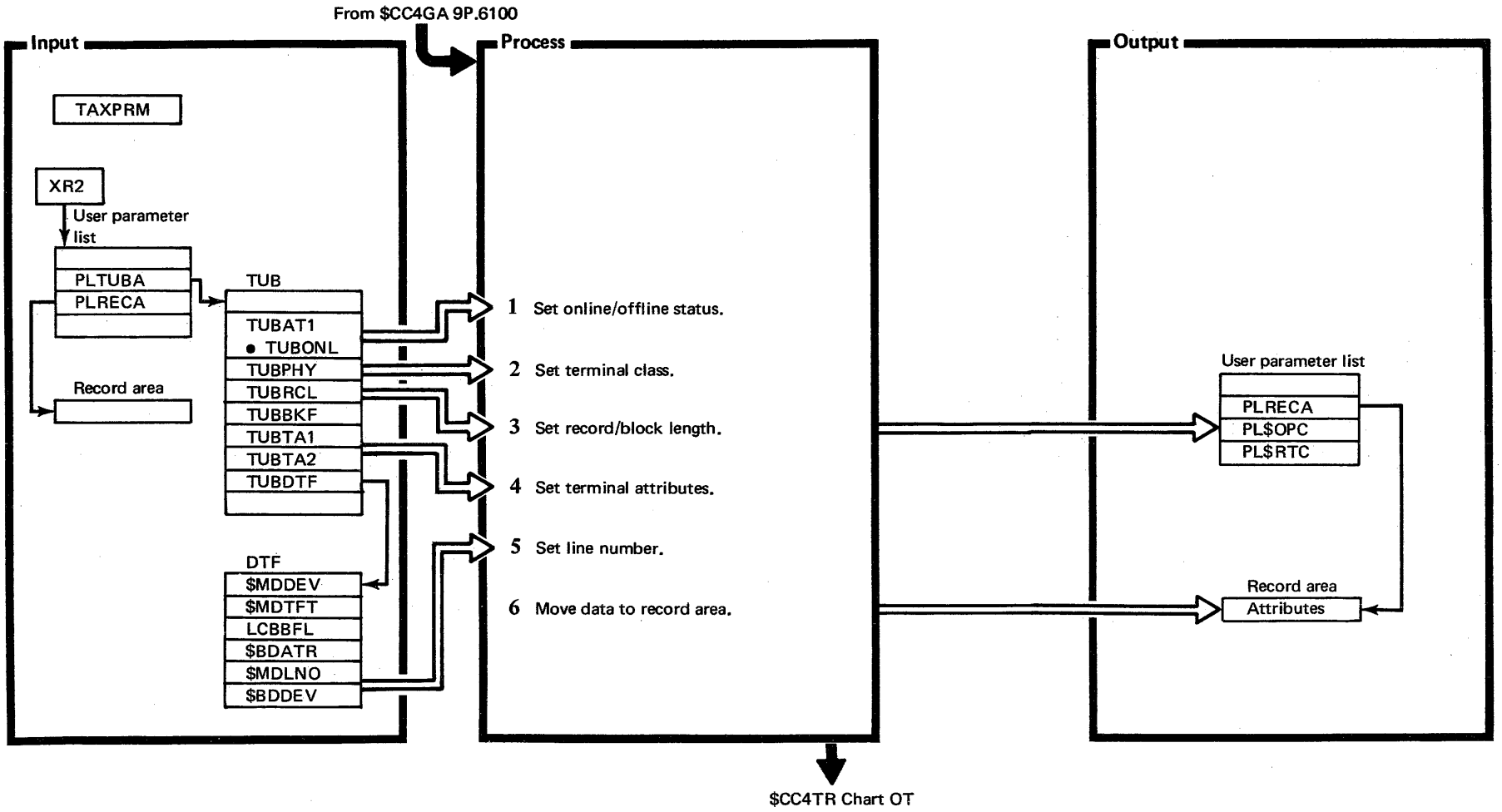


Diagram 9P.6120. $CC4GB

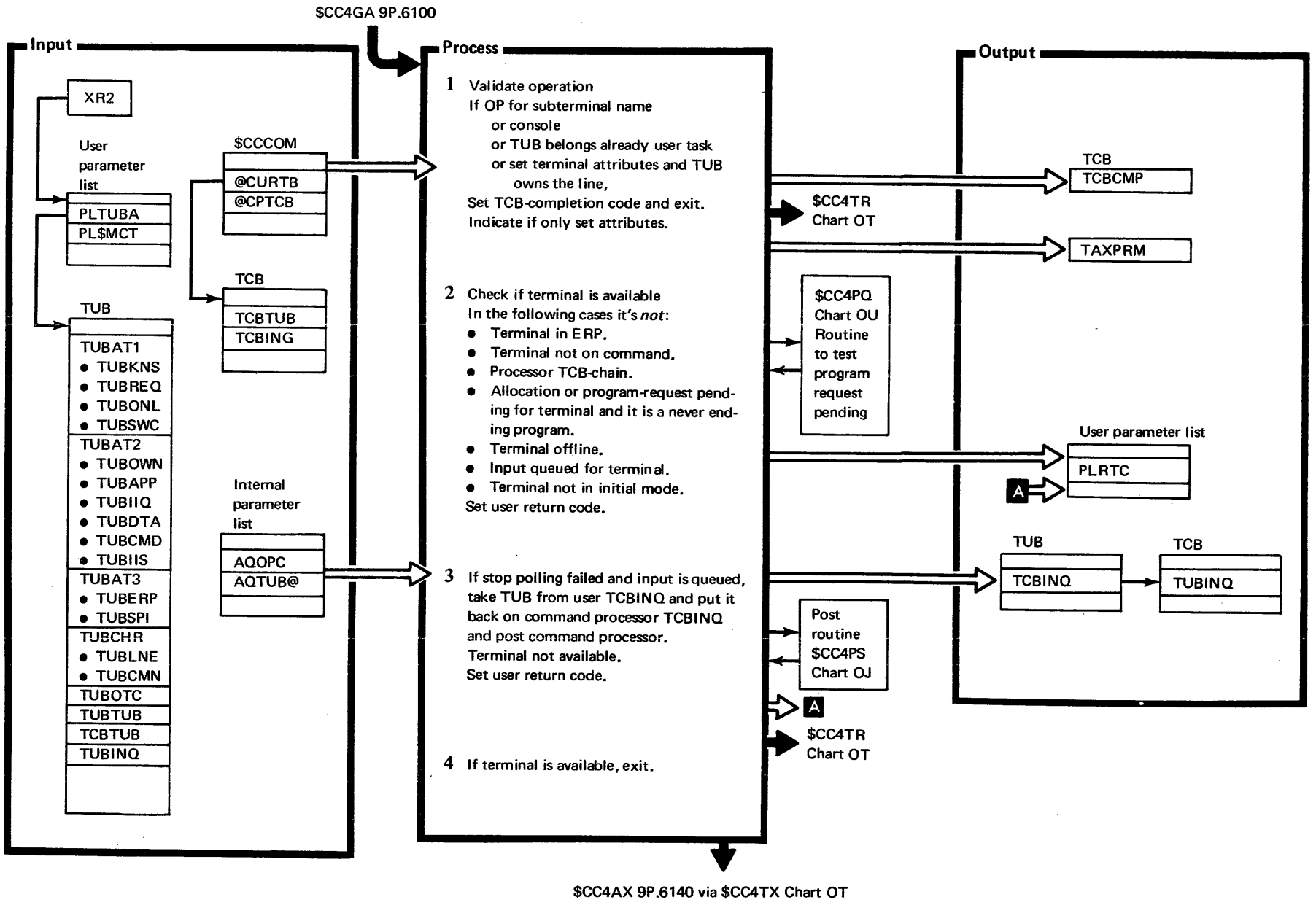


Diagram 9P.6130. $CC4AQ

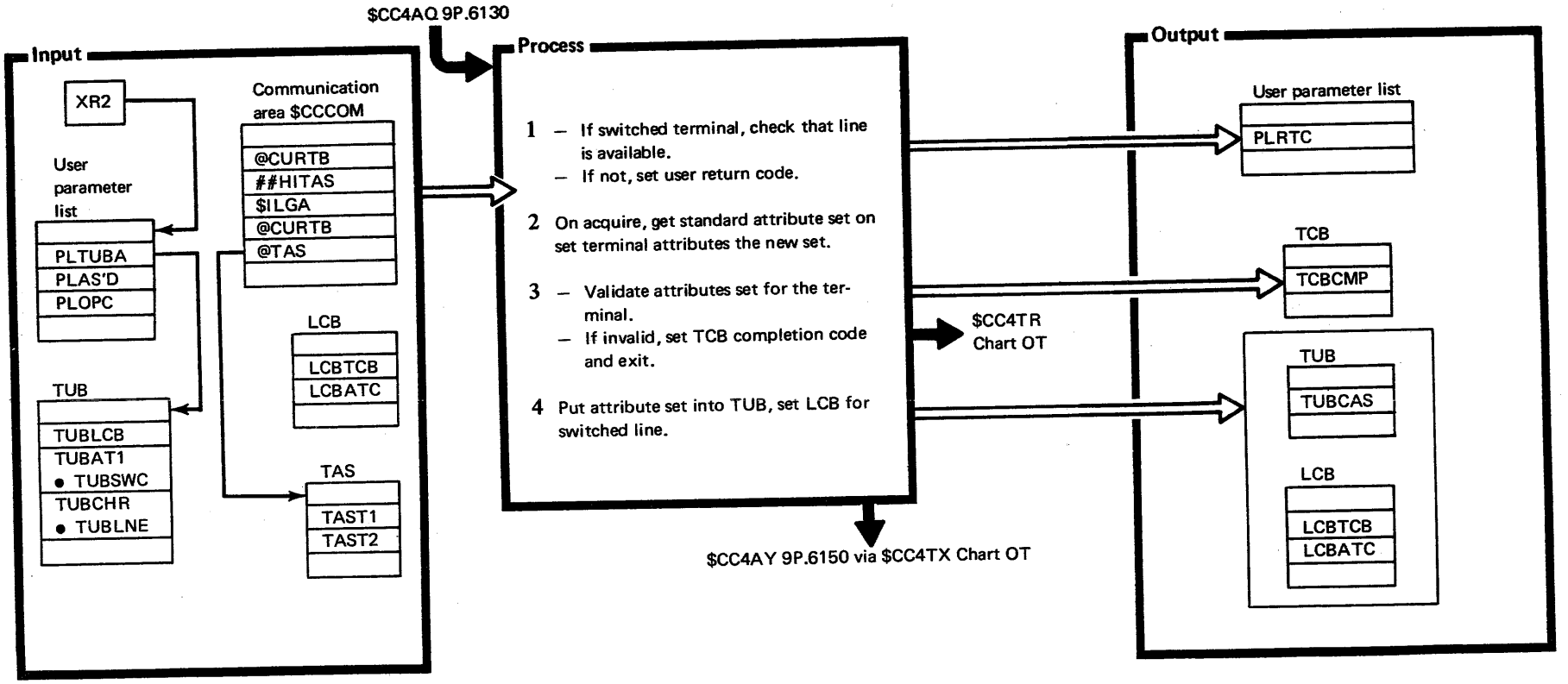


Diagram 9P.6140. $CC4AX

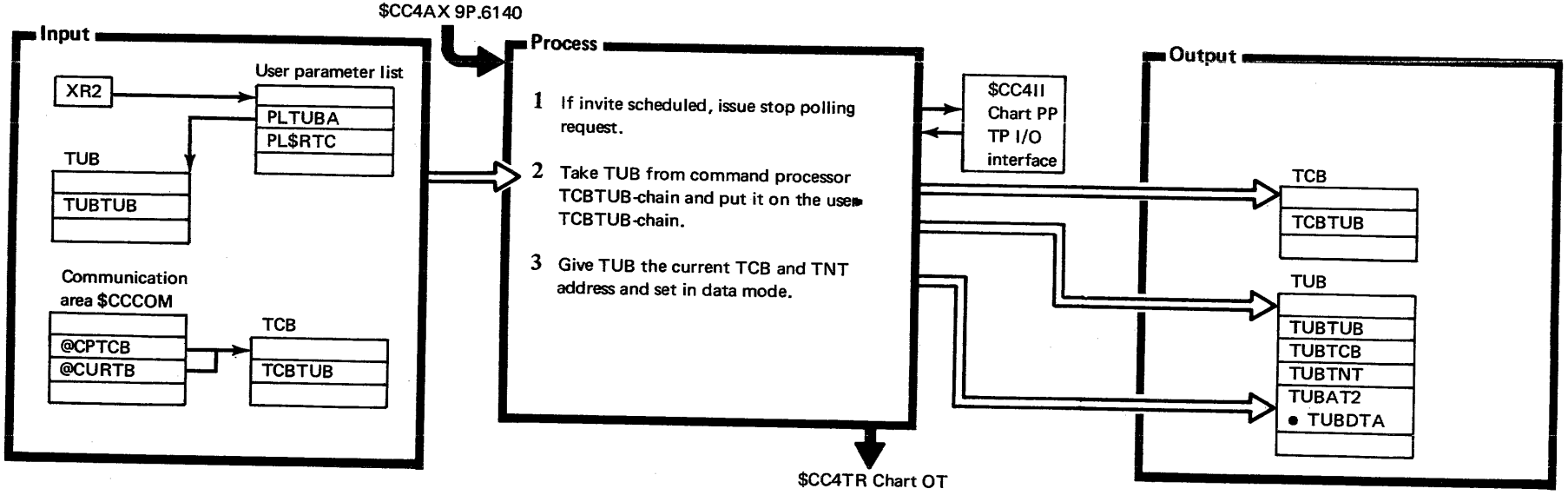


Diagram 9P.6150. $CC4AY

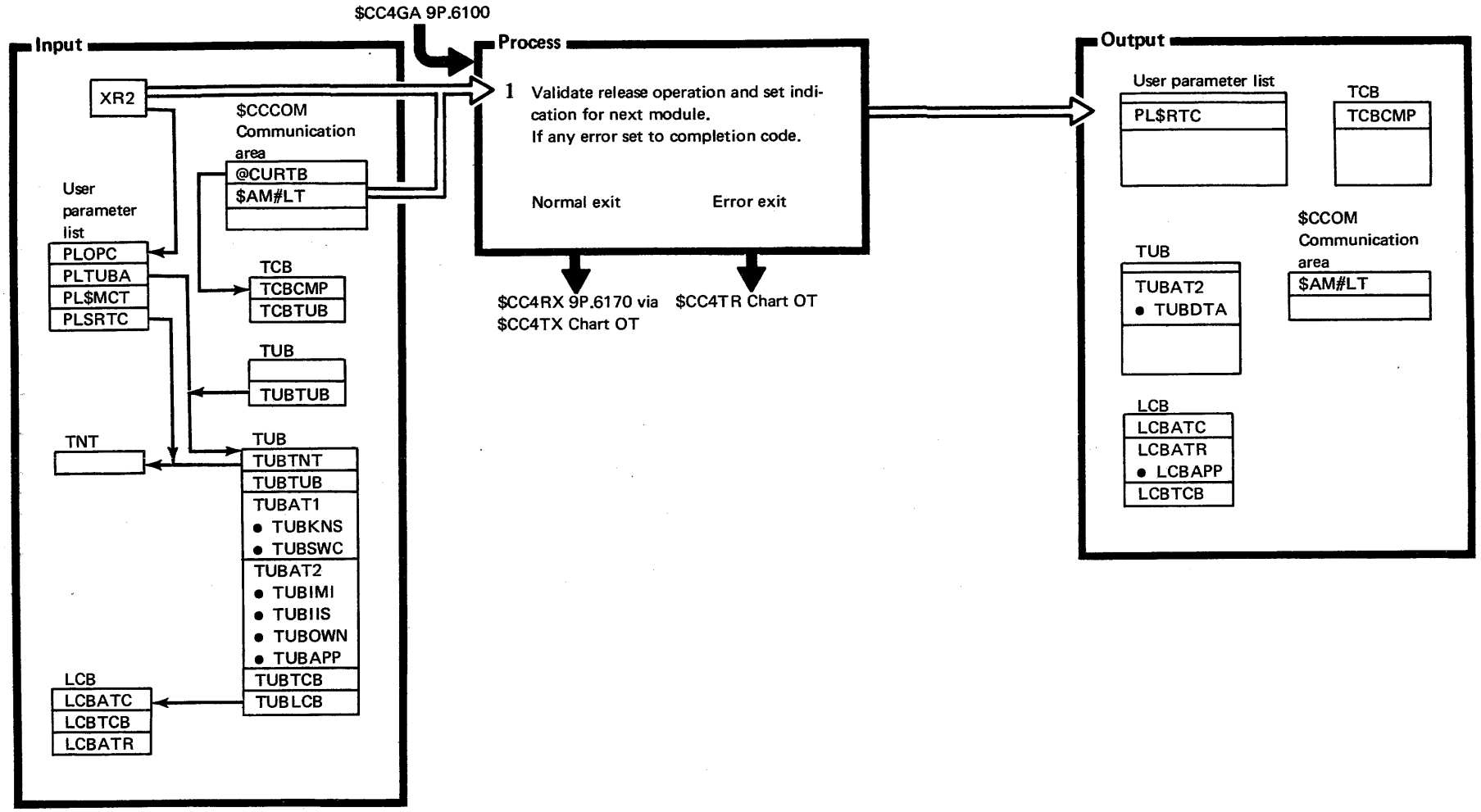


Diagram 9P.6160. $CC4RL

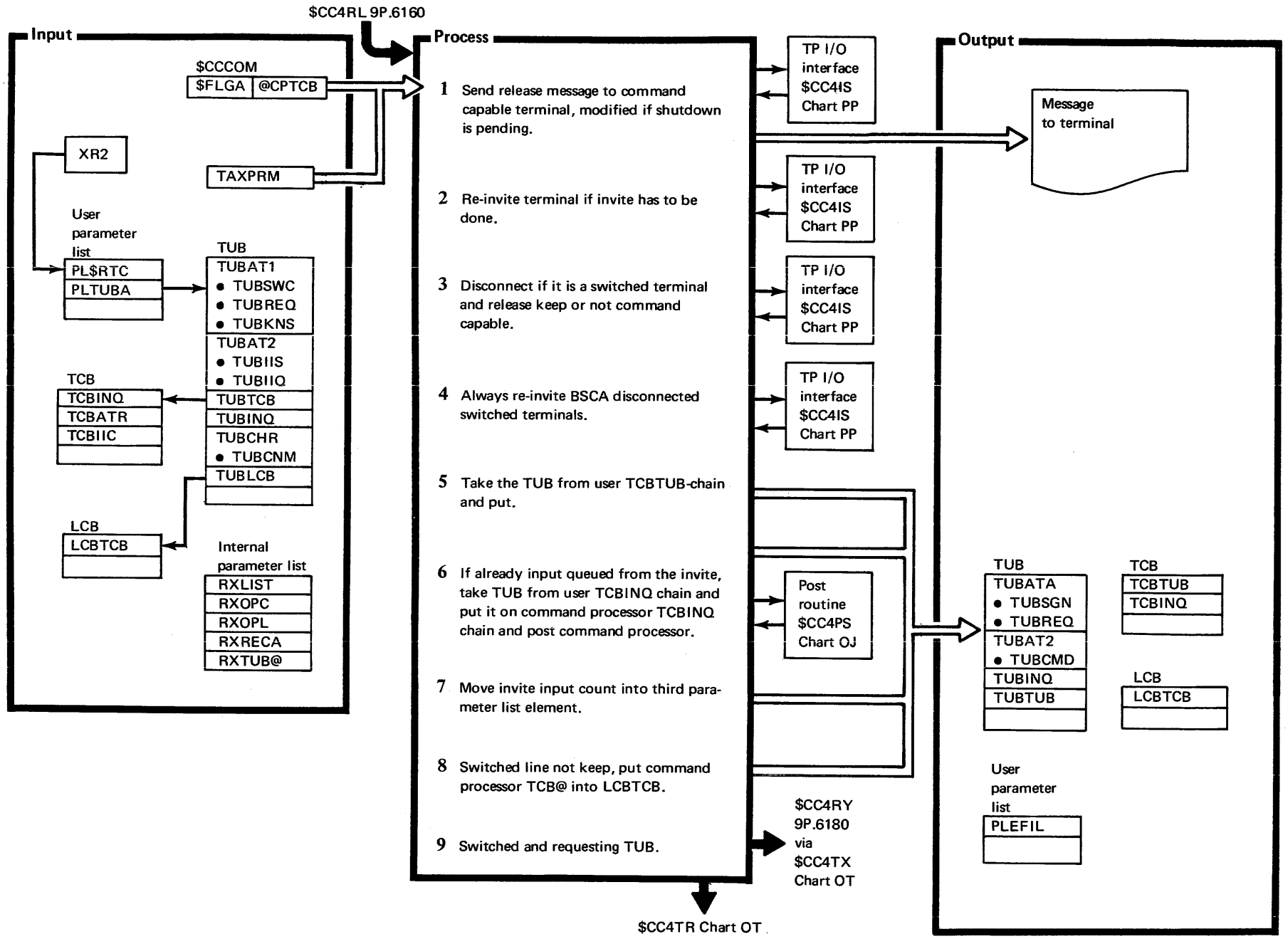


Diagram 9P.6170. $CC4RX

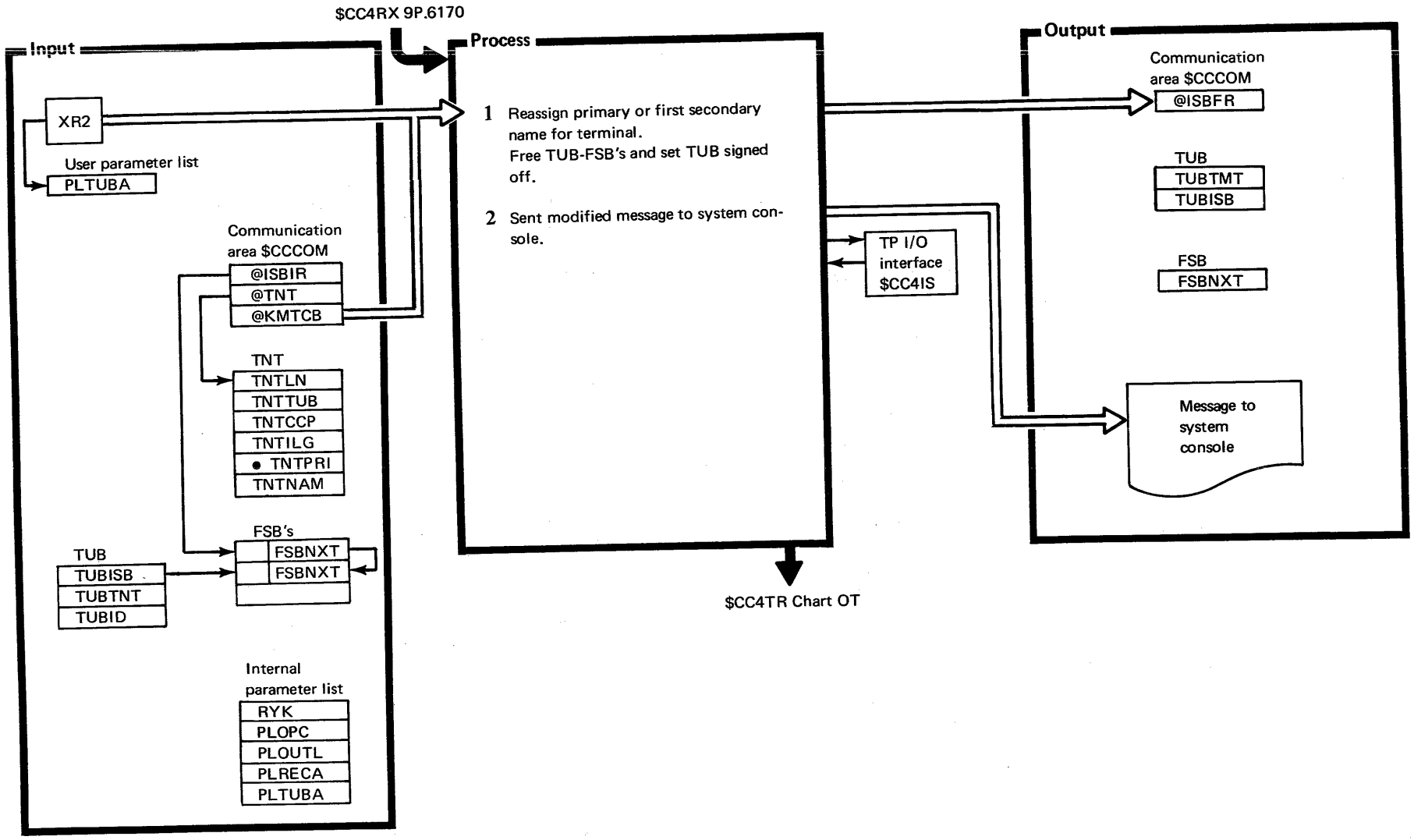


Diagram 9P.6180. $CC4RY

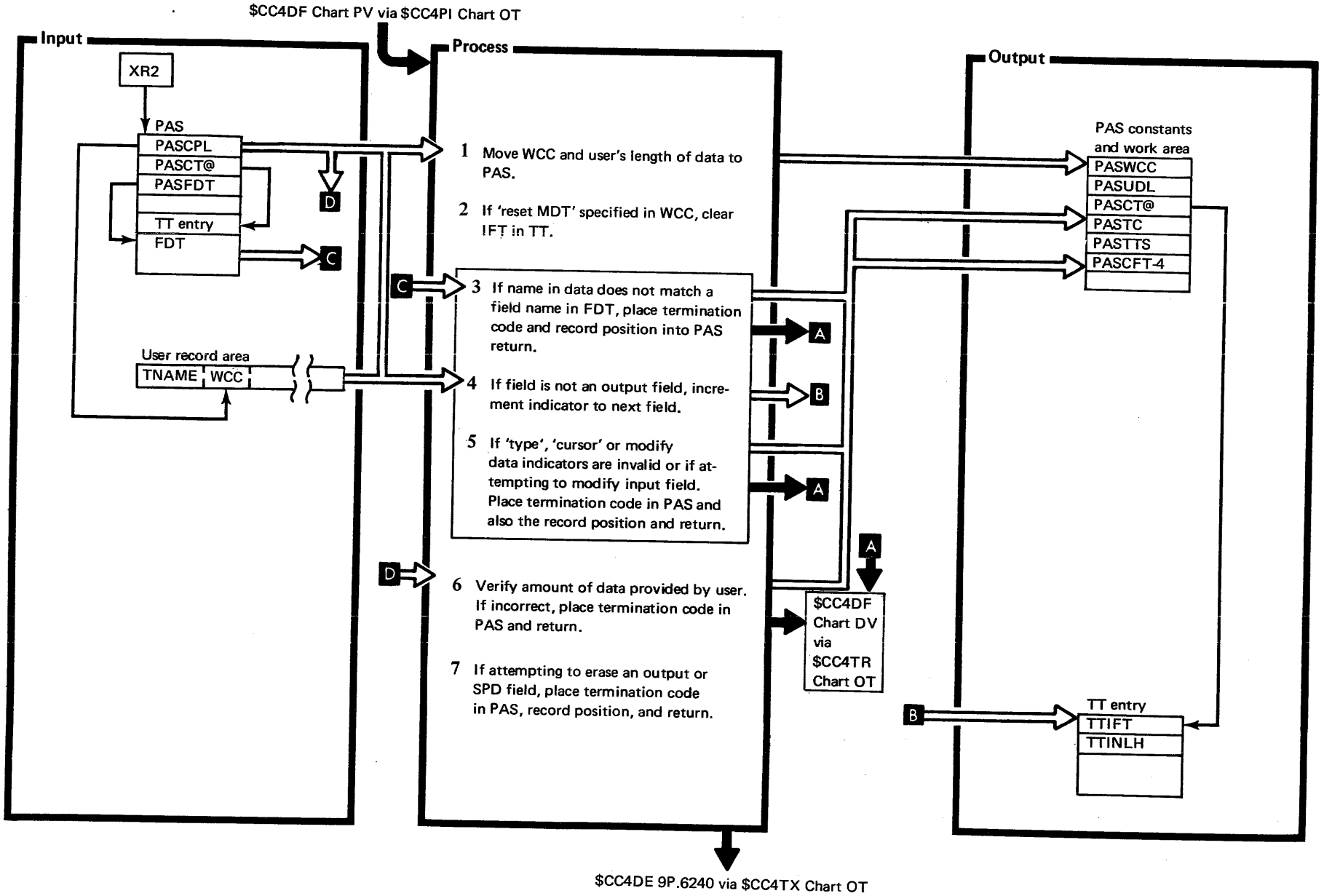


Diagram 9P.6210. $CC4DB

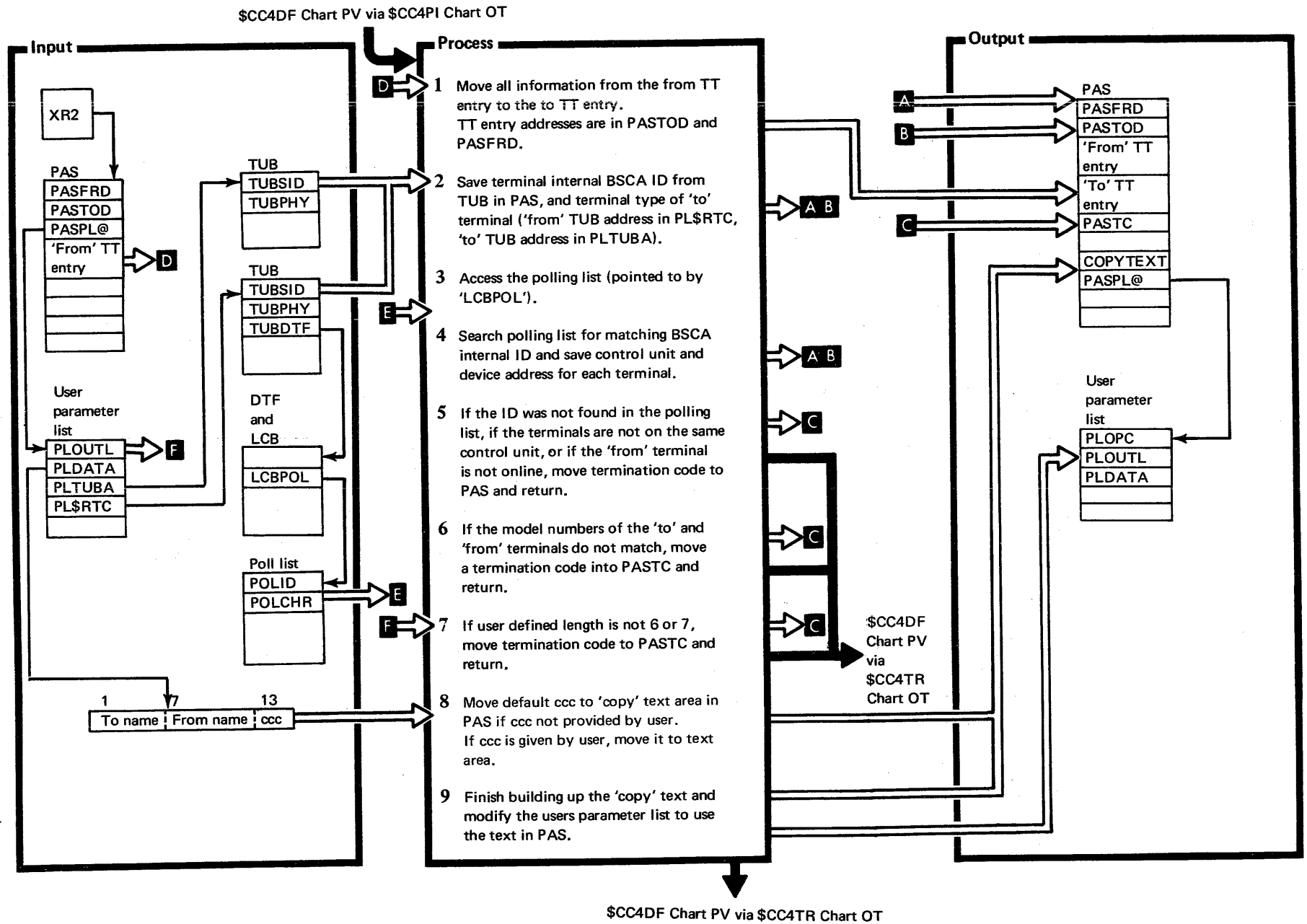
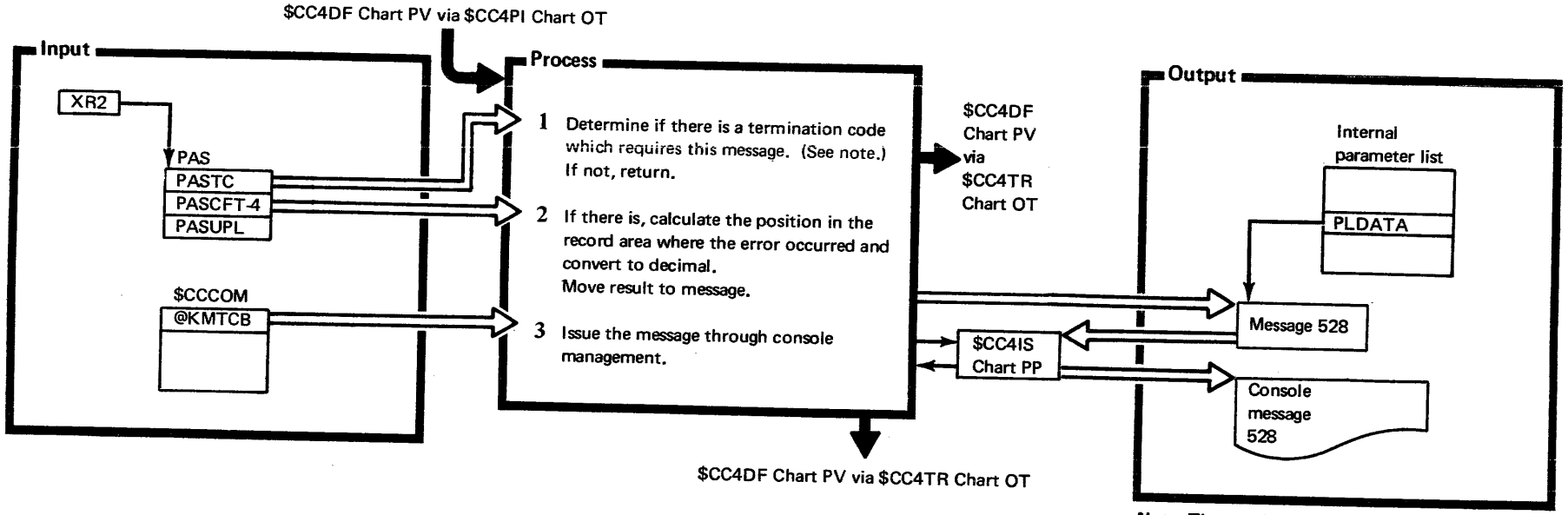


Diagram 9P.6220. $CC4DC



Note: The termination codes for which this message is given are EE, EF, F0, F1, and F4.

Diagram 9P.6230. $CC4DD

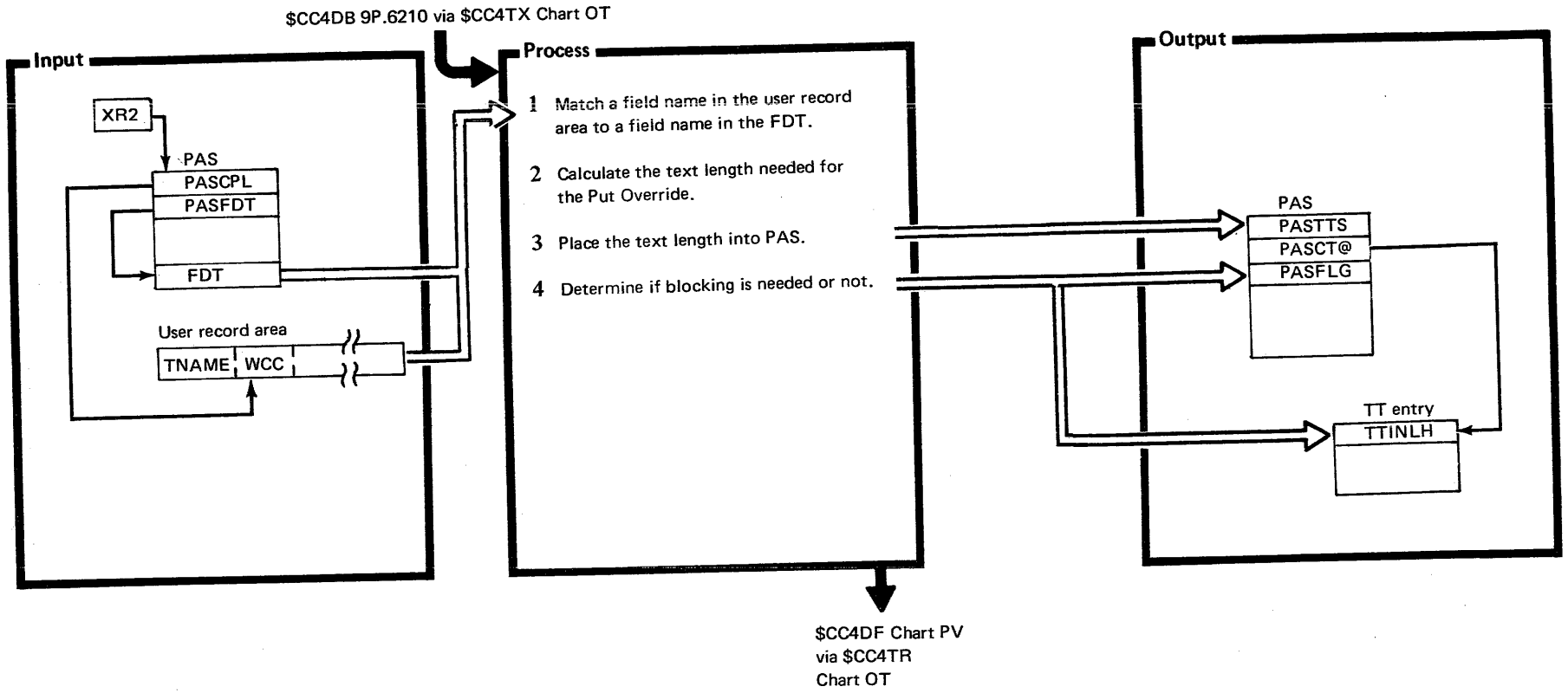


Diagram 9P.6240. $CC4DE

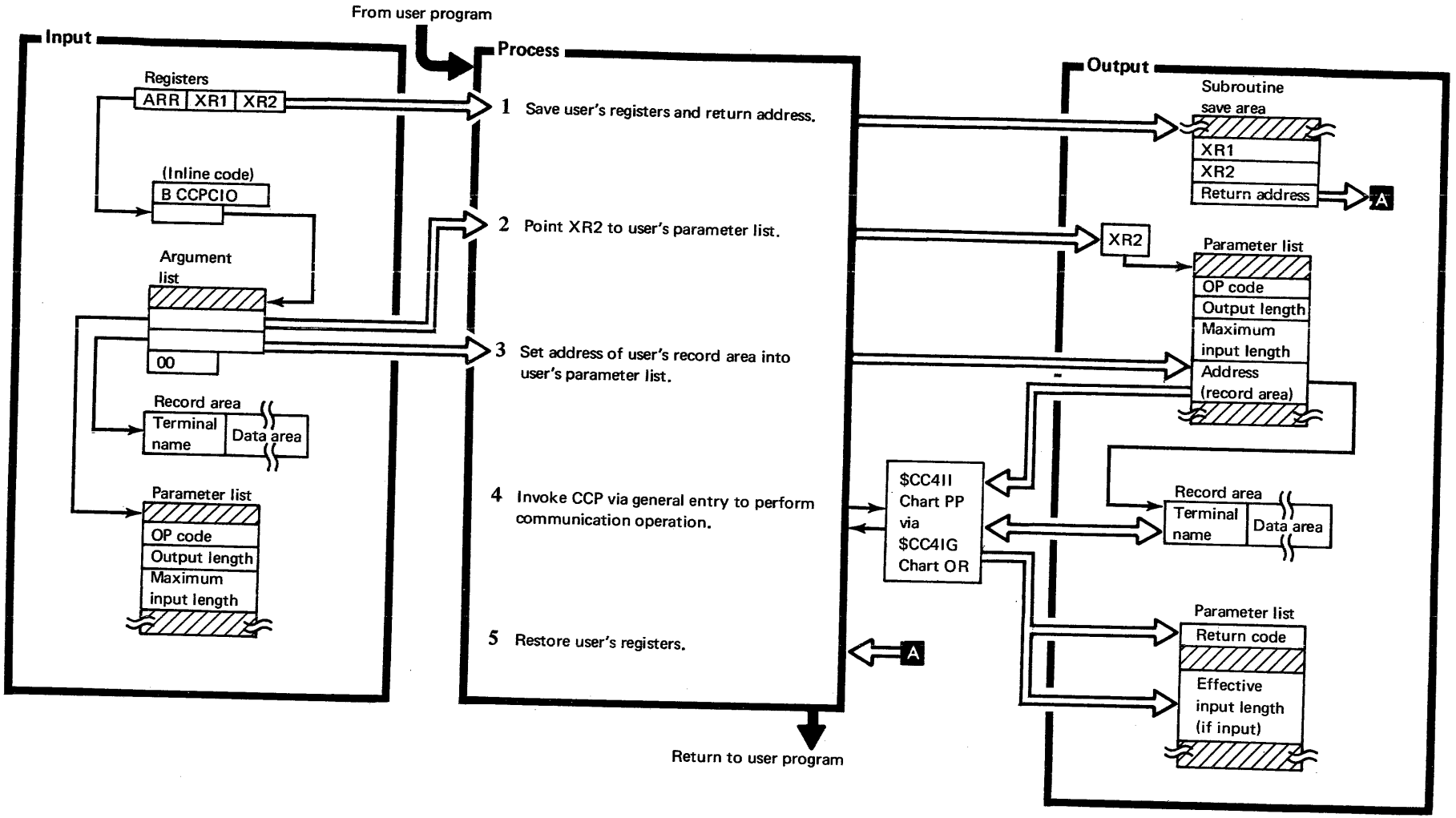


Diagram 9P.6300. CCPCIO

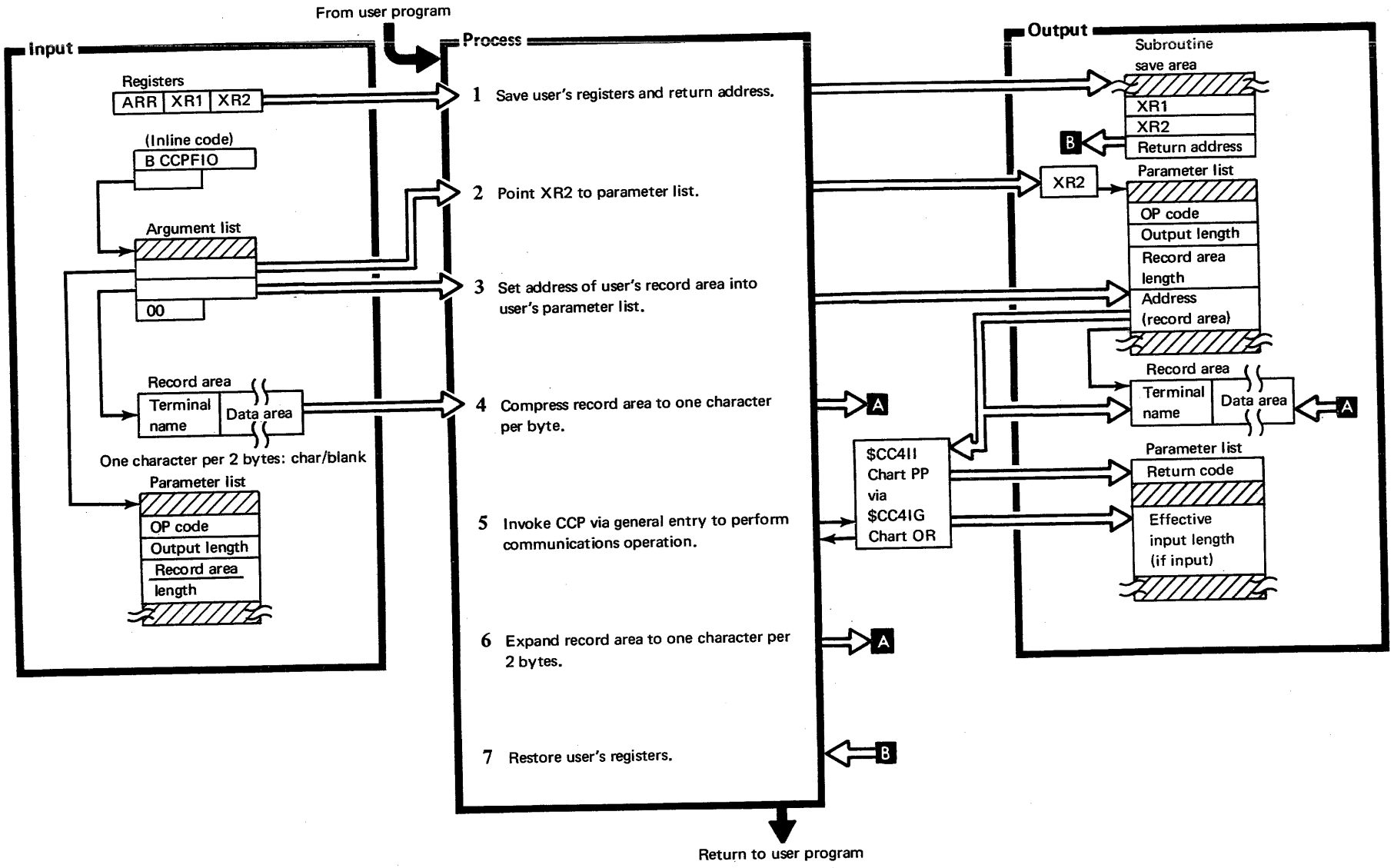


Diagram 9P.6310. CCPFIO

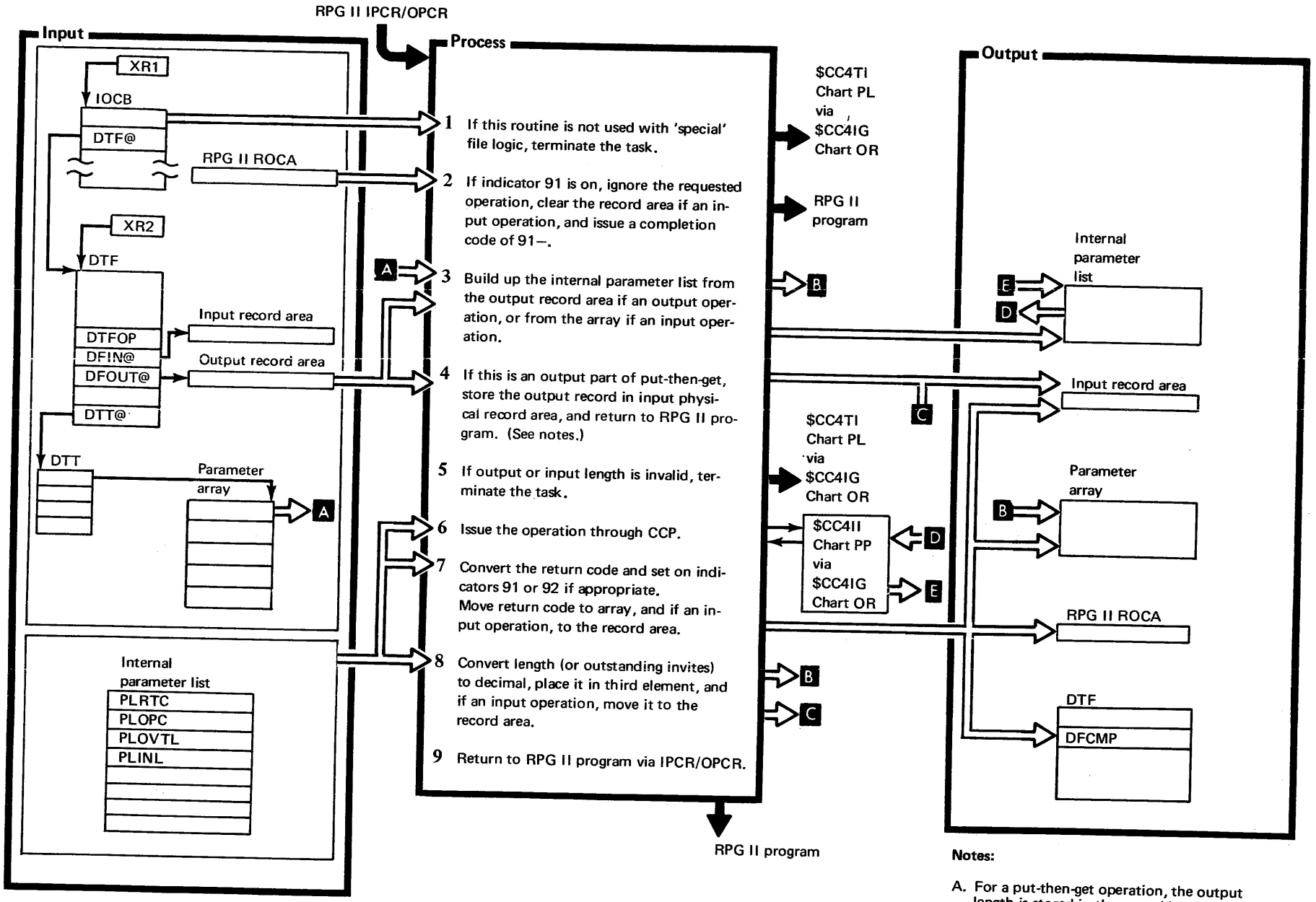


Diagram 9P.6320. SUBR92

Notes:

A. For a put-then-get operation, the output length is stored in the record length part of the DTF.

B. For a put-and-invite operation, the 'put' is executed first, then the 'invite' OP code is placed in the internal parameter list and step 6 is executed again.

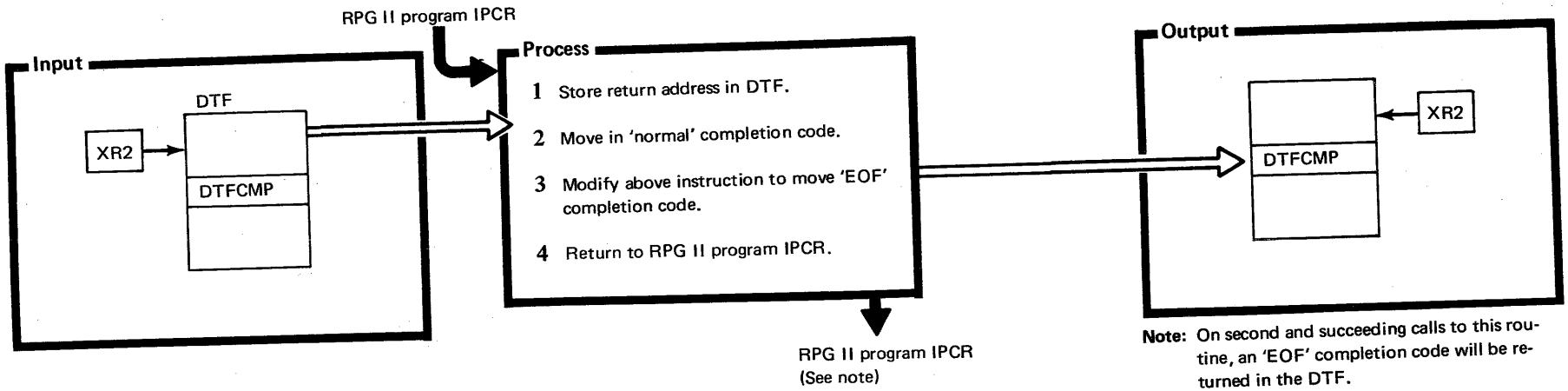


Diagram 9P.6330. SUBR93

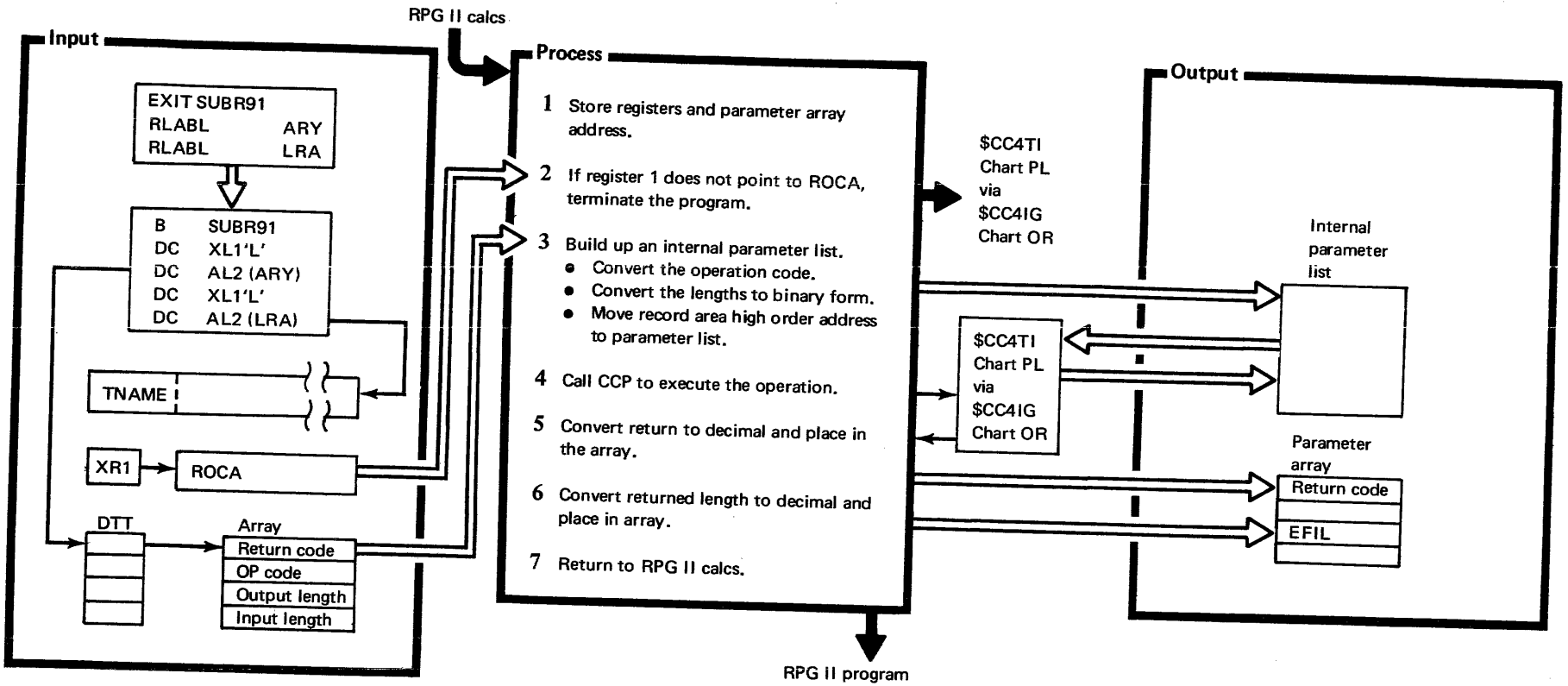
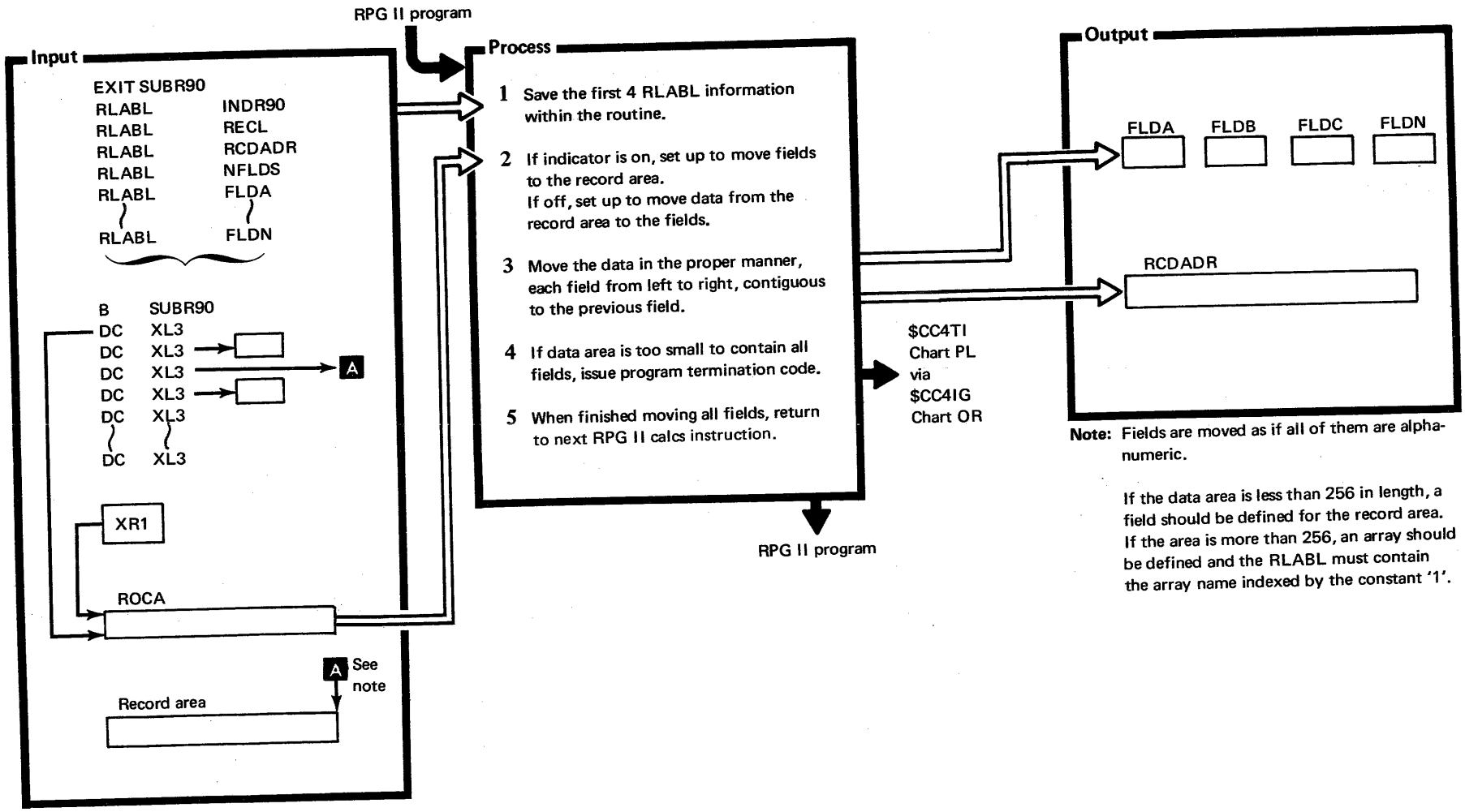


Diagram 9P.6340. SUBR91



Note: Fields are moved as if all of them are alpha-numeric.

If the data area is less than 256 in length, a field should be defined for the record area. If the area is more than 256, an array should be defined and the RLABL must contain the array name indexed by the constant '1'.

Diagram 9P.6350. SUBR90

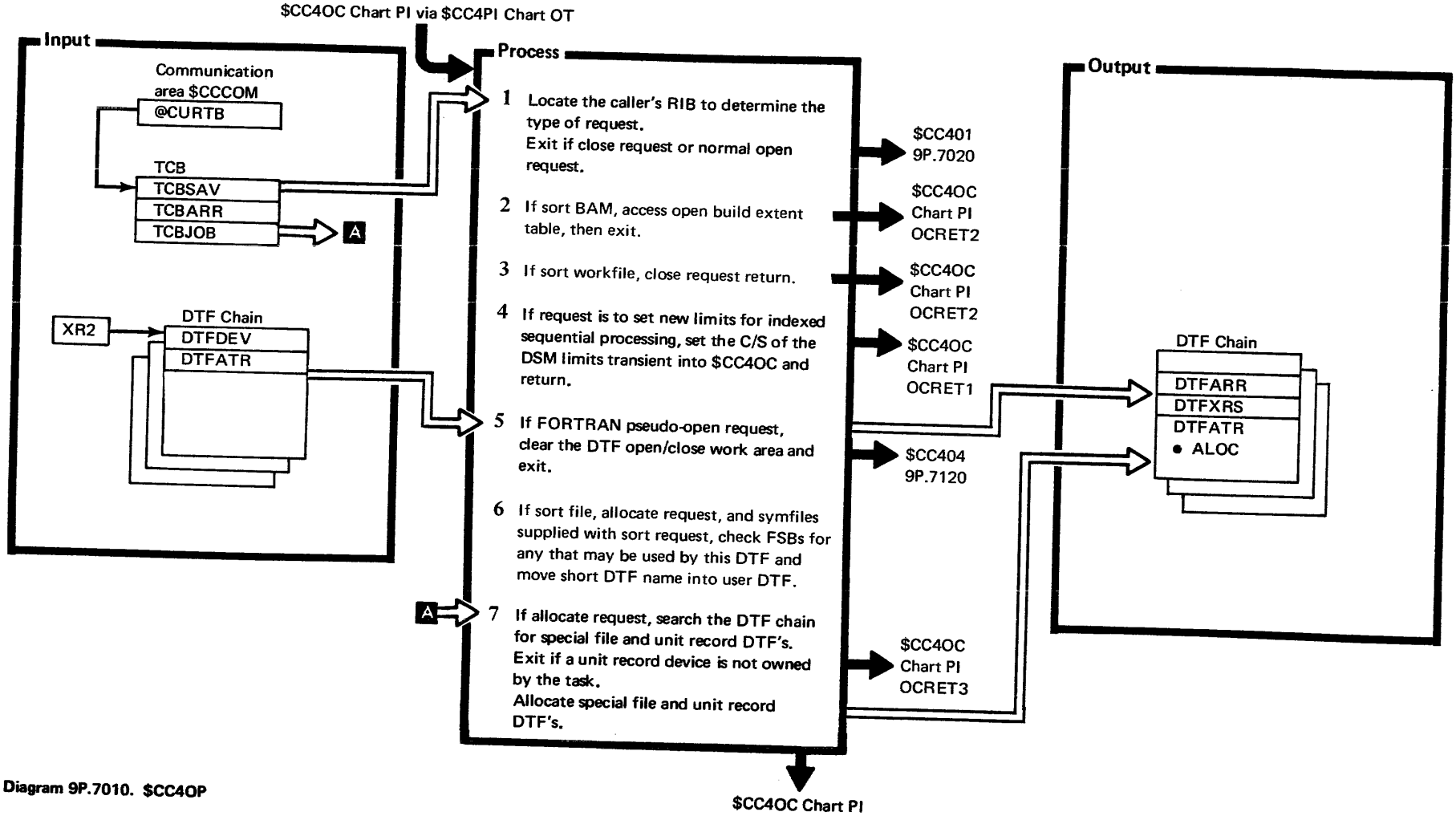


Diagram 9P.7010. $CC4OP

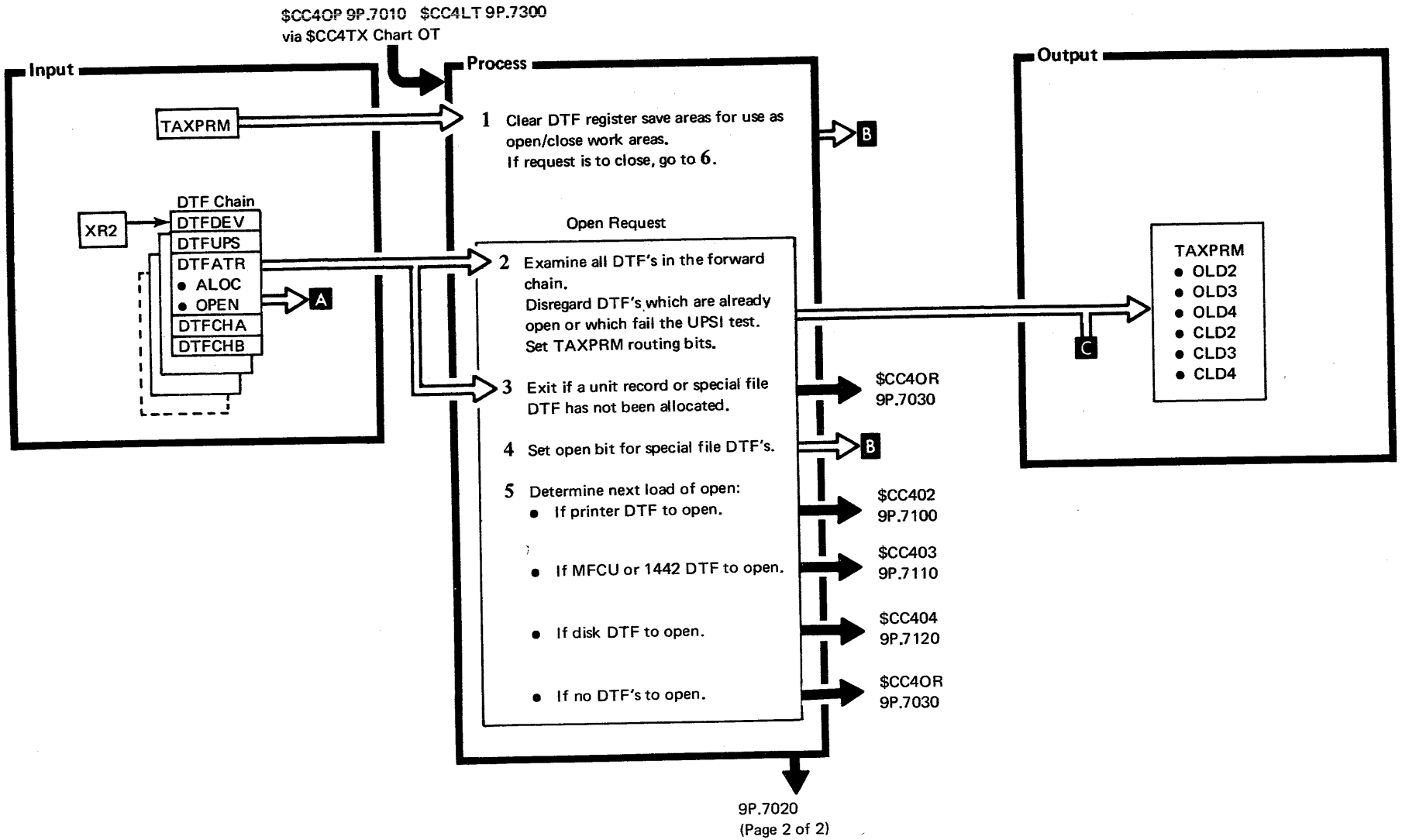


Diagram 9P.7020 (Part 1 of 2). $CC401

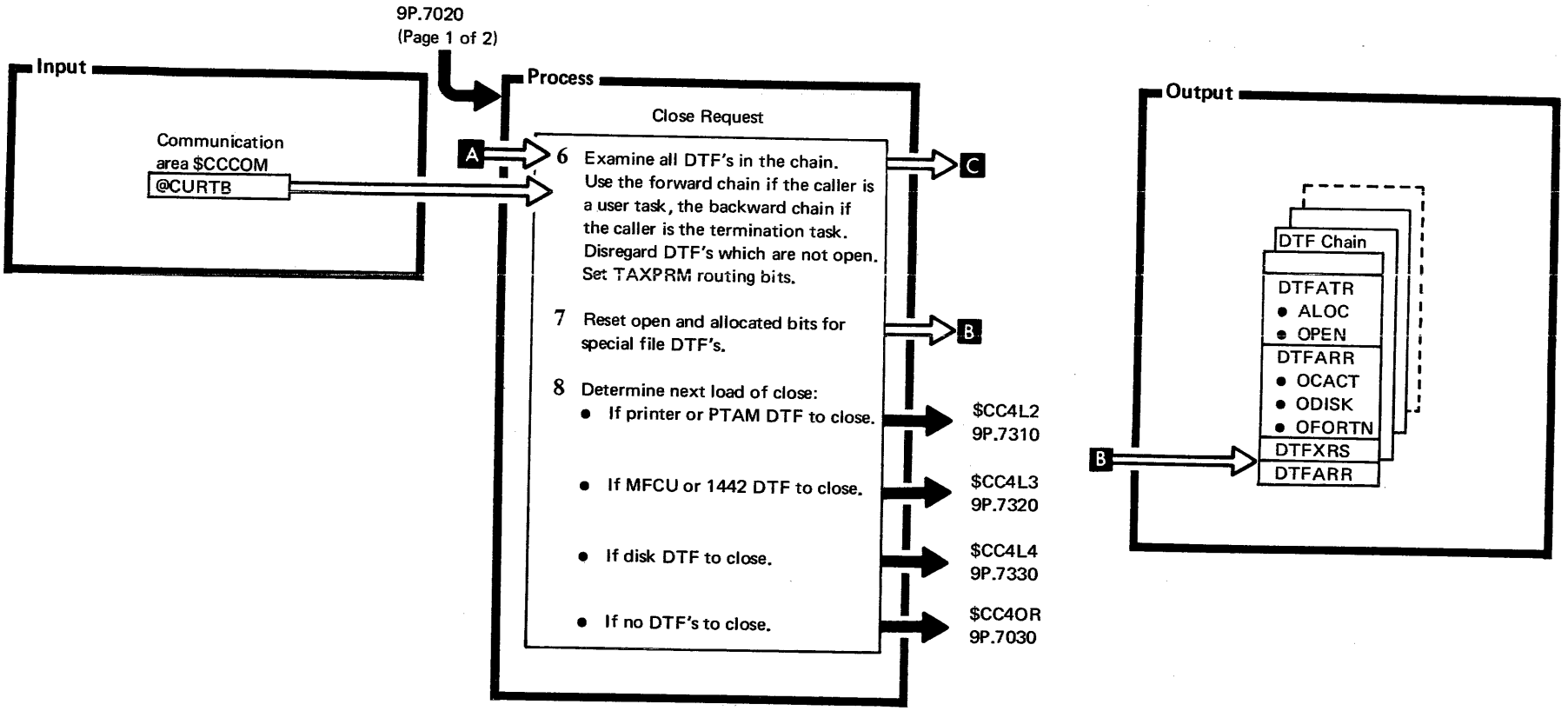


Diagram 9P.7020 (Part 2 of 2). $CC401

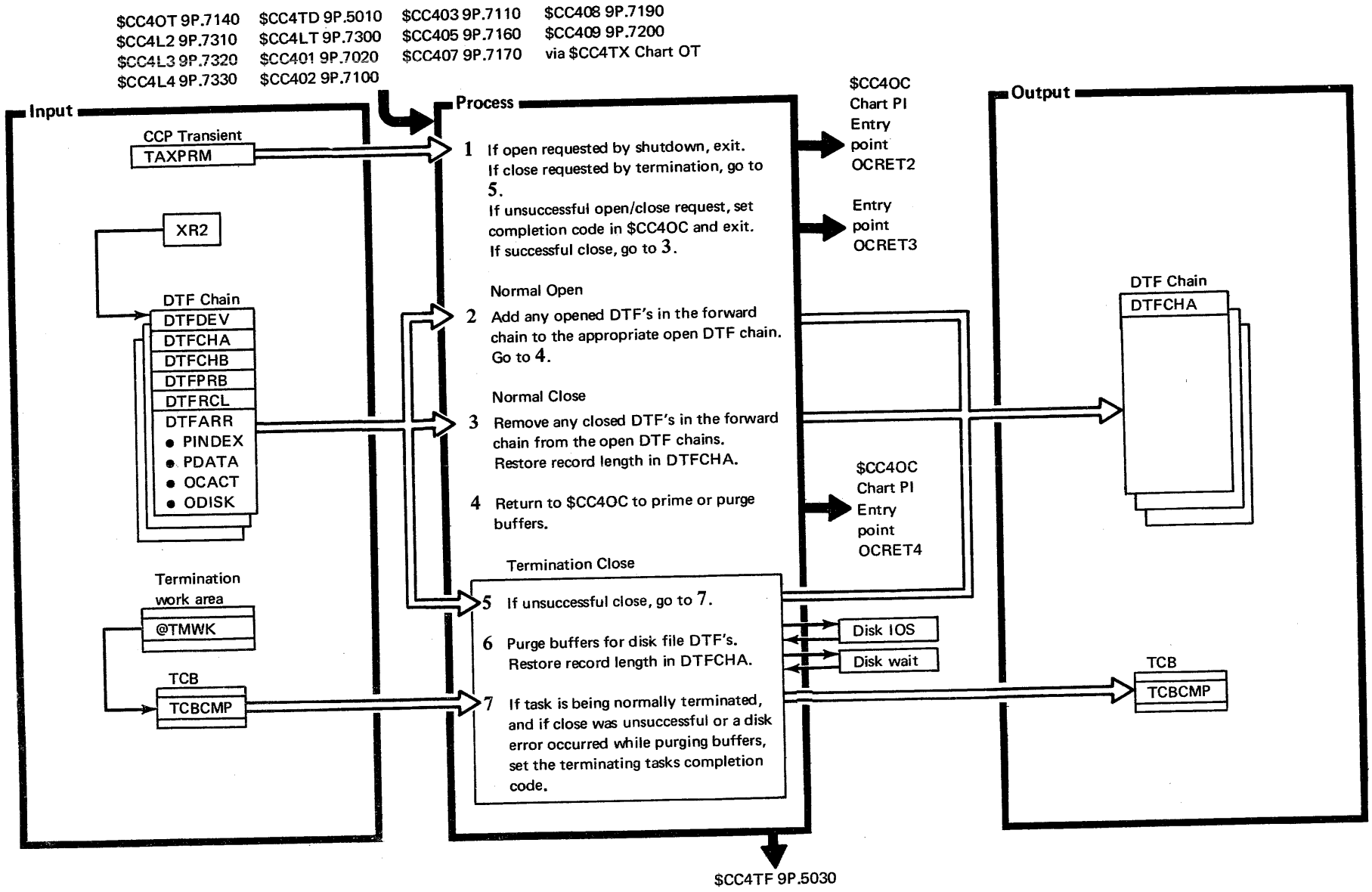


Diagram 9P.7030. $CC4OR

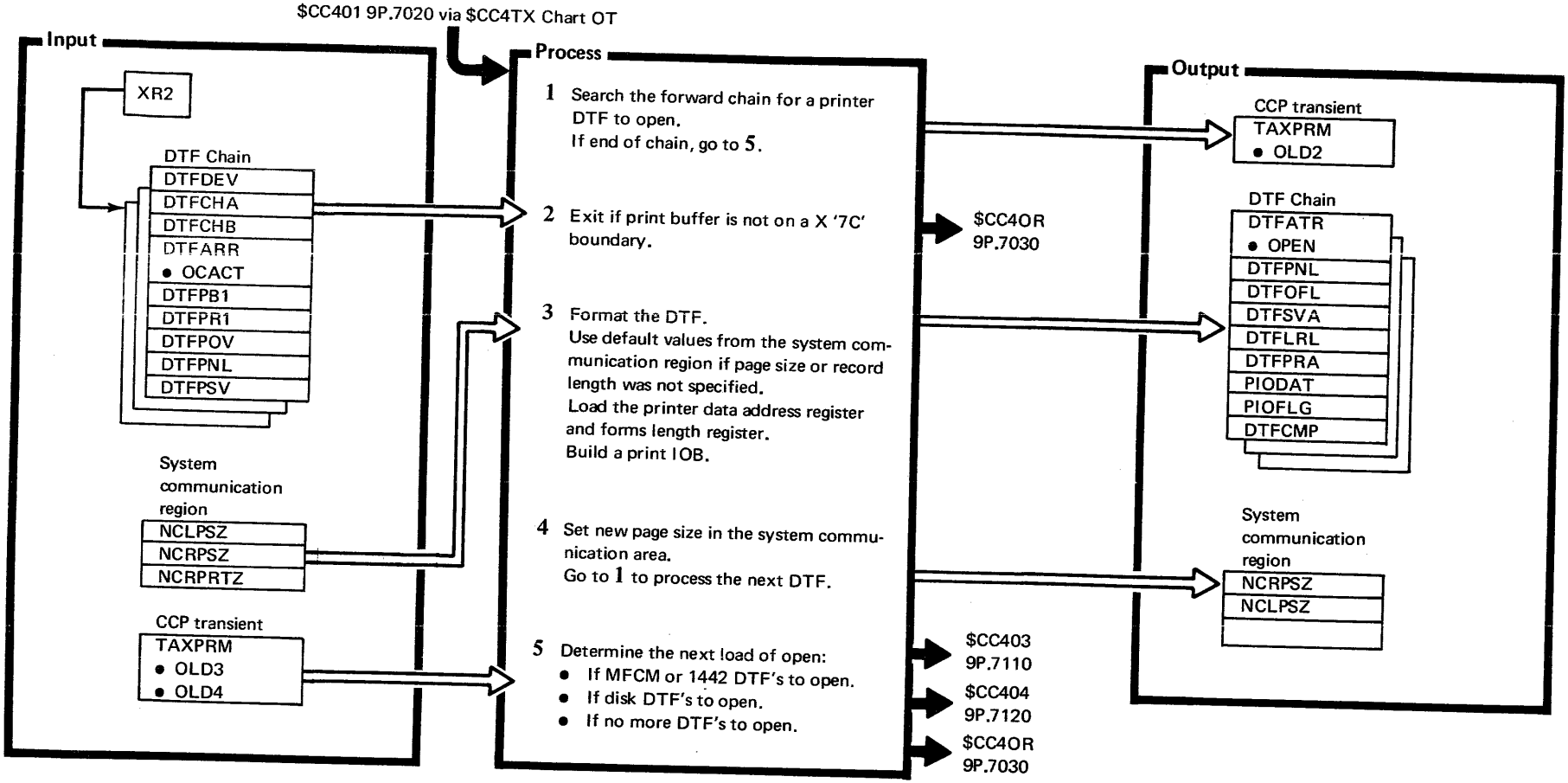


Diagram 9P.7100. $CC402 (Models 8, 10, and 12)

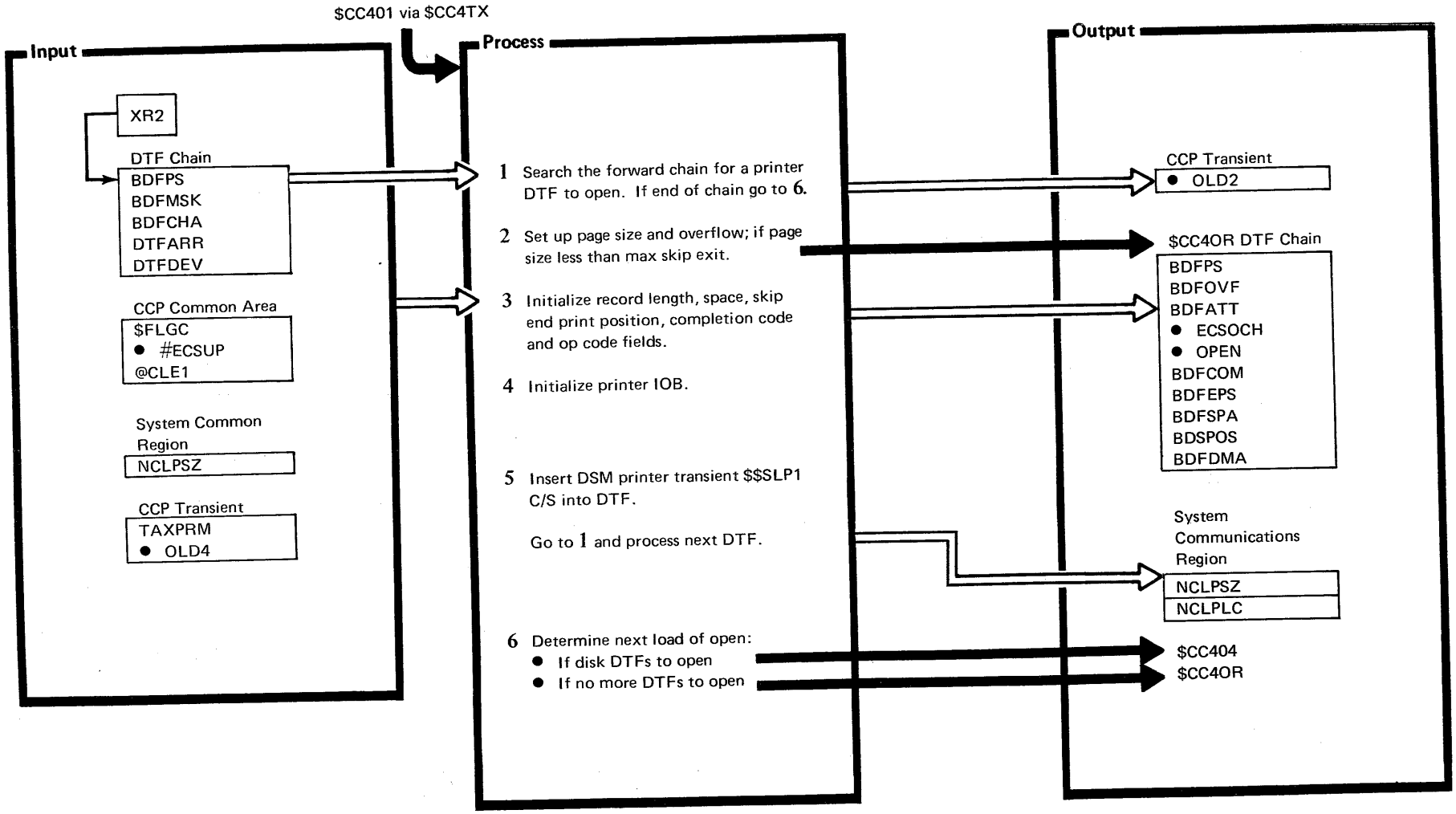


Diagram 9P.7105. $CC402 (Model 4)

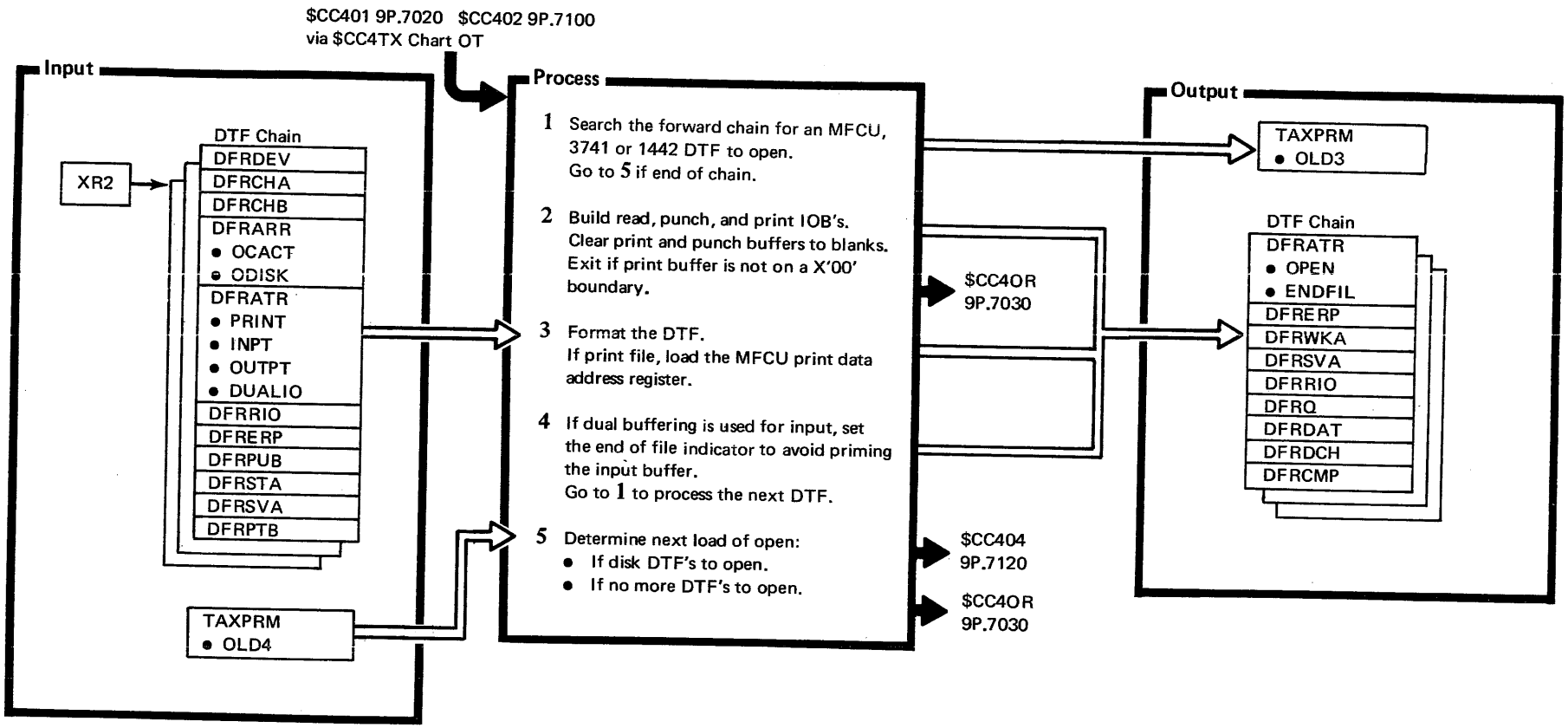
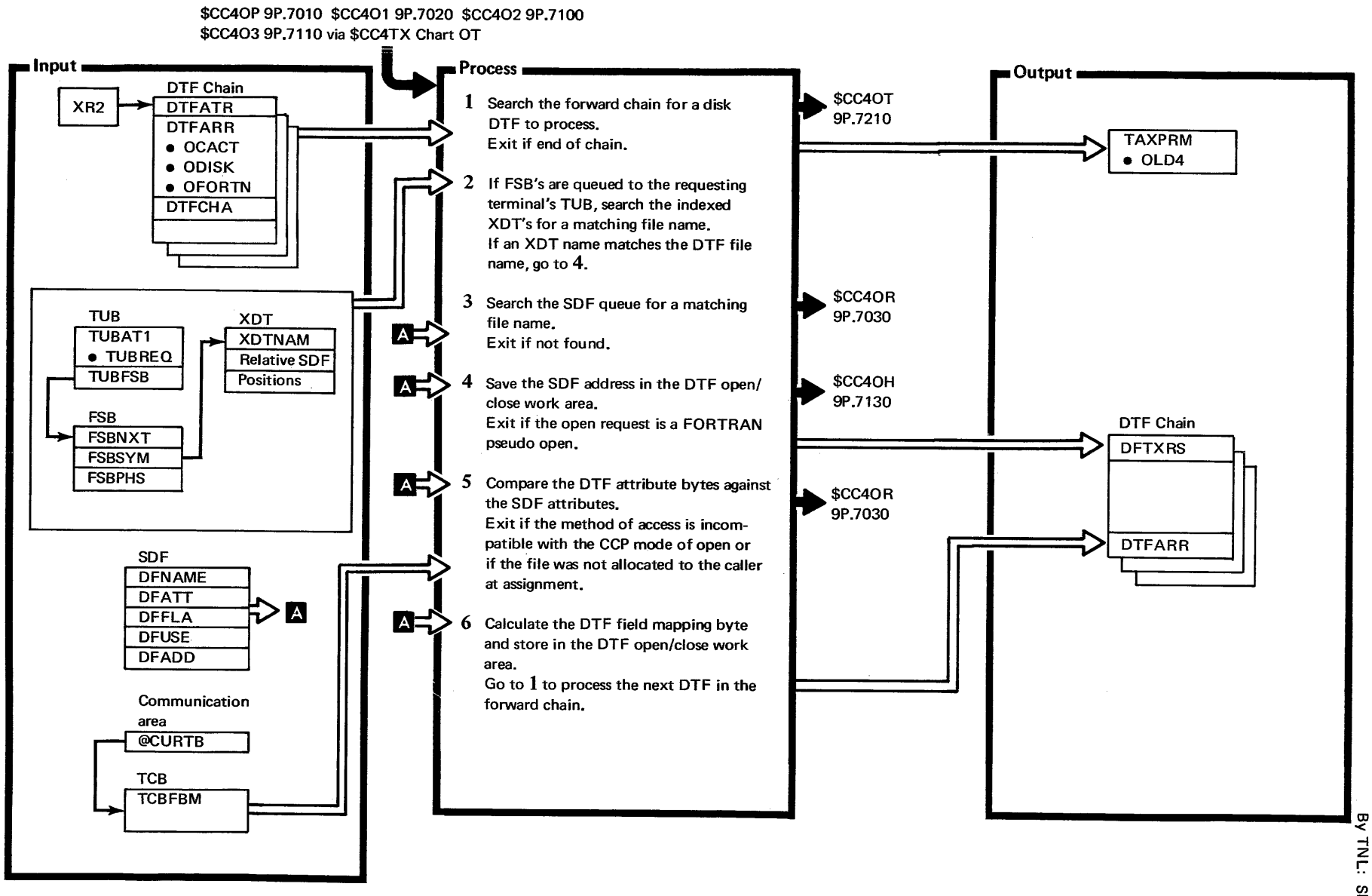


Diagram 9P.7110. $CC403



● **Diagram 9P.7120. $CC4O4**

Page of SY21-0531-2
Issued 24 June 1977
By TNL: SN21-5530

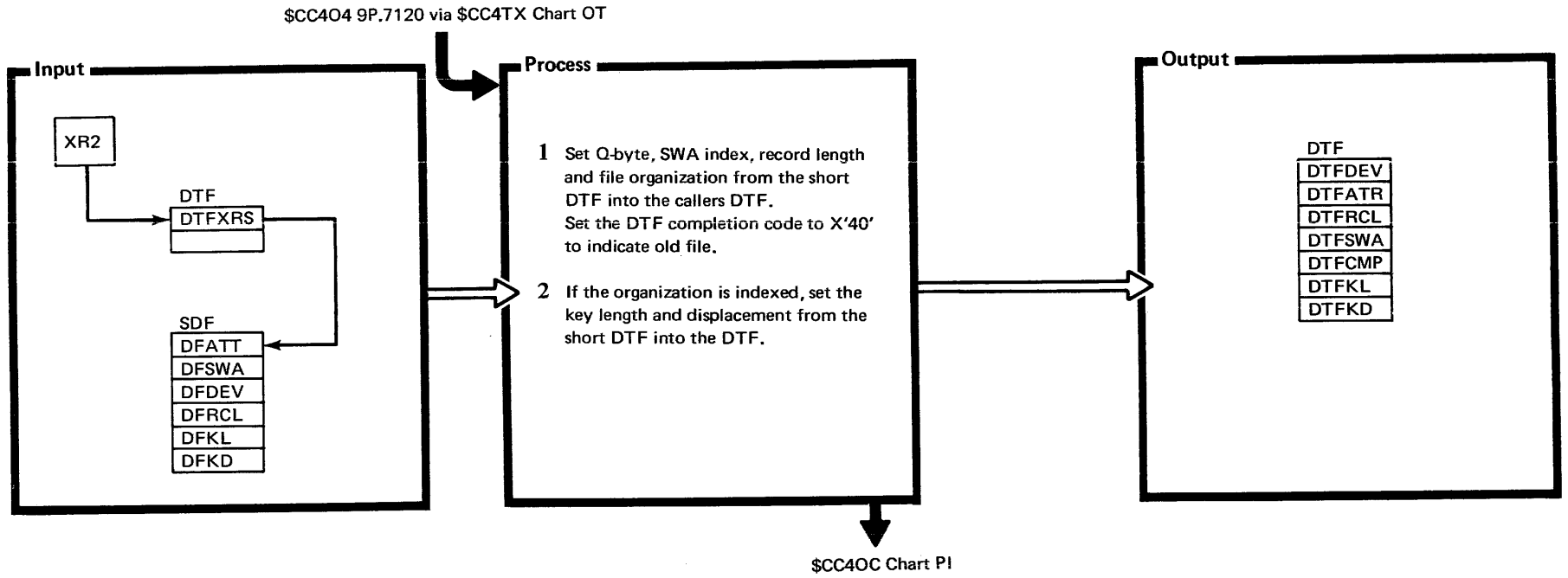
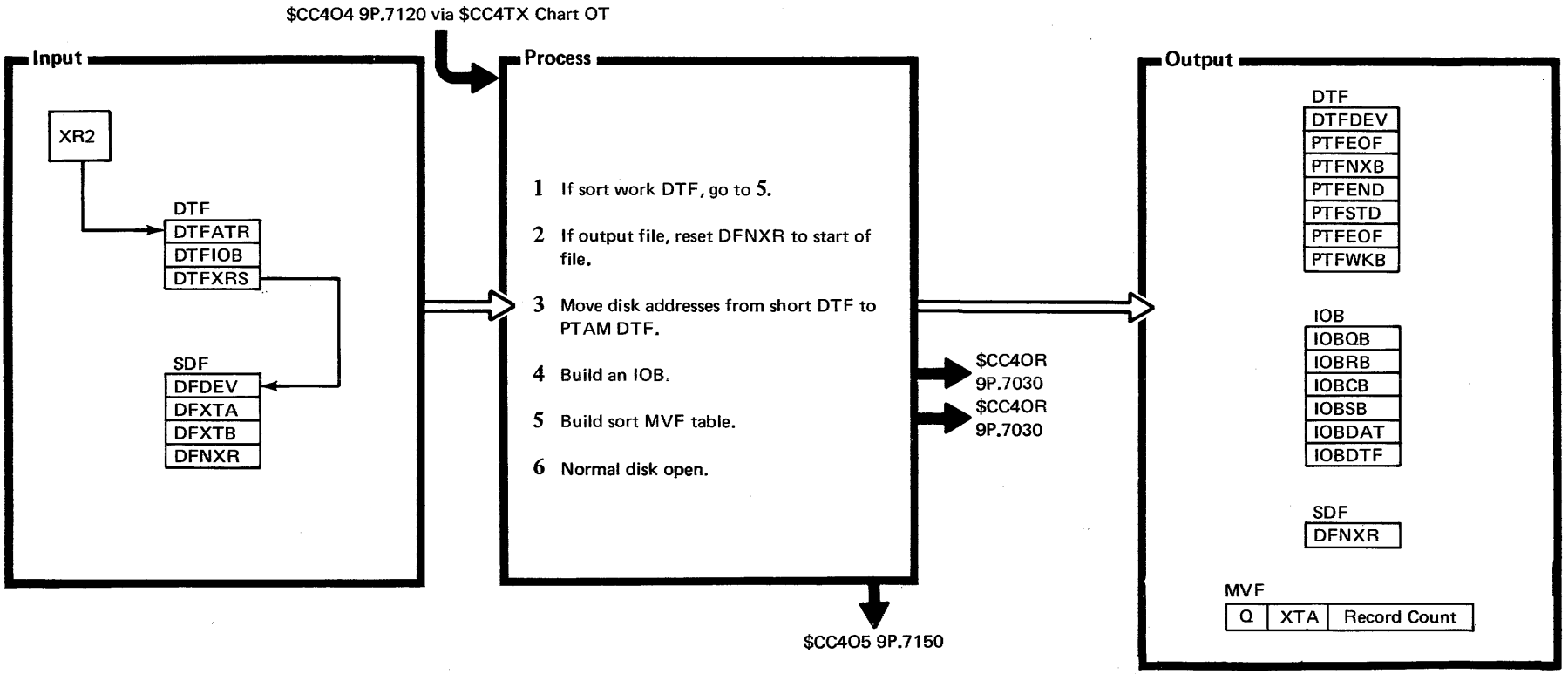
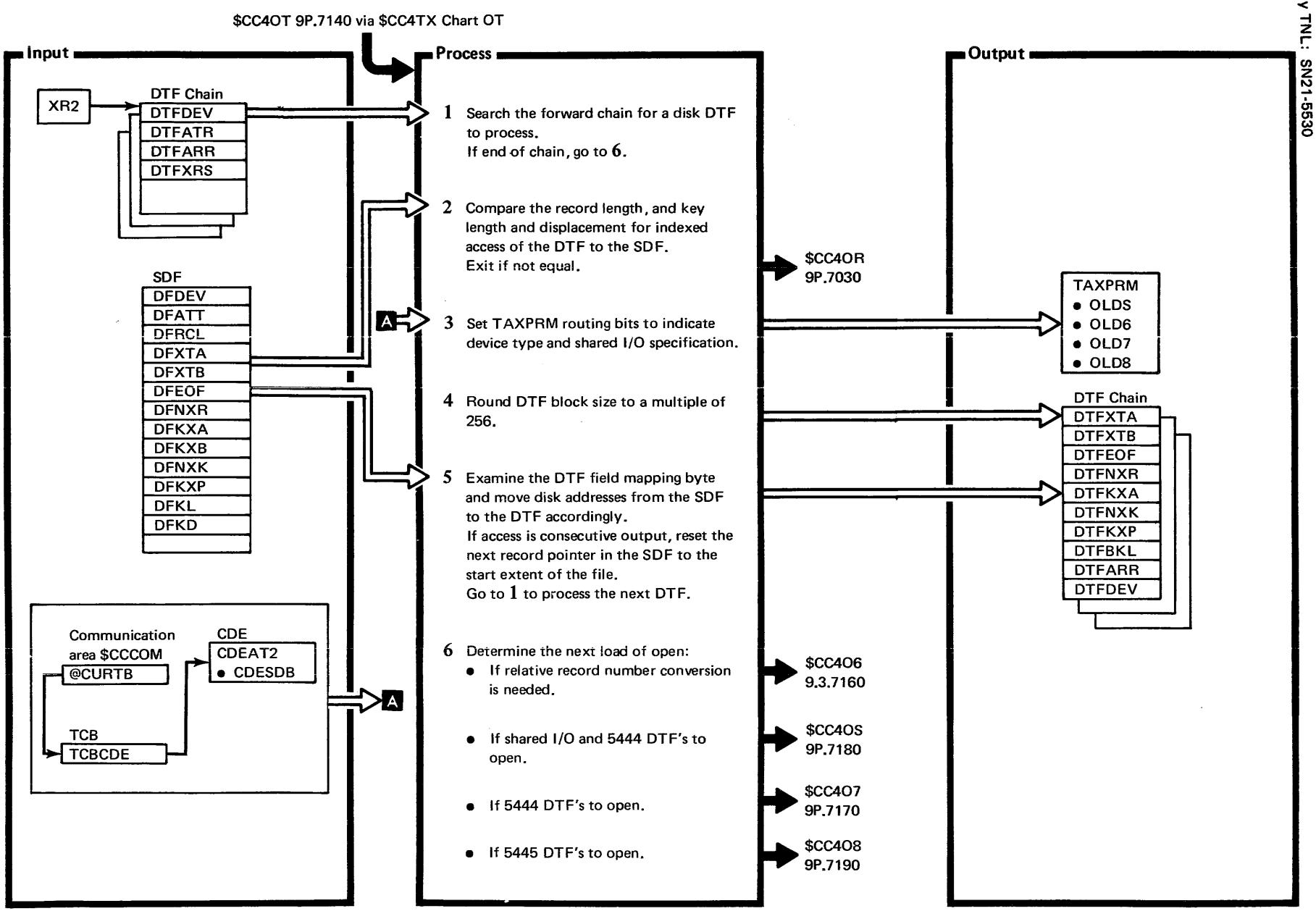


Diagram 9P.7130. $CC4OH



● **Diagram 9P.7140. $CC4OT**

Page of SY21-0531-2
Issued 24 June 1977
By TNL: SN21-5530



• **Diagram 9P.7150. $CC4O5**

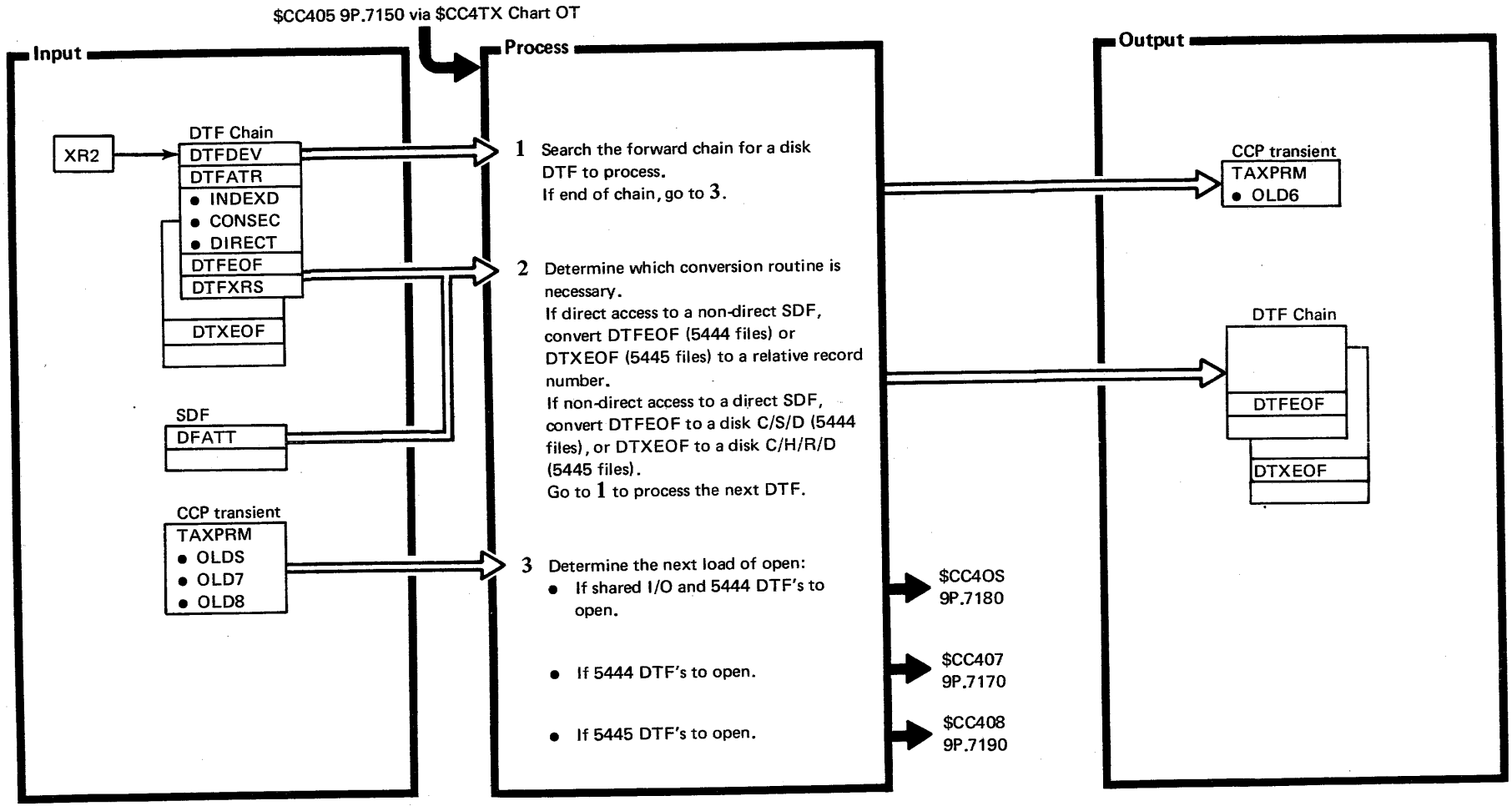


Diagram 9P.7160. $CC406

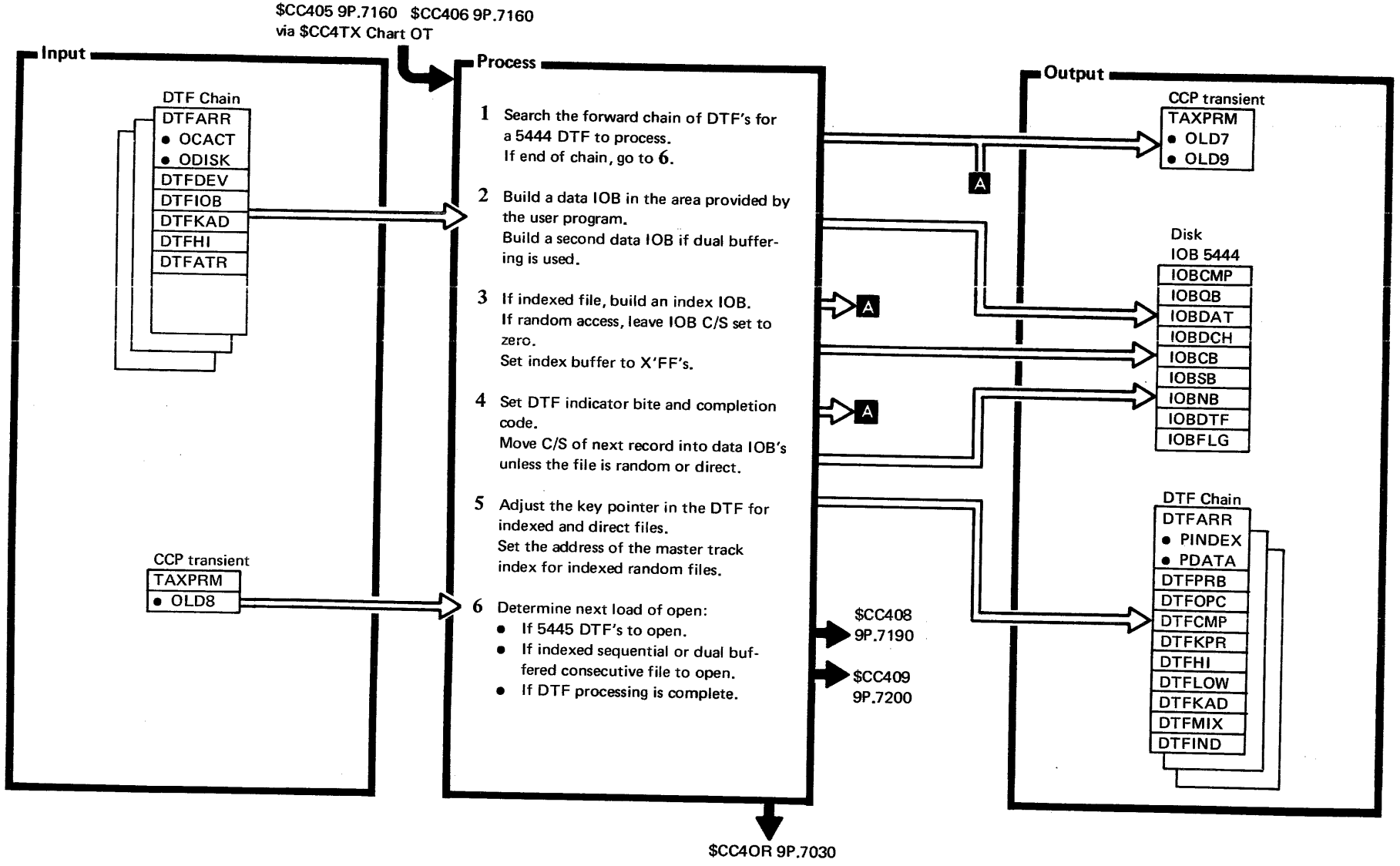


Diagram 9P.7170. $CC407

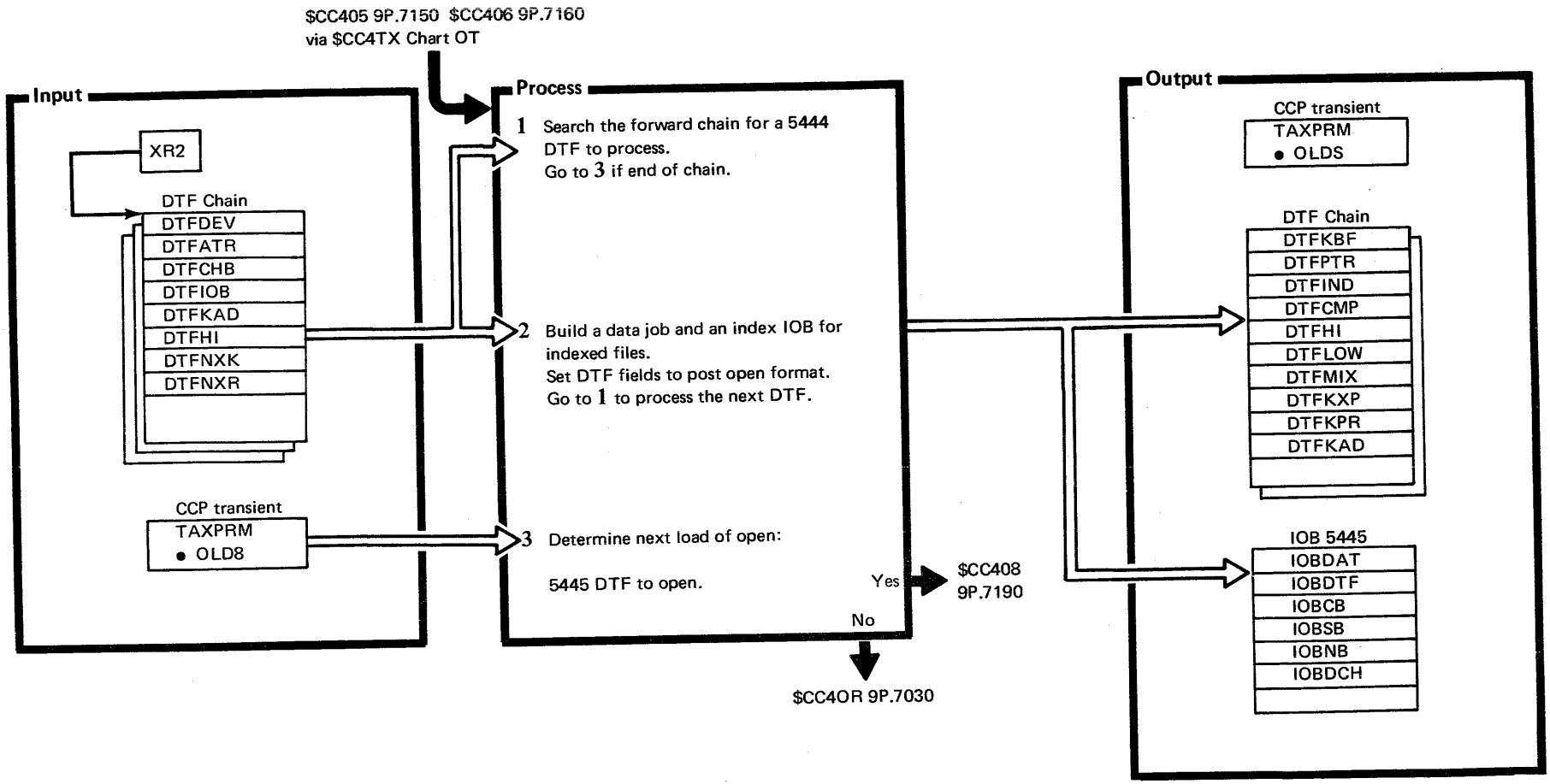


Diagram 9P.7180. $CC4OS

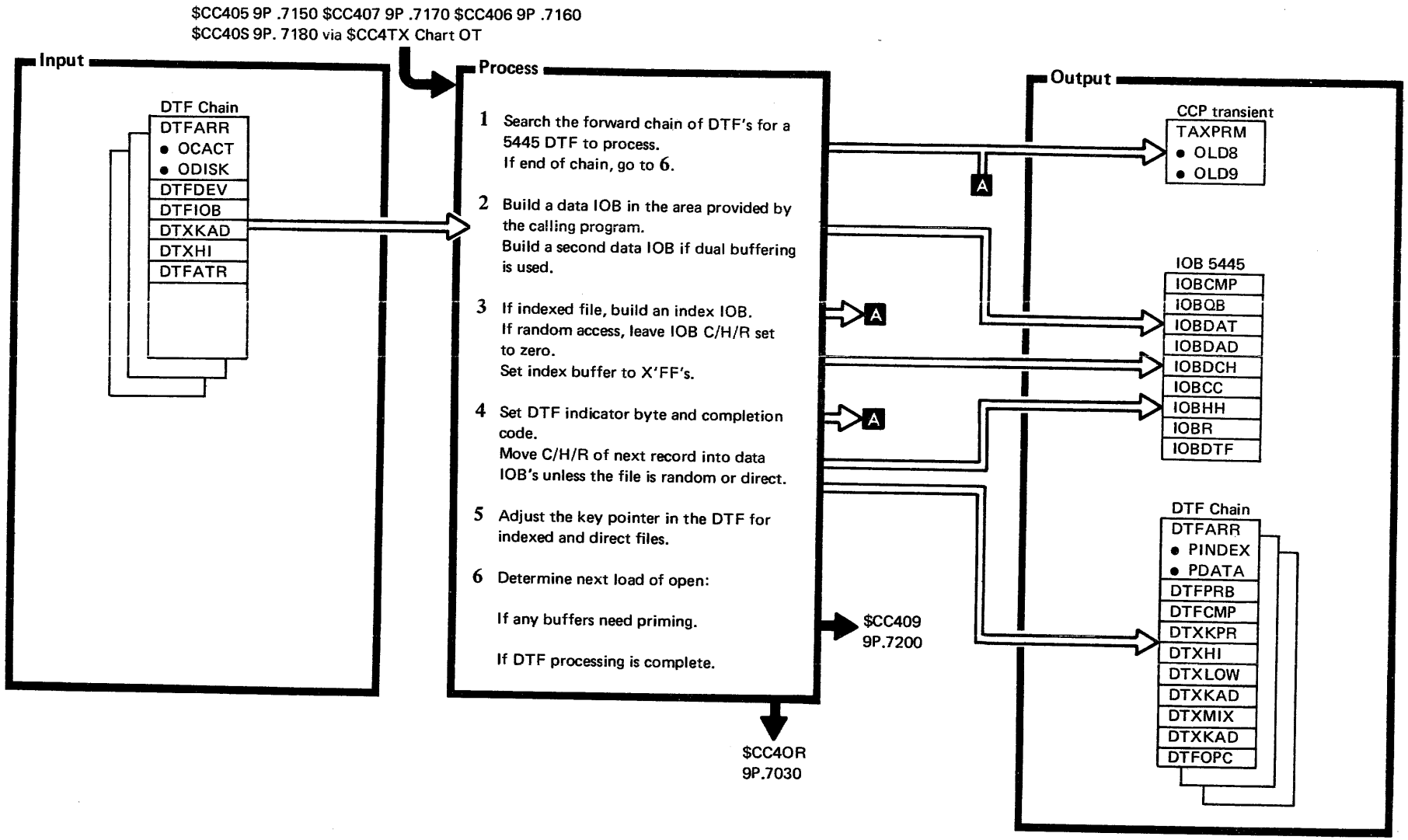


Diagram 9P.7190. $CC408 (Models 8, 10, and 12 Only)

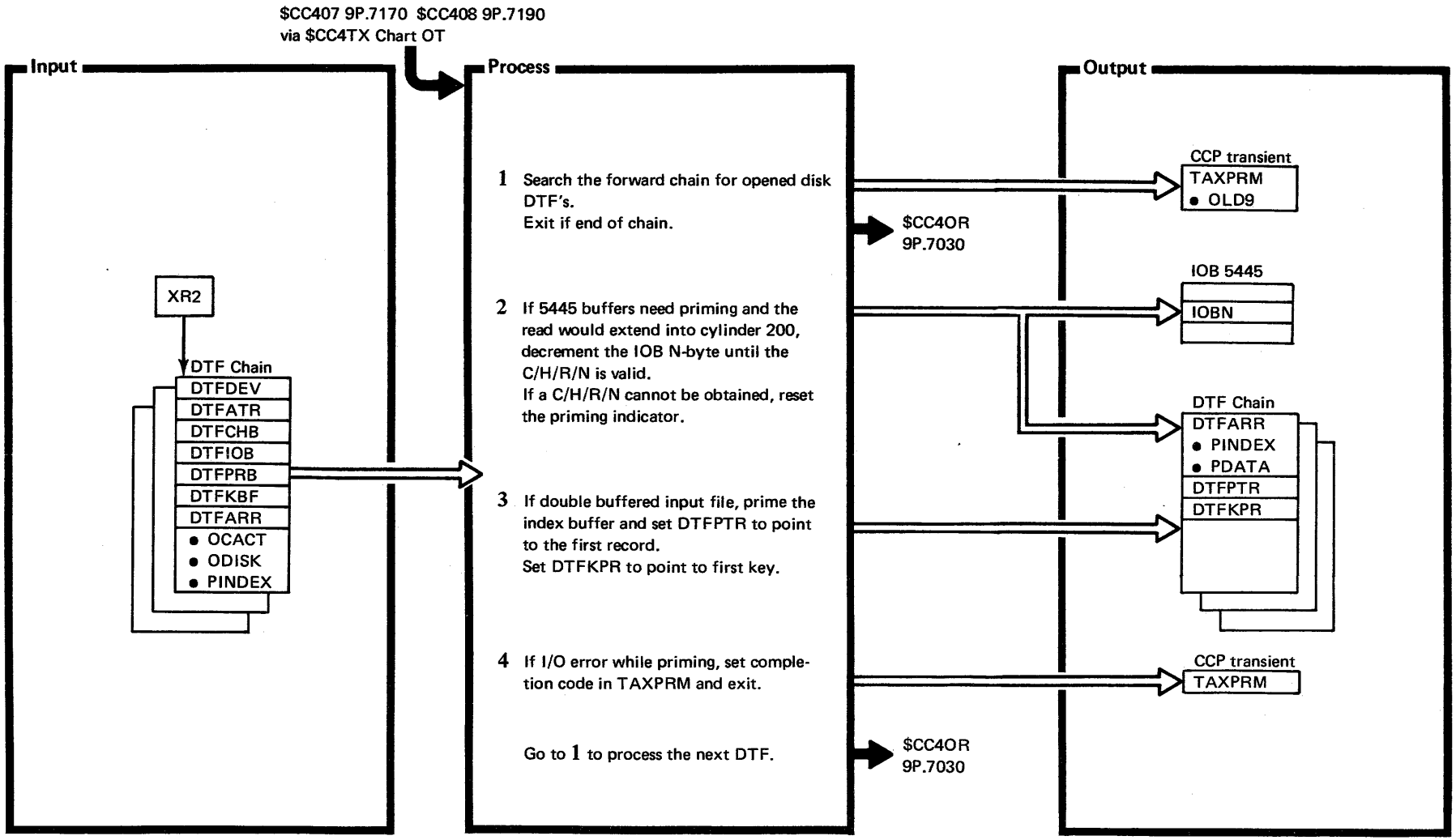


Diagram 9P.7200. $CC409

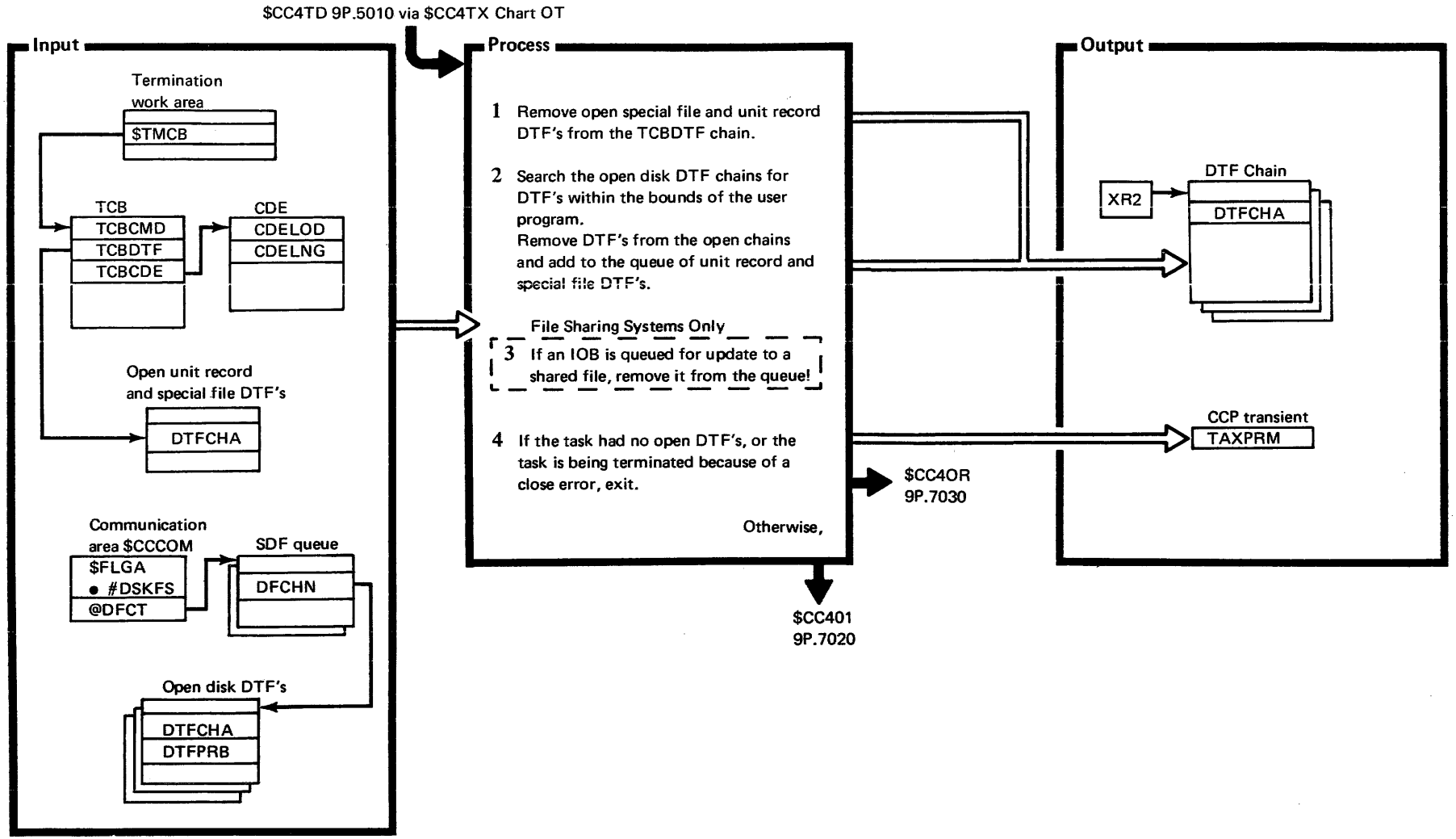


Diagram 9P.7300. $CC4LT

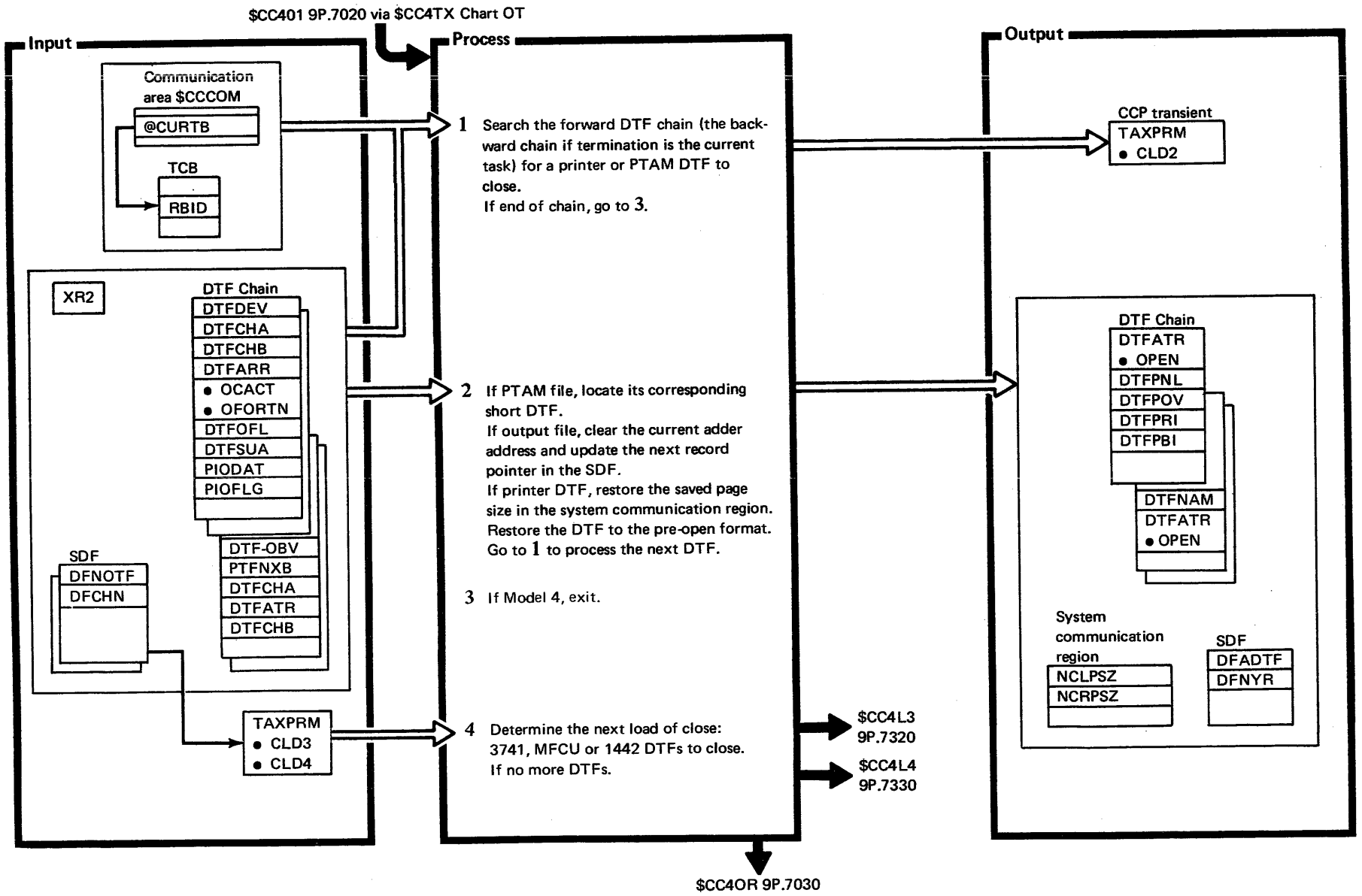


Diagram 9P.7310. $CC4L2

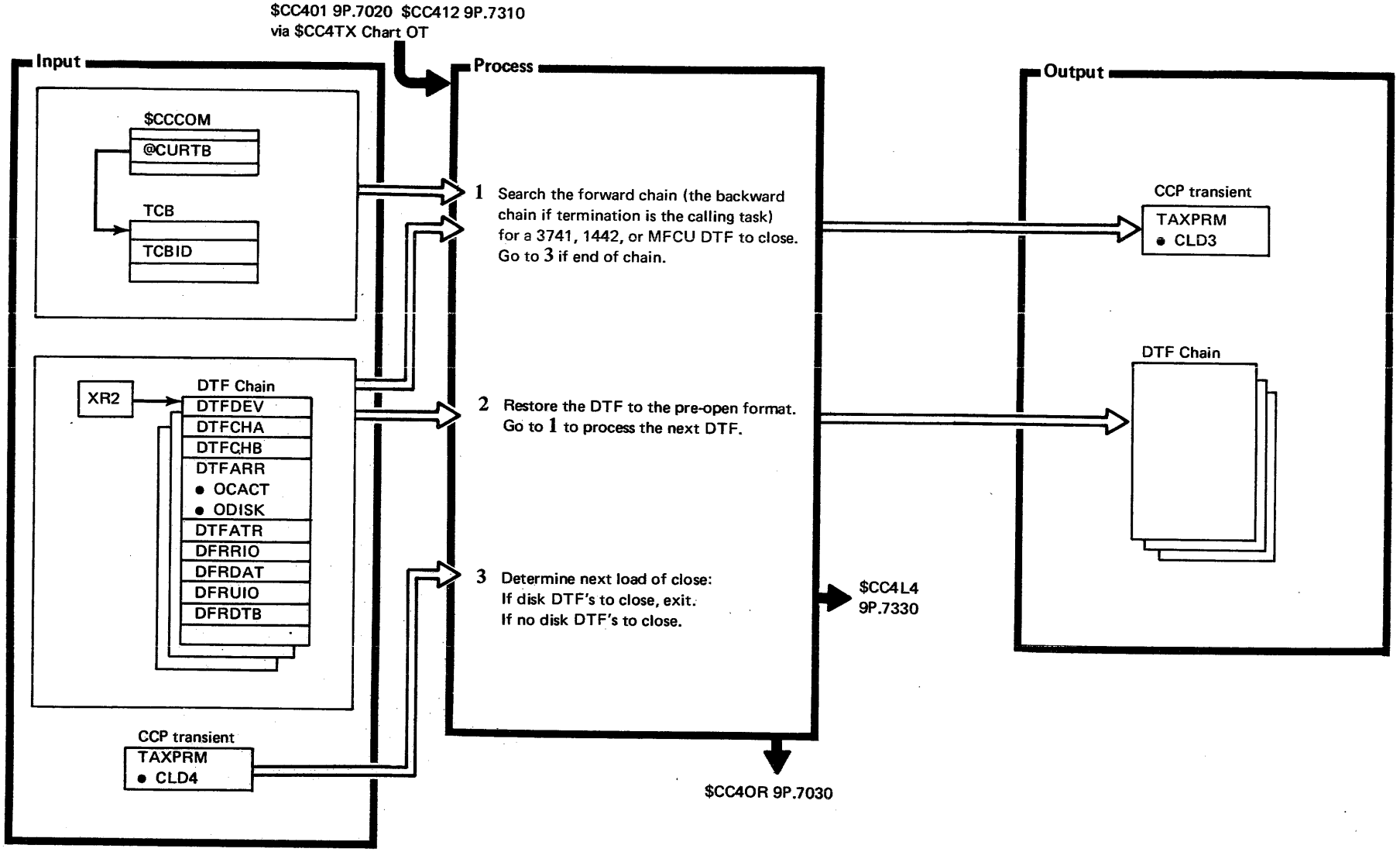


Diagram 9P.7320. $CC4L3

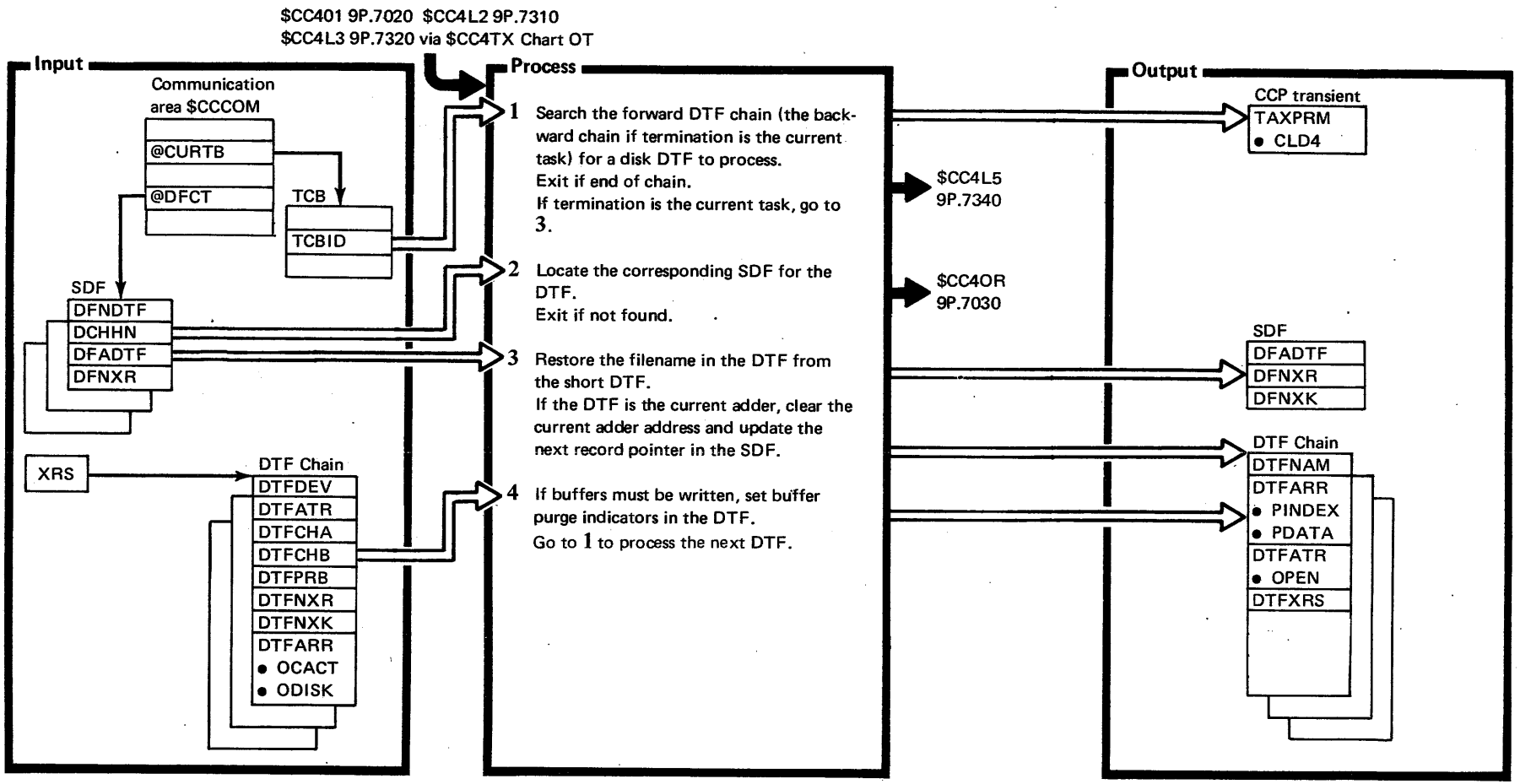


Diagram 9P.7330. $CC4L4

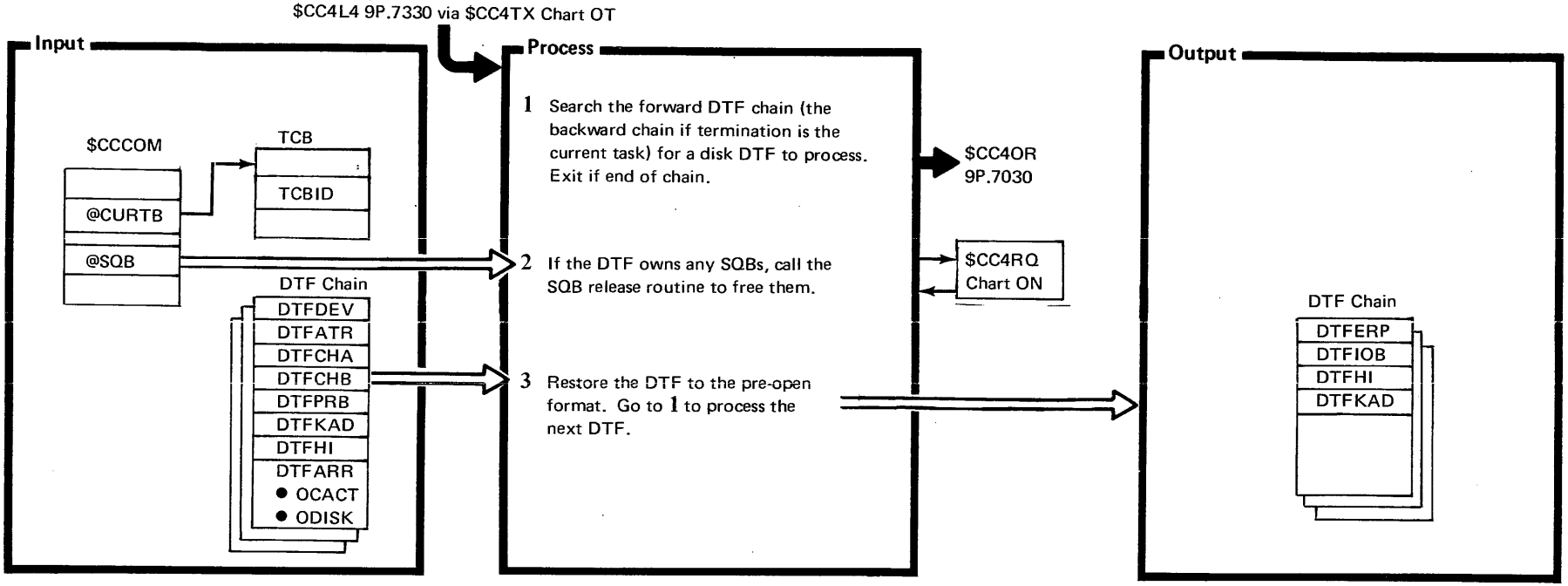
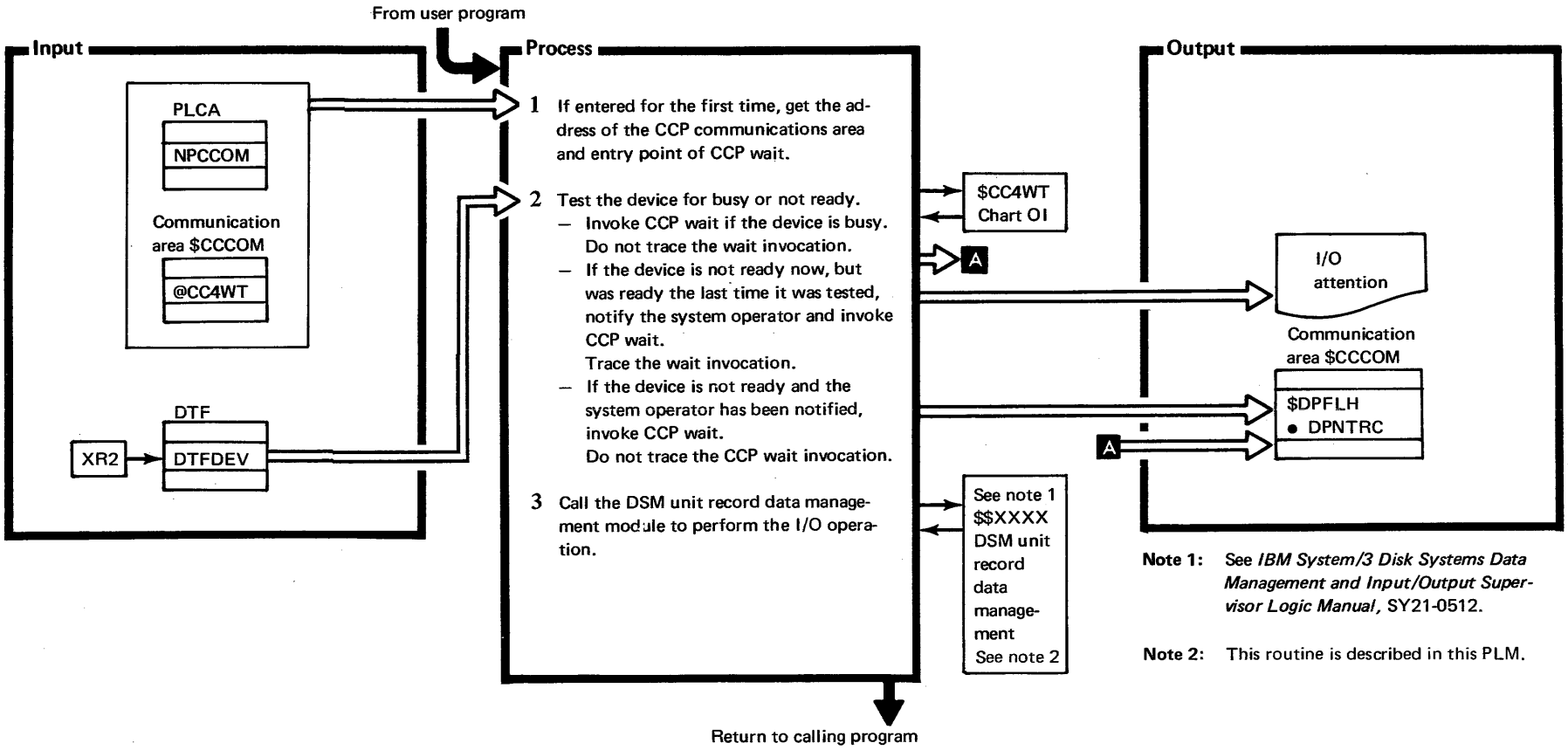


Diagram 9P.7340. $CC4L5



Note 1: See *IBM System/3 Disk Systems Data Management and Input/Output Supervisor Logic Manual*, SY21-0512.

Note 2: This routine is described in this PLM.

Diagram 9P.8010. $Nxxxx+$$Uxxx

Page of SY21-0531-2
Issued 24 June 1977
By TNL: SN21-5530

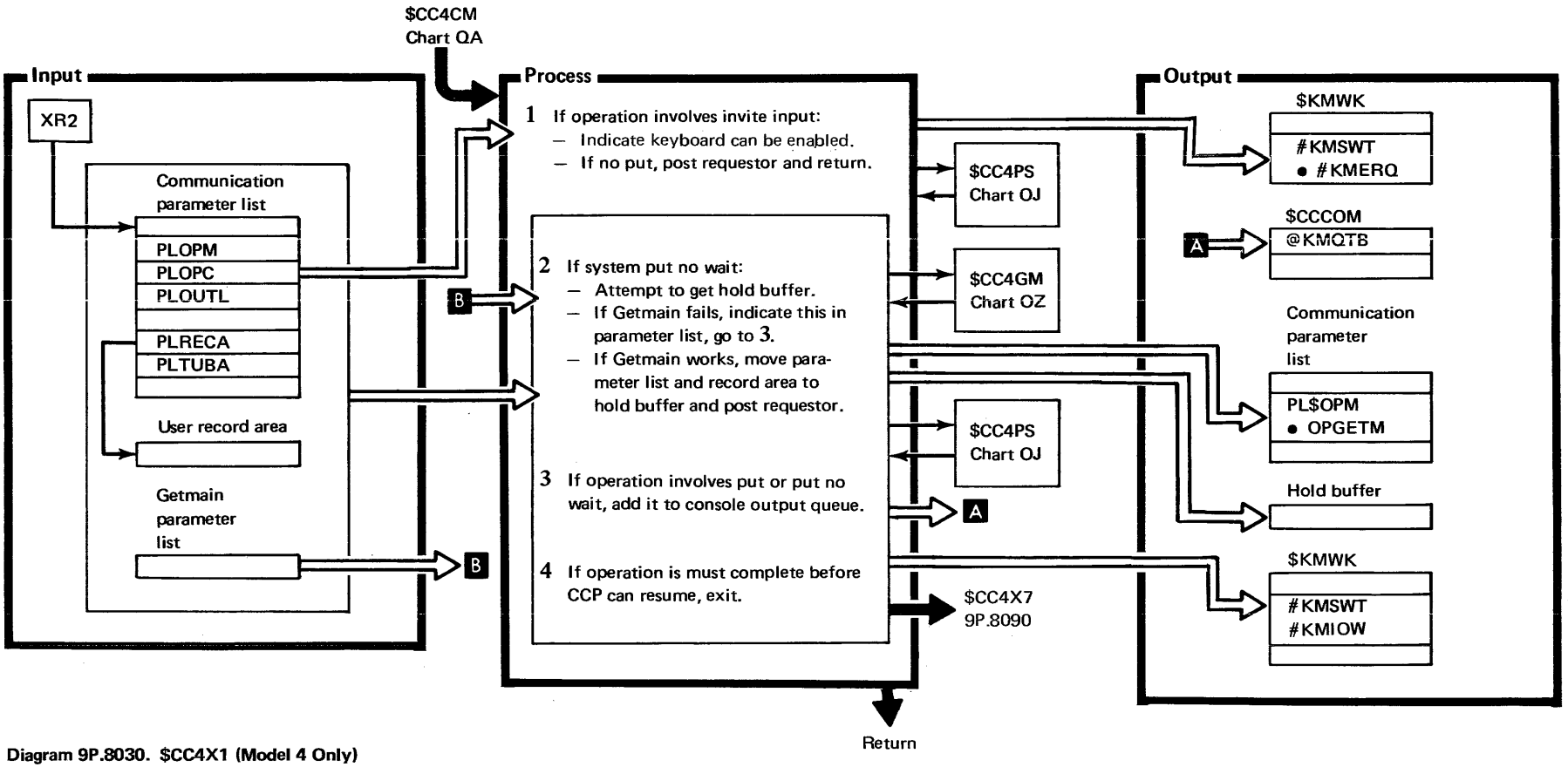


Diagram 9P.8030. $CC4X1 (Model 4 Only)

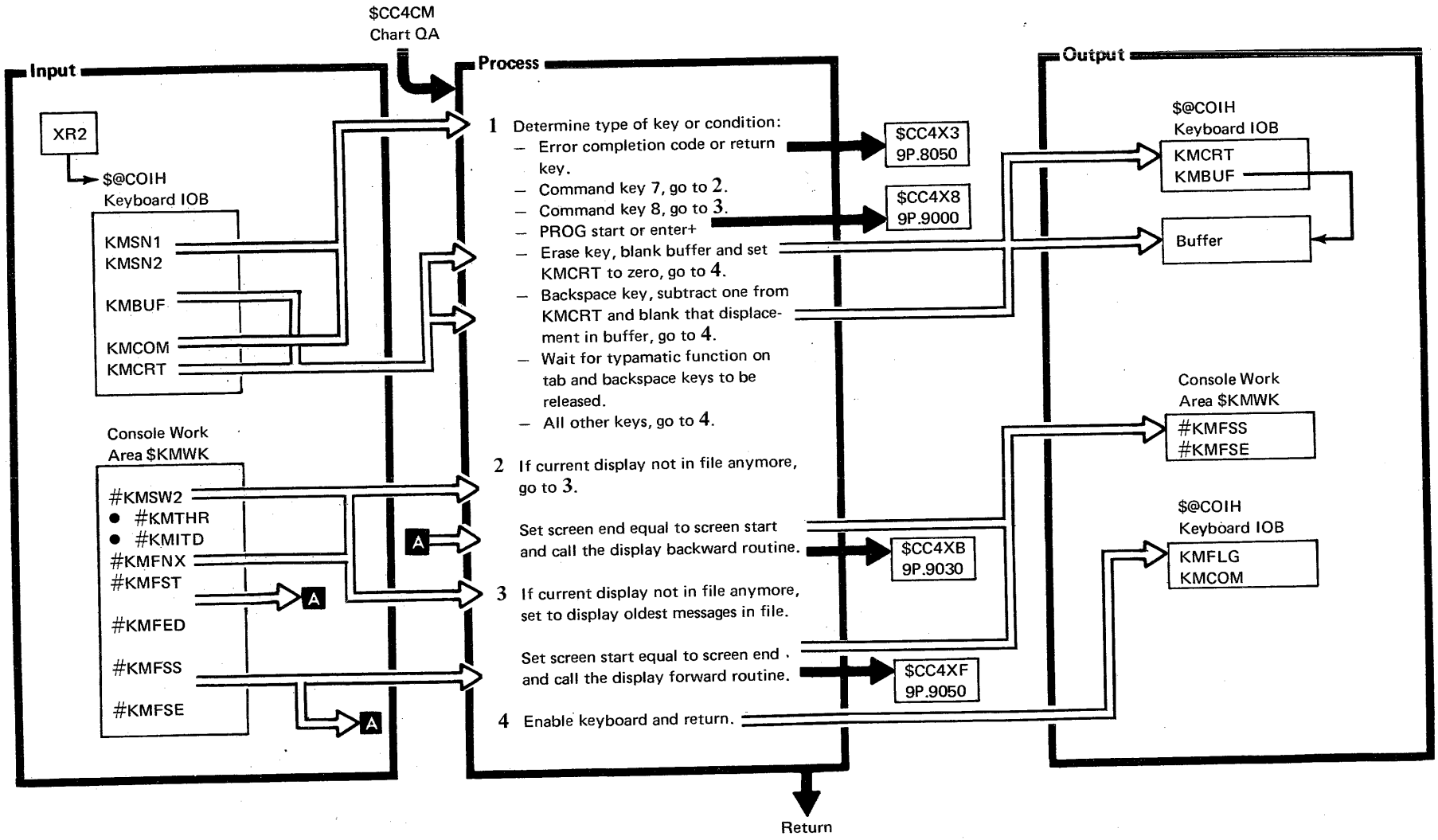


Diagram 9P.8040. $CC4X2 (Model 4 Only)

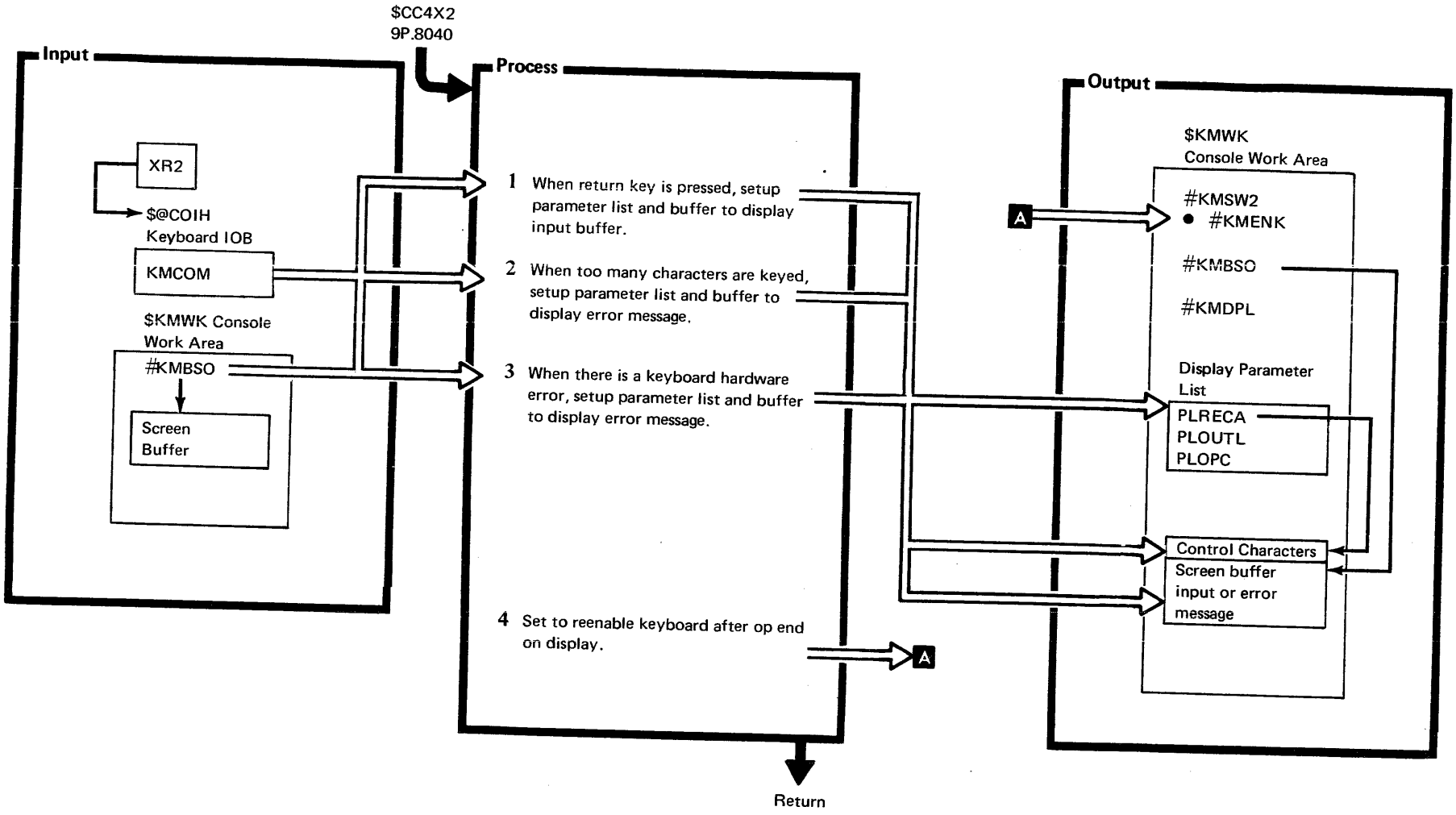


Diagram 9P.8050. $CC4X3 (Model 4 Only)

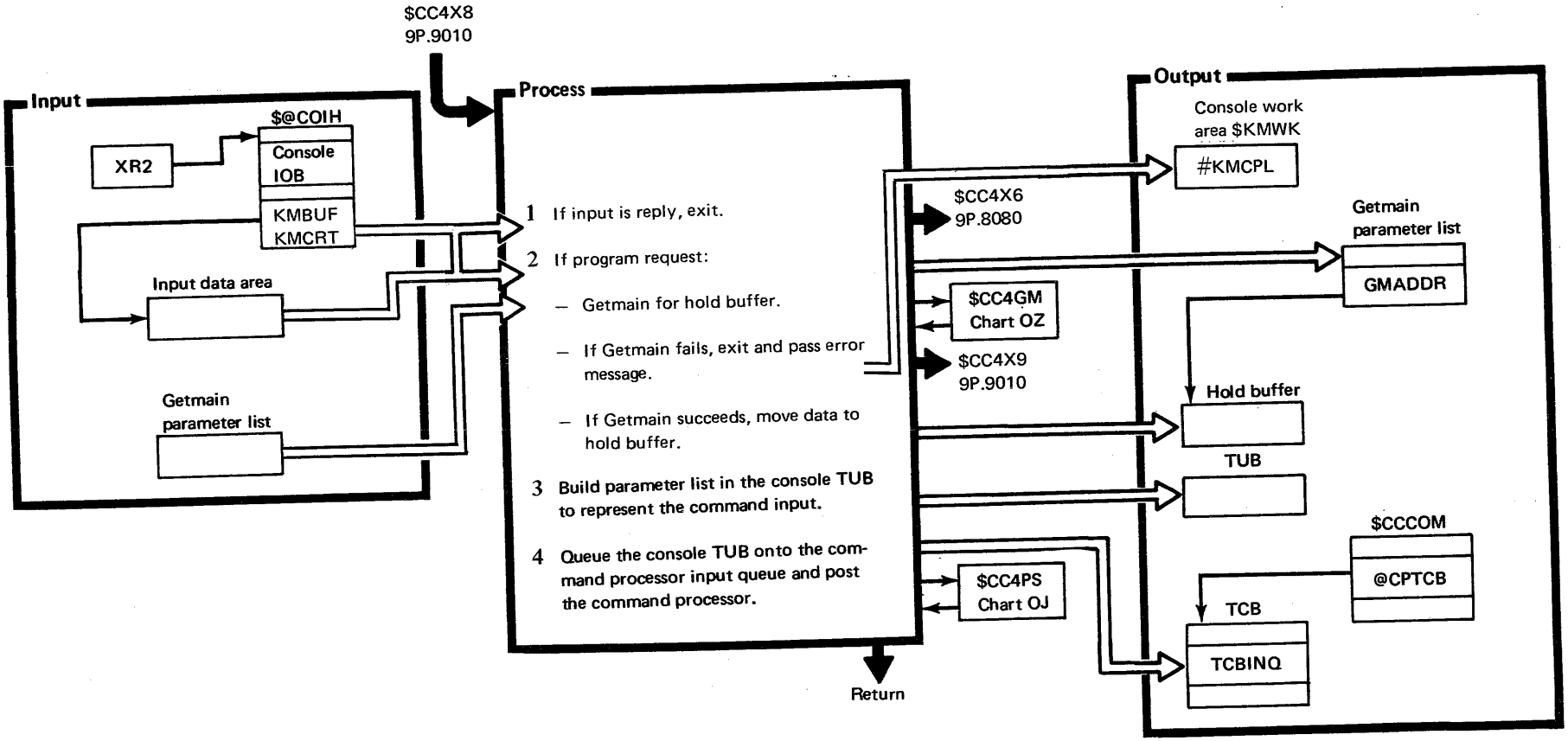


Diagram 9P.8060. $CC4X4 (Model 4 Only)

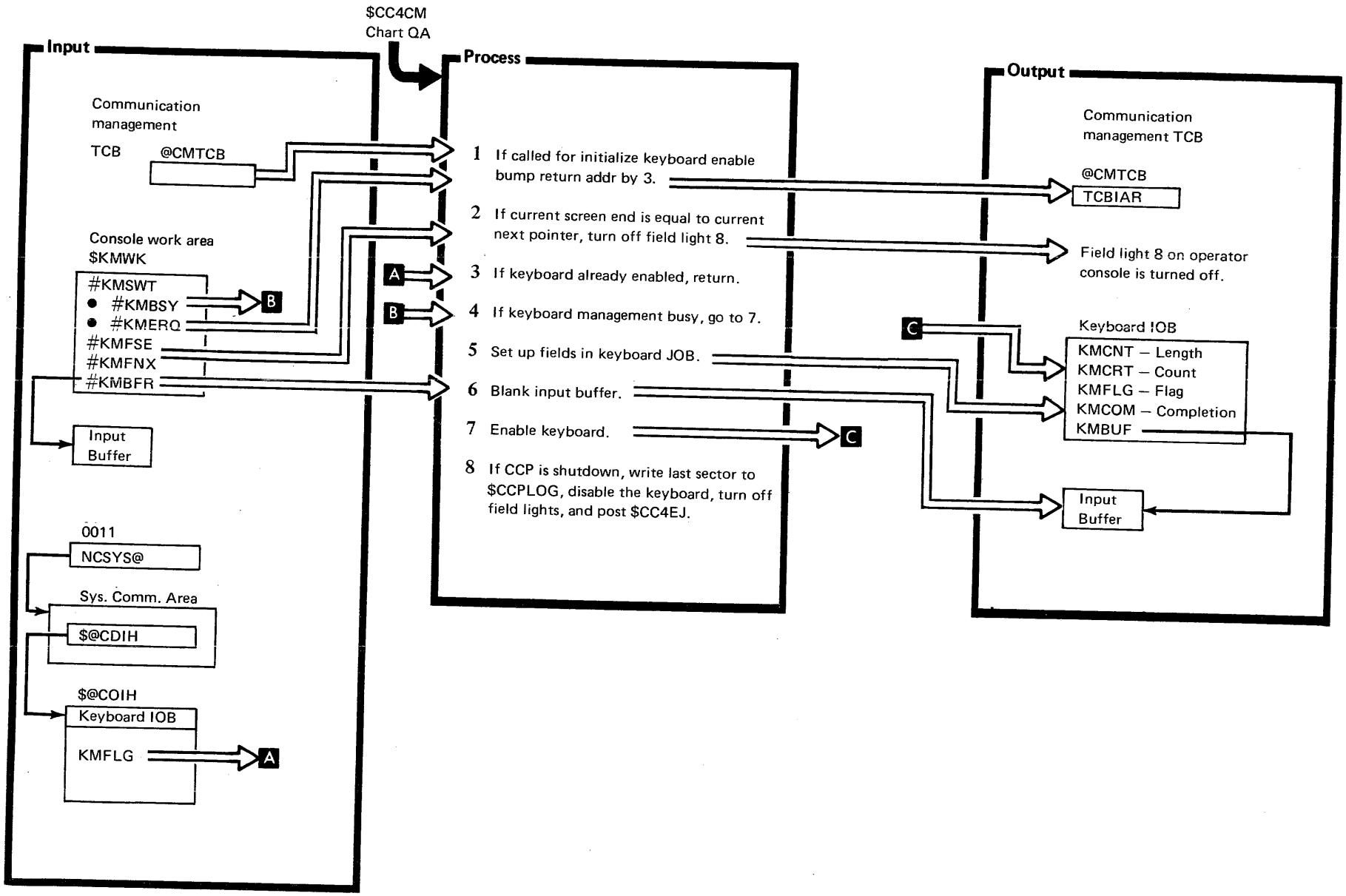


Diagram 9P.8070. $CC4X5 (Model 4 Only)

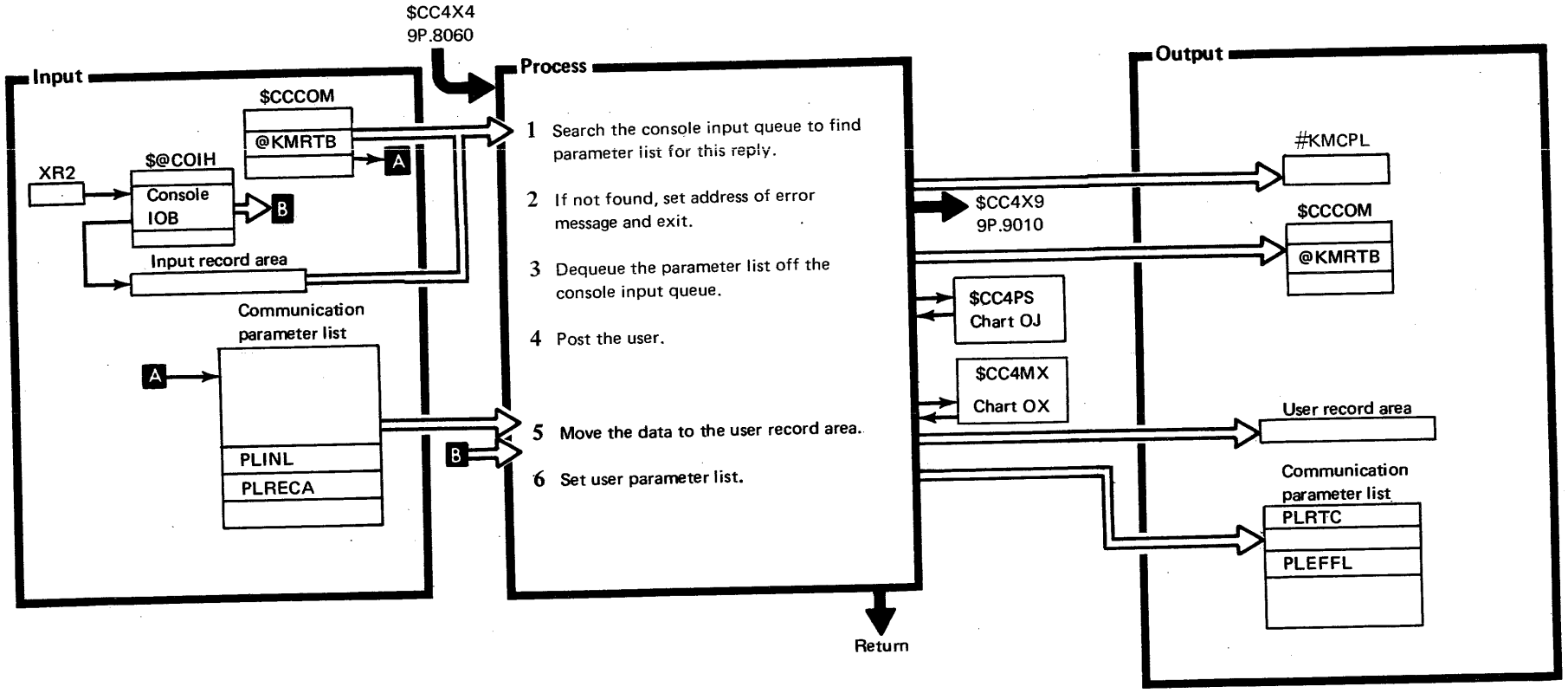


Diagram 9P.8080. $CC4X6 (Model 4 Only)

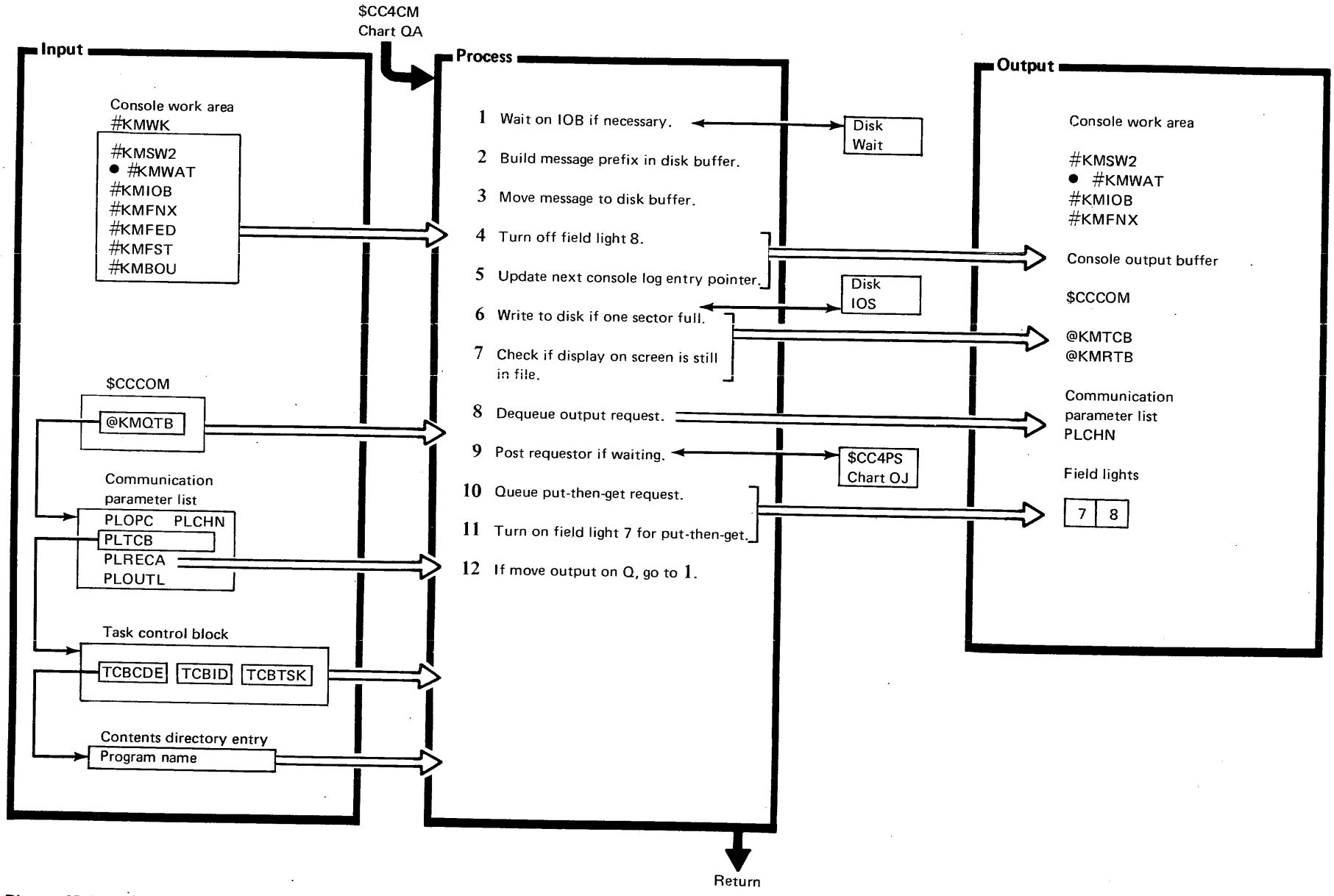


Diagram 9P.8090. $CC4X7 (Model 4 Only)

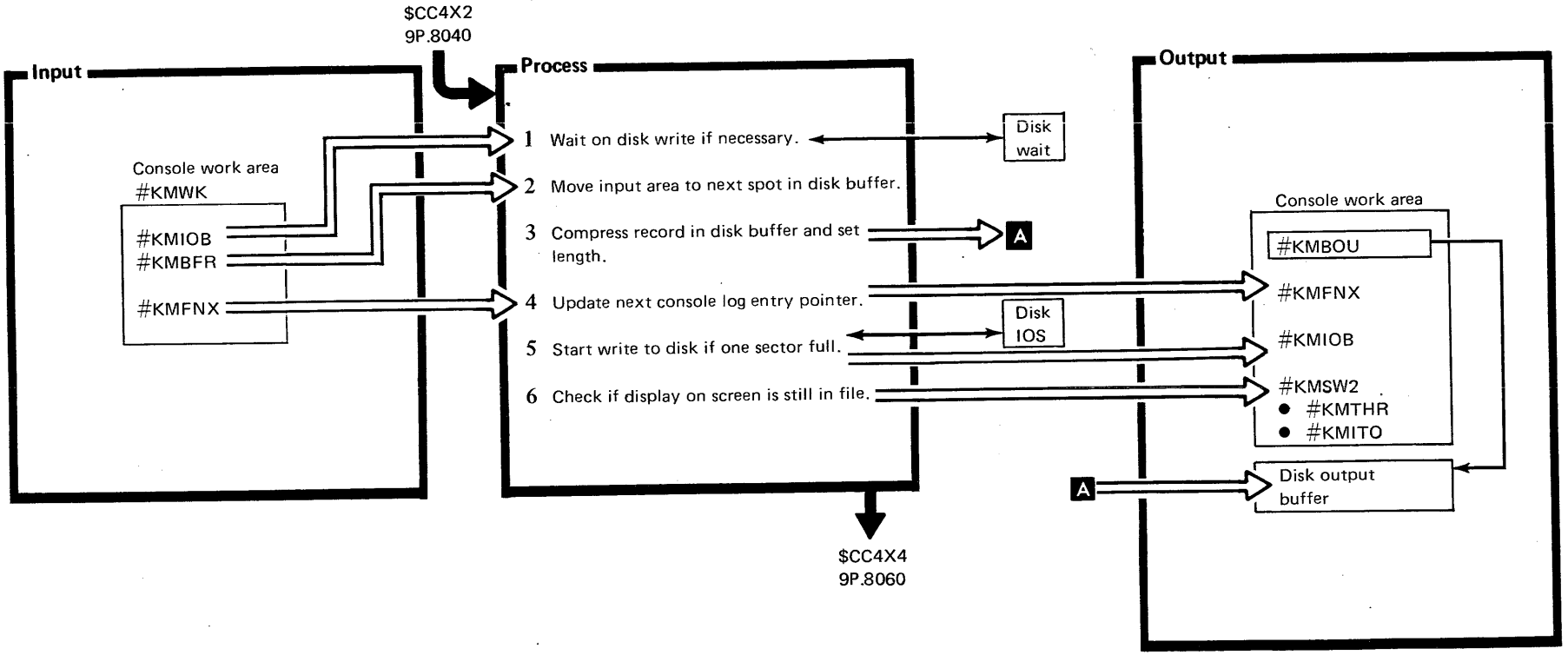
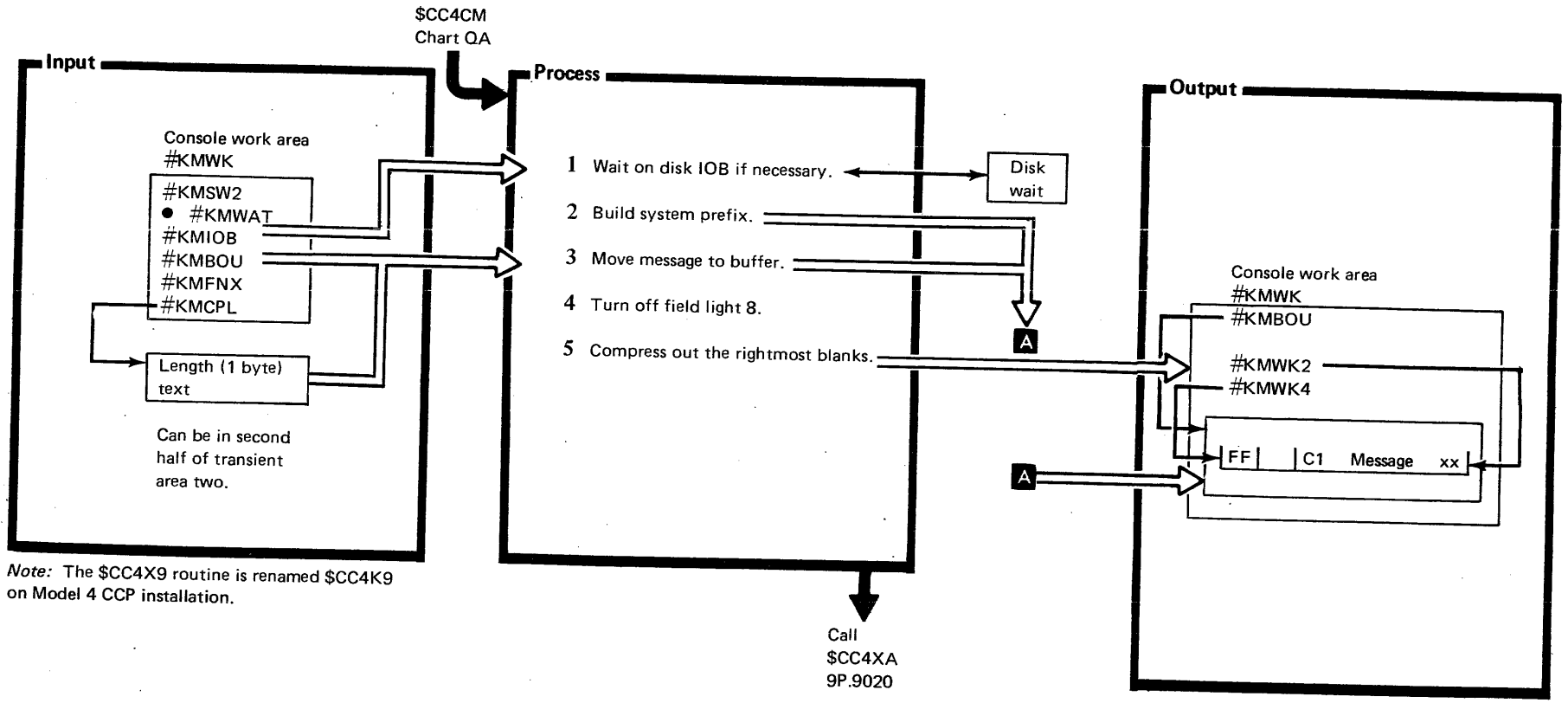


Diagram 9P.9000. $CC4X8 (Model 4 Only)



Note: The $CC4X9 routine is renamed $CC4K9 on Model 4 CCP installation.

Diagram 9P.9010. $CC4X9 (Model 4 Only)

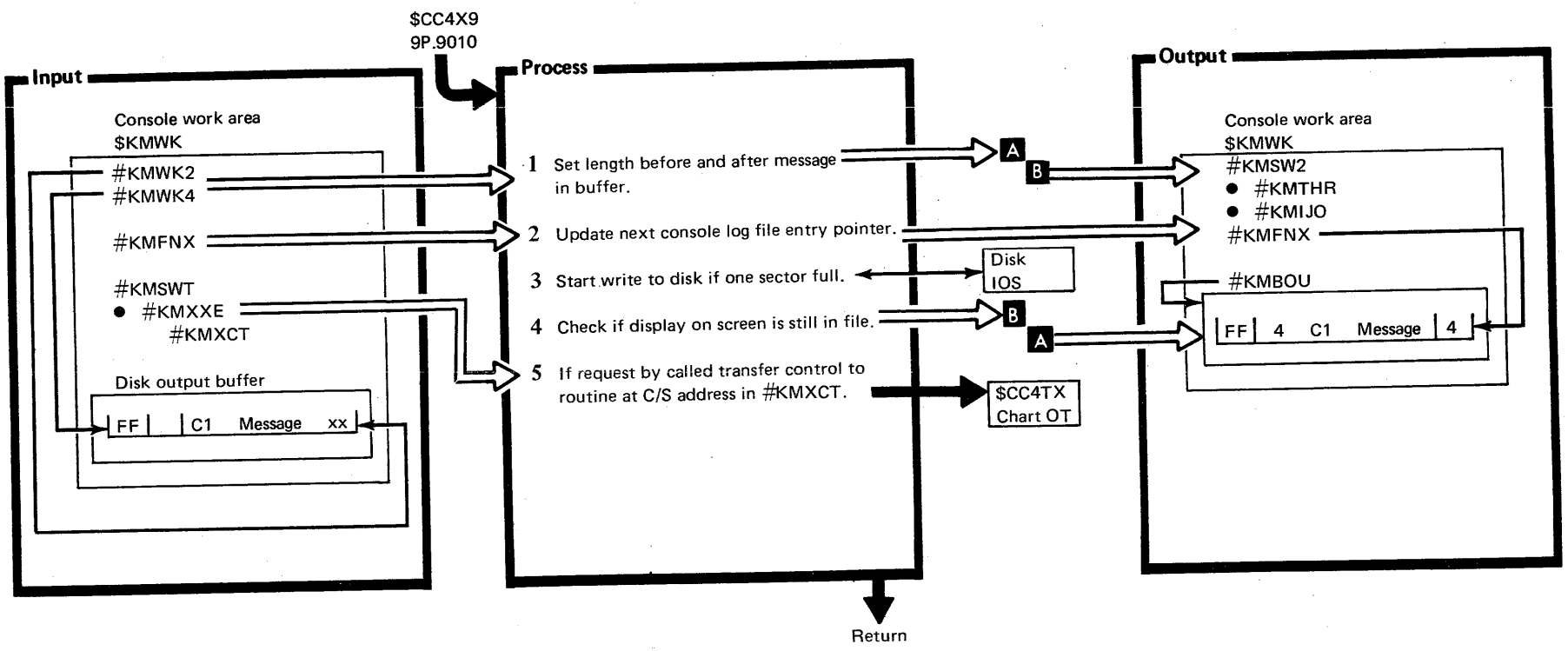


Diagram 9P.9020. $CCAXA (Model 4 Only)

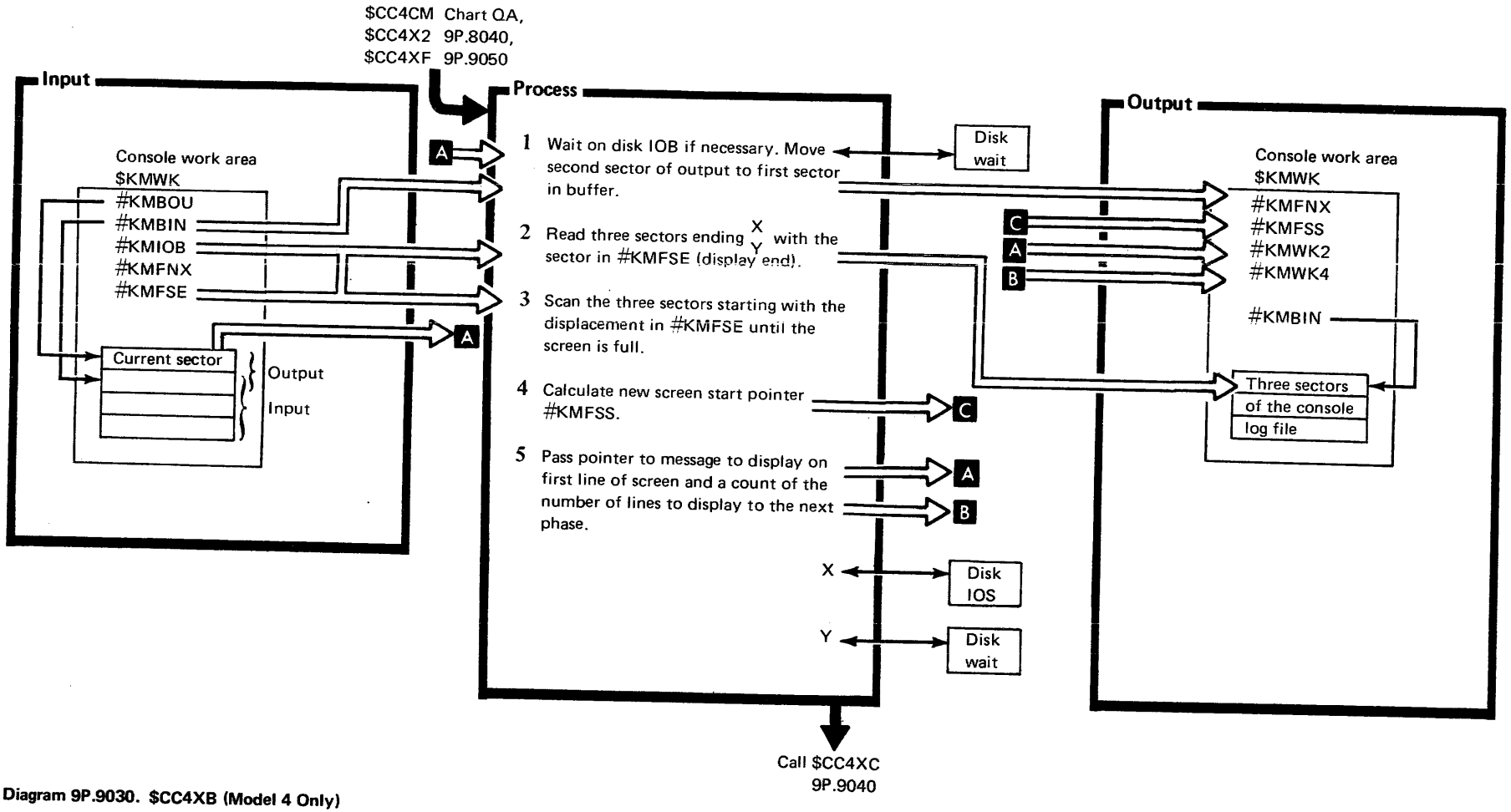


Diagram 9P.9030. $CC4XB (Model 4 Only)

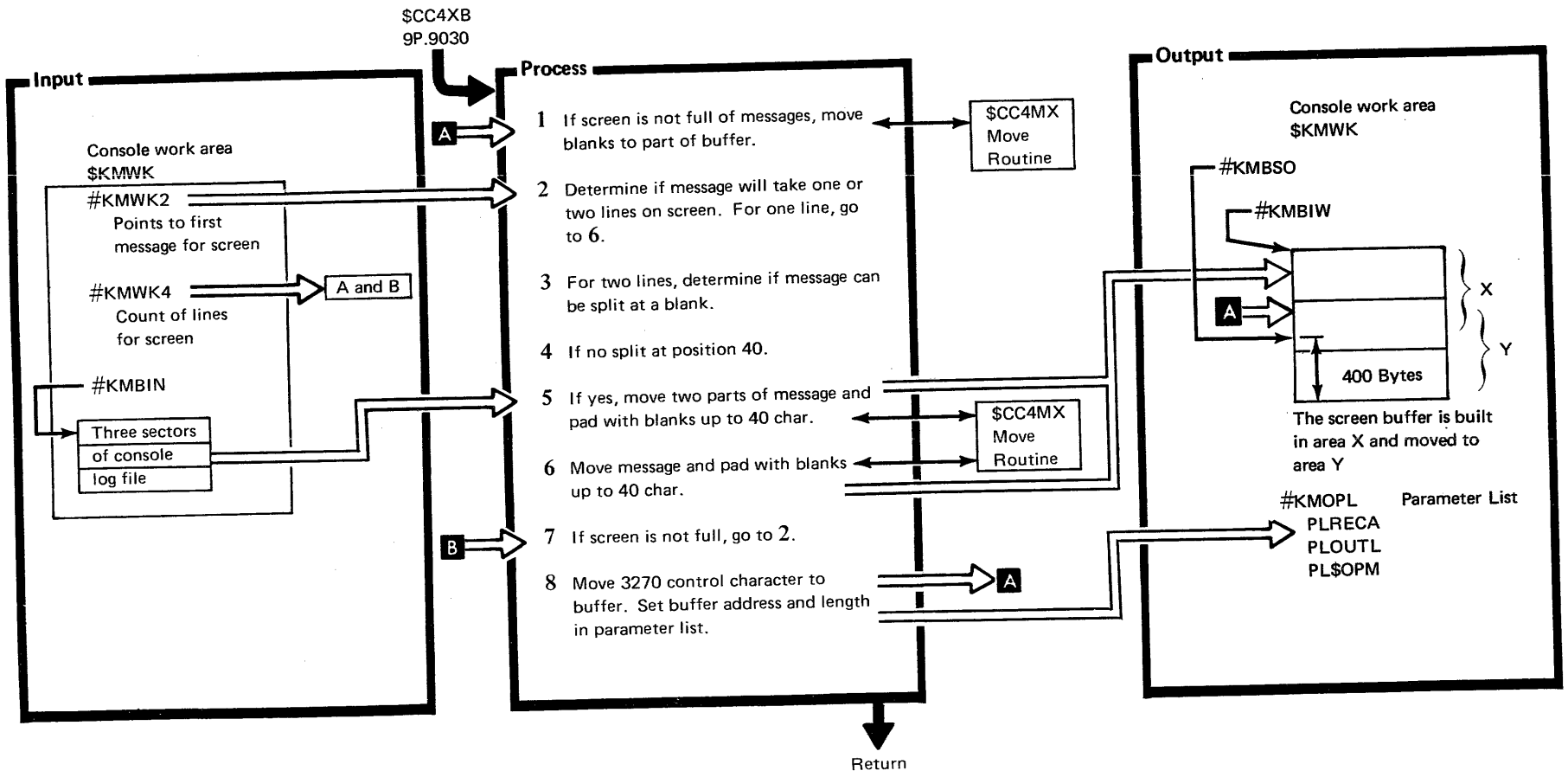


Diagram 9P.9040. $CC4XC (Model 4 Only)

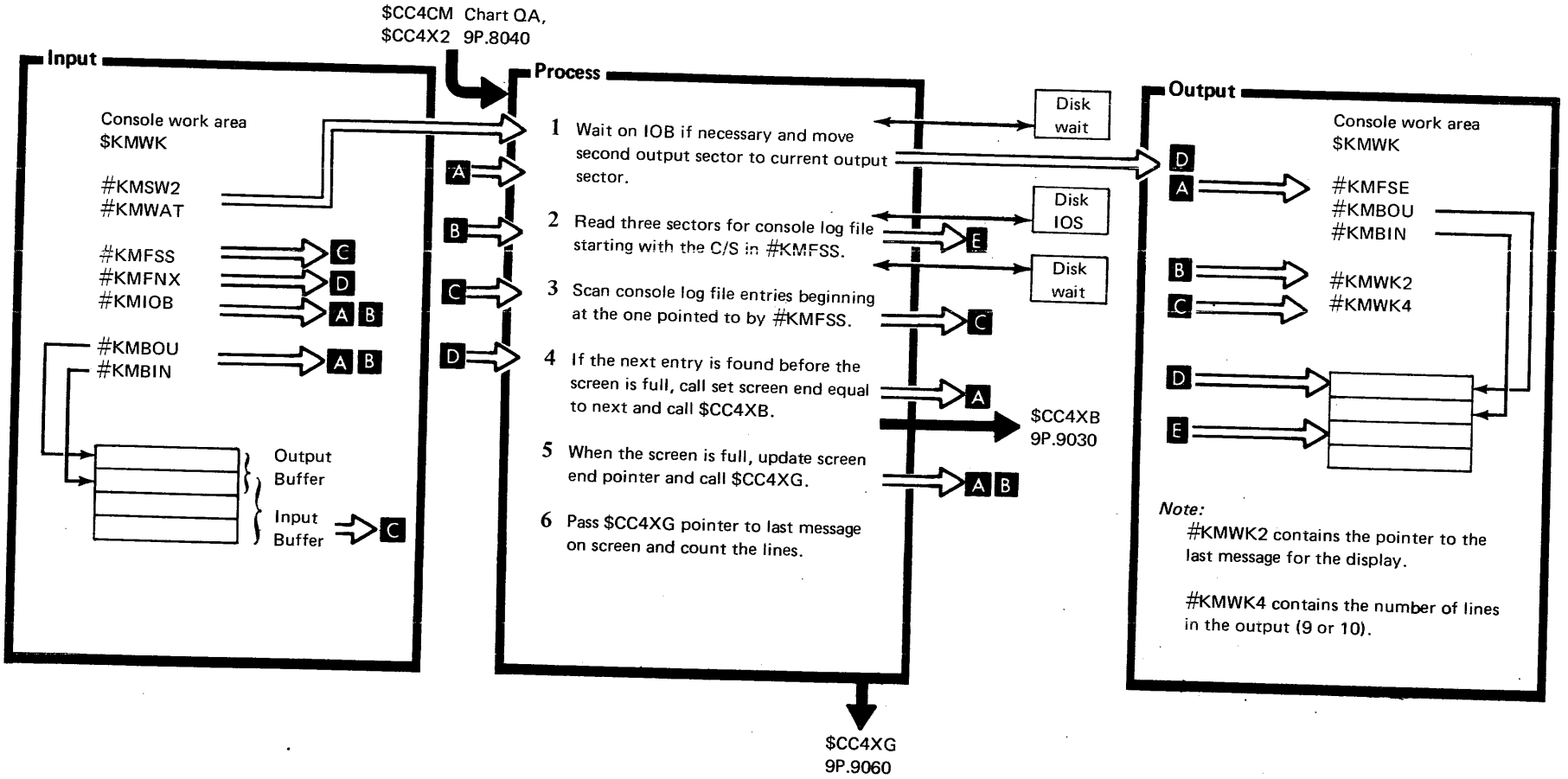


Diagram 9P.9050. $CC4XF (Model 4 Only)

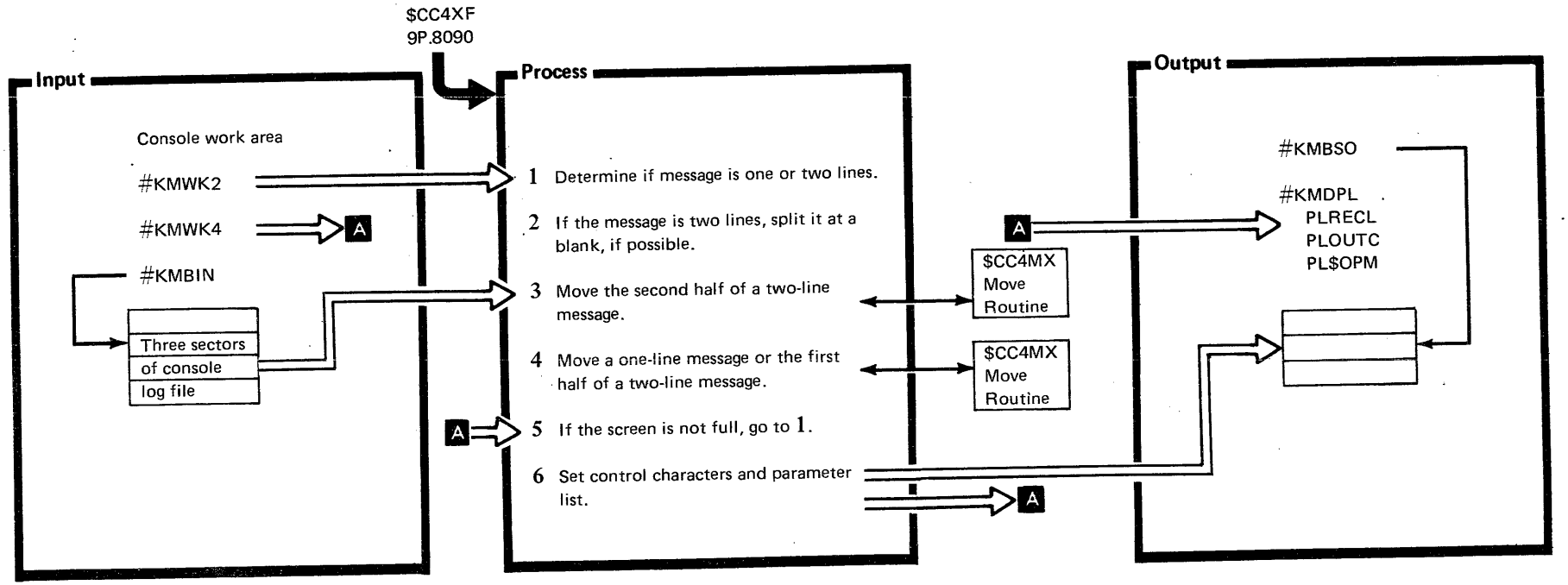


Diagram 9P.9060. $CC4XG (Model 4 Only)

Introduction

CCP Shutdown allows the system operator to terminate CCP and return control to Disk System Management. CCP Shutdown is invoked via either the Shutdown command, which allows all currently executing or queued user programs to complete, or the Cancel CCP command, which immediately terminates all user programs. The Shutdown module, $CC5SH, is loaded from the object library, relocated by transient $CC4EJ, and executed in the user program area. For information on how all programs that are running are informed that Shutdown has been requested, see index entries *CCP End Of Job* and *Termination.*

The system requirements for Shutdown are the same as for Startup and CCP Execution:

- 5404 Processing Unit (64K of MOSFET memory)
  or
  5410 Model A15 Processing Unit (24K of main storage)—Model 10
  or
  5412 Model B17 Processing Unit (48K FET memory)—Model 12
- 5444 Model 2 Disk Storage Drive—Model 10
  or
  5447 Model 1 Disk Storage Drive—Model 4
  or
  3340 Model C2 Direct Access Storage Facility—Model 12
- 5471 Printer-Keyboard
- 5203 or 5213 or 1403 Printer
- Teleprocessing line (BSCA, MLTA, or local display adapter)

Method of Operation

Shutdown (Figure 10-1 and Diagram 10M.0100), which runs under control of the Termination Task, changes the ID in Termination's TCB to the character S, indicating in the CCP trace table that Shutdown is now executing. Since certain parts of the Command Processor's TCB are used as work areas by $CC5SH, the TCB chain field (TCBNXT) is set to zero to assure that the Command Processor is not given control by the Dispatcher.

Build and Close Disk DTFs

Shutdown closes all disk files by building a normal disk DTF from each of CCP's short DTFs. A DTF is built by setting the file attributes, device code, and record length from the short DTF into an 85-byte area set aside to contain the full DTF. The DTF also contains an IOB address, which points to a 52-byte area, and a data area address, which points to a 512-byte area. CCP File Open is now called twice to build the DTF and IOB in a format suitable for DSM Close. If the open was not successful, error message 507 is printed and Shutdown proceeds to close the next disk file. If Open was successful, the file is closed by calling true DSM Close via General Entry with a RIB of 83. The true General Entry address is obtained from an area within $CC4IG, General Entry Intercept. This action occurs for each short DTF in the system.

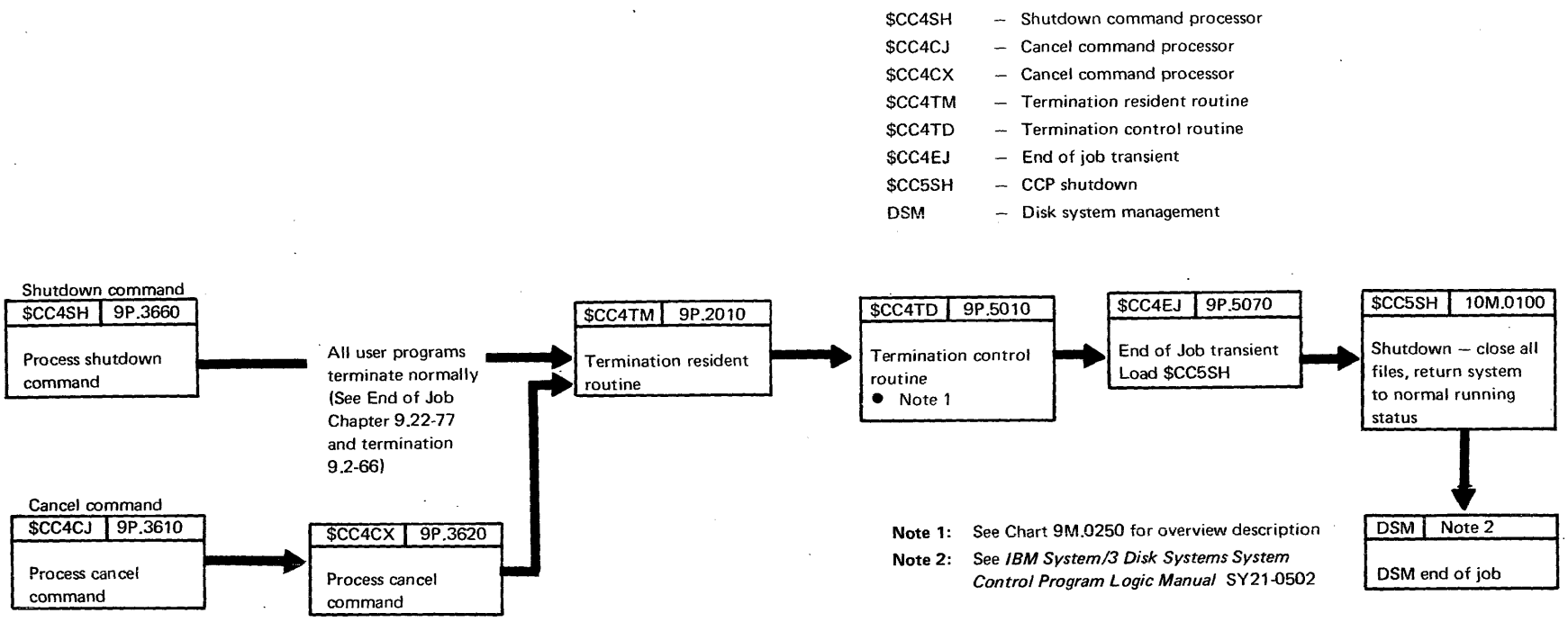
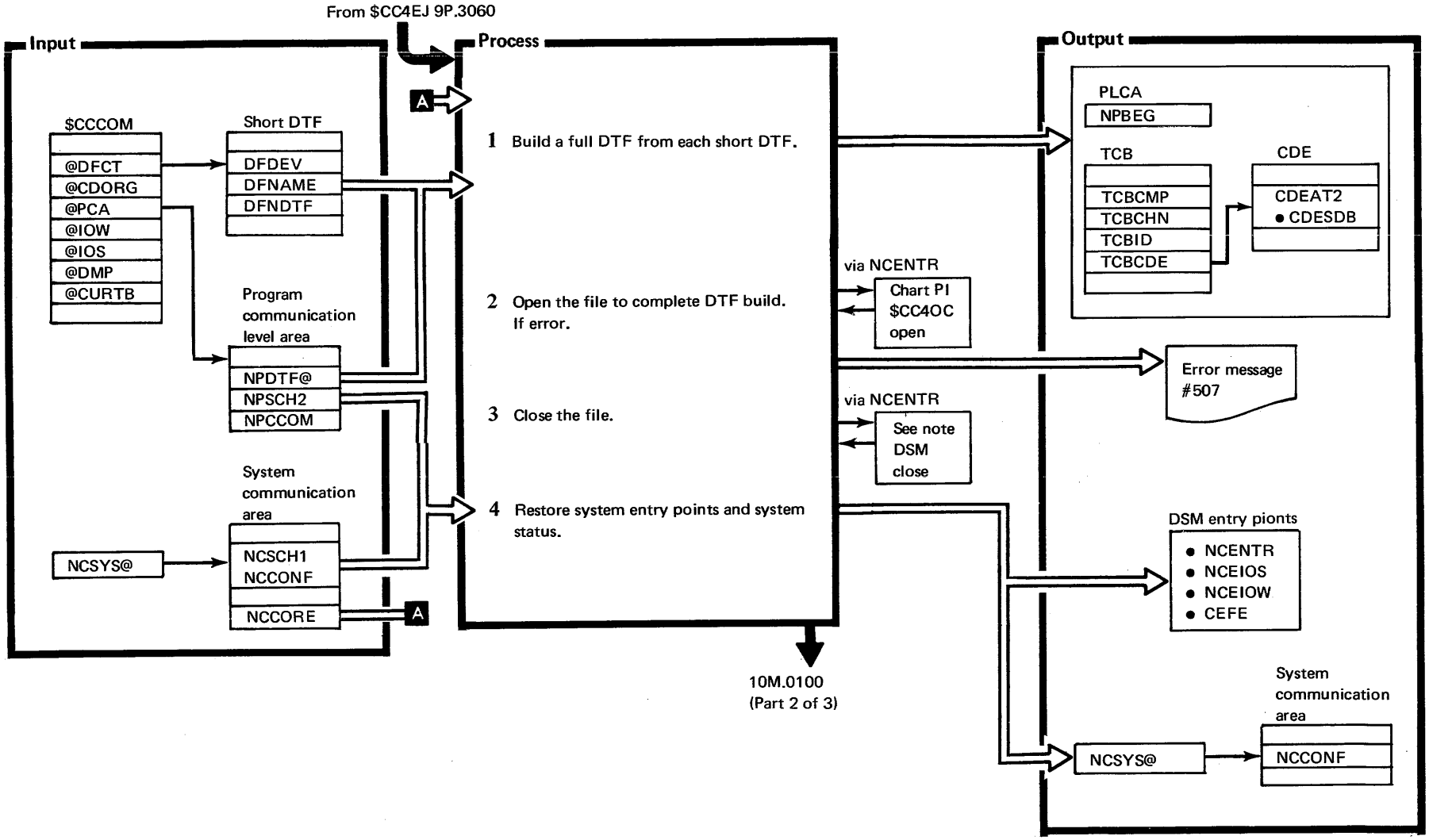


Figure 10-1. Control Flow of Shutdown

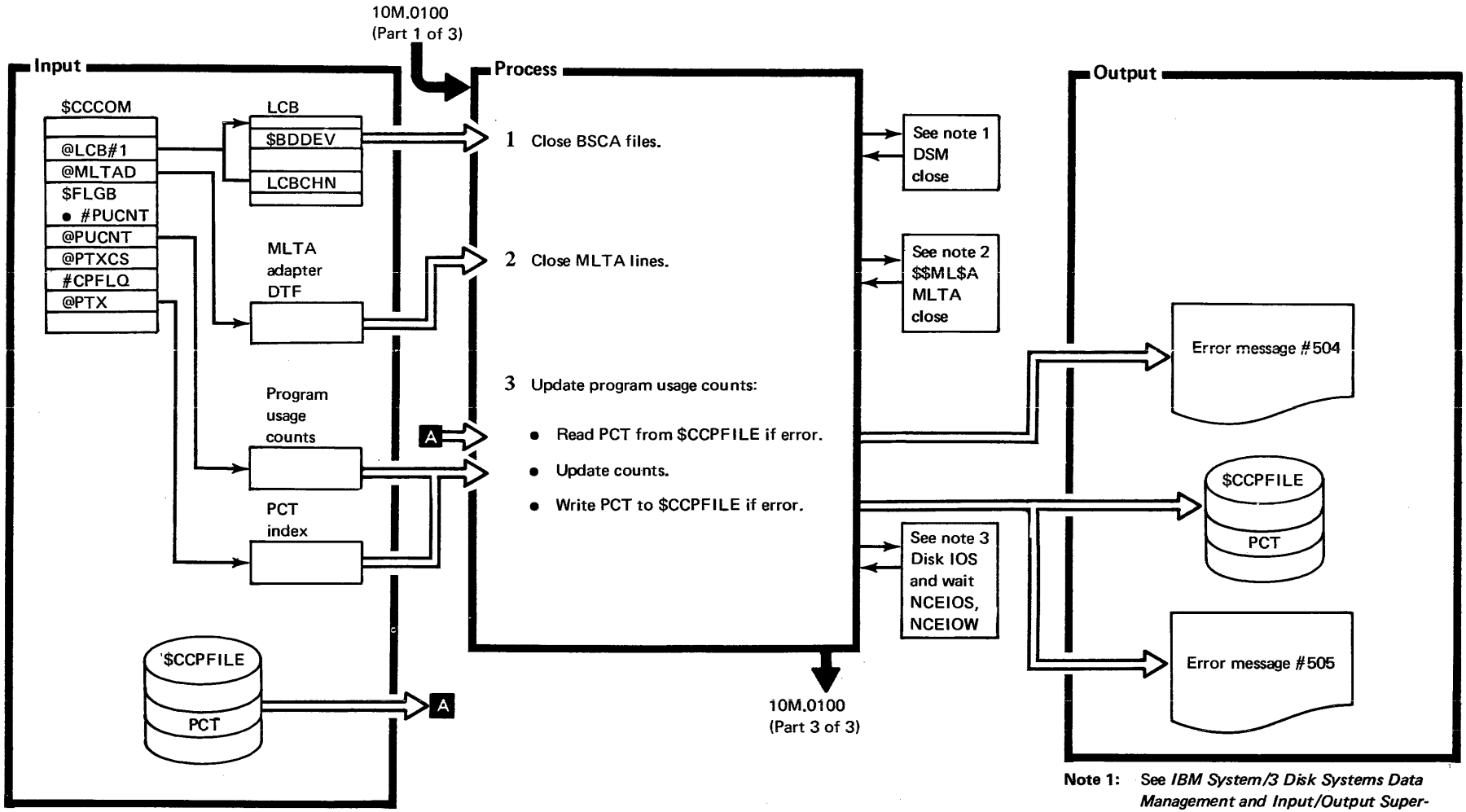


Note: See *IBM System/3 Models 4, 6, 8, and 10 Disk Systems System Control Program Logic Manual*, SY21-0502.

● **Diagram 10M.0100 (Part 1 of 3). Overall Flow of Shutdown**

Page of SY21-0531-2
Issued 24 June 1977
By TNL: SN21-5530

Page of SY21-0531-2
Issued 24 June 1977
By TNL: SN21-5530

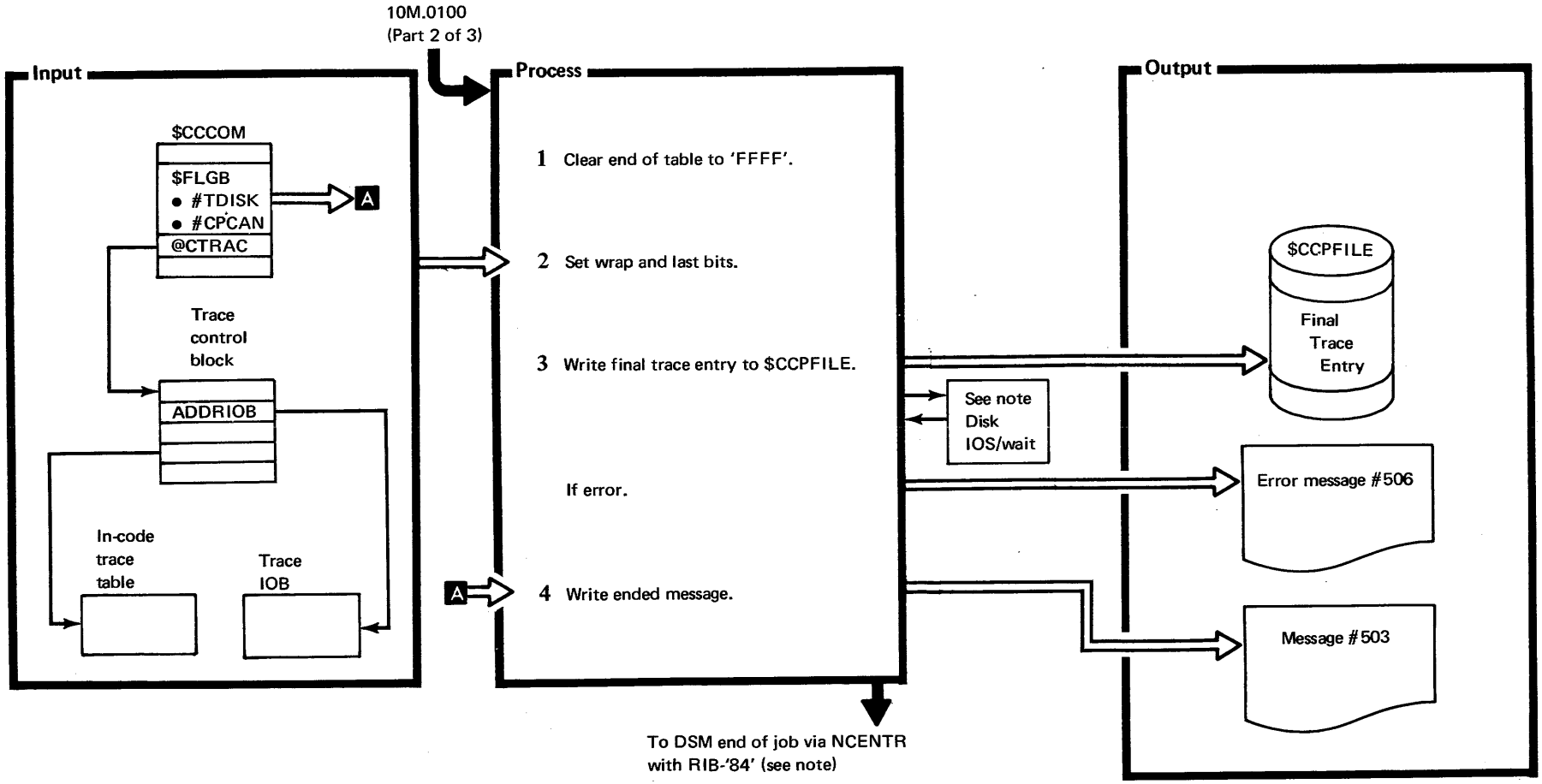


Note 1: See *IBM System/3 Disk Systems Data Management and Input/Output Supervisor Logic Manual*, SY21-0512.

Note 2: See *IBM System/3 Multiple Line Terminal Adapter RPQ Program Logic Manual*, SY21-0527.

Note 3: See *IBM System/3 Models 4, 6, 8, and 10 Disk Systems System Control Program Logic Manual*, SY21-0502.

Diagram 10M.0100 (Part 2 of 3). Overall Flow of Shutdown



Note: See *IBM System/3 Models 4, 6, 8, and 10 Disk Systems System Control Program Logic Manual*, SY21-0502.

Diagram 10M.0100 (Part 3 of 3). Overall Flow of Shutdown

Page of SY21-0531-2
Issued 24 June 1977
By TNL: SN21-5530

Restore DSM Entry Points

The next step in Shutdown is to restore the low main storage addresses of CEFE, General Entry, Disk IOS, and Disk Wait. If this is a DPF system, the IAR of the other program level is stored and set to point to a U-halt within $CC5SH for non-remap or within $@9CIN for remap in order to prevent the other program level from referencing the low main storage entry points while they are being restored. The low main storage entry points are then restored from areas in the CCP Communications Area where they were saved during CCP startup. The other program level is now released to continue processing.

Close BSCA Lines

All LCBs in the system are scanned, starting at the first LCB pointed to by LCB #1 in the CCP Communications Area. If a BSCA LCB is found, the line is closed via DSM Close. This continues until all LCBs have been processed.

Close MLTA Lines

The address of the MLTA adapter DTF is in the CCP Communications Area (@MLTAD). If MLTA is present, the MLTA Close transient ($$ML$A) is called to close the adapter DTF, which causes all line DTFs to be closed.

Update Program Usage Counts

If the Program Usage Count option was selected at CCP generation, a sector of PCT entries is read from $CCPFILE. If the read was not successful, error message 504 is printed and the remaining counts are not updated. If the read was successful, the program name in the PCT is compared to the first name in the PCT index. The index is a main storage table containing the program name corresponding to the last PCT entry in each sector on disk. If the names are equal, the entire sector has been updated and it is rewritten to $CCPFILE. If the write was unsuccessful, error message 505 is printed and the remaining counts are bypassed. Counts are updated from a main storage Program Usage Count Table, whose address is in the @PUCNT field in the CCP Communications Area. The table contains a two-byte count for each PCT entry. When a PCT index entry of X'FFFFFFFFFFFF' is passed, all PCT entries have been updated and written to $CCPFILE.

Write Out Last Disk Trace Entry

If CCP Trace to Disk was turned on by the system operator, the trace control block is found from the @CTRAC field in the CCP Communications Area. If the address of the next entry equals the address of the end of the resident trace table, the table is exactly filled. If the table is not completely filled, the remaining area in the resident table is filled with FF. This is done via the CCP Move Routine, $CC4MX. Bits 1 and 2 are set on in the first byte of the last entry in the resident table to indicate the last entry. If the disk address of the next entry is equal to the address of the first entry on disk, this group of entries causes the table on disk to wrap; therefore, bit 2 is set on in the first byte of the first entry to indicate a wrapped table. The resident table is now written to the next available address on disk using the trace IOB, the address of which is located in the trace control block. If a disk error occurs, error message 506 is printed.

Print Final Message and Exit

Message 503 is now printed, indicating that Shutdown or Cancel has completed, depending on what was requested by the system operator. This is determined by testing the #CPCAN bit in $FLGB field of the CCP Communications Area, which indicates that a Cancel was requested. Control is now returned to Disk System Management via an exit to General Entry with a RIB of 84 for end of job.

Program Organization

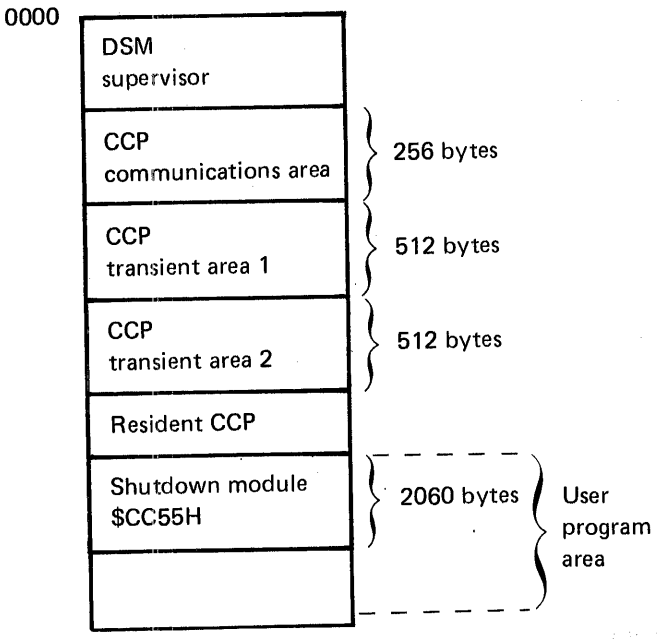


Figure 10-2. Storage Layout During Shutdown

CCP Shutdown ($CC5SH)

ENTRY POINT: SHNTRY
CHART: RM
FUNCTION/OPERATION:

- Calls CCP Open to build a full DTF from the CCP pseudo disk DTF. This DTF is built in an area within this routine.
- Calls true system close to close the rebuilt disk DTFs.
- Scans the LCB chain to find and close each BSCA DTF via system close.
- Closes the MLTA adapter and line DTFs via the MLTA close transient, $$ML$A.
- Restores the following system addresses:
  - Dump address at 0000.
  - General Entry at 0004.
  - IOS address at 0008.
  - Wait address at 000C.
- Resets the CCP running bits in the System Communications Area.
- Updates the program usage counts by reading the PCTs from $CCPFILE, incrementing the usage counts, and writing the PCTs back out to disk if the option was selected.
- Writes out to the disk trace table the last main storage trace entry if disk trace is on. The wrapped indication is set on in the table, if appropriate, and any remaining area in the trace entry is cleared to FFFF.
- Writes to the system operator a final Shutdown message indicating that Shutdown has completed.
- Exits to system EOJ.

INPUT:

- Pointers in the CCP Communications Area.
- Pointers in the System Communications Area.
- Address of true NCENTR located within $CC4IG.
- PCTs, residing within $CCPFILE on disk.
- PCT index in main storage.
- Program usage count table in main storage.
- Trace entry and the trace control block in main storage.

OUTPUT: Output will be the closed files and the system will be reset to normal running status. If a disk error occurs, one of the following messages is printed on the console:

- 504 disk error occurred while reading PCTs.
- 505 disk error occurred while writing PCTs.
- 506 disk error occurred while writing final disk trace entry.
- 507 error occurred while closing file XXXXXXXX.

EXITS: System EOJ.
NOTES: This program runs in the user program area and is loaded and called by $CC4EJ. Relocation of external references, such as $CCCOM, is also done by $CC4EJ via the table at the start of $CC5SH and a similar table within $CC4EJ.

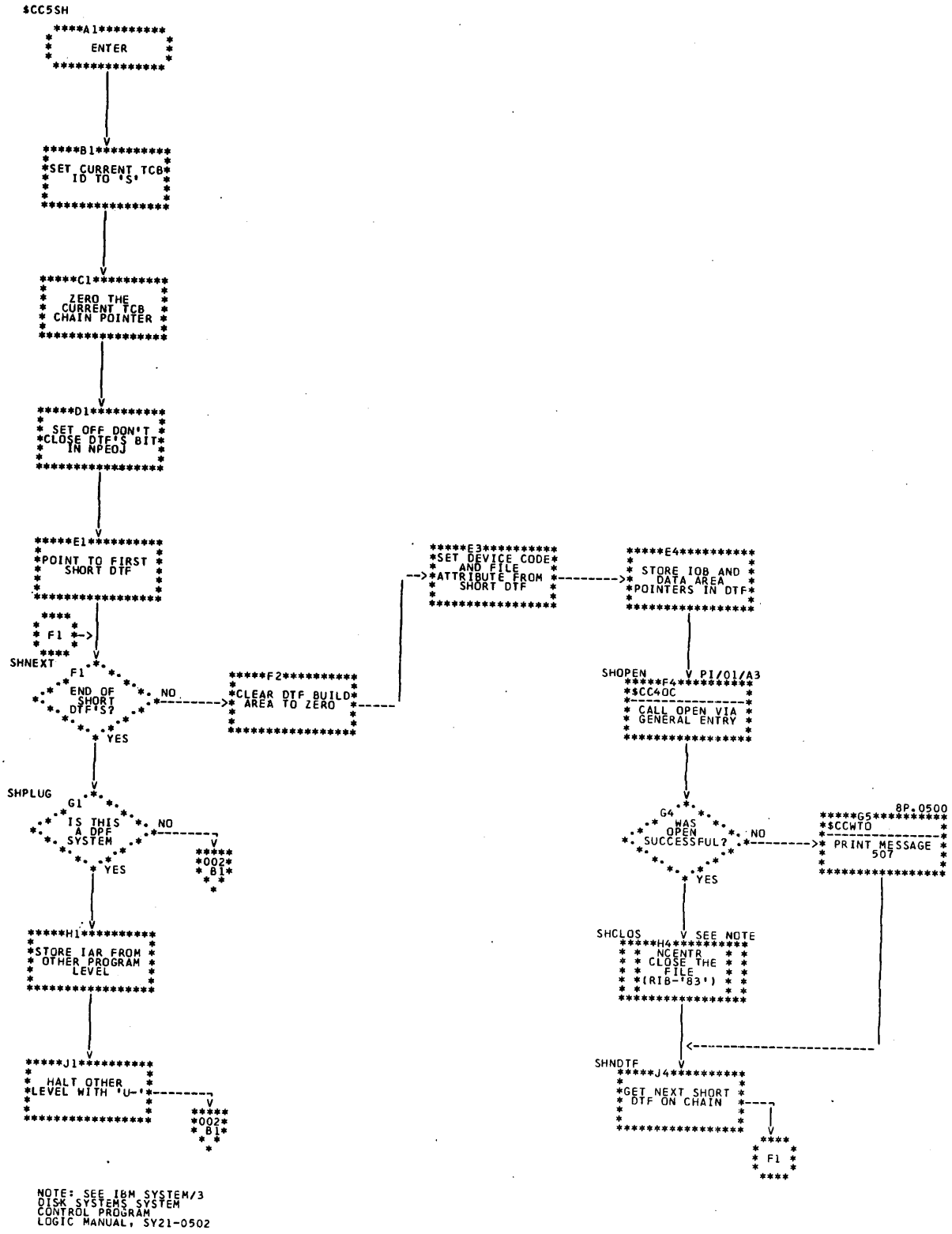
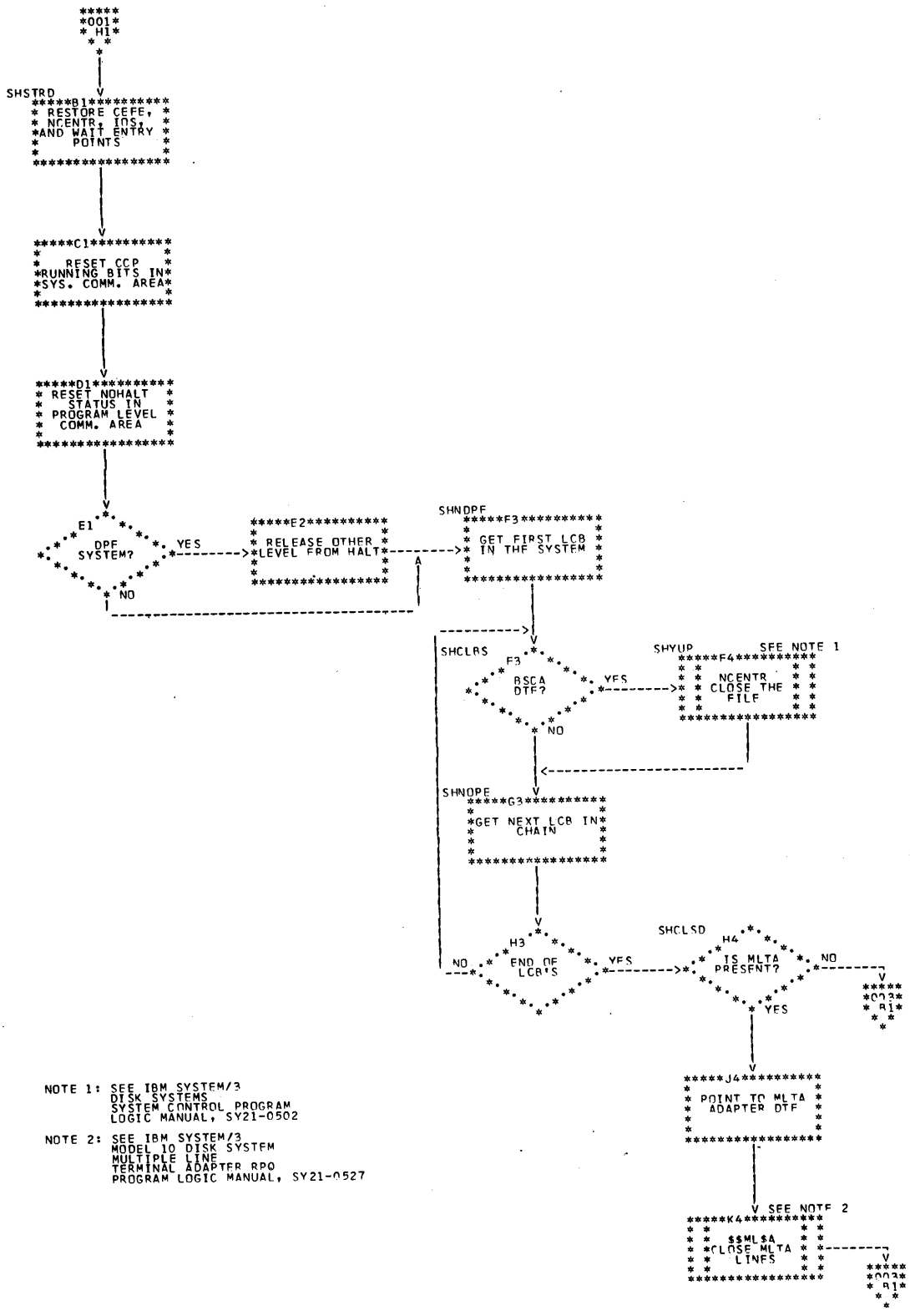


Chart RM (Part 1 of 4). CCP Shutdown ($CC5SH)



NOTE 1: SEE IBM SYSTEM/3
DISK SYSTEMS
SYSTEM CONTROL PROGRAM
LOGIC MANUAL, SY21-0502

NOTE 2: SEE IBM SYSTEM/3
MODEL 10 DISK SYSTEM
MULTIPLE LINE
TERMINAL ADAPTER RPQ
PROGRAM LOGIC MANUAL, SY21-0527

Chart RM (Part 2 of 4). CCP Shutdown ($CC5SH)

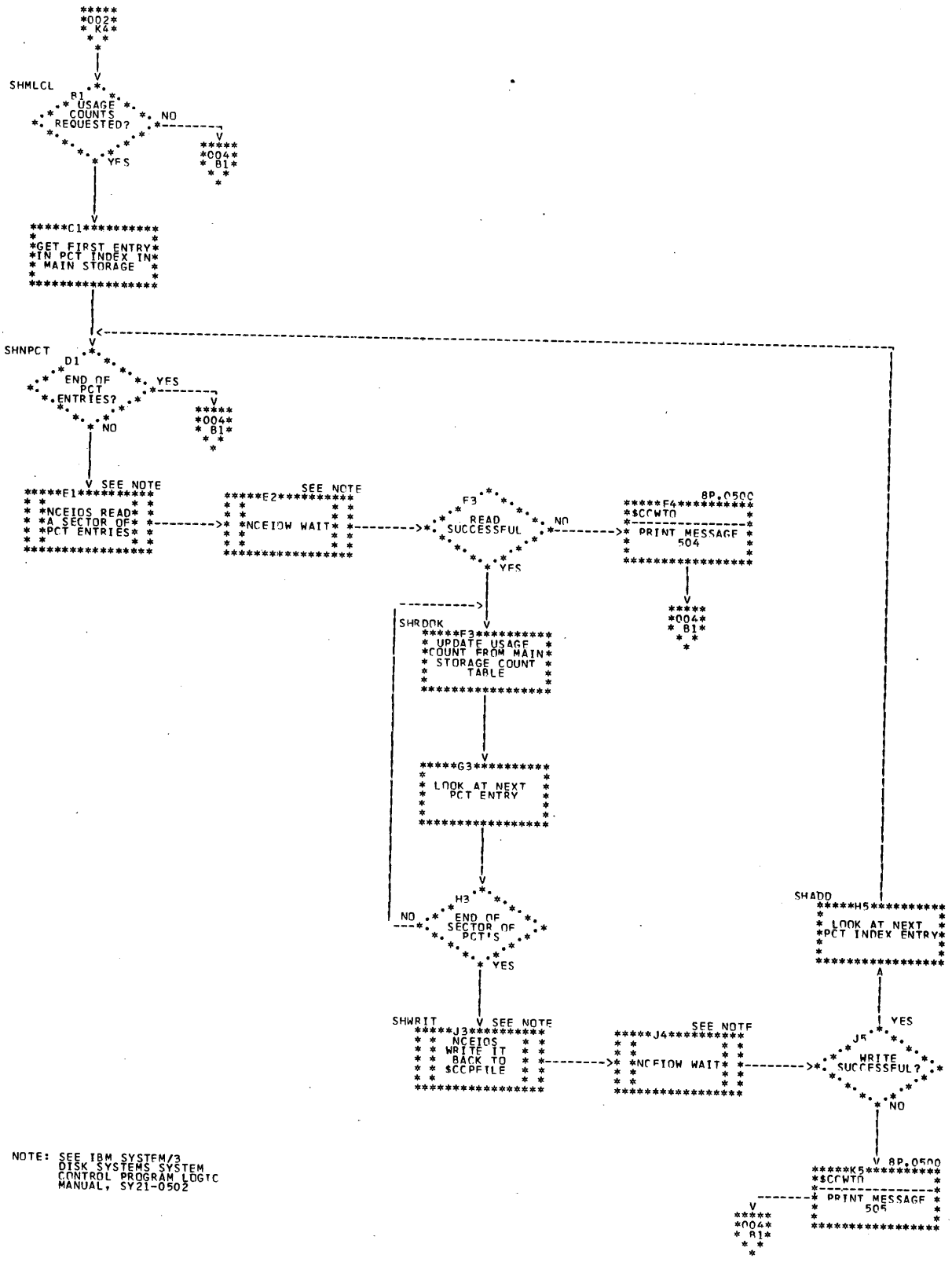


Chart RM (Part 3 of 4). CCP Shutdown ($CC5SH)

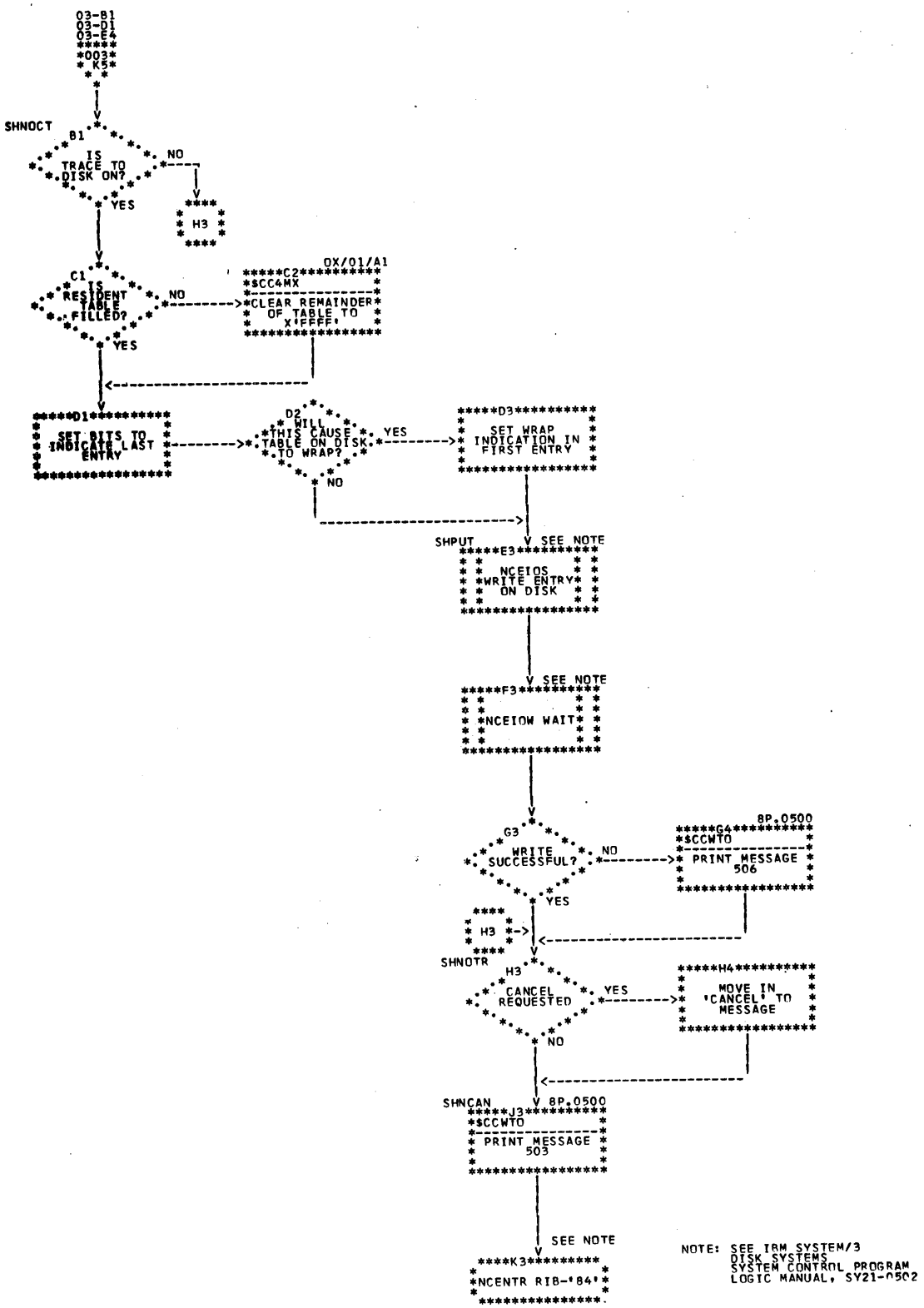


Chart RM (Part 4 of 4). CCP Shutdown ($CC5SH)

Chapter 11. Disk-to-Printer $CCPFILE Dump Program and Log List Program

Introduction

The Disk-to-Printer Dump Program ($CCPDD) is a stand-alone CCP support program that runs under control of DSM. It must be used whenever the contents of $CCPFILE main storage dump data or trace table data is to be dumped to the printer. Data is read from the disk, formatted according to printer size and data content, and printed on the system printer using printer data management routines. It does not run in the other program level or partition on a DPF system while CCP is running.

$CCPDD runs on any configuration that supports CCP.

Method of Operation

OCL Required

The following OCL must be used to initiate the running of program $CCPDD:

```
// LOAD $CCPDD,uu
// FILE UNIT-xx,NAME-$CCPFILE,PACK-nnnnnn
// RUN
```

where uu is the disk unit on which $CCPDD resides, xx is the disk unit on which the $CCPFILE dump data resides, and nnnnnn is the packname on which $CCPFILE resides. Follow this OCL with additional OCL records which specify what is to be dumped. Terminate the operation of this program with a /*.

Program Options

All dump options are selected by use of a control statement, which begins with // DUMP. A main storage dump can be specified by a control statement of the following form:

```
// DUMP n
```

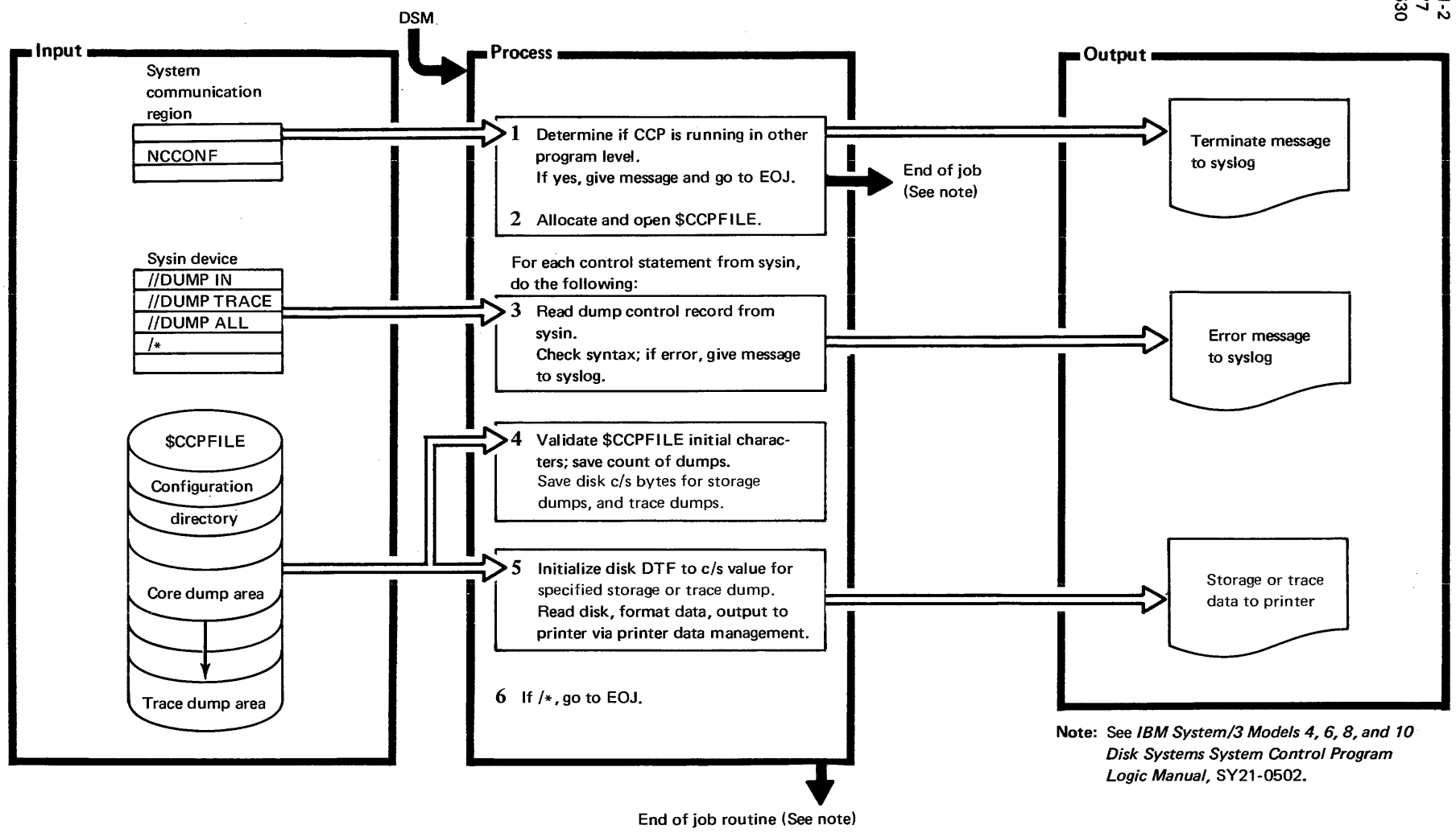
where n is the dump number. All main storage dumps residing on $CCPFILE can be asked for by the control statement // DUMP ALL. A partial main storage dump can be specified by the OCL statement:

```
// DUMP n,x'aaaa',x'bbbb' (Models 4, 8, and 10)
// DUMP n,x'aaaaaa',x'bbbbbb' (Model 12)
```

where n is the specified dump number (1 – 9), aaaa is the beginning main storage address, and bbbb is the ending main storage address. In this case, beginning addresses are rounded down to the next hex 100 address and the ending address is rounded up. Multiple dump selection requests can be made during one run.

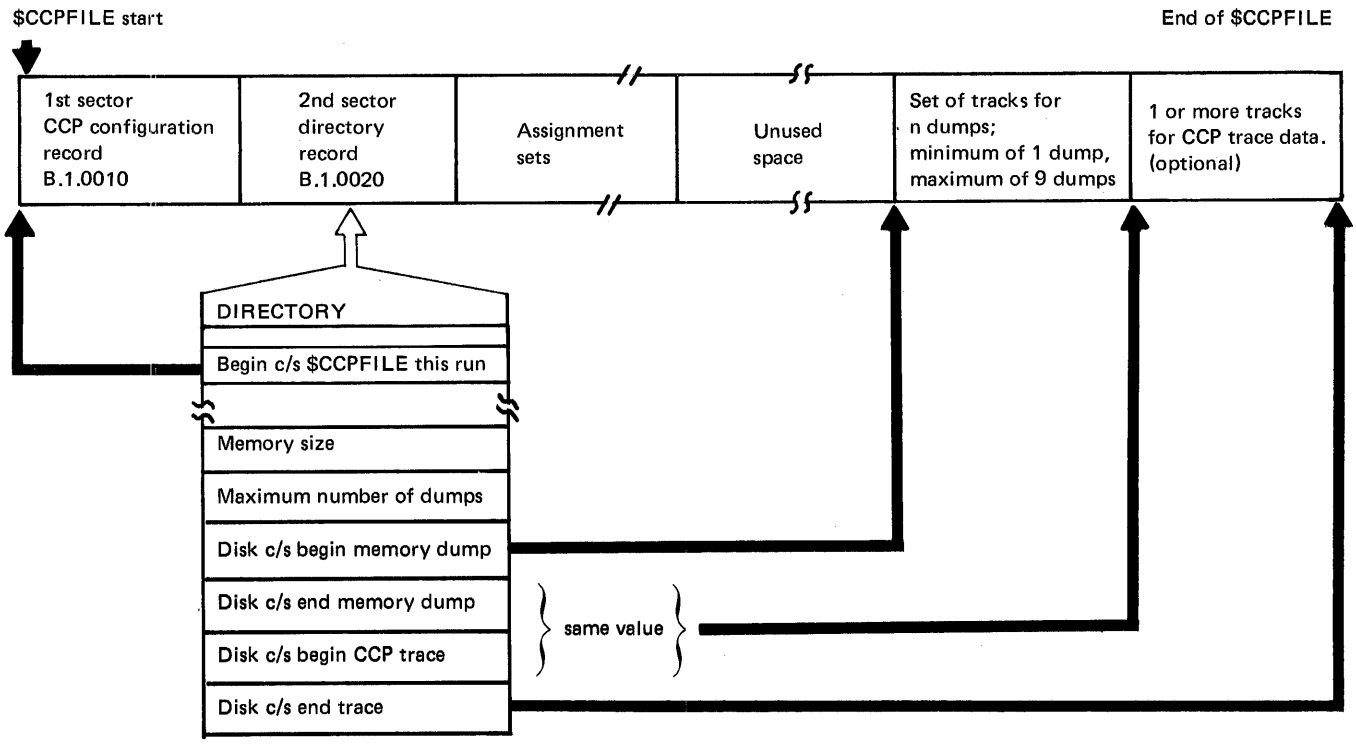
See Diagram 11M.0100 for an overview of $CCPDD and Figure 11-1 for a diagram for locating $CCPFILE dumps and traces.

Page of SY21-0531-2
Issued 24 June 1977
By TNL: SN21-5530



Note: See *IBM System/3 Models 4, 6, 8, and 10 Disk Systems System Control Program Logic Manual*, SY21-0502.

Diagram 11M.0100. Overall Flow of $CCPDD



Core dump memory size	Track usage number of tracks
24k	4
32k	6
48k	8
64k	11
80k	14
96k	16

Note: All dumps begin on a track boundary.

Figure 11-1. Schematic of Locating the Traces and Dumps

Program Organization

Disk-to-Printer $CCPFILE Dump Program ($CCPDD)

ENTRY POINT: DD0000
CHART: RZ
FUNCTION: Selectively dumps from disk, main storage memory, or trace dumps which were previously written on $CCPFILE by CCP.
INPUT:
- Control information from SYSIN.
- Data to be printed from $CCPFILE.

OUTPUT: Main storage dump or trace table dumps to printer.
EXIT: System end of job routine.
EXTERNAL REFERENCES: $$LPRT (printer I/O module)

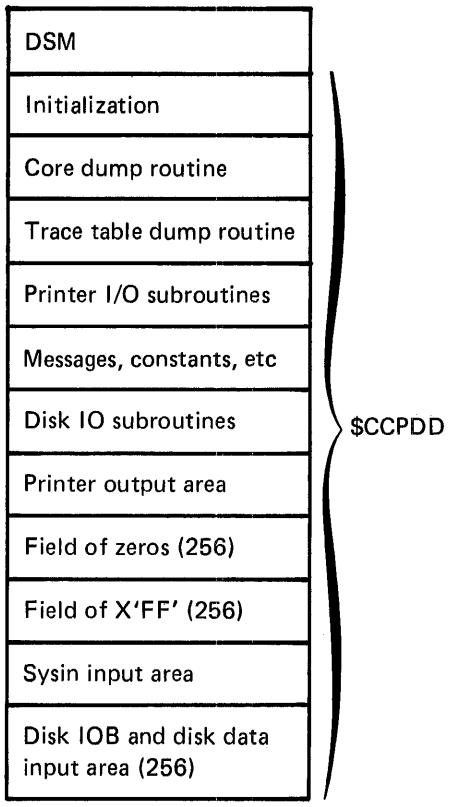
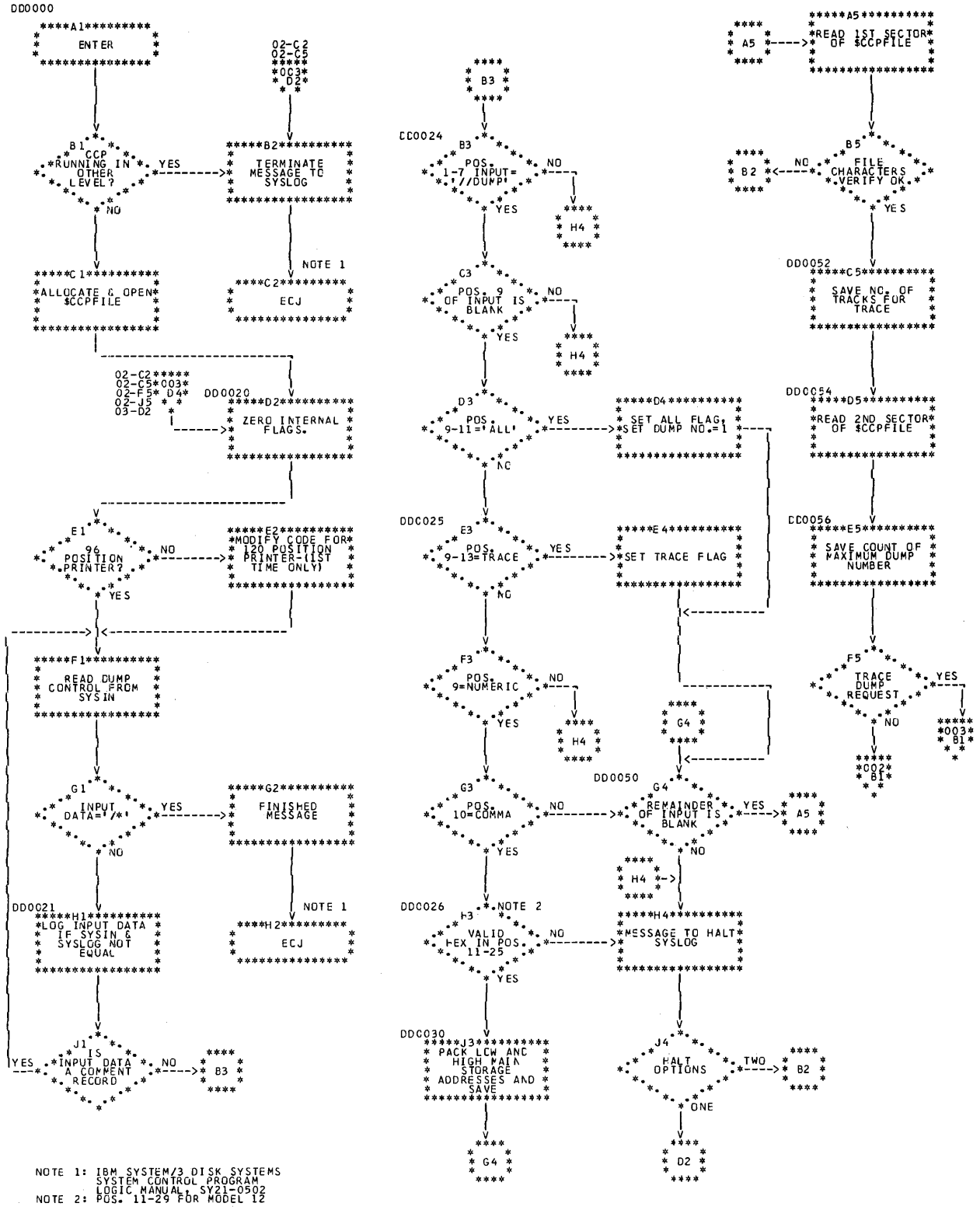


Figure 11-2. Storage Layout during $CCPDD



- **Chart RZ (Part 1 of 3). Disk-to-Printer $CCPFILE Dump Program ($CCPDD)**

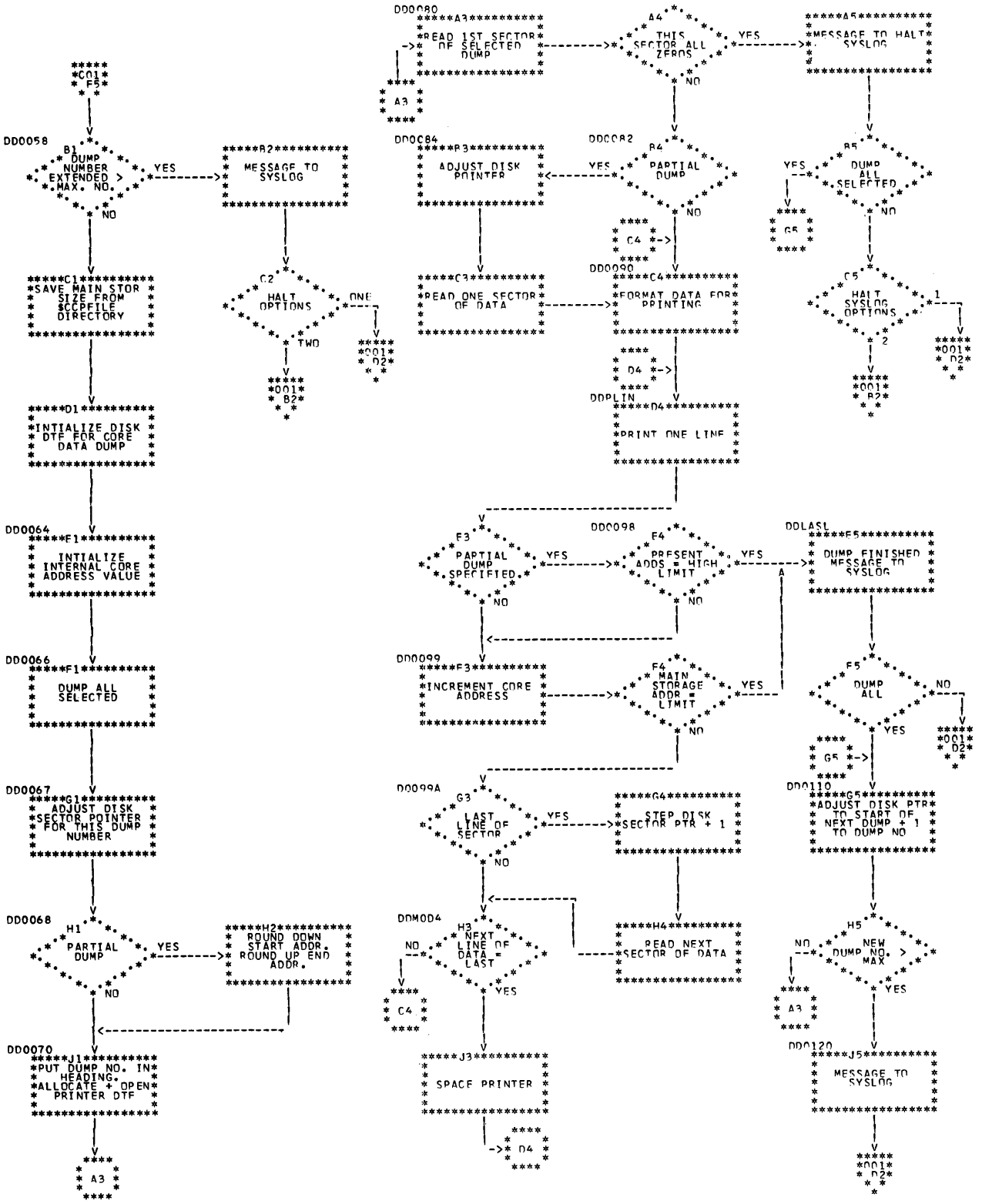


Chart RZ (Part 2 of 3). Disk-to-Printer $CCPFILE Dump Program ($CCPDD)

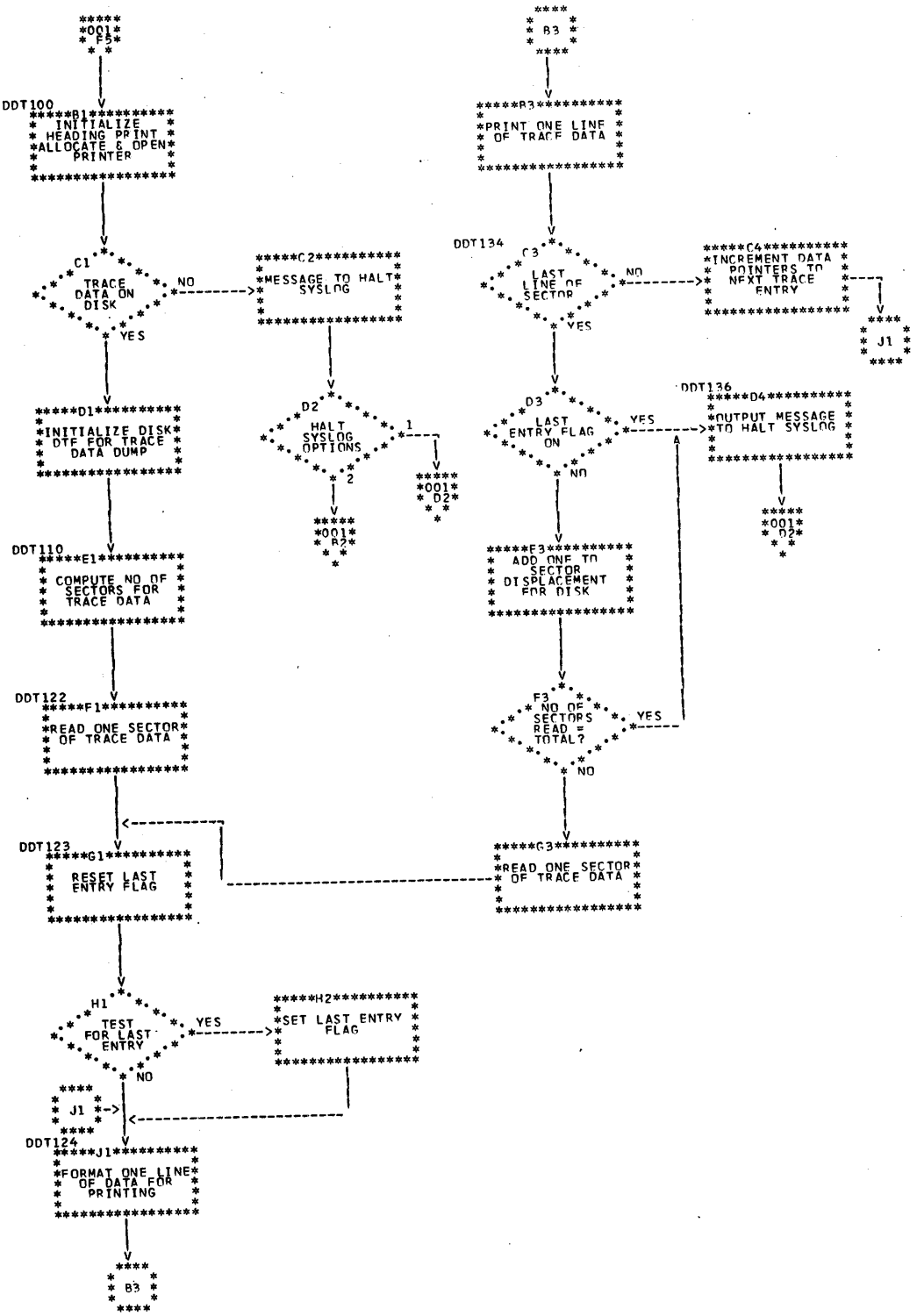


Chart RZ (Part 3 of 3). Disk-to-Printer $CCPFILE Dump Program ($CCPDD)

INTRODUCTION TO $CCPLL (MODEL 4 ONLY)

The log file dump program ($CCPLL) is a standalone CCP support program that runs under the control of DSM. It must be used whenever the contents of $CCPLOG file is to be printed. Data is read from the disk, formatted according to message lengths, and printed on the system printer using printer data management routines.

$CCPLL will run on any configuration that supports CCP.

METHOD OF OPERATION

OCL Required

The following OCL must be used to initiate the running of program $CCPLL:

```
// LOAD $CCPLL, uu
// FILE UNIT-xx, NAME-$CCPLOG, PACK-nnnnnn
// RUN
```

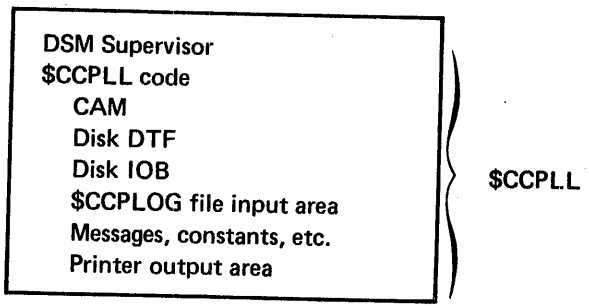
where uu is the disk unit on which $CCPLL resides, xx is the disk unit on which the $CCPLOG file resides, and nnnnnn is the packname on which $CCPLOG resides.

See Diagram 11M.0200 for an overview of $CCPLL.

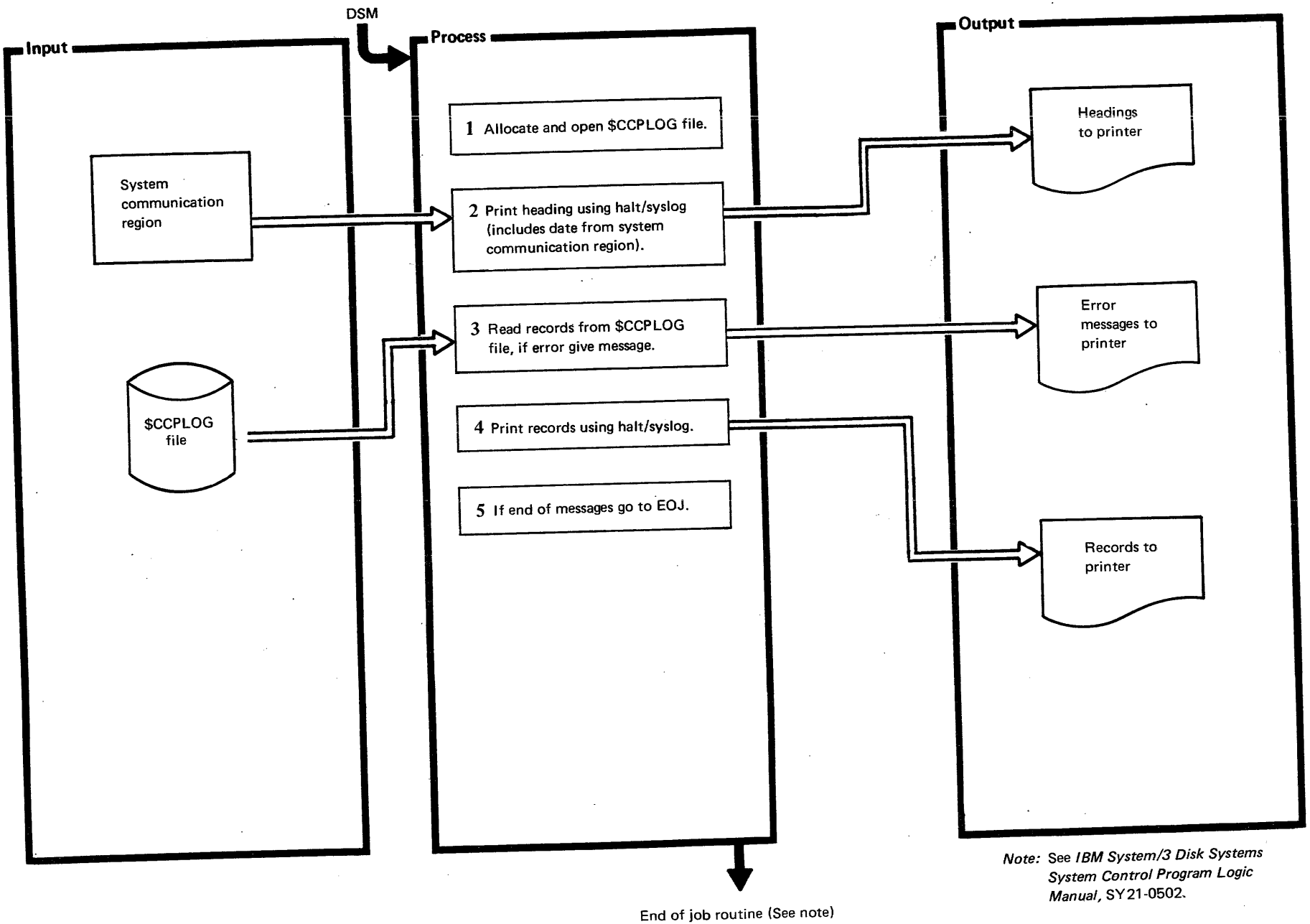
PROGRAM ORGANIZATION

Disk-to-Printer $CCPLOG File Program ($CCPLL)

ENTRY POINT: $CCPLL
CHART: YY
FUNCTION: Print all of the CCP messages which were previously written on $CCPLOG by CCP.
INPUT: Data to be printed from $CCPLOG.
OUTPUT: CCP messages printed on printer.
EXIT: System end of job routine.
EXTERNAL REFERENCES: Allocate, open, printer DM.



Storage Layout During $CCPLL



Note: See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.

Diagram 11M.0200. Overview of $CCPLL

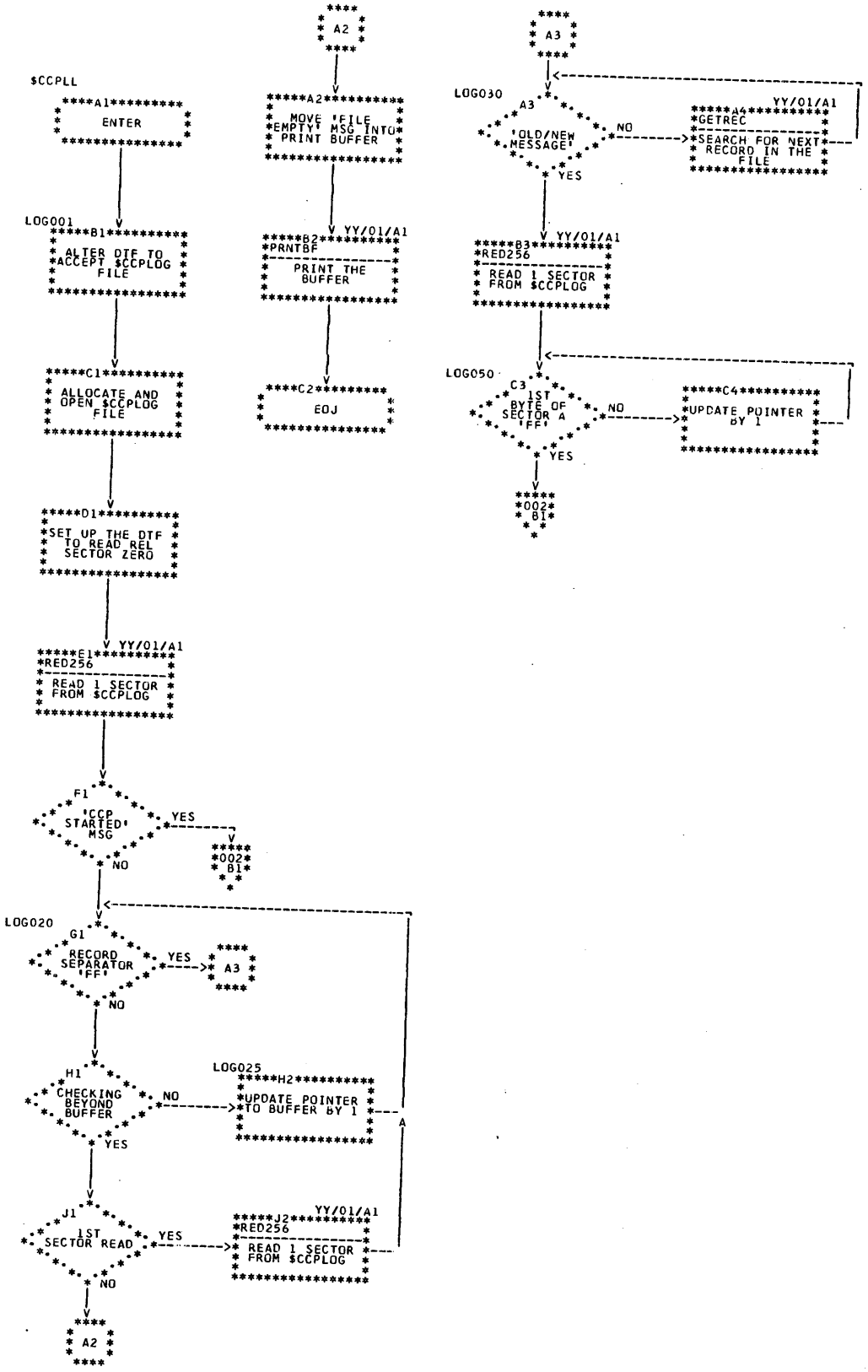


Chart YY (Part 1 of 3). Disk-to-Print $CCPLOG Dump Program ($CCPLL)

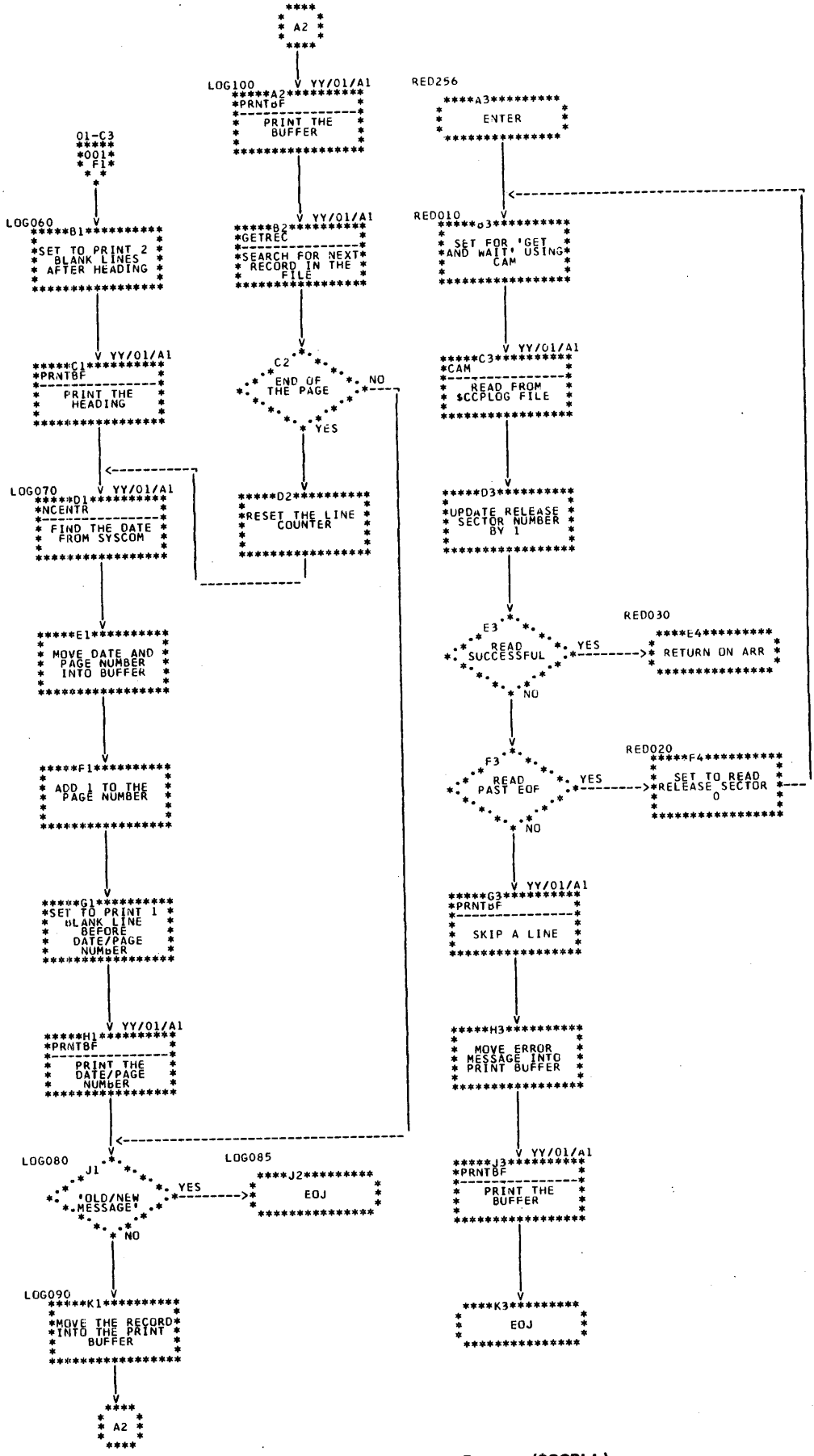


Chart YY (Part 2 of 3). Disk-to-Print $CCPLOG Dump Program ($CCPLL)

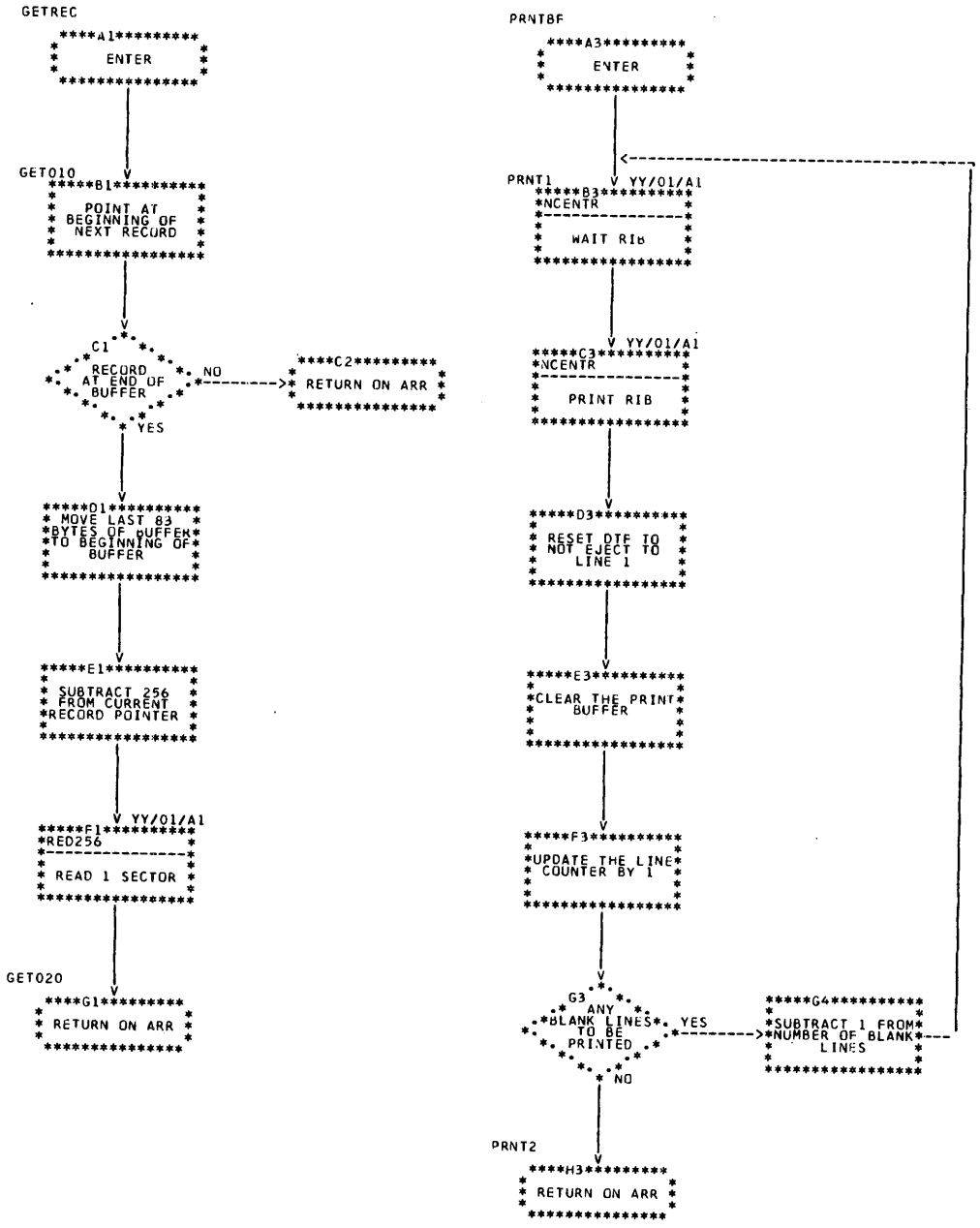


Chart YY (Part 3 of 3). Disk-to-Print $CCPLOG Dump Program ($CCPLL)

Chapter 12. Standalone Dump Programs (Models 8, 10 and 12 Only)

Introduction

Four MFCU loadable dump programs are provided as part of the CCP feature of DSM. Each dump differs from the others only in the specific character chain or train it is meant to run with. The four dumps are:

Program Name	*Chain/Train Supported*
CCPDAN	AN2/LC2 (48 character standard)
CCPDHN	HN2 (48 character COBOL/FORTRAN)
CCPDPN	PN2 (60 character)
CCPDTN	TN5 (120 character upper/lower case)

These card loadable dump programs are provided in case a severe error situation has destroyed the linkage in main storage to the DSM CEFE dump program. The availability of these dump programs itself emphasizes the importance of obtaining a main storage dump when attempting to diagnose an error condition while executing CCP.

The use of these dump programs is not necessarily limited to CCP execution time error conditions. The dumps are entirely generalized and can be used to dump main storage at any time.

Main Storage Requirements

The dump program itself occupies (and therefore destroys the contents of) 768 bytes of main storage. In addition, the bootstrap loader used to bring the dump program into main storage occupies the first 256 bytes of main storage.

Machine Requirements

- 5424 Multifunction Card Unit.
- 5203 or 1403 Printer, with chain/train of a character set matching the dump program selected.

Method of Operation

When PROGRAM LOAD is pressed with the Load Selector Dial set to CARD, a bootstrap loader is brought into main storage in bytes X'00-FF'. The loader issues halt CU.

The system operator dials an address, the leftmost two digits of which specify the high order position of the storage location at which the dump program is to be loaded (for example, 60 for load starting at X'6000'). The system operator then resets the halt. (For dump procedure and load addresses, see *IBM System/3 Models 10 and 12 Communications Control Program System Operator's Guide,* GC21-7581.)

The loader then loads the dump program into the 768 (X'300') bytes at the specified location, and turns control over to the dump program. The dump program issues halt 5E.

The system operator selects the bounds of storage to be dumped by dialing the high order digits of the address of the first and last 256-byte block to be dumped. That is, xxyy is dialed indicating that locations X'xx00' through X'yyFF' are to be dumped. The system operator then resets the halt.

The dump program dumps the selected area of main storage to the printer. Each print line dumps 32 bytes of main storage, both in hex and in character form. (A period (.) is displayed for an unprintable character.) On a 5203 with 96 character positions, not all the character representations can be printed; the dump is double spaced. Duplicate sets of 32 bytes are not printed, but are represented by an extra blank line in the dump; however, the last 32 bytes of a dump are always printed even if they are duplicates.

The dump program then returns to the 5E halt. The system operator can select another area of main storage to dump. The dump programs never issue an EJ halt.

Program Organization

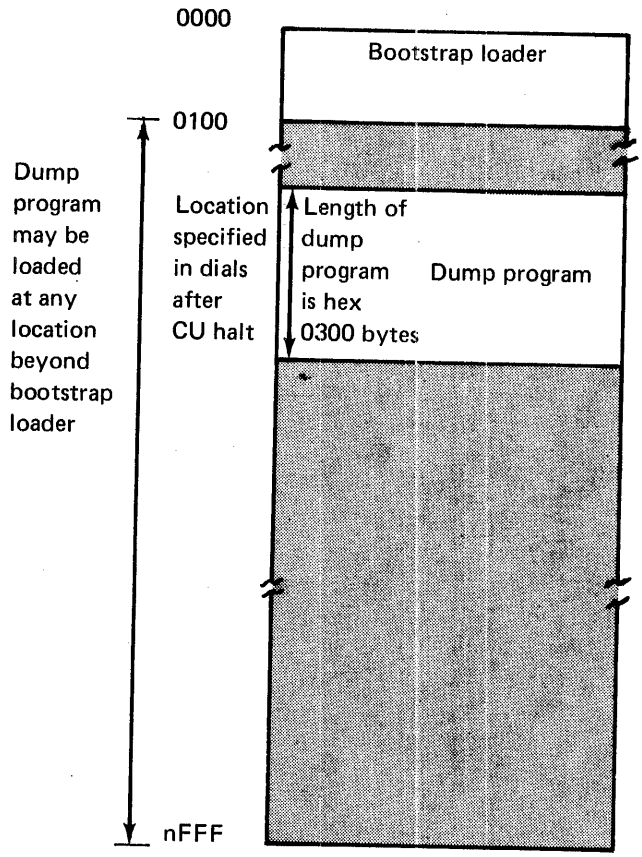


Figure 12-1. Storage Layout of Dump Program

Card-Loadable Main Storage Dump Programs (CCPDAN, CCPDHN, CCPDPN, CCPDTN)

ENTRY POINTS (from card bootstrap loader):

- CCPDAN
- CCPDHN
- CCPDPN
- CCPDTN

CHART: ZN

FUNCTION: Dumps main storage to printer.

INPUT:

- Main storage address bounds from operator via ADDRESS/DATA switches; on a 5E halt, the operator sets:
  - Switches 0 – 1 – High order digits of dump start address.
  - Switches 2 – 3 – High order digits of dump end address.
- Contents of main storage itself.

OUTPUT: Hex and character representations of the contents of the selected area of main storage. Each line represents 32 bytes of storage and contains:

- The address of the first of the 32 bytes.
- The hex representation of the 32 bytes.
- The character representation of as many of the 32 characters as are printable and that will fit in the available print positions.

(The first line of output appears after an eject to a new page. A page length of 66 lines is assumed – page overflow ejection is performed.)

HALTS, NORMAL:

- An initial 5E halt is issued to permit the operator to set the ADDRESS/DATA switches to the bounds of the dump.
- Upon completion of dumping specified area of storage, return to halt 5E. ADDRESS/DATA switches may be reset, permitting selection of another area of main storage to be dumped.

HALTS, ERROR:

- If, after a 5E halt, the setting of switches 0 – 1 (start address) is greater than the setting of switches 2 – 3 (end address), the 5E halt is immediately re-issued.
- If a printer error occurs, a PC halt is issued. Upon pressing START (or HALT/RESET), the error line is retried and dumping continues.

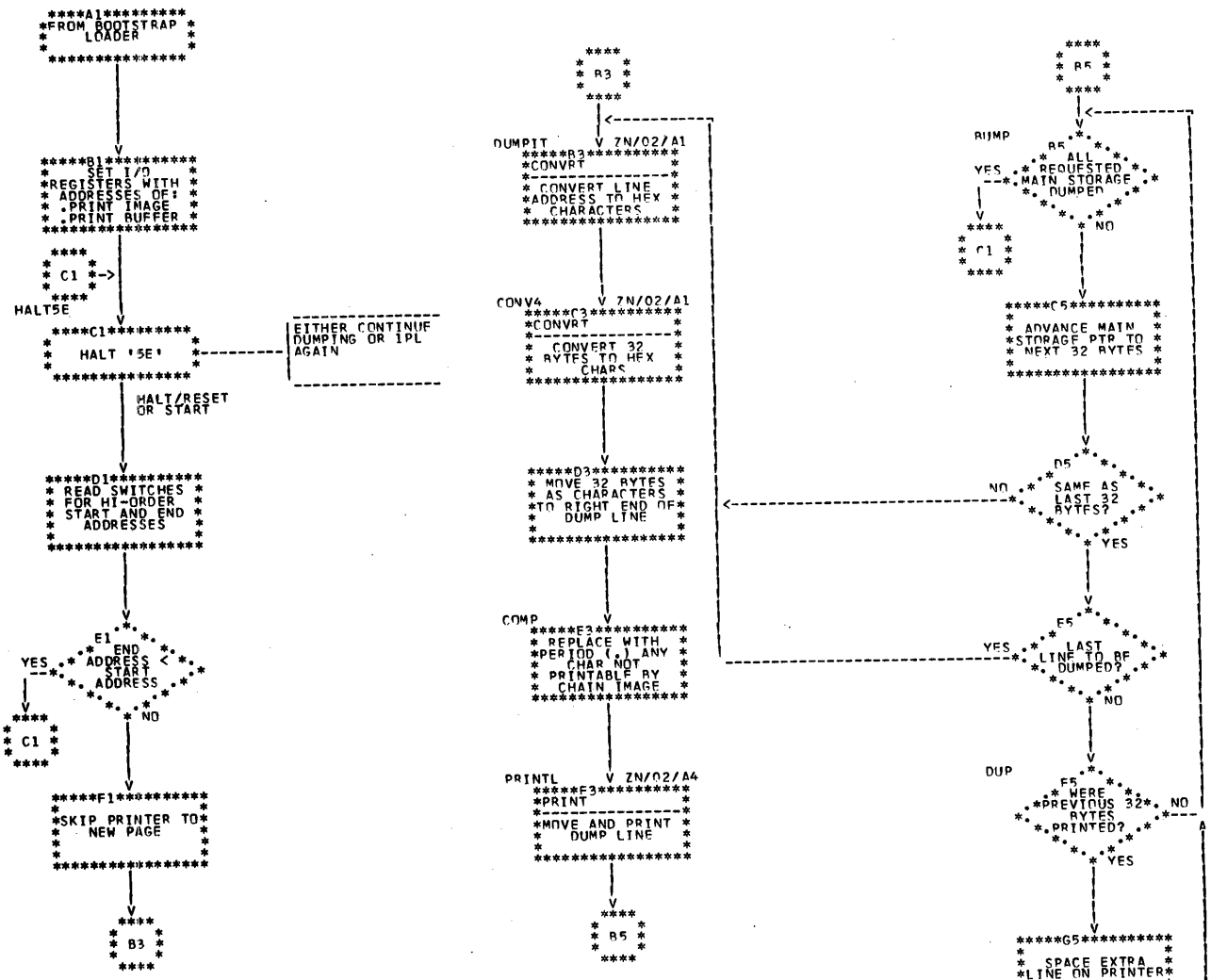


Chart ZN (Part 1 of 2). Card-Loadable Main Storage Dump Programs (CCPDAN, CCPDHN, CCPDPN, CCPDTN)

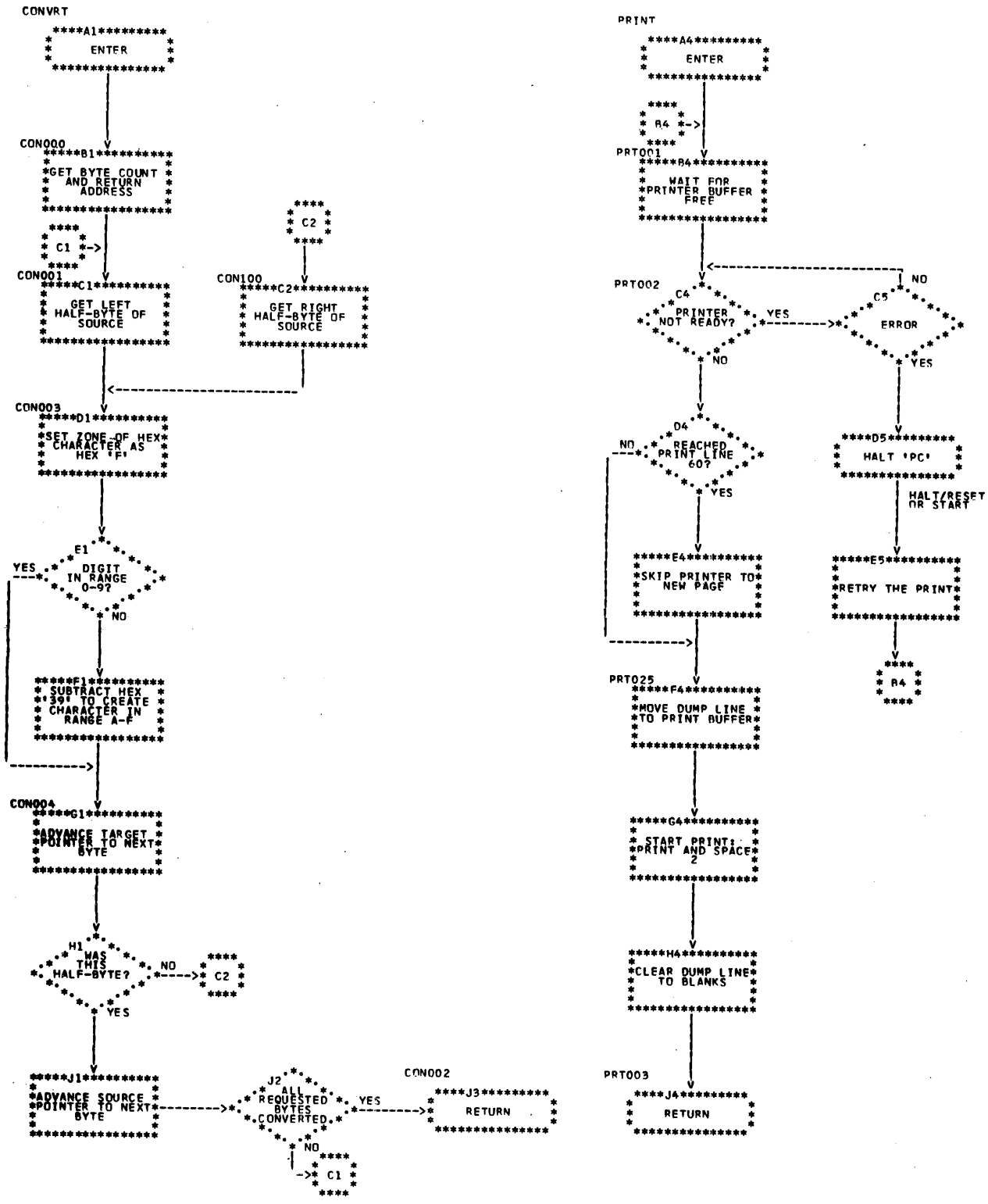


Chart ZN (Part 2 of 2). Card-Loadable Main Storage Dump Programs (CCPDAN, CCPDHN, CCPDPN, CCPDTN)

Chapter 13. Installation Verification Program (CCPIVP)

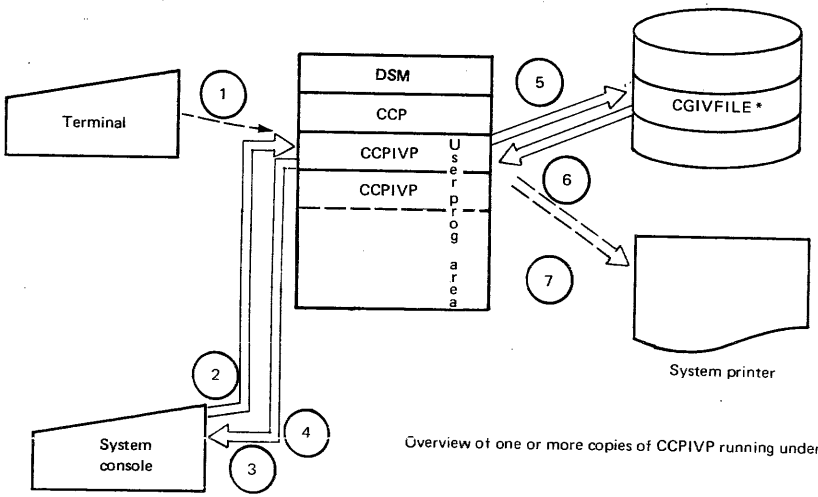
Introduction

CCPIVP is the installation verification program provided with CCP. After being included in an Assignment Build ($CCPAS) run, it can be used to verify a satisfactory generation of CCP. This program is distributed in two forms on the CCP distribution pack:

1. Object module CCPIVP that has been link-edited to include the necessary disk data management modules ($$CSOP, $$CSIP). The link edit map for this module is contained in Figure 13-1.

2. Relocatable module CCPIVR that has not been link-edited. Using the optional link edit statements punched during generation, CCPIVR can be link-edited to create a new CCPIVP that includes the necessary disk and printer data management modules to test the use of the printer under CCP.

On Model 4 CCPIVP is included, already link edited, and ready to run with the 5213 printer.



Overview of one or more copies of CCPIVP running under CCP.

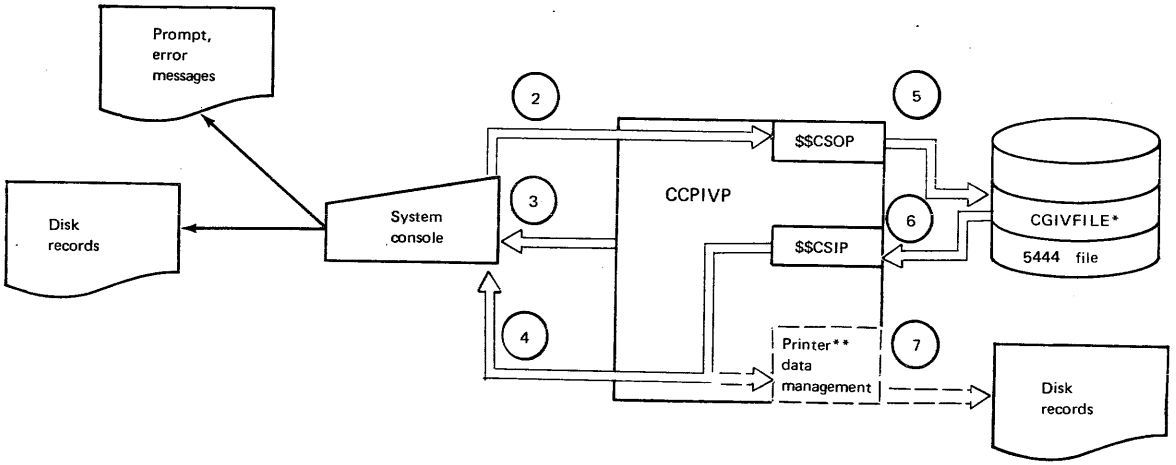
1. Program request from terminal (optional)
2. Program request from console, input data for disk record, selection of printer or console for disk file output, responses to error messages
3. Prompting or error messages
4. Disk records from created file
5. Disk records based on data with the program request and console input
6. Disk records from created file
7. Optional printing of created file

*CGIVFILE is the filename used in the disk DTFs within CCPIVP. If symbolic files are being used, the actual file name may differ according to the assignment set.

**On Models 8, 10, and 12, printer data management modules are optionally link edited based on the installation's needs.

If link edited from an RPG II pack used during CCP generation, the modules would be $$LPRT, $$UPRT.

If link edited from a non-RPG II pack used during CCP generation, the modules would be $$LPRT, $NLPRT.



Data path through CCPIVP

System printer

Figure 13-1. CCPIVP Method of Operation

Method of Operation

CCPIVP can be requested either from a command terminal or the console. Once loaded, CCPIVP communicates solely with the console, receiving input records and building disk records in a consecutive file. Based upon input from the console, the created file is dumped to either the console or the system printer.

Figure 13-2 shows:

- One or more copies of CCPIVP running under CCP. If more than one copy of CCPIVP is to be run concurrently under CCP, the following conditions must be in effect:

  a. Multitasking (MAXEUP>1) must be supported.
  b. Symbolic files must be used. Each copy of CCPIVP requires one disk file.
  c. CCPIVP must not be link-edited to use the printer.
  d. Sufficient main storage must be available for multiple copies of CCPIVP (approximately 4K per copy).

- Flow of data through a copy of CCPIVP.

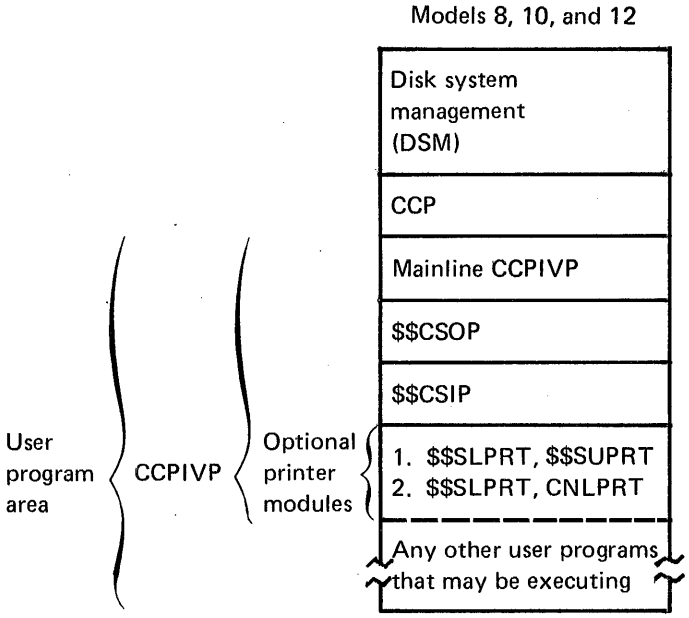


Figure 13-2 (Part 1 of 2). Main Storage Map of CCPIVP Running Under CCP

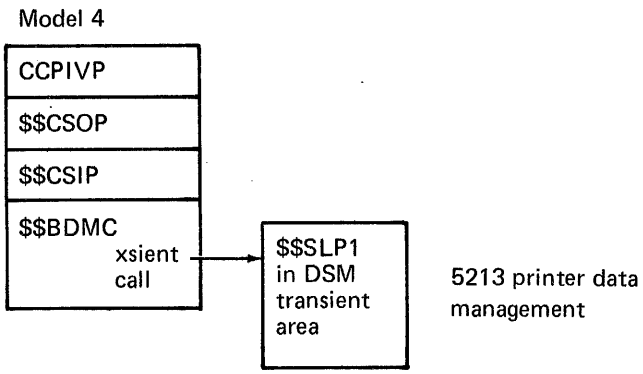


Figure 13-2 (Part 2 of 2). Main Storage Map of CCPIVP Running Under CCP

FUNCTIONS VERIFIED

The CCP functions that can be verified by running CCPIVP include:

- Program load from the console, and data with the program request.
- Program load from a terminal, and data with the program request.
- Resource management-core, disk and printer (if link-edited with CCPIVP).
- The following op codes to the console: put-wait, put-then-get, put-no-wait.
- Accept input op code to the requester.
- Release terminal op code to a non-console requester.
- If symbolic files are supported, use of the symbolic files and the /FILE terminal operator command.
- Multitasking (if supported) under CCP:

  a. If symbolic files are used, and
  b. CCPIVP does not use the printer.

- Open, create, and close a consecutive disk file. Open, list, and close the created file, verifying these file functions under CCP.
- If printer data management has been optionally link-edited, use of the printer under CCP. Use of halt/syslog under CCP by forcing a halt on the printer.

CCPIVP HALTS

CCPIVP issues stand-alone halts (HPLs) if:

- An unexpected return code was received from CCP.
- An unrecoverable printer or disk error occurred and control was returned to CCPIVP.

These halts should not occur and are given primarily so that dumps of main storage can be taken immediately upon detection of the condition.

Halt U0 (ABD1245 on Model 4)

Reason: This copy of CCPIVP was requested by a terminal and the release operation code failed.

Recovery:

- Press the HALT/RESET or START key; CCPIVP will go to end-of-job.
- Take a dump of the main storage contents by pressing SYSTEM RESET and START, or by using the appropriate stand-alone main storage dump program.

Halt H1 (ABD on Model 4)

Reason: Return code other than 00 or 01 received from accept input operation (can only happen if data has a program request).

Recovery:

- Press the HALT/RESET or START key; error message is written, input data required.
- Take a dump of the main storage contents by pressing SYSTEM RESET and START, or by using the appropriate stand-alone main storage dump program.

Note: If you enter the correct data and length of data and get error messages from CCPIVP, this may indicate that the CCP is passing invalid information in the parameter list or record area.

Halts U2 – UA

Reason: An unexpected return code: not 00 received on an output operation or not 00 or 01 received on an input operation. U2 through UA correspond to a particular operation issued in the program at a particular time.

Recovery:

- Press the HALT/RESET or START key; completion code is printed on the console and input is required.
- Take a dump of the main storage contents by pressing SYSTEM RESET and START, or by using the appropriate stand-alone main storage dump program.

Note: Model 4 CCPIVP does not use the U6 and U8 halts.

After pressing the HALT/RESET or START key on a U2 through UA halt, the following is printed:

TNAME-ssssss, UNEXPCTD RET CODE rrrr ENTER TA OR xx

ssssss	=	Symbolic terminal name of console (5471), always CONSOL.
rrrr	=	Return code in hexadecimal.
TA	=	Operation retried.
xx	=	Any characters other than TA; program goes to EOJ.

Halt Ud (ABD123 on Model 4)

Reason: An error has occurred using the disk file and control has returned to the application program indicating a non-recoverable condition.

Recovery:

- Press the HALT/RESET or START key; CCPIVP goes directly to EOJ.
- Take a dump of the main storage contents by pressing SYSTEM RESET and START, or by using the appropriate stand-alone main storage dump program.

Halt UP (ABD2345 on Model 4)

Reason: An error has occurred in using the printer and control has returned to CCPIVP indicating a non-recoverable error.

Recovery:

- Press the HALT/RESET or START key; CCPIVP goes directly to EOJ.
- Take a dump of the main storage contents by pressing SYSTEM RESET and START, or by using the appropriate stand-alone main storage dump program.

```
                OVERLAY LINKAGE EDITOR CORE USAGE MAP AND CROSS REFERENCE LIST

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

 1500         0       CCPIVR          0C04    3076
 2104         2       $$CSOP          001D      29    CCPIVR
 2121         2       $$CSIP          0027      39    CCPIVR
 2148         2       $$SRBR          0079     121    $$CSIP  $$CSOP
 21C1         2       $$SRUA          0026      38    $$CSIP  $$CSOP
 21E7         2       $$SRDF          001C      28    $$CSOP
 2203         2       $$SRTC          001C      28    $$CSIP  $$CSOP  $$SRDI
 2203                   DMSRLO
 2214                   DMSRTC                        $$CSIP  $$CSOP
 2217                   DMSRER                        $$CSIP  $$CSOP  $$SRDI
 221F         2       $$SRMO          0081     129    $$SRBR
 22A0         2       $$SRSB          0043      67    $$SRBR
 22E3         2       $$SRDI          0038      56    $$SRSB  $$SRBR
 2302                   DMSRPD                        $$SRSB
 22FB                   DMSRRD                        $$SRSB
 231B         2       $$SRBP          002F      47    $$SRSB  $$SRUA

OL100 I  THE TOTAL CORE USED BY CCPIVP IS  3658 DECIMAL.
OL101 I  THE START CONTROL ADDRESS OF THIS MODULE IS 1500.
OL104 I  TOTAL NUMBER OF LIBRARY SECTORS REQUIRED IS  16
         NAME-CCPIVP,PACK-CCPOBJ,UNIT-R1,RETAIN-P,LIBRARY-O
```

Note: This version of CCPIVP is not link edited to run with the printer.

Figure 13-3. Sample Linkage Edit Map of CCPIVP (Models 8, 10, 12)

Installation Verification Program (CCPIVP)

ENTRY POINT: CCPIVP
CHART: ZP
INPUT:
- Data with program request (8 characters). Specified to be MM/DD/YY but can be any 8 characters.
- Input from console to build disk file (3 characters).
- Input from console to end input for disk file (/*).
- Input from console to dump to printer or console (P for printer, any other character for console).
- Input from created disk file—DDD*PPPPPPPP*NNN where:
  - DDD – three characters keyed from input 2 above.
  - PPPPPPPP – 8 characters keyed with program request.
  - NNN – 3-digit record count kept by CCPIVP.
- Input from console when unexpected return code message issued:
  - TA – try again, repeat failed operation.
  - Any other character, go to abnormal end of job.

OUTPUT:
- Messages to console – normal operational messages:
  - Prompting messages.
  - End of job.
- Disk records to console or printer.
- Messages to console – errors:
  - Invalid data with program request.
  - Data from console not 3 characters or /*.
  - P entered from console but printer data management not included at link edit.
  - Abnormal end of job.
  - Unexpected return code.

EXTERNAL ROUTINES:
- $$CSOP – consecutive output data management.
- $$LPRT, $NLPRT – for printer on non-RPG II packs (Models 8, 10, and 12).
- $$LPRT, $$UPRT – for printer on RPG II packs (Models 8, 10, and 12).
- $$BDMC (Model 4 CCPIVP)

EXITS, NORMAL:
- Disk file build complete.
- End of job for CCPIVP.

EXIT, ERROR: Program has encountered unrecoverable errors; disk file closed.

NOTES:
- Two disk DTFs; both use filename CGIVFILE.
- In order to handle symbolic files and release of requester, both DTFs are opened before input is accepted so that the actual file name is placed in both DTFs when closed.
- CCP Generation, Assignment and link edit must be run before loading under CCP.
- $$LPRT is a weak EXTRN (Models 8, 10, 12 CCPIVP).
- Stand-alone halts are issued when 'should not occur' situations occur:
  - Unexpected return codes.
  - An unrecoverable disk or printer error has occurred and CCPIVP has received control.

These halts are issued at different points within CCPIVP providing the user with places to dump main storage.

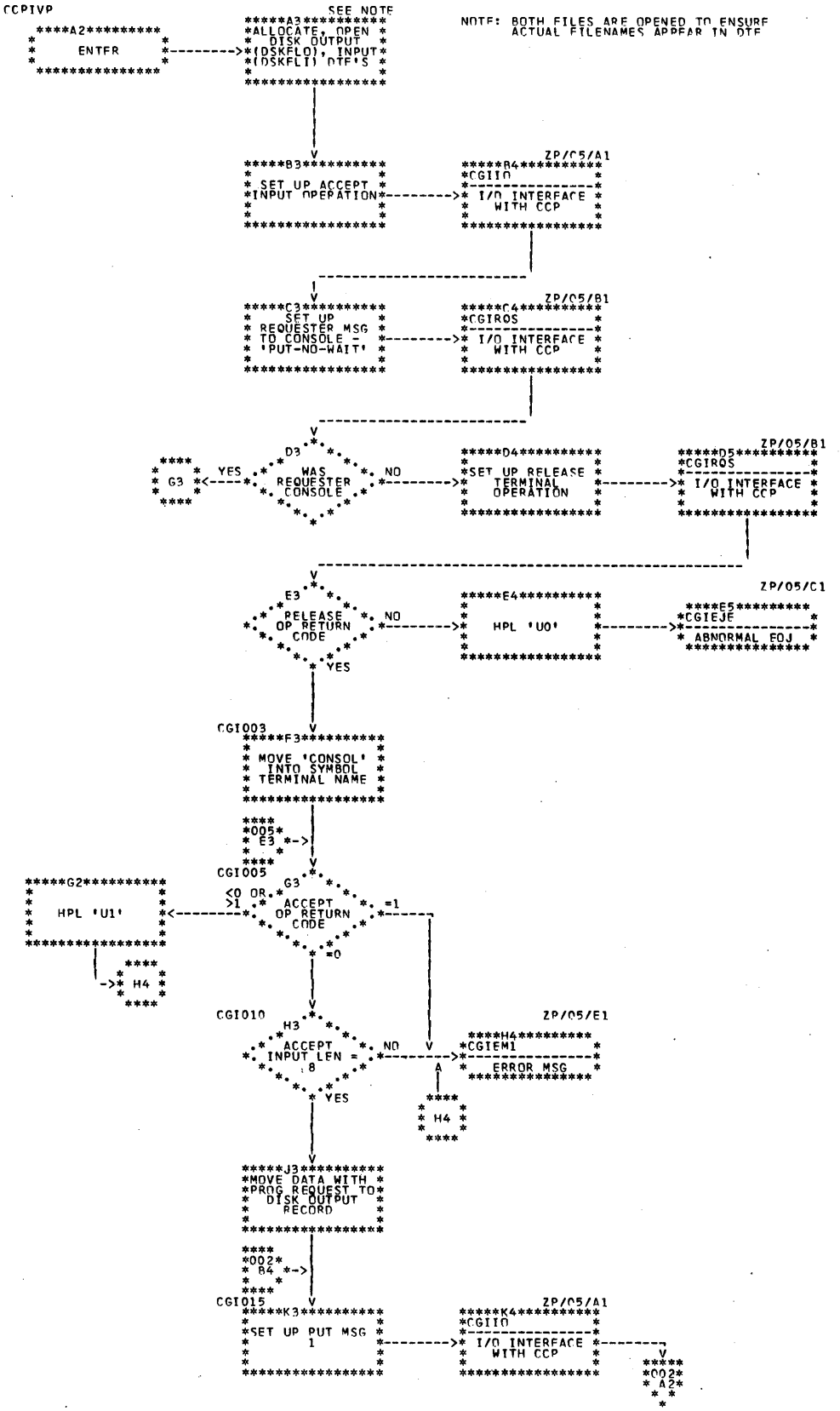
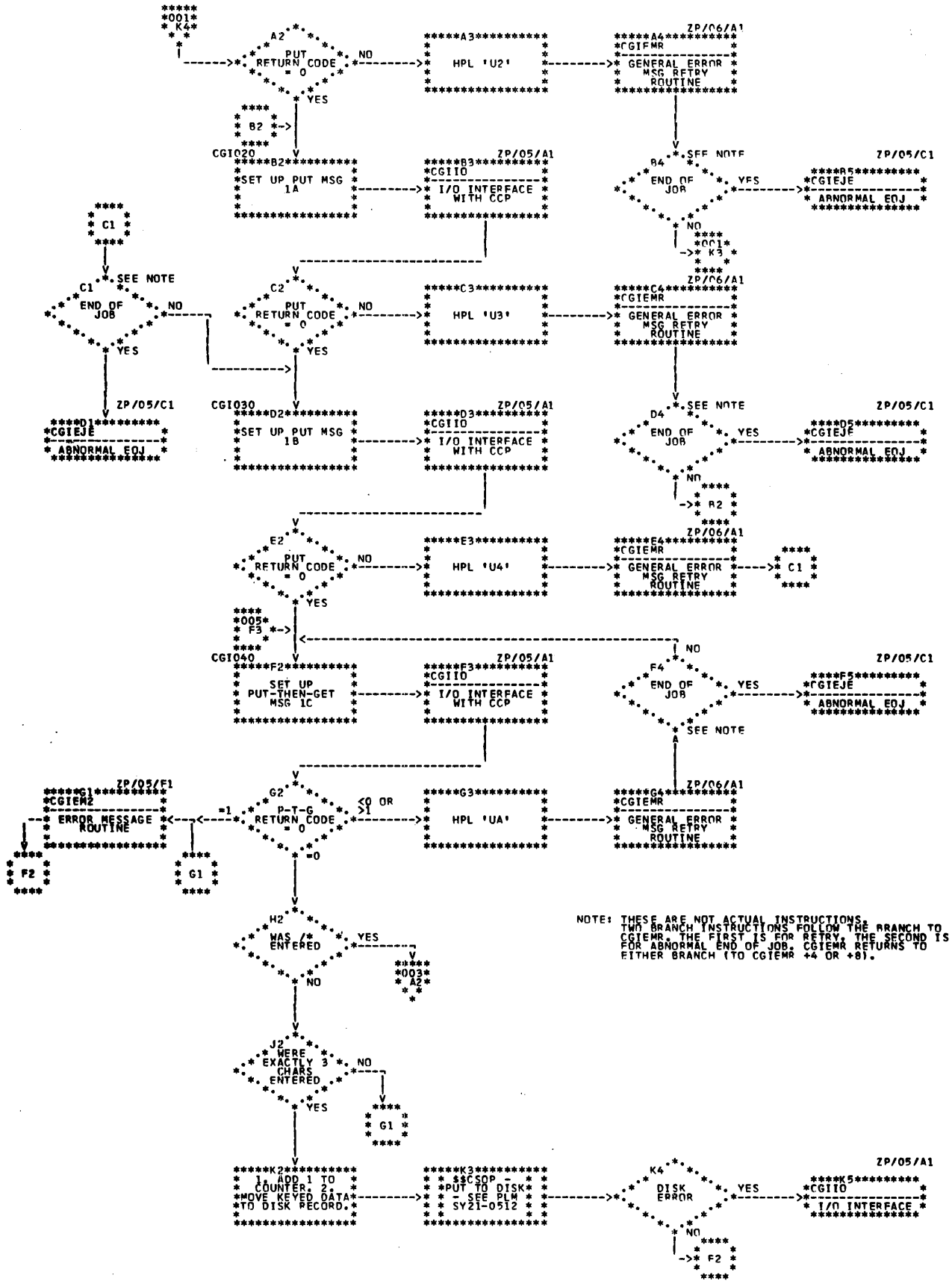


Chart ZP (Part 1 of 6). Installation Verification Program (CCPIVP)



NOTE: THESE ARE NOT ACTUAL INSTRUCTIONS. TWO BRANCH INSTRUCTIONS FOLLOW THE BRANCH TO CGIEMR. THE FIRST IS FOR RETRY, THE SECOND IS FOR ABNORMAL END OF JOB. CGIEMR RETURNS TO EITHER BRANCH (TO CGIEMR +4 OR +8).

Chart ZP (Part 2 of 6). Installation Verification Program (CCPIVP)

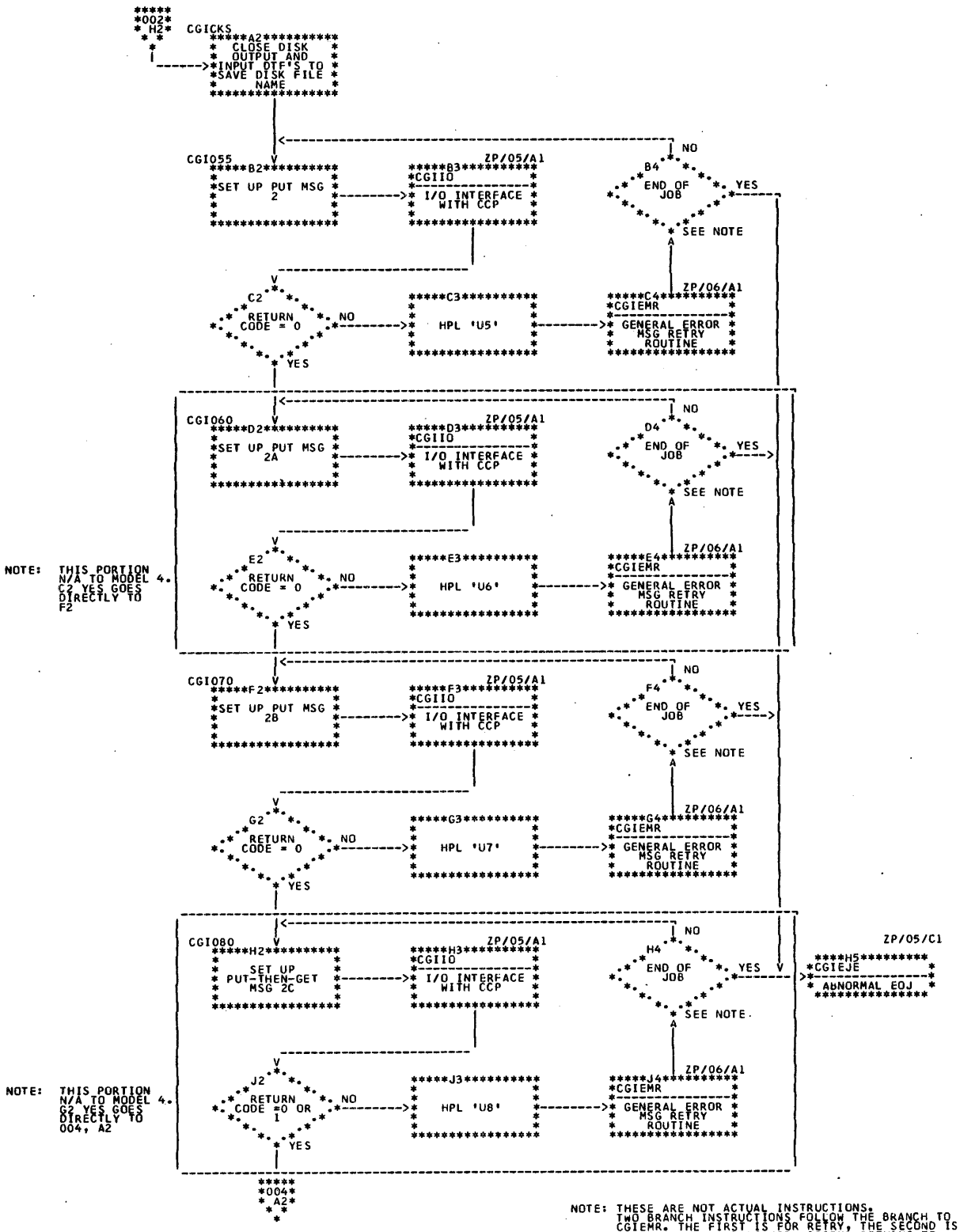


Chart ZP (Part 3 of 6). Installation Verification Program (CCPIVP)

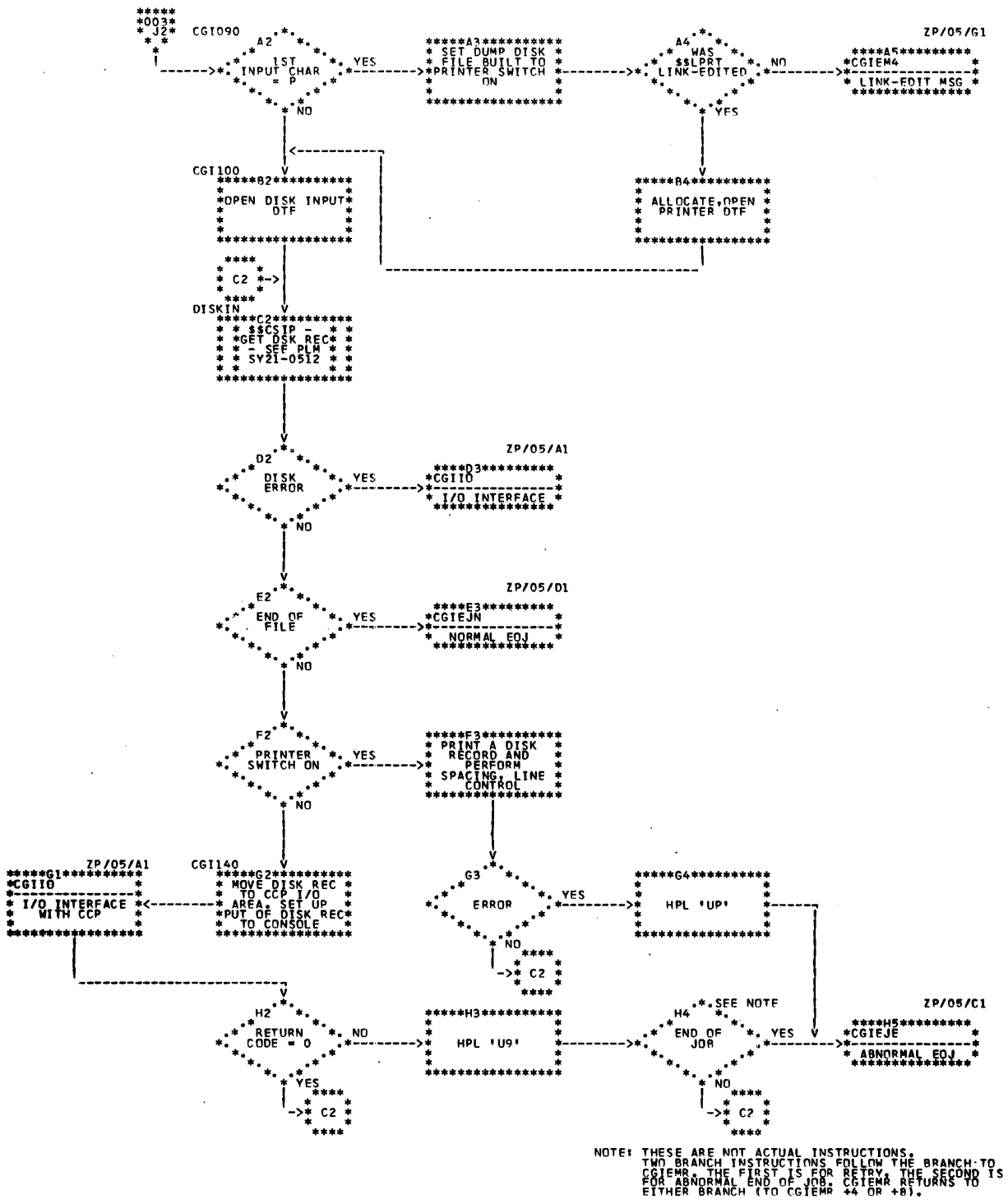
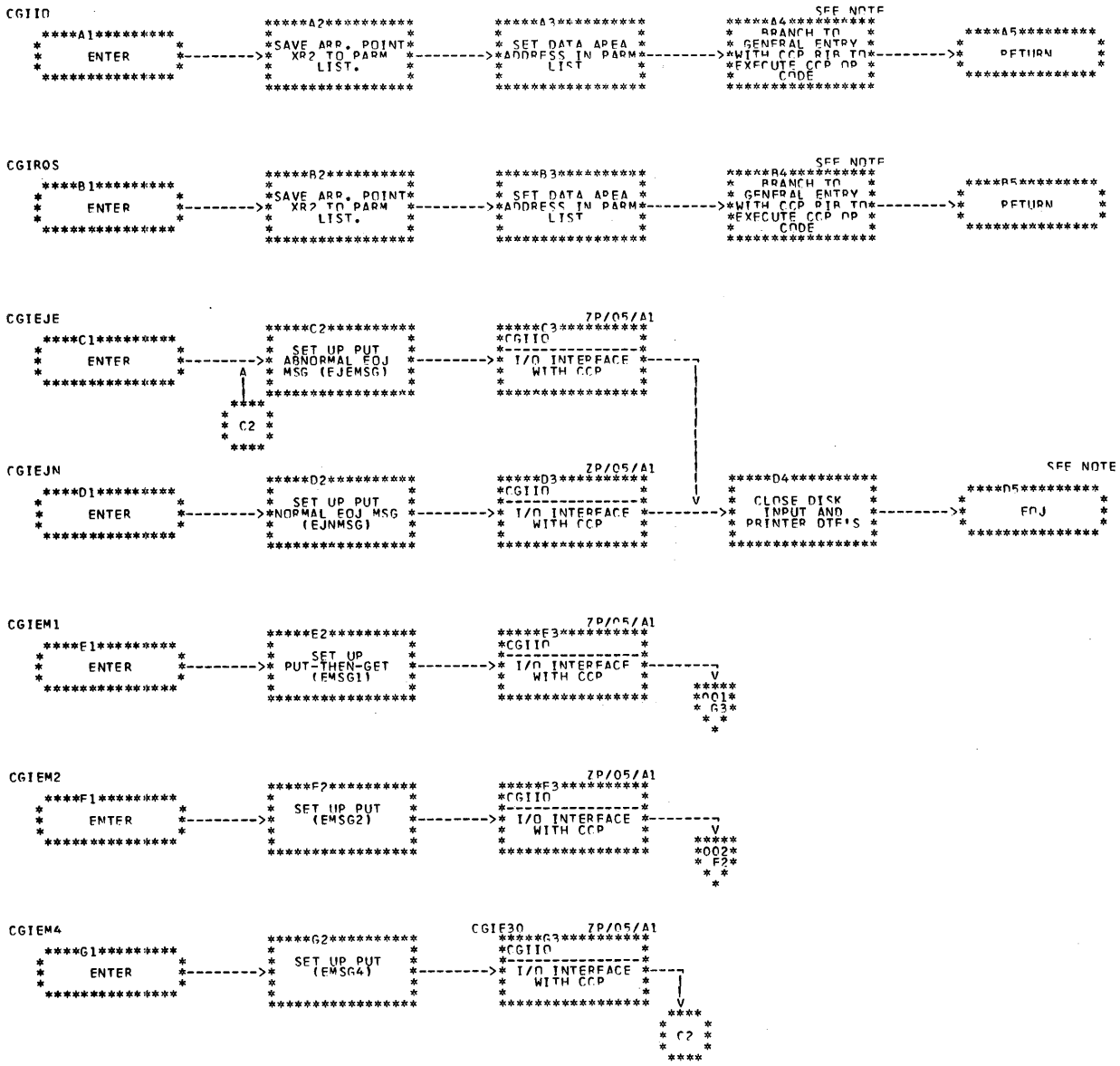


Chart ZP (Part 4 of 6). Installation Verification Program (CCPIVP)



NOTE: SEE IBM SYSTEM/3 DISK SYSTEMS SYSTEM CONTROL PROGRAM LOGIC MANUAL, SY21-0502.

Chart ZP (Part 5 of 6). Installation Verification Program (CCPIVP)

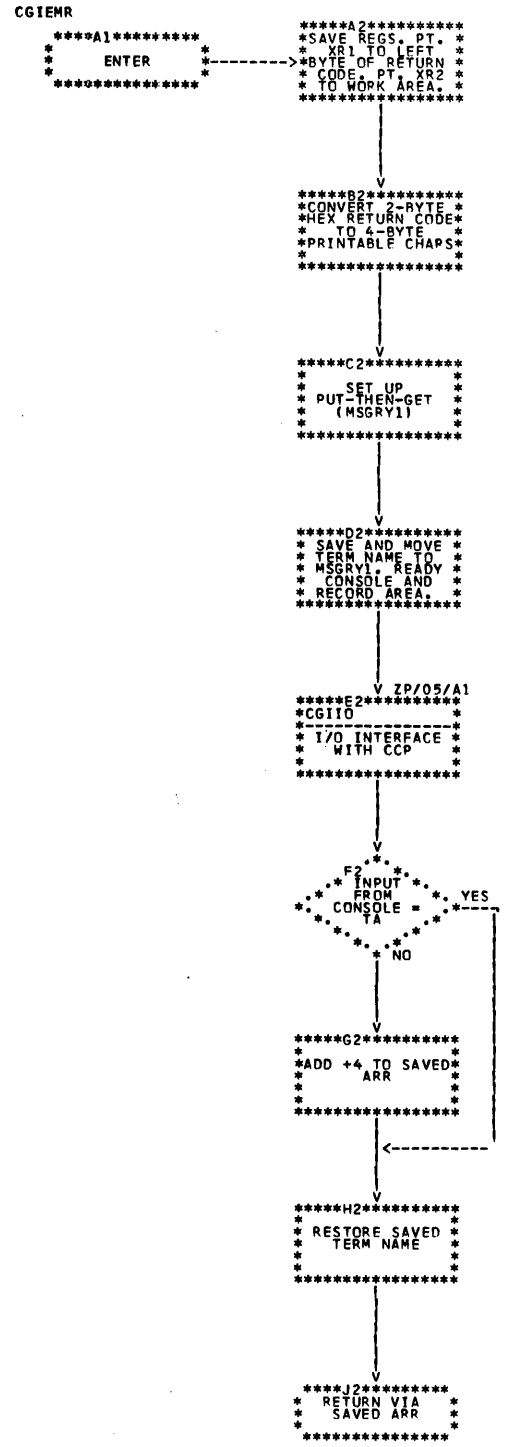


Chart ZP (Part 6 of 6). Installation Verification Program (CCPIVP)

Chapter 14. File Recovery Program ($CCPRB)

Introduction

The File Recovery Program ($CCPRB) is a stand-alone CCP support program that runs under control of DSM.

The Disk File Recovery Program's purpose is to assist the CCP user in disk file recovery if the CCP is abnormally terminated.

A normal CCP termination is by the shutdown procedure, which includes updating file pointers and closing of the files. Any other method of termination is considered abnormal, such as a power failure, a process check, or a U — halt.

A CCP abnormal termination does not update the disk file pointers or close the files. This condition leaves sequential and indexed files with added records that are inaccessible by normal methods.

$CCPRB must be the first program run after an abnormal CCP termination. The IPL must be from the same pack for which the CCP IPL process was performed. All disk packs that CCP uses must be in their original location and online. On a DPF system, the file recovery program must be loaded if the same level CCP was running.

Method of Operation

The following OCL statements are required:

```
// LOAD $CCPRB,unit
// FILE UNIT-unit,PACK-packname,NAME-$CCPFILE
// RUN
```

The // FILE statement must describe the same $CCPFILE that the abnormally terminated CCP used.

File recovery is based on the format 1's in the SWA built during CCP startup. CCP startup saves the first twelve format 1's (three sectors) in the $CCPFILE. The three sectors saved are placed in the three sectors preceding the CCP dump area. When saved, the last three bytes of the first format 1 contain a pointer to the last, used format 1 in the SWA.

The file recovery program retrieves these twelve format 1's by use of the // FILE statement for the $CCPFILE and the dump area pointer in the $CCPFILE directory. The remaining format 1's are retrieved directly from the SWA on the system pack.

Each format 1 is checked and the following actions taken:

Open Consecutive Add or Load Files: If the user requests the ADD option, the format 1 end-of-data pointer is set to equal the end-of-file pointer.

If the user requests the CLOSE option, the format 1 pointers are left as they were.

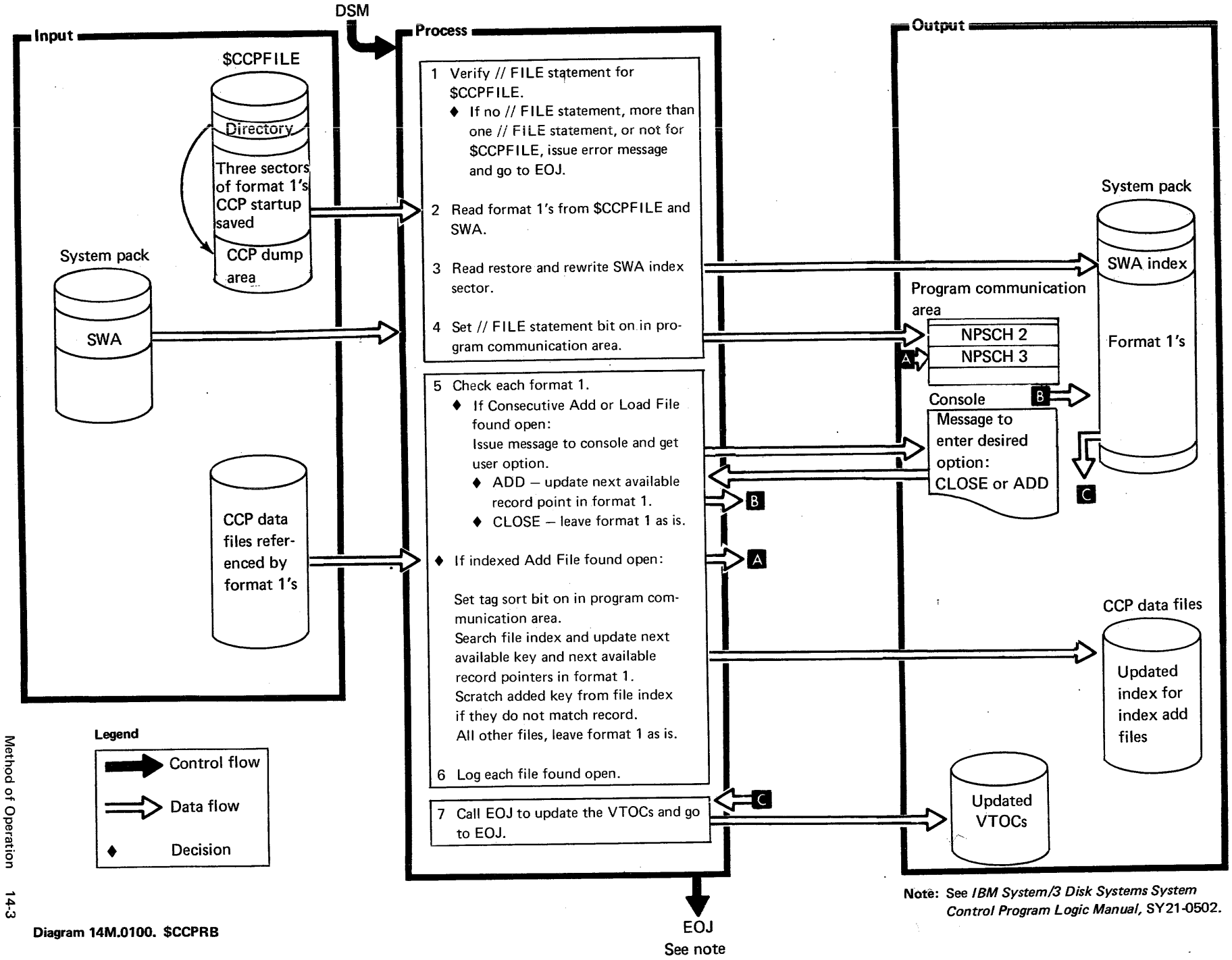
Open Indexed Add Files: The end-of-index pointer is set to the first free position following the last good key. The last good key is determined by searching the added keys for the X'FF' following the last added key. The record pointed to by the key preceding the X'FF' is checked for the same key. If the keys do not match, the preceding key is used for the same check. When a match of keys is found, the key location within the record is checked.

If the record does *not* end in the same sector that contains the key, then the preceding key is used as the last key. If the record does end in the same sector that contains the key, then this key is used as the last key. All keys between the last good key and the X'FF' after the last added key are set to X'FF'. The end-of-data pointer is set to the first free data position based on the last good key in the index.

All Other Files: The format 1's are left as they were.

Updating the VTOCs: End-of-job updates the required VTOCs based on the updated format 1's in the SWA and closes all files.

See Diagram 14M.0100 for an overview of $CCPRB.



Note: See *IBM System/3 Disk Systems System Control Program Logic Manual*, SY21-0502.

Diagram 14M.0100. $CCPRB

File Recovery Program ($CCPRB)

ENTRY POINT: RBEGIN
CHART: None
FUNCTION: Recovers added records to CCP files if the CCP is abnormally terminated. Recovery is attempted on consecutive add, consecutive load, and indexed add files.
OPERATIONS:
- Checks for valid // FILE statement for $CCPFILE.
- Reads three sectors of format 1's that CCP startup saved in $CCPFILE.
- Reads other seven sectors of format 1's from SWA.
- Restores and rewrites SWA index sector.
- Checks each format 1 and updates as follows:
  - If not open, does nothing.
  - If open direct file, does nothing.
  - If consecutive add or load, gives the user an option:
    CLOSE – Leaves pointers as they were. All records added by the last CCP run are lost.
    ADD – Makes end-of-data pointer equal to the end-of-file pointer.
  - If indexed add:
    - Sets end-of-index pointer to first free position following the last good key.
    - Sets end-of-data pointer to first free position determined by the index.
    - Determines the last good key.
  - If all other files are consecutive and indexed, does nothing.
- Logs each file found open.

INPUT:
- Three sectors of format 1's from $CCPFILE.
- Seven sectors of format 1's from system SWA.
- Data files opened format 1's point to.

OUTPUT:
- Format 1's in SWA with updated end-of-data and end-of-index pointers.
- Updated VTOC (updated by EOJ)

EXIT: EOJ is called with RIB of X'84'.
EXTERNAL REFERENCES:
- System communication area
- Program communication area

Appendix A. System Directory

The system directory contains quick-reference tables that can be used to find information within this manual as well as to find CCP module/phase listings in the Microfiche. Each directory table contains:

- Production pack module/phase names.
- PID (Program Information Department) pack module/phase names.
- Entry points.
- Descriptive module/phase names.
- A summary of function for each module/phase.
- Diagram identifications of method of operation and HIPO diagrams that describe the modules/phases listed.
- Flowchart identifications of flowcharts that describe the modules/phases listed.

The directory tables are arranged to reflect the organization of the manual. That is, the Generation table is first, the SCP Generator table is second, the Assignment Build table is third, and so on. Module/phase names within each table are arranged in alphabetic sequence; they are not necessarily mentioned in the order in which the modules/phases are executed.

Generation/Installation

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CC1BF	$CC1BF	EXECBF	Build Initial Contents of $CCPFILE	Clears $CCPFILE to all blanks. Creates initial configuration and directory records.	None	AB
$CC1BL	$CC1BL	$CC1BL	Creates $CCPLOG	Creates $CCPLOG file.		AA
$CC1PP[1]	$CC1PP	PPEXEC	CCP Generation Utility	Prints user input to Generation and diagnostics issued. If no errors, punches or reorganizes input to second pass of Generation.		

SCP Generator (Models 8, 10, and 12 Only)

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CGNBX	$CGNBX	BXI000	Build XREF File Phase	Reads the work file ($WORK2) and creates the cross reference sort file. Initializes working storage, print area, and listing header for $CGNSX.	3M.0010 3M.0020	HA-HF
$CGNCM	$CGNCM	CMI000	Source Compression Phase	Processes all source statements following the control cards. Builds the intermediate text file and decodes and tests the source statements.	3M.0010 3M.0020	BB-BH
$CGNIN	$CGDRV	INI000	Processor Initialization Phase	Initializes the processor	3M.0010 3M.0020	BA
$CGNPE	$CGNPE	PEI000	ESL Output Phase	Sorts and prints the ESL listing if LIST is specified. Initializes COMMON for $CGNPS.	3M.0010 3M.0020	FA-FF
$CGNPS	$CGNPS	PSI000	Source/Object Output Phase	Generates object code for machine instructions and DCs. Puts out object code. Builds printer image. Prints source statements and generated object code. Performs source time instructions and processes Group 1 and Group 2 instructions.	3M.0010 3M.0020	GA-GI
$CGNSB	$CGNSB	SBI000	Symbol Table Build Phase	Assigns the location counter values and builds the symbol table.	3M.0010 3M.0020	CA-CK

[1]Models 8, 10, and 12 only

SCP Generator (Models 8, 10, and 12 Only) (continued)

Phase	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CGNSF	$CGNSF	SFI000	Symbol Table Overflow Phase	Tests the intermediate text from the statement following the overflow to the end of the text for symbols that have been defined previously. (Previously defined symbols are marked.) Resolves all references to symbols that are defined in the present symbol table. (Resolved symbols are marked.)	3M.0010 3M.0020	DA-DD
$CGNSS	$CGNSS	SSI000	Symbol Substitution Phase	Substitutes the symbol value and attribute from the last (or only) symbol table into the intermediate text term records. Builds an ESL table for module name and EXTRN and ENTRY statements.	3M.0010 3M.0020	EA-EE
$CGNSX	$CGNSX	SXI000	Merge and List XREF Phase	Merges the XREF sort file. Generates the XREF listing. Prints the error summary statements. Fetches Overlay Linkage Editor if object output is required.	3M.0010 3M.0020	IA-IE
CAM	CAM	CAM001	Compiler Access Method	Retrieves or loads as many as 255 sectors at a time.	3M.0010	JA

Assignment Build

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CCPAS	$CCPAS	SA0000	Assignment Build Mainline	Assignment Build control	4M.0100	4P.0100 4P.0110 4P.0120 4P.0130 4P.0140
$CC2BL	$CCYBL	BL0000	BSCALINE State-ment Processor	Builds LCT table entry.	4M.0100	4P.0500
$CC2BT	$CCYBT	BT0000	BSCATERM State-ment Processor	Builds TUT table entry	4M.0100	4P.0600
$CC2DF	$CCYDF	DF0000	DISKFILE/ SYMFILEState-ment Processor	Builds FCT table.	4M.0100	4P.1000
$CC2EA	$CCYEA	$CC2EA	Error message module	Contains messages for Assign Build.	None	None
$CC2EB	$CCYEB	$CC2EB	Error message module	Contains messages for Assign Build.	None	None
$CC2EC	$CCYEC	$CC2EC	Error message module	Contains messages for Assign Build.	None	None
$CC2ED	$CCYED	$CC2ED	Error message module	Contains messages for Assign Build.	None	None
$CC2E1	$CCYE1	$CC2E1	Error message module	Contains messages for Assign Build.	None	None
$CC2E2	$CCYE2	$CC2E2	Error message module	Contains messages for Assign Build.	None	None
$CC2E3	$CCYE3	$CC2E3	Error message module	Contains messages for Assign Build.	None	None
$CC2E4	$CCYE4	$CC2E4	Error message module	Contains messages for Assign Build.	None	None
$CC2E5	$CCYE5	$CC2E5	Error message module	Contains messages for Assign Build.	None	None
$CC2E6	$CCYE6	$CC2E6	Error message module	Contains messages for Assign Build.	None	None
$CC2E7	$CCYE7	$CCYE7	Error message module	Contains messages for Assign Build.	None	None
$CC2E8	$CCYE8	$CC2E8	Error message module	Contains messages for Assign Build.	None	None

Assignment Build (continued)

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CC2E9	$CCYE9	$CC2E9	Error message module	Contains messages for Assign Build.	None	None
$CC2HK	$CCYHK	$CC2HK	End of SET Routine	Modifies $CCPFILE.	4M.0100	4P.1200
$CC2ML[1]	$CCYML	$CC2ML	MLTALINE Statement Processor	Builds SLCT table entry.	4M.0100	4P.0700
$CC2MT[1]	$CCYMT	$CC2MT	MLTATERM Statement Processor	Builds TUT table entry.	4M.0100	4P.0800
$CC2PG	$CCYPG	$CC2PG	Program Statement Processor	Builds PCT table.	4M.0100	4P.1100
$CC2SP	$CCYSP	$CC2SP	Program Statement Parameter List	Lists data only.	4M.0100	None
$CC2SS	$CCYSS	SS0000	Syntax Scan Routine	Performs syntax check on parameters of input control statements.	4M.0100	4P.0200
$CC2SY	$CCYSY	$CC2SY	SYSTEM Statement Processor	Builds SIT table.	4M.0100	4P.0300
$CC2TA	$CCYTA	$CC2TA	TERMATTR Statement Processor	Builds TAT table.	4M.0100	4P.0400
$CC2TN	$CCYTN	TN0000	TERM Name	Builds TNT table.	4M.0100	4P.0900

Assignment List

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CCPAL	$CCPAL	ALA000	Assignment List Module	Lists assignment set from specifications.	5M.0100	5P.0100 5P.0200 5P.0300 5P.0400

User Security Information Build (Models 8, 10, and 12 Only)

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CCPAU[1]	$CCPAU	AU0000	User Security Data Module	Puts user security data to the user security module on disk.	6M.0100	MU

[1]Models 8, 10, and 12 only.

Display Format Generator

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CCPDF	$CCPDF	BLDFMT	DFGR—Build LOMMON and open files	Builds OLE common area and opens $WORK and $SOURCE files.	7M.0200	MV
$CC2CF	$CCZCF	BLDFMT	DFGR—Print and diagnose display specifications	Logs the display format specifications, diagnoses errors, and builds the format and common area.	7M.0400	MY
$CC2CP	$CCZCP	$CC2CP	DFGR—Format generation print and file-build	Logs information about the printer format being generated, and copies the format to a $WORK file on disk.	7M.0500	MZ
$CC2CR	$CCZCR	BLDINP	DFGR—Read multiple from $SOURCE file	Reads display format specifications from $SOURCE file.	7M.0300	MW

Printer Format Generator

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CCPPF	$CCPPF	BLDLOM	PFGR--Build LOMMON and open files	Builds OLE common area and opens $WORK and $SOURCE files.	7M.0800	M1
$CC2CG	$CCZCG	BLDFMT	PFGR--Print and diagnose printer format specifications	Logs the printer format specifications, diagnoses errors, and builds the format and common area.	7M.1000	M3
$CC2CQ	$CCZCQ	BLDFIL	PFGR--Format generation print and file-build	Logs information about the printer format being generated and copies the format to a $WORK file on disk.	7M.1100	M4
$CC2CS	$CCZCS	BLDINP	PFGR--Read multiple from $SOURCE file	Reads printer format specifications from $SOURCE file.	7M.0900	M2

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CCP	$CCP	SPEXEC	Startup Root Module	Accepts operator specifications and begins initializing CCP.	8M.0100	8P.0010
$CC3CE	$CC3CE	CEEXEC	Build CDEs	Builds Contents Directory Entry blocks.	8M.0100	8P.0150
$CC3CR	$CC3CR	CRBEGN	Cross Reference Transients	Cross-references CCP transients.	8M.0100	8P.0050
$CC3CX	$CC3CX	CXEXEC	REP Card Processor	Reads REP cards from system input, patches code initialized prior to $CC3CX	8M.0100	8P.0210
$CC3DF	$CC3DF	DFEXEC	Open Disk Files and Compress DTFs	Opens disk files, completes short DTFs, builds CCP-built master indexes.	8M.0100	8P.0180
$CC3DL	$CC3DL	DLEXEC	Build Skeleton DTFs and Symbolic File Entries	Validates OCL for disk files, creates and initializes short DTFs, builds symbolic file names and disk file reference pointers.	8M.0100	8P.0170
$CC3EJ	$CC3EJ	EJEXEC	Final External Startup Phase	Searches program pack object library directory for first level transients, sets designated terminals offline, re-sets storage bounds for dynamic TP buffer and user program area, and opens MLTA lines.	8M.0100	8P.0220
$CC3FB	$CC3FB	FBEXEC	Build File Specification Blocks	Builds file specification blocks.	8M.0100	8P.0200
$CC3FS	$CC3FS	FSEXEC	Facility Suppression	Accepts operator specifications and turns on designated suppress bits in $CCPFILE.	8M.0100	8P.0080
$CC3FX	$CC3FX	FXEXEC	DFF Format Indexer	Builds DFF format index in $CCPFILE.	8M.0100	8P.0040
$CC3IP	$CC3IP	IPEXEC	Initialize Program Control Table	Searches program pack and system pack object library directories for programs in program control table.	8M.0100	8P.0090
$CC3LD	$CC3LD	LDEXEC	Load Base Module and Initialize $CCCOM	Loads $CC4, CCTR, CCSA, CCBS, CCML and initializes many miscellaneous fields in $CCCOM.	8M.0100	8P.0100

Startup (continued)

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CC3LO[1]	$CC3LO	LOEXEC	Startup Phase Locator	Searches object library directory for Startup phases, and initializes roller list.	8M.0100	8P.0030
$CC3MV	$CC3MV	MVEXEC	Load DF and #3 into PL1	Initializes $@@BIN and $@CIN, sets up ATRs and loads $CC4DF and $CC4#3 into PL1.	8M.0100	8P.00215
$CC3PX	$CC3PX	PXEXEC	Build PCT Index	Builds index to Program Control Table names.	8M.0100	8P.0160
$CC3QB	$CC3QB	QBEXEC	Build Sector Enqueue Blocks	Builds sector enqueue blocks.	8M.0100	8P.0190
$CC3RO	$CC3RO	ROEXEC	Startup Phase Roller	Loads Startup phases consecutively.	8M.0100	8P.0020
$CC3RT	$CC3RT	RTEXEC	Relocate Transients	Relocates CCP transients to current execution address.	8M.0100	8P.0060
$CC3TA	$CC3TA	TAEXEC	Read in Terminal Attribute Table	Reads in terminal attribute table.	8M.0100	8P.0110
$CC3TB	$CC3TB	TBEXEC	Build Teleprocessing Control Blocks	Builds TP DTF/LCB and related control areas for both BSCA and MLTA.	8M.0100	8P.0120
$CC3TC	$CC3TC	TCEXEC	Build TUB and TNT Entries	Builds Terminal Unit Blocks and Terminal Name Table.	8M.0100	8P.0130
$CC3UB	$CC3UB	UBEXEC	Build Task Control Blocks	Builds user Task Control Blocks.	8M.0100	8P.0140
$CC3US	$CC3US	USEXEC	Unsuppress Facilities	Turns off all suppress bits in $CCPFILE.	8M.0100	8P.0070

[1] Models 8, 10, and 12 only.

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
CCML	CCML	$$MLDT MLT001 MLT002	CCP MLTA Trace Routine	Loadable CCP MLTA trace routine which builds a circular trace table for program level SIOs, interrupt level entries, and EXTRNs.	None	None
CCSA	CCSA	CCSA	Trace Halt Service Aid	Facilitates halting CCP at specified trace entries.	None	PZ
CCTR	CCTR	CCTR	CCP Trace	Saves information about significant events in CCP processing in chronological order.	None	PY
$CC4AB	$CCIAB	$CC4AB	Diagnose and Process Accept Input	Processes accept input requests prior to wait or completion of $CC4II.	9M.0050	9P.6010
$CC4AC	$CCIAC	$CC4AC	Process Satisfied Accept Input	Moves data, Freemains invite buffer, and returns.	9M.0050	9P.6020
$CC4AD	$CCAAD	$CC4AD	Disk Allocation Transient	Allocates disk files to user program.	9M.0240	9P.4060
$CC4AL	$CCSAL	$CC4AL	First Unit Record Allocation Transient	Diagnoses allocation requests.	9M.0210	9P.3550
$CC4AM	$CC4AM	$CC4AM	Allocation Resident Routine	Passes control to allocation transients or user program.	9M.0240	PF
$CC4AN	$CCSAN	$CC4AN	Second Unit Record Allocation Transient	Sets requests in SWA common sector.	9M.0210	9P.3560
$CC4AP	$CCSAP	$CC4AP	Third Unit Record Allocation Request	Sets requests in partition index.	9M.0210	9P.3570
$CC4AQ	$CCIAQ	$CC4AQ	First Terminal Acquire Processor	Validates and checks for terminal available.	9M.0050	9P.6130
$CC4AS	$CCSAS	$CC4AS	First Assignment Transient	Validates assign command and its operands.	9M.0210	9P.3520
$CC4AT	$CCSAT	$CC4AT	Third Assignment Transient	Validates STT# operand, changes terminal name if necessary.	9M.0210	9P.3530
$CC4AU	$CC5AU	$CC4AU	Second Assignment Transient	Checks that assigned TUB characteristics are compatible with previously assigned TUB.	9M.0210	9P.3540

CCP Execution (continued)

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CC4AX	$CCIAX	$CC4AX	Second Terminal Acquire Processor	Validates terminal attribute set.	9M.0050	9P.6140
$CC4AY	$CCIAY	$CC4AY	Third Terminal Acquire Processor	Stop invites and allocates the TUB.	9M.0050	9P.6150
$CC4A1	$CCAA1	$CC4A1	Allocation Control Transient	Sets up allocation and controls for the first pass route control.	9M.0240	9P.4010
$CC4A2	$CCAA2	$CC4A2	Final Allocations Transient	Allocates requesting terminal.	9M.0240	9P.4090
$CC4BA	$CCBBA	BASTRT	3270 Sense/Status Transient	Analyzes 3270 sense/status and logs the results.	9M.0110	9P.2370
$CC4BB	$CCBBB	BBSTRT	Blank Pad Transient	For record mode operations, if OUTL is less than record length, pads to correct length with blanks.	9M.0110	9P.2321
$CC4BC	$CCBBC	BCSTRT	BSCA Cancel Receive Transient	Handles priority stops of line or stop invites for system and user. Issues cancel to BSCA.	9M.0130	9P.2510
$CC4BD	$CCBBD	BDSTRT	BSCA ERP Cleanup Transient	Cleans internal indicators after ERP is run.	9M.0160	9P.2341
$CC4BE	$CCBBE	BESTRT	BSCA Error Handler/Router Transient	Determines error situations and handles them. Sets up to log conditions to operator.	9M.0110	9P.2330
$CC4BF	$CCBBF	BFSTRT	BSCA Stop Invite Transient	Handles system/user stop invites and calls.	9M.0130	9P.2520
$CC4BI	$CCBBI	BISTRT	DME Inquiry Transient	Analyzes input record for DME character string.	9M.0110	9P.2320
$CC4BL	$CCBBL	BLSTRT	BSCA Error Logger Transient	Logs out operator message to console log.	9M.0110	9P.2340
$CC4BN	CCBBN	CC4BN	Level 2 Interrupt Handler	Handles level 2 interrupt.	None	9P.2345
$CC4BP	$CCBBP	BPSTRT	BSCA Purge Transient	Handles system purge I/O request and setup to stop invite if gets outstanding.	9M.0130	9P.2500
$CC4BQ	$CCBBQ	BQSTRT	BSCA Stop Invite Queue Analysis Transient 1	Analyzes the LCB line Q for stops after BSCA cancels the input from $CC4BC (see *$CC4BU* for Transient 2)	9M.0110	9P.2325

CCP Execution (continued)

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CC4BR	$CCBBR	BRSTRT	Operation Reject Transient	Rejects TP requests that conflict with current BSCA line operations.	9M.0110	9P.2360
$CC4BS	$CCBBS	$CC4BS	BSCA Switched Line Purge	Purges operation parameter list on a BSCA line.	9M.0130	9P.2530
$CC4BT	$CCBBT	$$BSMT	BSCA Trace Interface	Interfaces with BSCA trace.	None	QZ
$CC4BU	$CCBBU	BUSTRT	BSCA Stop Invite Queue Analysis Transient 2	Analyzes the LCB line Q for stops after BSCA cancels (see *$CC4BQ* for Transient 1).	None	9P.2325
$CC4BX	$CCBBX	$CC4BX	Message Module Negative Response to Polling	Sets message parameters in *$CCCOM*, and invokes console management.	None	9P.2342
$CC4B0	$CCBB0	B0STRT	3270 Format Transient	Formats output for the 3270, re-formats the input from the 3270.	9M.0140	9P.2390
$CC4B1	$CCBB1	B1STRT	Minimum System Transient 1	Normal resident code executed in the transient area for minimum system.	None	9P.2710
$CC4B2	$CCBB2	B2STRT	Minimum System Transient 2	Normal resident code executed in the transient area for minimum system.	None	9P.2720
$CC4B3	$CCBB3	B3STRT	Minimum System Transient 3	Normal resident code executed in the transient area for minimum system.	None	9P.2730
$CC4B5	$CCBB5	B5STRT	3735 Sense/Status Transient	Analyzes 3735 status messages and logs the results.	9M.0110	9P.2380
$CC4B7	$CC4B7	B7STRT	3741 Sense/Status Transient	Analyzes 3741 status messages and logs the results.	9M.0110	9P.2385
$CC4B9	$CCBB9	$CC4B9	Snap Dump of BSCA Trace	Snap dumps the BSCA trace in main storage.	9M.0160	9P.2350
$CC4CA	$CCSCA	$CC4CA	Display User's Command for DFF Main Storage	Lists DFF tasks.	9M.0210	9P.3790
$CC4CG	$CCSCG	$CC4CG	Message Command, System Operator to Terminal	Does a put-no-wait to receiving terminal if able to receive; if not, calls message module.	9M.0210	9P.3400
$CC4CJ	$CCSCJ	$CC4CJ	System Operator Cancel Command Processor	Validates cancel command, sets task or CCP cancel indicators.	9M.0210	9P.3610

CCP Execution (continued)

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CC4CM	$E000	N/A	Communications Management Task	Controls CCP teleprocessing I/O operations.	9M.0090 9M.0095 (Model 4)	QA through QW, RF
$CC4CN	$CCTCN	$CC4CN	Terminal Release Command Processor	Validates release command and frees the terminal from the program.	9M.0220	9P.3260
$CC4CP	$CC4CP	$CC4CP	CMD Processor Resident Routine	Routes control to processor transients and waits for work.	9M.0200	PC
$CC4CR	$CCACR	$CC4CR	First Main Storage Allocation Transient	Builds control blocks.	9M.0240	9P.4020
$CC4CR	$CCDCR	$CC4CR	First Main Storage Allocation Transient	Builds control blocks (remap only).	9M.0240	9P.4020
$CC4CS	$CCACS	$CC4CS	Second Main Storage Allocation Transient	Builds control blocks.	9M.0240	9P.4030
$CC4CT	$CCACT	$CC4CT	Third Main Storage Allocation Transient	Builds control blocks.	9M.0240	9P.4040
$CC4CX	$CCSCX	$CC4CX	Cancel CCP Processor	Marks all tasks to be terminated and posts termination. Notifies command terminals of cancel.	9M.0210	9P.3620
$CC4C1	$CCSC1	$CC4C1	Display Command Control	Does a syntax check of input operand and calls display transient or issues error message.	9M.0210	9P.3700
$CC4C2	$CCSC2	$CC4C2	Display Replies Command	Display user messages that are awaiting replies.	9M.0210 None	9P.3710 9P.8020 (Model 4)
$CC4C3	$CCSC3	$CC4C3	Display TERMATTR Command	Lists a specified terminal or all terminals by symbolic name.	9M.0210	9P.3720
$CC4C4	$CCSC4	$CC4C4	Display Terminals Command One	Lists by physical ID one or all terminals.	9M.0210	9P.3730

CCP Execution (continued)

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CC4C5	$CCSC5	$CC4C5	Display Users Command One	Prints header line and first part of status line.	9M.0210	9P.3740
$CC4C6	$CCSC6	$CC4C6	Display Queue Command	Lists by name any programs that are queued to run.	9M.0210	9P.3750
$CC4C7	$CCSC7	$CC4C7	Display Users Command by ID	Lists all resources used by task.	9M.0210	9P.3760
$CC4C8	$CCSC8	$CC4C8	Display Terminals Command Two	Lists all terminals.	9M.0210	9P.3770

This page intentionally left blank.

CCP Execution (continued)

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CC4C9	$CCSC9	$CC4C9	Display Users Command Two	Lists disk files used by task.	9M.0210	9P.3780
$CC4DA	$CCADA	DAA000	Initialize PAS Work Area and Constants	Initializes program appended storage for a task which uses DFF.	9M.0240	9P.4050
$CC4DB	$CCFDB	DBA000	Diagnose Use of DFF Put Override	Diagnoses DFF put override list.	9M.0060	9P.6210
$CC4DC	$CCFDC	DCA000	Build DFF Copy Parameter List	Diagnoses use of copy operation and builds 3270 text.	9M.0060	9P.6220
$CC4DD	$CCFDD	DDA000	Issue System Console Message 528	Builds message 528 and calculates position error occurred.	9M.0060	9P.6230
$CC4DE	$CC4DE	DEA000	Calculate 3270 Put Override Text Length.	Calculates the 3270 put override text length.	9M.0060	9P.6240
$CC4DF	$CC4DF	DFA000	Display Format Control Routine	Provides logical support for the 3270 System under CCP.	9M.0060	9P.6200
$CC4DI	$E055	$CC4DI	CCP File Sharing Enqueue/Dequeue Routine	Permits file sharing between user tasks.	None	ON
$CC4DM	$CCTDM	$CC4DM	Data Mode Escape Transient	Issues initial command interrupt mode I/O to terminal.	9M.0200	9P.3100
$CC4DP	$E040	$CC4DP	Task Dispatcher and I/O Scheduler	Dispatches ready system and user tasks; performs task switching on interrupts; schedules disk I/O; performs CCP waits.	9M.0070	OH
$CC4EA	$CCEEA	$CC4EA	Command Processor Message Module	Contains messages, moves messages to Command Processor work area.	9M.0230	9P.3060
$CC4EB	$CCEEB	$CC4EB	Command Processor Message Module	Contains program request error messages.	9M.0230	9P.3060
$CC4EC	$CCEEC	$CC4EC	Command Processor Message Module	Signs off error messages.	9M.0210 9M.0220	9P.3060
$CC4ED	$CCEED	$CC4ED	Command Processor Message Module	Signs off error messages.	9M.0220	9P.3060

CCP Execution (continued)

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CC4EE	$CCEEE	$CC4EE	Command Processor Message Module	Signs off error messages.	9M.0210	9P.3060
$CC4EF	$CCEEF	$CC4EF	Command Processor Message Module	Contains messages.	9M.0210 9M.0220	9P.3060
$CC4EG	$CCEEG	$CC4EG	Command Processor Message Module	Contains messages.	9M.0210	9P.3060
$CC4EH	$CCEEH	$CC4EH	Command Processor Message Module	Contains messages.	9M.0210	9P.3060
$CC4EJ	$CCXEJ	$CC4EJ	End-of-Job Transient	Stops all system activity and loads $CC5SH.	9M.0290	9P.5070
$CC4EL	$CCEEL	$CC4EL	Command Processor Message Module	Contains error messages for online test.	9M.0210	9P.3060
$CC4EP	$CCEEP	$CC4EP	Command Processor Message Module	Contains messages.	9M.0210	9P.3060
$CC4EU	$CCEEU	$CC4EU	Command Processor Message Module	Contains messages, moves messages to Command Processor work area.	9M.0210	9P.3060
$CC4E1	$CCEE1	$CC4E1	Command Processor Message Module	Contains messages.	9M.0210	9P.3060
$CC4E2	$CCEE2	$CC4E2	Command Processor Message Module	Contains messages.	9M.0210	9P.3060
$CC4E3	$CCEE3	$CC4E3	Command Processor Message Module	Contains messages.	9M.0210	9P.3060
$CC4E4	$CCEE4	$CC4E4	Command Processor Message Module	Contains messages.	9M.0210	9P.3060
$CC4E5	$CCEE5	$CC4E5	Command Processor Message Module	Contains messages.	9M.0210	9P.3060

CCP Execution (continued)

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CC4FL	$CCTFL	$CC4FL	First File Command Transient	Validates file commands.	9M.0200	9P.3120
$CC4F2	$CCTF2	$CC4F2	Second File Command Transient	Queues and dequeues FSBs.	9M.0200	9P.3130
$CC4GA	$CCIGA	CC4GA	First Get-Terminal-Attributes Transient	Processes allocation status.	9M.0050	9P.6100
$CC4GB	$CCIGB	$CC4GB	Second Get-Terminal-Attributes Transient	Processes remaining attributes and moves to record area.	9M.0050	9P.6120
$CC4HF	$CC4HF	$CC4HF	CCP RPG II Halt Processor Transient	Handles RPG II halt requests and routes them to $CC4H1.	9M.0300	9P.1010
$CC4HH	$CC4HH	$CC4HH	CCP Halt/Syslog Transient, Handler User Task Halt	Handles halt/syslog requests from user tasks which involve unconditional halts.	9M.0300	9P.1030
$CC4H1	$CC4H1	$CC4H1	CCP Halt/Syslog Transient, Non-Communications Task.	Handles halt/syslog requests from other than the Communications Management Task.	9M.0300 None	9P.1020 9P.1025 (Model 4)
$CC4H2	$CC4H2	$CC4H2	CCP Halt/Syslog Transient, Communications Task	Handles halt/syslog requests from the Communications Management Task.	9M.0300 None	9P.1040 9P.1045 (Model 4)
[1] $CC4H4	$CC4H4	$CC4H4	Model 4CCP Halt Syslog Conversion Transient	Handles conversion of Halt/Syslog codes to light indicators.	None	9M.0345
$CC4IB	$CC4IB	$CC4IB	BSCA Interrupt Appendage	Analyzes all interrupts from BSCA, indicates when op ending interrupts are received.	9M.0030	OD
$CC4IC	$CC4IC	$CCYIC	Console Interrupt Intercept	Indicates occurrence of console interrupt in $CCCOM.	9M.0030	OC
[2] $CC4IG	$E050	$CC4IG	General Entry Intercept	Intercepts and serializes DSM requests.	9M.0040	OR

[1] Model 4 only.
[2] Models 8, 10, and 12 only.

CCP Execution (continued)

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CC4IH	$E044	$CC4IH	Interrupt Handler Common Subroutine	Interrupts routing and register control.	None	OF
$CC4II	$E060	$CC4II	I/O Interface Routine	Teleprocessing I/O servicing interface.	9M.0050	PP
$CC4IM	$CC4IM	$$MLTD MLT001 MLT002	MLTA Interrupt Intercept	Analyzes MLTA interrupt and adds to TP op end count in $CCCOM if needed.	9M.0030	OE
$CC4IO	$E040	$CC4IO	CCP Disk I/O Intercept Routine	Intercepts and routes disk IOS requests.	None	OM
$CC4IS	$E060	$CC4IS	I/O Interface Routine	Teleprocessing I/O servicing interface.	9M.0060	PP
$CC4IW	$E040	$CC4IW	CCP Disk I/O Wait Intercept Routine	Intercepts disk I/O and waits.	None	OM
$CC4JA	$CC4JA	$CC4JA	MLTA Translate Transient, Output	Translates from EBCDIC to 2740/2741 PTTC/EBCD code.	9M.0145	9P.2620
$CC4JB	$CC4JB	$CC4JB	MLTA Translate Transient, Output	Translates from EBCDIC to 2740/2741 correspondence code.	9M.0145	9P.2620
$CC4JC	$CC4JC	$CC4JC	MLTA Translate Transient, Output	Translates from EBCDIC to 1050 PTTC/EBCD code.	9M.0145	9P.2620
$CC4JD	$CC4JD	$CC4JD	BSCA EBCDIC to ASCII Transient	Translates from EBCDIC to ASCII.	9M.0140	9P.2620
$CC4JE	$CC4JE	$CC4JE	BSCA ASCII to EBCDIC Transient	Translates from ASCII to EBCDIC.	9M.0145	9P.2620
$CC4J1	$CC4J1	$CC4J1	MLTA Translate Transient, Input	Translates from 2740/2741 PTTC/EBCD code to EBCDIC.	9M.0120	9P.2300
$CC4J2	$CC4J2	$CC4J2	MLTA Translate Transient, Input	Translates from 2740/2741 PTTC/EBCD code to EBCDIC.	9M.0120	9P.2300
$CC4J3	$CC4J3	$CC4J3	MLTA Translate Transient, Input	Translates from 2740/2741 correspondence code to EBCDIC.	9M.0120	9P.2300
$CC4J4	$CC4J4	$CC4J4	MLTA Translate Transient, Input	Translates from 1050 PTTC/EBCD to EBCDIC.	9M.0120	9P.2300

CCP Execution (continued)

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CC4J5	$CC4J5	$CC4J5	MLTA Translate Transient, Input	Translates from 2740/2741 code to upper case EBCDIC.	9M.0120	9P.2300
$CC4J6	$CC4J6	$CC4J6	MLTA Translate Transient, Input	Translates from 2740/2741 correspondence code to upper case EBCDIC.	9M.0120	9P.2300
$CC4J7	$CC4J7	$CC4J7	MLTA Translate Transient, Input	Translates from 2740/2741 PTTC/EBCD code to upper case EBCDIC.	9M.0120	9P.2300
$CC4J8	$CC4J8	$CC4J8	MLTA Translate Transient, Input	Translates from 1050 PTTC/EBCD code to upper case EBCDIC.	9M.0120	9P.2300
$CC4J9	$CC4J9	$CC4J9	MLTA Translate Transient, Output	Translates from EBCDIC to 2740/ 2741 PTTC/EBCD code.	9M.0145	9P.2620
$CC4KA	$CCKKA	$CC4KA	Console Management Transient, Console Controller	Routes control to other console management transients to service the console without letting rest of CCP run.	9M.0150	9P.2100
$CC4KB	$CCKKB	$CC4KB	Console Management Transient, Start Console I/O (Without Parameter List)	Starts console I/O requests by Communications Management transients for which no parameter list was built.	9M.0150	9P.2090
$CC4K1	$CCKK1	$CC4K1	Console Management Transient, Schedule Console I/O	Schedules console I/O requests from non-communications management tasks (routed through $CC4II or $CC4IS).	9M.0100	9P.2000
$CC4K2	$CCKK2	$CC4K2	Console Management Transient, Start Output Operation	Starts console output using the first parameter list in the console queue.	9M.0100 9M.0150	9P.2010
$CC4K3	$CCKK3	$CC4K3	Console Management Transient, Handle Output Op End	Freemains and posts upon op end of console output. Routes control to ERP transients if necessary.	9M.0100 9M.0150	9P.2020
$CC4K4	$CCKK4	$CC4K4	Console Management Transient, Handle Input Op End	Routes command input to the Command Processor. Routes control to proper transient for other than errorless command input.	9M.0100	9P.2050

CCP Execution (continued)

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CC4K5	$CCKK5	$CC4K5	Console Management Transient, Handle REQ Key Interrupt	Blanks the console buffer and enables the console for input.	9M.0100	9P.2040
$CC4K6	$CCKK6	$CC4K6	Console Management Transient, Handle Input Reply	Handles all console replies to a task.	9M.0100	9P.2060
$CC4K7	$CCKK7	$CC4K7	Console Management Transient, Handle Input Error	Issues error message to system operator.	9M.0100	9P.2070
$CC4K8	$CCKK8	$CC4K8	Console Management Transient, Handle Output Error	Handles output error and issues message if necessary.	9M.0100	9P.2030
$CC4K9	$CCKK9	$CC4K9	Console Management Transient, Schedule Console I/O for the Communications Manager	Assures that console output request from the Communications Manager is scheduled either with or without a parameter list.	9M.0100 9M.0150	9P.2080
$CC4LT	$CCLLT	$CC4LT	Termination Close Interface	Locates and dequeues all open DTFs for a terminating program.	9M.0250	9P.7300
$CC4L2	$CCLL2	$CC4L2	Close Load 2, Printer and PTAM Files	Closes line printer and PTAM file DTFs.	9M.0280	9P.7310
$FC4L3	$CCLL3	$CC4L3	Close Load 3, MFCU and 1442 Files	Closes MFCU and 1442 DTFs.	9M.0280	9P.7320
$CC4L4	$CCLL4	$CC4L4	Close Load 4, Disk Files	Closes 5444 and 5445 disk DTFs.	9M.0280	9P.7330
$CC4L5	$CCLL5	$CC4L5	Close Load 5, Disk DTFs	Frees SQBs and restores disk DTFs to pre-open format.	None	9P.7340
$CC4MA	$CCMMA	$CC4MA	Communications Management MLTA Error Transient, First Level	Analyzes MLTA completion code, forms CCP return code and either returns or reroutes.	9M.0160	9P.2200

CCP Execution (continued)

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CC4MB	$CCCMB	$CC4MB	Communications Management MLTA Error Transient, Switched Line Disconnect	Purges all I/O in the line queue for a switched line disconnect due to hardware error.	9M.0160	9P.2250
$CC4MC	$CCMMC	$CC4MC	Communications Management MLTA Error Transient, Buffer Busy Line Rescheduler	Reschedules the parameter lists in in the line queue taking into consideration the buffer busy condition just sensed.	9M.0160	9P.2230
$CC4MD	$CCCMD	$CC4MD	Communications Management Common Error Transient	Determines whether terminal in error should be placed in CCP error recovery status.	9M.0160	9P.2240
$CC4ME	$CCMME	$CC4ME	Communications Management MLTA Error Transient, Line Closed Processor	Varies offline all terminals online and cleans up line queue.	9M.0160	9P.2220
$CC4MF	$CCCMF	$CC4MF	Communications Management MLTA Error Transient, Switched Line Disconnect Sign Off	Processes through sign off a command mode terminal on a switched line which was disconnected because of a hardware error.	9M.0160	9P.2260
$CC4MG	$CCTMG	$CC4MG	Message Command, Terminal To System Operator	Issues message if data truncated.	9M.0220	9P.3250
$CC4MM	$CC4MM	$CC4RM $CC4GM $CC4FM	Getmain/Freemain Service Subroutine, Minimum System	Allocates/de-allocates a main storage segment in a main storage pool.	9M.0020	OZ
$CC4MP	$CCCMP	$CC4MP	Communications Management Transient, Put-to-Terminal in CCP ERP	Processes all puts requested to terminals in CCP error recovery status.	9M.0130	9P.2400
$CC4MS	$CC4MS	$CC4RM $CC4GM $CC4FM	Getmain/Freemain Service Subroutine, Less Minimum System	Allocates/de-allocates a main storage segment in a main storage pool.	9M.0020	OZ

CCP Execution (continued)

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CC4MT	$CCMMT	$CC4MT	Start MLTA Online Test	Puts control information in MLTA DTF and issues MLTA online test request.	9M.0145	9P.2600
$CC4MX	$CC4MX	$CC4MX $CC4MV	Generalized Move-Service Subroutine	Moves data of any length within main storage.	None	OX
$CC4MZ	$CCBMZ	$CC4MZ	OLT Test REQ for MLTA	Completes the syntax check of and initiates a MLTA OLT request.	9M.0210	9P.3440
$CC4M1	$CCBM1	$$BSMS	3270 only MLMP IOCS	Minimum version of MLMP for CCP 3270-only system.	None	RG
$CC4M9	$CCMM9	$CC4M9	Snap Dump of MLTA Trace	Snap dumps the MLTA trace table in main storage.	9M.0160	9P.2210
$CC4NC	$CCINC	$CC4NC	Terminal Release Command Processor	Diagnoses and processes release command.	None	9P.6030
$CC4NM	$CCTNM	$CC4NM	Name Command Processor	Changes current terminal name.	9M.0220	9P.3240
$CC4OC	$CC4OC	$CC4OC	Open/Close/ Allocate Interface Mainline	Calls open/close/allocate transient, calls DSM limits transients, primes and purges buffers, invokes Termination.	9M.0260 9M.0270 9M.0280	PI
$CC4OF	$CCTOF	$CC4OF	First/Off Transient	Validates sign off command.	9M.0220	9P.3220
$CC4OG	$CCTOG	$CC4OG	Second/Off Transient	Puts terminal offline and informs system and terminal operator.	9M.0220	9P.3230
$CC4OH	$CCOOH	$CC4OH	FORTRAN Pseudo Open	Performs pseudo open for disk files.	9M.0270	9P.7130
$CC4OP	$CCOOP	$CC4OP	Open/Close/ Allocate Initial Transient	Determines type of request, performs unit record and special file DTF allocation.	9M.0260 9M.0270 9M.0280	9P.7010
$CC4OR	$CCOOR	$CC4OR	Open/Close Return Transient	Sets return address.	9M.0270 9M.0280	9P.7030
$CC4OS	$CCOOS	$CC4OS	Shared I/O Open Routine	Opens 5444 DTFs for shared I/O.	9M.0270	9P.7180
$CC4OT	$CCOOT	$CC4OT	PTAM File Open BAM File Open	Opens PTAM file DTFs. Builds BAM MVF table.	9M.0270	9P.7140

CCP Execution (continued)

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CC4O1	$CC001	$CC4O1	Open/Close Routing Transient	Routes control through open/close. Opens or closes special file DTFs.	9M.0270	9P.7020
$CC4O2	$CC002	$CC4O2	Open Load 2, Printer Files	Opens line printer DTFs.	9M.0270 None	9P.7100 9P.7105 (Model 4)
$CC4O3	$CC003	$CC4O3	Open Load 3, MFCU and 1442 Files	Opens MFCU and 1442 DTFs.	9M.0270 None	9P.7110 9P.7105 (Model 4)
$CC4O4	$CC004	$CC4O4	Open Load 4, Disk DTF Validation	Determines if disk file DTFs are valid to open.	9M.0270	9P.7120
$CC4O5	$CC005	$CC4O5	Open Load 5, Disk DTF Field Mapping	Sets appropriate disk addresses into disk DTFs.	9M.0270	9P.7150
$CC4O6	$CC006	$CC4O6	Open Load 6, Relative Record Number Conversion Routines	Converts a relative record number to a disk address or a disk address to a relative record number.	9M.0270	9P.7160
$CC4O7	$CC007	$CC4O7	Open Load 7, 5444 Files	Opens 5444 disk DTFs.	9M.0270	9P.7170
$CC4O8[1]	$CC008	$CC4O8	Open Load 8, 5445 Files	Opens 5445 disk DTFs.	9M.0270	9P.7190
$CC4O9	$CC009	$CC4O9	Open Load 9, Buffer Priming	Primes index buffers for sequential files, second data buffer for dual buffered consecutive files.	9M.0270	9P.7200
$CC4PC	$CCGPC	$CC4PC	Command Processor First Load	Performs initial command validation and routes control to Command Processor transients.	9M.0200	9P.3010
$CC4PF	$CCGPF	$CC4PF	Command Processor Stop Polling Failed Routine	Informs terminal operator that last input was ignored.	9M.0200	9P.3020
$CC4PG	$CCMPG	$CC4PG	Communications Management Transient, MLTA Purge I/O Processor	Dequeues put-to-terminal requests and routes control to stop invite input transients.	9M.0130	9P.2420

[1] Models 8, 10, and 12 only.

CCP Execution (continued)

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CC4PI	$CC4PI	$CC4PI	Transient Area Handler	Reads transients into transient areas and passes control between transients and resident code.	9M.0048	OT
$CC4PK	$CCGPK	$CC4PK	Command Processor Console Command Validator	Routes control to appropriate system operator command transients.	9M.0200 9M.0210	9P.3040
$CC4PQ	$CC4PI	$CC4PQ	Test for Request for a Terminal/ Line Subroutine	Determines if a pending program request exists for a switched BSCA line.	None	OU
$CC4PR	$CCGPR	$CC4PR	Command Processor Return Transient	Sends messages to terminals or console, frees TP buffers, and re-invites commands.	9M.0200	9P.3050
$CC4PS	$E046	$CC4PS	CCP Post Routine	Posts indication that an awaited event has occurred.	9M.0050	OJ
$CC4PT	$CCGPT	$CC4PT	Command Processor Term Command Validator	Routes control to appropriate terminal operator command transients.	9M.0200 9M.0220	9P.3030
$CC4QQ	$CCTQQ	$CC4QQ	/Q Processing Transient	Sets or resets Q status of terminals.	9M.0200	9P.3110
$CC4RC	$CCRRC	$CC4RC	Program Request Multiple Requesting Terminal Support Add-On	Allocates terminal to and parts the requested program.	9M.0230	9P.3310
$CC4RE	$CCSRE	$CC4RE	Resume Command One	Searches TCB chain for active tasks and issues messages.	9M.0210	9P.3645
$CC4RF	$CCSRF	$CC4RF	Resume Command Two	Searches TCB chain for active tasks and issues messages.	9M.0210	9P.3650
$CC4RL	$CCIRL	$CC4RL	First Terminal Release Processor	Validates command.	9M.0050	9P.6160
$CC4RN	$CCTRN	$CC4RN	/Run Command Processor Transient	Processes command interrupt mode terminal back to data mode.	9M.0220	9P.3270
$CC4RP	$CCSRP	$CC4RP	ERP Command Processor	Validates the command and retries or bypasses the I/O in error as specified.	9M.0210	9P.3410

CCP Execution (continued)

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CC4RX	$CCIRX	$CC4RX	Second Terminal Release Processor	Sends released message and de-allocates the TUB.	9M.0050	9P.6170
$CC4RY	$CCIRY	$CC4RY	Third Terminal Release Processor	Frees up names/FSBs and sends console message.	9M.0050	9P.6180
$CC4R1	$CCRR1	$CC4R1	First Program Request Transient	Verifies syntax, sets parameters for other request processors.	9M.0230	9P.3300
$CC4R2	$CCRR2	$CC4R2	Second Program Request Transient	Reads PCT.	9M.0230	9P.3320
$CC4R3	$CCRR3	$CC4R3	Third Program Request Transient	Looks up program name.	9M.0230	9P.3340
$CC4R4	$CCRR4	$CC4R4	Fourth Program Request Transient	Validates disk file.	9M.0230	9P.3350
$CC4R5	$CCRR5	$CC4R5	Sixth Program Request Transient	Validates terminals and unit record devices.	9M.0230	9P.3370
$CC4R6	$CCRR6	$CC4R6	Seventh Program Request Transient	Processes final request.	9M.0230	9P.3380
$CC4R7	$CCRR7	$CC4R7	Fifth Program Request Transient	Validates main storage availability.	9M.0230	9P.3360
$CC4R7	$CCDR7	$CC4R7	Fifth Program Request Transient	Validates main storage availability (remap only).	9M.0230	9P.3360
$CC4R8	$CCRR8	$CC4R8	Program Request PCT I/O Error Handler	Handles PCT I/O error message.	9M.0230	9P.3330
$CC4SB[1]	$CCMSB	$CC4SB	Communications Management Transient, MLTA Stop Invite Input — Abort the MLTA Line	Issues abort to MLTA line in the processing of a stop invite input.	None	9P.2460

[1] Models 8, 10, and 12 only.

CCP Execution (continued)

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CC4SC[1]	$CCMSC	$CC4SC	Communications Management Transient, MLTA Start Code Analysis	Analyzes non-zero MLTA start code and cancels CCP if invalid.	9M.0145	9P.2610
$CC4SF	$CCMSF	$CC4SF	Communications Management Transient, MLTA Stop Invite Input Failed	Processes a stop invite input request after determination that stop invite input failed.	None	9P.2440
$CC4SH	$CCSSH	$CC4SH	Shutdown Command Processor	Diagnoses and processes the shutdown command.	9M.0210	9P.3660
$CC4SK[1]	$CCMSK	$CC4SK	Communications Management Transient, Handle MLTA Op End for Operation without Parameter List	Processes op end for operation for which the parameter list has previously been dequeued (or may never have existed).	9M.0120	9P.2280
$CC4SO	$CCTSO	$CC4SO	Terminal Sign On Command Processor	Validates sign on command, calls user security routine if specified.	9M.0220	9P.3200
$CC4SP[1]	$CCMSP	$CC4SP	Communications Management Transient, MLTA Stop Invite Input Processor	Determines status of request to stop invite input.	9M.0130	9P.2430
$CC4SQ[1]	$CCMSQ	$CC4SQ	Communications Management Transient, MLTA Stop Invite Input — Abort Op End	Analyzes the op end of an abort issued to stop invite input at an MLTA terminal.	9M.0120	9P.2270
$CC4SS[1]	$CCSSS	$CC4SS	Suspend Command	Sets off bits in each active user task.	9M.0210	9P.3630
$CC4SS	$CCSSS	$CC4SS	Suspend Command	Suspend init and suspend command.	None	9P.3635 (Model 4)
$CC4ST	$CCSST	$CC4ST	Suspend Command	Suspend user and suspend command.	None	9P.3640

[1] Models 8, 10, and 12 only.

CCP Execution (continued)

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CC4SU	$CC4SU	SUEXEC	Exit from Startup, Entrance to Resident Control	Plugs DSM console interrupt handler, resets keyboard, issues CCP STARTED message, and invites input from console and online command terminals.	9M.0200	8P.0230
$CC4S0	$CCBSO	SOSTRT	3275 Switched Format Transient	Formats output for the 3275 switched; reformats input from the 3275 switched.	9M.0140	9P.2390
$CC4S2	$CCCS2	$CC4S2	Communications Management Transient, MLTA Stop Invite Input — User Input Op End Handler	Handles user stop invite request for invite input which has already op ended.	None	9P.2450
$CC4TB	$CCATB	$CC4TB	Terminal Allocation Routine	Allocates and sets attributes to program selected terminals.	9M.0240	9P.4070
$CC4TD	$CCXTD	$CC4TD	First Termination Routine	Controls termination, main storage de-allocate, and TCB re-initialize.	9M.0250	9P.5010
$CC4TE	$CCSTE	$CC4TE	Trace Command	Checks if trace is in the system and if so, sets on bit.	9M.0210	9P.3600
$CC4TF	$CCXTF	$CC4TF	Termination Files De-allocation	De-allocates disk/unit record devices from user.	9M.0250	9P.5030
$CC4TI	$CC4TI	$CC4TI	User Task Termination Interface	Interfaces between CCP functions.	9M.0250	PL
$CC4TK	$CCXTK	$CC4TK	Termination Console Message Transient	Issues message to console when a task terminates abnormally.	9M.0250	9P.5025
$CC4TM	$CC4TM	$CC4TM	Termination Task Mainline Routine	Provides the resident interface required to the Termination Task transient routines.	None	PL
$CC4TN	$CCXTN	$CC4TN	Termination Routine	Stops TP I/O user task.	9M.0250	9P.5040
$CC4TP	$CCXTP	$CC4TP	Termination Polling Restart #1	Determines terminal/restart to be done.	9M.0250	9P.5050
$CC4TR	$CC4PI	$CC4TR	Transient Return	Returns transient control router.	9M.0048	OT

CCP Execution (continued)

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CC4TS	$CCSTS	$CC4TS	First Level OLT Request	Begins syntax check of online test request.	9M.0210	9P.3420
$CC4TT	$CC4TT	$CC4TT	CCP Trace Interface Routine	Provides an interface to the dynamically loaded CCP trace module.	None	OV
$CC4TW	$CCXTW	$CC4TW	Termination Dump Routine	Dumps storage to $CCPFILE when a task terminates abnormally.	9M.0250	9P.5020
$CC4TX	$CC4PI	$CC4TX	Transient Transfer of Control Routine	Passes control between CCP transients.	9M.0048	OT
$CC4TY	$CCXTY	$CC4TY	TP Polling Restart #2	Writes and invites command capable terminals.	9M.0250	9P.5060
$CC4TZ	$CCSTZ	$CC4TZ	Complete OLT Request for BSCA	Completes the syntax check of and initiates a request for BSCA OLT.	9M.0210	9P.3430
$CC4T1	$CCMT1	$CC4T1	OLT Transient for MLTA	Formats the TUB and LCB for selected terminal.	9M.0130	9P.2410
$CC4T2	$CCMT2	$CC4T2	Complete MLTA OLT	Services op end interrupt for MLTA OLT.	9M.0120	9P.2290
$CC4UR	$CCAUR	$CC4UR	Unit Record Allocation Transient	Allocates unit record devices to tasks.	9M.0240	9P.4080
$CC4VA	$CCSVA	$CC4VA	Vary Command Processor	Validates the command and transfers control to $CC4VB.	9M.0210	9P.3500
$CC4VB	$CCSVB	$CC4VB	Second Vary Command Processor	Processes the terminal offline or online as specified.	9M.0210	9P.3510
$CC4WC	$CCCWC	$CC4WC	Communications Management Transient, Locate Phone Number	Reads STT from disk and locates phone number to call for user operation to switched line which is not connected.	9M.0140 9M.0145	9P.2630
$CC4WD	$CCCWD	$CC4WD	Communications Management Transient, Issue Switched Line Connection Message	Issues message specifying information about switched line connection for user operation to switched line which is not connected.	9M.0140 9M.0145	9P.2640
$CC4WR	$CCCWR	$CC4WR	Communications Management Transient, Translation Error	Processes a parameter list for which a translation error has occurred.	9M.0145 9M.0160	9P.2310

CCP Execution (continued)

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CC4WT	$E046	$CC4WT	CCP Wait Routine	Processes CCP asynchronous wait.	None	OI
$CC4XA[1]	$CCKXA	$CC4XA	Puts CM Output	Put console management output to console log file.	None	9P.9020
$CC4XB[1]	$CCKXB	$CC4XB	Display Backward	Gets 5-10 messages from log file.	None	9P.9030
$CC4XC[1]	$CCKXC	$CC4XC	Display Backward	Puts 5-10 messages to console screen.	None	9P.9040
$CC4XF[1]	$CCKXF	$CC4XF	Display Forward	Gets 5-10 messages from log file.	None	9P.9050
$CC4XG[1]	$CCKXG	$CC4XG	Display Forward	Puts 5-10 messages to console screen.	None	9P.9060
$CC4X1[1]	$CCKX1	$CC4X1	Console Management	Schedules console operations.	None	9P.8030
$CC4X2[1]	$CCKX2	$CC4X2	Console Input	Handles console input interrupt.	None	9P.8040
$CC4X3[1]	$CCKX3	$CC4X3	Error Messages	Handles errors and return key.	None	9P.8050
$CC4X4[1]	$CCKX4	$CC4X4	Input Complete	Handles completed input.	None	9P.8060
$CC4X5[1]	$CCKX5	$CC4X5	Enable Keyboard	Enables keyboard.	None	9P.8070
$CC4X6[1]	$CCKX6	$CC4X6	Reply to a Get	Handles console reply to a get.	None	9P.8080
$CC4X7[1]	$CCKX7	$CC4X7	Puts Output to a File	Puts output to console log file.	None	9P.8090
$CC4X8[1]	$CCKX8	$CC4X8	Puts Input to a File	Puts input to console log file.	None	9P.9000
$CC4X9[1]	$CCKX9	$CC4X9	Puts CM Output	Puts console management output to console log file.	None	9P.9010
$CC4YA	$CC4YA	$CC4YA	User First Sign On Transient	Calls halt routine. Provides hook as dummy user sign on routine.	9M.0220	9P.3210
CCPSAV	$CC4PI	CCPSAV CCPRST CCPRET	CCP Register Save and Restore	Saves and restores registers.	None	OV
CMBSKP	$E092	CMBSKP	BSCA Poll Skip Bit Routine	Sets skip bits on/off in BSCA polling list or switched ID list.	None	RF
CMFRMN	$E080	CMFRMN CMREQ	Accept TP Requests	Accepts and processes user TP requests (queues or initiates the request).	9M.0090	QA QE

[1] Model 4 only.

CCP Execution (continued)

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
CMGINL	$E092	CMGINL	Operation Input Length Determination Routine	Calculates input length and sets truncate data on buffer overflow.	None	QU
CMIVGM	$E087	CMIVGM	Invite Buffer Analysis	Obtains buffer space for invite operations.	None	QK
CMOPND	$E070	CMOPND	Op End Analysis	Analyzes TP op ends and initiates I/O completion, return to user, and rescheduling of TP line.	9M.0090	QA QC
CMPSRQ	$E090	CMPSRQ	Post TP Scheduled	Posts requester that TP operation has been processed and scheduled.	None	QS
CMRSCH	$E085	CMRSCH CMPAII CMTSRQ	Reschedule TP Line	Schedules work on non-busy lines and stops polling and starts output.	9M.0090	QA QG
CMSET	$E090	CMSET	Format Put-No-Wait Area	Obtains and formats storage for put-no-wait.	None	QM
CMSTOR	$E090	CMSTOR	Getmain Size Determination	Calculates main storage required for a particular TP request.	None	QL
CPHALT	$CC4PI	CPHALT	CCP Stand-Alone Halt Routine	Initiates CCP U- halt and dump.	None	OV

User Subroutines

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$$UFFF[1]	$$UFFF	$$UFFF	Unit Record Intermediary	Tests unit record device for busy or not ready to avoid looping on SIO.	None	9P.8010
$$UFPP[1]	$$UFPP	$$UFPP	Unit Record Intermediary	Tests unit record device for busy or not ready to avoid looping on SIO.	None	9P.8010
$$UFPR[1]	$$UFPR	$$UFPR	Unit Record Intermediary	Tests unit record device for busy or not ready to avoid looping on SIO.	None	9P.8010
$$UFPU[1]	$$UFPU	$$UFPU	Unit Record Intermediary	Tests unit record device for busy or not ready to avoid looping on SIO.	None	9P.8010
$$UFRD[1]	$$UFRD	$$UFRD	Unit Record Intermediary	Tests unit record device for busy or not ready to avoid looping on SIO.	None	9P.8010

[1] Models 8, 10, and 12 only.

User Subroutines (continued)

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$$UFRP[1]	$$UFRP	$$UFRP	Unit Record Intermediary	Tests unit record device for busy or not ready to avoid looping on SIO.	None	9P.8010
$$UFRU[1]	$$UFRU	$$UFRU	Unit Record Intermediary	Tests unit record device for busy or not ready to avoid looping on SIO.	None	9P.8010
$$UPRT	$$UPRT	$$UPRT	Unit Record Intermediary	Tests unit record device for busy or not ready to avoid looping on SIO.	None	9P.8010
$$URFF[1]	$$URFF	$$URFF	Unit Record Intermediary	Tests unit record device for busy or not ready to avoid looping on SIO.	None	9P.8010
$NARFF[1]	$NARFF	$NARFF	Unit Record Intermediary	Tests unit record device for busy or not ready to avoid looping on SIO.	None	9P.8010
$NLPRT[1]	$NLPRT	$NLPRT	Unit Record Intermediary	Tests unit record device for busy or not ready to avoid looping on SIO.	None	9P.8010
$NMFFF[1]	$NMFFF	$NMFFF	Unit Record Intermediary	Tests unit record device for busy or not ready to avoid looping on SIO.	None	9P.8010
$NMFPP[1]	$NMFPP	$NMFPP	Unit Record Intermediary	Tests unit record device for busy or not ready to avoid looping on SIO.	None	9P.8010
$NMFPR[1]	$NMFPR	$NMFPR	Unit Record Intermediary	Tests unit record device for busy or not ready to avoid looping on SIO.	None	9P.8010
$NMFPU[1]	$NMFPU	$NMFPU	Unit Record Intermediary	Tests unit record device for busy or not ready to avoid looping on SIO.	None	9P.8010
$NMFRD[1]	$NMFRD	$NMFRD	Unit Record Intermediary	Tests unit record device for busy or not ready to avoid looping on SIO.	None	9P.8010
$NMFRP[1]	$NMFRP	$NMFRP	Unit Record Intermediary	Tests unit record device for busy or not ready to avoid looping on SIO.	None	9P.8010
$NMFRU[1]	$NMFRU	$NMFRU	Unit Record Intermediary	Tests unit record device for busy or not ready to avoid looping on SIO.	None	9P.8010
CCPCIO[1]	CCPCIO	CCPCIO	COBOL Communications Service Subroutine	Passes user programs communications requests to CCP.	9M.0420	9P.6300
CCPFIO[1]	CCPFIO	CCPFIO	FORTRAN Communication Service Subroutine	Passes user programs communications requests to CCP.	9M.0430	9P.6310

[1] Models 8, 10, and 12 only.

User Subroutines (continued)

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
SUBR90	SUBR90	SUBR90	Move Fields To/From Record Area	Extracts fields from a record area or moves fields to a record area.	None	9P.6350
SUBR91	SUBR91	SUBR91	RPG II EXIT/ RLABL Support	Supports EXIT/RLABL TP I/O for CCP.	9M.0370 9M.0380	9P.6340
SUBR92	SUBR92	SUBR92	RPG II SPECIAL File Support	Support SPECIAL files in RPG II for CCP.	9M.0370 9M.0380	9P.6320
SUBR93	SUBR93	SUBR93	RPG II False File Support	Subroutine to a guide false file logic main storage in RPG II for CCP.	None	9P.6330

Shutdown

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CC5SH	$CC5SH	$CC5SH	Shutdown	Returns system to normal DSM status.	10M.0100	RM

Disk-to-Printer Dump

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CCPDD	$CCPDD	DD0000	Disk-to-Printer Dump	Dumps main storage and trace data to printer from $CCPFILE.	11M.0100	RZ
$CCPLL	$CCPLL	$CCPLL	Log List	Lists $CCPLOG.	11M.0200	YY

Standalone Dump Programs (Models 8, 10, and 12 Only)

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
CCPDAN	CCPDAN	CCPDAN	MFCU-Loadable Main Storage Dump, AN or LC Chain	Dumps main storage to printer between selected bounds.	None	ZN
CCPDHN	CCPDHN	CCPDHN	MFCU-Loadable Main Storage Dump, NN Chain	Dumps main storage to printer between selected bounds.	None	ZN

Standalone Dump Programs (continued)

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
CCPDPN	CCPDPN	CCPDPN	MFCU-Loadable Main Storage Dump, PN Chain	Dumps main storage to printer between selected bounds.	None	ZN
CCPDTN	CCPDTN	CCPDTN	MFCU-Loadable Main Storage Dump, TN Chain	Dumps main storage to printer between selected bounds.	None	ZN

Standalone CCP Support Program

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
$CCPRB	$CCPRB	RBEGIN	File Recovery Program	Assists the CCP user in disk file recovery if the CCP is abnormally terminated.	14M.0100	None

Installation Verification Program

Module	PID Name	Entry Point	Descriptive Name	Function	Method of Operation ID	Program Organization ID
CCPIVP	CCPIVP	CCPIVP	Installation Verification Program	Verifies installed CCP (object module link edited).	(Figure 13-2)	ZP
CCPIVR	CCPIVR	CCPIVR	Installation Verification Program	Verifies installed CCP (relocatable module).	(Figure 13-2)	ZP

Appendix B information was added to *IBM System/3 Models 4, 8, 10, and 12 CCP Data Areas and Diagnostic Aids,* SY21-0048.

Appendix C information was added to *IBM System/3 CCP Messages Manual,* GC21-5170.

SCP GENERATOR DATA AREAS (MODELS 8, 10, AND 12 ONLY)

All data areas used by more than one phase of the SCP Generator are described in this chapter:

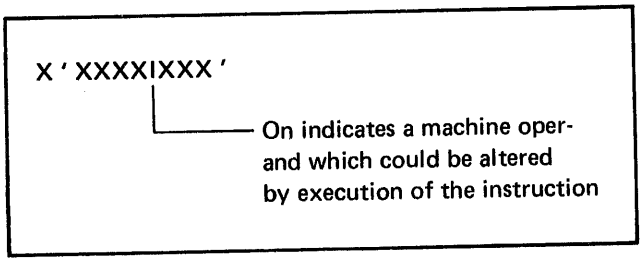
- Communications Area (COMARA). Part of $CGNIN—Contains the transfer vector and an address table for all other tables in $CGNIN (Figure B-1).
- Common Area (COMMON). Part of $CGNIN—Used for interphase constants, working storage, and communication (Figures B-2 through B-7). The COMMON Interphase Parameter Usage Chart shows which fields in COMMON are used by each phase of the SCP Generator (Figure B-8).
- Printer Communications Area. Part of $CGNPE—Contains printer DTF, printer buffer, and listing header (Figure B-9).
- Symbol Table—Contains symbols found in the source program (Figures B-10 and B-11).
- Work File ($WORK2)—A disk scratch file used for intermediate storage throughout execution of the SCP Generator (Figures B-12, B-13, and B-14).

Symbol Definition Record (translated name record)

Bytes 0-5	Symbol padded with blanks
Byte 6	Type byte X'01XXXXXX'
Bytes 7, 8	Statement number where defined
Byte 9	Not used

Symbol Reference Record (translated term record)

Bytes 0-5	Symbol padded with blanks
Byte 6	Type byte X'10XXXXXX'
Bytes 7, 8	Statement number of the reference
Byte 9	X'XXXXIXXX'



Appendix C. Halts Cross Reference

This appendix lists SCP Generation halts and specifies the meaning of each halt and the module that issues the halt. Halts are listed alphabetically.

CCP halts are listed in the *IBM System/3 Models 4, 8, 10, and 12 Communication Control Program Data Areas and Diagnostic Aids*, SY21-0048-0.

SCP GENERATOR HALTS

The following halts are subhalts to a U-halt (1234 halt on Model 4).

Halt	Meaning	Issued By
GE	Error in Generation code	$CGNSX
OB	End of extent on Object File ($WORK)	$CGNPE $CGNPS
SC	EOF on Source File ($SOURCE) with no /* read	$CGNCM
WK	End of extent on Work File ($WORK2)	$CGNCM $CGNSF $CGNSS

Appendix D. Macros Used by CCP Generation

First Pass

Macro	*Purpose*
$EIOD	Define unit-record and disk devices to be supported
$EFAC	Define CCP facilities to be supported
$EPLG	Define programming language to be supported
$ESEC	Define sign on security support to be used
$EFIL	Define size requirements in $CCPFILE
$EMLA	Define MLTA lines and features to be supported
$EMLD	Define MLTA devices to be supported
$EBSC	Define BSCA lines and features to be supported
$EBSD	Define BSCA device to be supported
$EGEN	Define miscellaneous requirements and generate second pass input

Second Pass

Macro	*Purpose*
$E000	Set system-wide global variables
$E001	Set system control global variables
$E002	Set MLTA global variables
$E003	Set BSCA global variables
$EEQU	Generate equates for common values
$EDSM	Generate equates for DSM supervisor values
$EKIO	Generate equates for console support
$ECOM	Generate equates for CCP Communications Area
$ETCB	Generate equates for task control block
$ETCC	Generate equates for task completion codes
$ECDE	Generate equates for Contents directory entry
$EIOB	Generate equates for disk IOB
$ESQB	Generate equates for sector enqueue block
$ECPL	Generate equates for communications parameter list
$ETAS	Generate equates for terminal attributes set
$ETUB	Generate equates for terminal unit block
$ELCB	Generate equates for line control block
$ETNT	Generate equates for Terminal Name Table
$ETML	Generate equates for translate/more parameter list
$EBEQ	Generate equates for BSCA values
$E030	Generate CCP Communications Area
$E031	Generate transient relocation constants
$E032	Generate resident Startup initialization routine

Second Pass (continued)

Macro	*Purpose*
$E033	Generate initial IOCS pointers
$E035	Generate system work areas, TCBs, and console TUBs
$E038	Generate first level transient list
$E040	Generate routines $CC4DP, $CC4IO, $CC4IW
$E042	Intercepts all branches to zero
$E044	Generate routine $CC4IH
$E046	Generate routines $CC4WT, $CC4PS
$E050	Generate routine $CC4IG
$E055	Generate routine $CC4DI
$E058	Generate DCs for relocation of $CC4II
$E060	Generate routines $CC4II, $CC4IS
$E065	Generate maintenance space and any user security work area
$E070	Generate routine $CC4CM — Part 1
$E072	Generate routine $CC4CM — Part 2
$E075	Generate routine $CC4CM — Part 3
$E077	Generate routine $CC4CM — Part 4
$E080	Generate routine $CC4CM — Part 5
$E082	Generate routine $CC4CM — Part 6
$E085	Generate routine $CC4CM — Part 7
$E087	Generate routine $CC4CM — Part 8
$E090	Generate routine $CC4CM — Part 9
$E092	Generate routine $CC4CM — Part 10
$E095	Generate routine $CC4CM — Part 11

Appendix E information was added to *IBM System/3 CCP Messages Manual,* GC21-5170.

Appendix E. Messages Cross Reference

This appendix lists the CCP messages and specifies the module that contains each message and the module that issues each message. The messages are listed in twelve groups:

1. CCP Generation Messages
2. SCP Generator Messages
3. Assignment Build Messages
4. Assignment List Messages
5. User Security Routine Messages
6. Display Format Generator Messages
7. Startup Messages
8. System Operator Messages
9. Terminal Operator Messages
10. CCP Dump Messages
11. Installation Verification Program (CCPIVP) Messages
12. Display Format Test Routine Diagnostic Messages

Messages are arranged alphamerically by message ID within each group.

CCP Generation Messages

Message ID	Message Text	Contained In	Issued By
CC005	$EIOD STATEMENT NOT IN PROPER SEQUENCE	$EIOD	$EIOD
CC010	INVALID 'CARD' PARAMETER – MUST BE MFCU/ 1442/'MFCU, 1442'/NO	$EIOD	$EIOD
CC015	INVALID 'PRINTER' PARAMETER – MUST BE 5203/1403/NO	$EIOD	$EIOD
CC020	INVALID 'DISKS' PARAMETER – MUST BE R2/ 'R2,F2'/NO	$EIOD	$EIOD
CC025	INVALID 'D5445' PARAMETER – MUST BE D1/'D1,D2'/NO	$EIOD	$EIOD
CC026	INVALID 'N3741' PARAMETER – MUST BE YES/Y/NO/N	$EIOD	$EIOD
CC050	$EFAC STATEMENT OUT OF SEQUENCE – OR PRECEDING STATEMENT ERROR	$EFAC	$EFAC
CC055	INVALID 'MAXEUP' PARAMETER – MUST BE DIGIT IN RANGE 1-8	$EFAC	$EFAC
CC060	INVALID 'DPF' PARAMETER – MUST BE YES/Y/NO/N	$EFAC	$EFAC
CC065	INVALID 'FSHARE' PARAMETER – MUST BE YES/Y/NO/N	$EFAC	$EFAC
CC067	FSHARE–YES SPECIFIED WITH MAXEUP-1 – TREATED AS FSHARE–NO	$EFAC	$EFAC
CC075	INVALID 'SYMFIL' PARAMETER – MUST BE YES/Y/NO/N	$EFAC	$EFAC
CC080	INVALID 'PGMCNT' PARAMETER – MUST BE YES/Y/NO/N	$EFAC	$EFAC
CC085	INVALID 'ESCAPE' PARAMETER – MUST BE '6 CHARS'/X'12 CHARS'/NO	$EFAC	$EFAC
CC090	INVALID 'FORMAT' PARAMETER – MUST BE YES/Y/NO/N	$EFAC	$EFAC
CC091	INVALID 'PRUF' PARAMETER–MUST BE YES/Y/NO/N	$EFAC	$EFAC
CC100	$EPLG STATEMENT OUT OF SEQUENCE–OR PRECEDING STATEMENT ERROR	$EPLG	$EPLG
CC100	$EPLG STATEMENT OUT OF SEQUENCE – OR PRECEDING STATEMENT ERROR	$EPLG	$EPLG
CC105	INVALID 'LANG' PARAMETER – MUST BE COBOL/ RPGII/ASSEM/FORTRAN	$EPLG	$EPLG

CCP Generation Messages (continued)

Message ID	Message Text	Contained In	Issued By
CC110	DUPLICATE $EPLG STATEMENT FOR RPGII LANGUAGE	$EPLG	$EPLG
CC110	DUPLICATE $EPLG STATEMENT FOR COBOL LANGUAGE	$EPLG	$EPLG
CC110	DUPLICATE $EPLG STATEMENT FOR FORTRAN LANGUAGE	$EPLG	$EPLG
CC110	DUPLICATE $EPLG STATEMENT FOR ASSEM LANGUAGE	$EPLG	$EPLG
CC115	INVALID 'PPUNIT' PARAMETER – MUST BE F1/F1/R2/F2	$EPLG	$EPLG
CC120	MISSING 'LANG' OPERAND – MUST BE SPECIFIED	$EPLG	$EPLG
CC125	MISSING 'PPUNIT' OPERAND – MUST BE SPECIFIED	$EPLG	$EPLG
CC150	$ESEC STATEMENT OUT OF SEQUENCE – OR PRECEDING STATEMENT ERROR	$ESEC	$ESEC
CC155	INVALID 'SECURE' PARAMETER – MUST BE CCP/USER/NO	$ESEC	$ESEC
CC160	INVALID 'LUSI' PARAMETER – ONLY 0 IS VALID UNLESS SECURE–USER	$ESEC	$ESEC
CC165	LUSI–0 SPECIFIED, OR 'LUSI' OPERAND OMITTED WITH SECURE–USER	$ESEC	$ESEC
CC170	INVALID 'LUSI' PARAMETER – MUST BE NUMBER IN RANGE 1-4096	$ESEC	$ESEC
CC200	$EFIL STATEMENT OUT OF SEQUENCE – OR PRECEDING STATEMENT ERROR	$EFIL	$EFIL
CC205	INVALID 'SETS' PARAMETER – MUST BE NUMBER IN RANGE 1-25	$EFIL	$EFIL
CC210	INVALID 'PROGS' PARAMETER – MUST BE NUMBER IN RANGE 1-255	$EFIL	$EFIL
CC215	INVALID 'DFILES' PARAMETER – MUST BE NUMBER IN RANGE 0-50	$EFIL	$EFIL
CC220	INVALID 'TERMS' PARAMETER – MUST BE NUMBER IN RANGE 1-254	$EFIL	$EFIL
CC225	INVALID 'DUMPS' PARAMETER – MUST BE NUMBER IN RANGE 1-9	$EFIL	$EFIL
CC230	INVALID 'CORE' PARAMETER – MUST BE 24K/32K/48K/64K (Model 10 only)	$EFIL	$EFIL

CCP Generation Messages (continued)

Message ID	Message Text	Contained In	Issued By
CC230	INVALID 'CORE' PARAMETER – MUST BE 48K/ 64K (Model 12 only)	$EFIL	$EFIL
CC240	MISSING 'FLUNIT' OPERAND – MUST BE SPECIFIED	$EFIL	$EFIL
CC242	INVALID 'FLUNIT' PARAMETER – MUST BE R1/F1/ R2/F2	$EFIL	$EFIL
CC244	MISSING 'FLPACK' OPERAND – MUST BE SPECIFIED	$EFIL	$EFIL
CC246	INVALID 'FLPACK' PARAMETER – MUST BE 1-6 CHARACTERS	$EFIL	$EFIL
CC248	INVALID 'TRKLOC' PARAMETER – MUST BE NUMBER IN RANGE 8-405	$EFIL	$EFIL
CC250	$EMLA STATEMENT OUT OF SEQUENCE – OR PRECEDING STATEMENT ERROR	$EMLA	$EMLA
CC255	INVALID 'LINES' PARAMETER – MUST BE NUMBER IN RANGE 0-8	$EMLA	$EMLA
CC260	MISSING 'LINES' OPERAND – MUST BE SPECIFIED IF STATEMENT USED	$EMLA	$EMLA
CC265	LINES-0, BUT OTHER KEYWORD SPECIFIED WITH NON-DEFAULT PARAMETER	$EMLA	$EMLA
CC270	INVALID 'XLATE' PARAMETER – MUST BE YES/Y/NO/N	$EMLA	$EMLA
CC300	$EMLD STATEMENT OUT OF SEQUENCE – OR PRECEDING STATEMENT ERROR	$EMLD	$EMLD
CC305	STATEMENT USED, BUT NO MLTA LINES SPECIFIED	$EMLD	$EMLD
CC310	MISSING 'TYPE' OPERAND – MUST BE SPECIFIED	$EMLD	$EMLD
CC315	INVALID 'TYPE' PARAMETER – MUST BE MLTA TERMINAL DESIGNATION	$EMLD	$EMLD
CC320	MISSING 'XMCODE' OPERAND – MUST BE SPECIFIED	$EMLD	$EMLD
CC325	INVALID 'XMCODE' PARAMETER – MUST BE CORR/ PTTCEBCE/PTTCBCD	$EMLD	$EMLD
CC330	XMCODE–PTTCEBCD NOT VALID FOR TERMINAL TYPE SPECIFIED	$EMLD	$EMLD
CC330	XMCODE–PTTCBCD NOT VALID FOR TERMINAL TYPE SPECIFIED	$EMLD	$EMLD
CC330	XMCODE–CORR NOT VALID FOR TERMINAL TYPE SPECIFIED	$EMLD	$EMLD

CCP Generation Messages (continued)

Message ID	Message Text	Contained In	Issued By
CC400	$EBSC STATEMENT OUT OF SEQUENCE – OR PRECEDING STATEMENT ERROR	$EBSC	$EBSC
CC405	MISSING 'BSCA' OPERAND – MUST BE SPECIFIED IF STATEMENT USED	$EBSC	$EBSC
CC410	INVALID 'BSCA' PARAMETER – MUST BE NUMBER IN RANGE 0-2	$EBSC	$EBSC
CC415	BSCA-0, BUT OTHER OPERAND SPECIFIED WITH NON-DEFAULT PARAMETER	$EBSC	$EBSC
CC417	INVALID 'DA' PARAMETER – MUST BE YES/Y/NO/N	$EBSC	$EBSC
CC419	IF DA-YES – MUST SPECIFY BSCA-1 or BSCA-2	$EBSC	$EBSC
CC420	INVALID 'DIAL' PARAMETER – MUST BE YES/Y/NO/N	$EBSC	$EBSC
CC425	INVALID 'PP' PARAMETER – MUST BE YES/Y/NO/N	$EBSC	$EBSC
CC430	INVALID 'MP' PARAMETER – MUST BE YES/Y/NO/N	$EBSC	$EBSC
CC435	INVALID 'CS' PARAMETER – MUST BE YES/Y/NO/N	$EBSC	$EBSC
CC440	BSCA PRESENT BUT NO LINE TYPES SPECIFIED	$EBSC	$EBSC
CC445	INVALID 'GETMSG' PARAMETER – MUST BE YES/Y/NO/N	$EBSC	$EBSC
CC450	INVALID 'ITB' PARAMETER – MUST BE YES/Y/NO/N	$EBSC	$EBSC
CC455	INVALID 'RECSEP' PARAMETER – MUST BE TWO HEX DIGITS	$EBSC	$EBSC
CC460	INVALID 'ASCII' PARAMETER – MUST BE YES/Y/NO/N	$EBSC	$EBSC
CC465	INVALID 'EBCDIC' PARAMETER – MUST BE YES/Y/NO/N	$EBSC	$EBSC
CC470	BSCA PRESENT BUT NEITHER TRANSMISSION CODE IS USED	$EBSC	$EBSC
CC472	EBCDIC–YES AND ASCII–YES SPECIFIED WITH BSCA-1 – NOT POSSIBLE	$EBSC	$EBSC
CC480	INVALID 'RESPOL' PARAMETER – MUST BE YES/Y/NO/N	$EBSC	$EBSC
CC482	RESPOL–YES SPECIFIED WITH CS–NO – TREATED AS RESPOL–NO	$EBSC	$EBSC
CC485	INVALID 'AUTORS' PARAMETER – MUST BE YES/Y/NO/N	$EBSC	$EBSC

CCP Generation Messages (continued)

Message ID	Message Text	Contained In	Issued By
CC487	AUTORS–YES SPECIFIED WITH MS–NO – TREATED AS AUTORS–NO	$EBSC	$EBSC
CC490	INVALID 'SPRNCY' PARAMETER – MUST BE YES/Y/ NO/N	$EBSC	$EBSC
CC492	XPRNCY–YES SPECIFIED WITH EBCDIC–NO – NOT VALID	$EBSC	$EBSC
CC500	$EBSD STATEMENT OUT OF SEQUENCE – OR PRE-CEDING STATEMENT ERROR	$EBSD	$EBSD
CC505	$EBSD STATEMENT USED, BUT NO BSC ADAPTERS SPECIFIED	$EBSD	$EBSD
CC510	MISSING 'TYPE' OPERAND – MUST BE SPECIFIED	$EBSD	$EBSD
CC515	INVALID 'TYPE' PARAMETER – MUST BE BSCA TERMINAL DESIGNATION	$EBSD	$EBSD
CC520	3271 CONTROL UNIT REQUIRES CONTROL STATION SUPPORT	$EBSD	$EBSD
CC525	'TYPE' 3275 REQUIRES CONTROL STATION OR SWITCHED SUPPORT	$EBSD	$EBSD
CC600	$EGEN STATEMENT OUT OF SEQUENCE – OR PRECEDING STATEMENT ERROR	$EBSD	$EBSD
CC605	DUPLICATE $EGEN STATEMENT – CONTENTS IGNORED	$EGEN	$EGEN
CC610	MISSING 'DSUNIT' OPERAND – MUST BE SPECIFIED	$EGEN	$EGEN
CC615	INVALID 'DSUNIT' PARAMETER – MUST BE R1/F1	$EGEN	$EGEN
CC620	MISSING 'CCUNIT' OPERAND – MUST BE SPECIFIED	$EGEN	$EGEN
CC625	INVALID 'CCUNIT' PARAMETER – MUST BE R1/F1/ R2/F2	$EGEN	$EGEN
CC630	MISSING 'WKUNIT' OPERAND – MUST BE SPECIFIED	$EGEN	$EGEN
CC635	INVALID 'WKUNIT' PARAMETER – MUST BE R1/F1/ R2/F2 OR SERIES OF 3	$EGEN	$EGEN
CC637	WKUNIT/WKPACK ERROR – PACKS ****** AND ****** BOTH ON UNIT **	$EGEN	$EGEN
CC640	MISSING 'WKPACK' OPERAND – MUST BE SPECIFIED	$EGEN	$EGEN
CC645	INVALID 'WKPACK' PARAMETER – MUST BE 1-6 CHAR NAME OF SERIES OF 3	$EGEN	$EGEN

CCP Generation Messages (continued)

Message ID	Message Text	Contained In	Issued By
CC650	MISSING 'DIUNIT' OPERAND – MUST BE SPECIFIED	$EGEN	$EGEN
CC655	INVALID 'DIUNIT' PARAMETER – MUST BE R1/F1/ R2/F2	$EGEN	$EGEN
CC660	'CCUNIT' SAME AS 'DIUNIT' – NOT PERMITTED	$EGEN	$EGEN
CC665	'PPUNIT' FROM $EPLG STATEMENT SAME AS 'DIUNIT' – NOT PERMITTED	$EGEN	$EGEN
CC670	INVALID 'MINRES' PARAMETER – MUST BE YES/Y/ NO/N	$EGEN	$EGEN
CC675	MINRES – YES REQUIRES MAXEUP – 1	$EGEN	$EGEN
CC680	INVALID 'CARD' PARAMETER – MUST BE YES/Y/NO/N	$EGEN	$EGEN
CC700	NO MLTA OR BSCA SUPPORT SPECIFIED – AT LEAST ONE REQUIRED	$EGEN	$EGEN
CC705	FORMAT–YES SPECIFIED IN $EFAC REQUIRES GETMSG–YES IN $EBSC	$EGEN	$EGEN
CC710	NO PROGRAMMING LANGUAGE SUPPORTED – AT LEAST ONE REQUIRED	$EGEN	$EGEN
CC715	FORMAT–YES IN $EFAC REQUIRES 3270 DISPLAY DEVICE IN $EBSD	$EGEN	$EGEN
CC720	ESCAPE SPECIFIED IN $EFAC REQUIRES MINRES-NO	$EGEN	$EGEN
CC725	PGMCNT-YES SPECIFIED IN $EFAC REQUIRES MINRES-NO	$EGEN	$EGEN
CC730	SYMFIL-YES SPECIFIED IN $EFAC REQUIRES MINRES-NO	$EGEN	$EGEN
CC735	XLATE-NO SPECIFIED IN $EMLA REQUIRES MINRES-NO	$EGEN	$EGEN
CC740	MULTIPLE TP LINES REQUIRE MINRES-NO	$EGEN	$EGEN
CC745	DPF-YES SPECIFIED IN $EFAC REQUIRES MINRES-NO	$EGEN	$EGEN
CC750	FSHARE-YES SPECIFIED IN $EFAC REQUIRES MINRES-NO	$EGEN	$EGEN

CCP Generation Messages (continued)

Message ID	Message Text	Contained In	Issued By
CC901	INVALID STATEMENT TYPE OR SOURCE LIBRARY ON WRONG PACK	$CC1PP	$CC1PP
CC902	INVALID STATEMENT TYPE, OR PREVIOUS ERROR ON STATEMENT	$CC1PP	$CC1PP
CC903	A KEYWORD USED IS NOT VALID FOR THIS STATEMENT TYPE	$CC1PP	$CC1PP
CC904	A PARAMETER IS MISSING OR HAS INVALID FORM	$CC1PP	$CC1PP
CC905	A DELIMITER IS EITHER INVALID OR WRONGLY PLACED IN THE STATEMENT	$CC1PP	$CC1PP
CC906	COMMA AFTER LAST OPERAND BUT NO CONTINUATION INDICATOR	$CC1PP	$CC1PP
CC907	ON CONTINUATION CARD, COLS. 1-13 MUST BE BLANK – THEY ARE NOT	$CC1PP	$CC1PP
CC909	ERROR '**' ISSUED BY MACRO PROCESSOR – POSSIBLE CCP ERROR	$CC1PP	$CC1PP
CC990	INCOMPLETE INPUT TO CCP GENERATION	$CC1PP	$CC1PP
CC991	NOT A VALID STATEMENT FOR CCP GENERATION	$CC1PP	$CC1PP

SCP Generator Messages

Message ID	Message Text	Contained In	Issued By
	TOTAL STATEMENTS IN ERROR IN THIS GENERATION = ****		SCP Generator

Assignment Build Messages

Message ID	Message Text	Contained In	Issued By
CA002	T DELIMITING APOSTROPHE MISSING	$CC2E1	$CC2SS
CA003	T INVALID KEYWORD FOR THIS STATEMENT OR PAST COL 71	$CC2E1	$CC2SS
CA004	T VALUE FOR KEYWORD IS NOT YES/NO	$CC2E1	$CC2SS
CA005	T KEYWORD GIVEN PREVIOUSLY IN THIS STATEMENT	$CC2E1	$CC2SS
CA006	T DATA IS IN COLUMN 72	$CC2E1	$CC2SS
CA007	T VALUE IS NOT NUMERIC OR INVALID SIZE NUMBER GIVEN	$CC2E1	$CC2SS
CA008	T VALUE IS NOT CORRECT RESPONSE FOR THIS KEYWORD	$CC2E1	$CC2SS
CA009	T SUBLIST NOT VALID FOR THIS KEYWORD	$CC2E1	$CC2SS
CA011	T INCORRECT USE OF SUBLIST VALUES	$CC2E1	$CC2SS
CA012	T TOO MANY OR TOO FEW CHARACTERS IN VALUE	$CC2E1	$CC2SS
CA013	T INCORRECT USE OF SPLIT–VALUE	$CC2E1	$CC2SS

Assignment Build Messages (continued)

Message ID	Message Text	Contained In	Issued By
CA014	T KEYWORD EXPECTED OR INCORRECT USE OF PUNCTUATION	$CC2E1	$CC2SS
CA015	T VALUE GIVEN IS TOO LONG	$CC2E2	$CC2SS
CA020	I FOLLOWING STATEMENTS ARE SYNTAX SCANNED ONLY	$CC2E2	$CCPAS
CA025	T REQUIRED KEYWORD NOT GIVEN	$CC2E2 $CC2E2	$CC2SS $CC2SY
CA026	W STARTUP DEFAULT SET DELETED	$CC2E2	$CCPAS
CA027	T FOUND HYPHEN INSTEAD OF COMMA OR APOSTROPHE	$CC2E2	$CC2SS
CA028	T TOO MANY STATEMENTS GIVEN	$CC2E2 $CC2E2 $CC2E2 $CC2E2 $CC2E2	$CC2BT $CC2MT $CC2TN $CC2DF $CC2PG
CA029	T TOO MANY ENTRIES GIVEN	$CC2E2 $CC2E2	$CC2SS $CC2PG
CA031	TERMINATION MESSAGES	$CC2E2	$CC2HK
CA032	WARNING MESSAGES	$CC2E2	$CC2HK
CA033	T INVALID TERMINAL ID	$CC2E2 $CC2E2	$CC2BT $CC2MT
CA040	T DISK ERROR OCCURRED ON EITHER $CCPWORK OR $CCPFILE	$CC2E2	$CCPAS
CA041	T STATEMENT NOT IN PROPER SEQUENCE	$CC2E3 $CC2E3	$CCPAS $CC2DF
CA042	W FILE ACCESS METHOD CONFLICTS WITH PROGRAM LANGUAGE	$CC2E3	$CC2PG
CA043	W SET ID NOT FOUND IN DIRECTORY	$CC2E3	$CCPAS
CA044	T CREATE SET ID EXISTS ALREADY	$CC2E3	$CCPAS
CA045	T CCP OR AUX PROG EXECUTING IN OTHER LEVEL	$CC2E3	$CCPAS
CA046	W REPLACE SET ID NOT FOUND, ASSUME ACTION–CREATE	$CC2E3	$CCPAS
CA051	T STATEMENT DOES NOT START WITH '//'	$CC2E3	$CCPAS

Assignment Build Messages (continued)

Message ID	Message Text	Contained In	Issued By
CA052	T INVALID STATEMENT IDENTIFIER	$CC2E3	$CCPAS
CA053	T $CCPFILE UNUSABLE	$CC2E3	$CCPAS
CA054	T SYSIN READ ERROR	$CC2E3	$CCPAS
CA055	T SYMBOLIC NAME REQUIRED FOR ALL DEFINED TERMINALS	$CC2E3	$CC2TN
CA060	T NO MORE ROOM IN $CCPFILE OR $CCPWORK	$CC2E3	$CCPAS
CA061	T INVALID ADDR/POLL CHARACTERS	$CC2E4	$CC2BT
CA070	T ATTRIBUTES SET WAS NOT SPECIFIED FOR THIS TERMINAL	$CC2E4	$CC2PG
CA071	T DUPLICATE REQUIRED TERMINALS INVALID	$CC2E4	$CC2PG
CA072	T TERMINAL NAME INVALID	$CC2E4	$CC2TN
CA073	T ADDR INVALID WITH LINE TYPE SW OR PP	$CC2E4	$CC2MT
CA074	T ADDR REQUIRED WITH LINE TYPE CS OR CW	$CC2E4	$CC2MT
CA075	T TERMINAL TYPE 2740 INVALID WITH RECEIVE INTERRUPT	$CC2E4	$CC2MT
CA076	T DELAY/AUTOPOLL IS INVALID WITH LINE TYPE SW OR PP	$CC2E4	$CC2ML
CA077	T ADDR/POLLCHAR INVALID WITH LINE TYPE SW OR PP	$CC2E4	$CC2BT
CA078	W PGMREQL LESS THAN MINIMUM REQUIRED, ASSUME DEFAULT	$CC2E4	$CC2SY
CA079	T PHONE NUMBER CONTAINS AN INVALID NUMERIC DIGIT	$CC2E4	CC2TN
CA080	T AUTOCALL INVALID FOR MLTA LINE	$CC2E4	$CC2MT
CA082	W PRUF$Z REQUIRES PRUFLNG AND DFF PROG, PRUF$Z IGNORED	$CC2ED	$CC2PG
CA083	W FORMAT NAME MUST BEGIN WITH $Z, PRUF$Z IGNORED	$CC2ED	$CC2PG
CA084	W DFF PROG WITH PRUFLNG REQUIRES PRUF$Z, PRUF IGNORED	$CC2ED	$CC2PG
CA085	W PRUF NOT SUPPORTED AT GENERATION, PRUF IGNORED	$CC2ED	$CC2PG

Assignment Build Messages (continued)

Message ID	Message Text	Contained In	Issued By
CA086	W PRUF REQUIRES PGMDATA-YES, ASSUME-YES	$CC2ED	$CC2PG
CA087	T INDEXED FILES NOT VALID IN 5444 SIMULATION AREA (Model 12 only)	$CC2ED	$CC2DF
CA098	W MINTPBUF TOO SMALL, CALCULATED VALUE USED	$CC2E5 $CC2E5	$CC2SY $CC2HK
CA099	I PREVIOUS SET NOT PROCESSED	$CC2E5	$CCPAS
CA101	T SET ID NOT VALID	$CC2E5	$CCPAS

Assignment Build Messages (continued)

Message ID	Message Text	Contained In	Issued By
CA102	T MINUPA VALUE INVALID	$CC2E5	$CC2SY
CA103	T MAXEUP GREATER THAN GIVEN AT GENERATION	$CC2E5	$CC2SY
CA104	W SQB MUST BE ZERO FOR ONE USER PROGRAM, VALUE IGNORED	$CC2E5	$CC2SY
CA105	W SOURCE STATEMENT FROM SOURCE MEMBER IS IGNORED	$CC2E5	$CCPAS
CA106	T SQB VALUE NOT ALLOWED, GENERATION CONFLICT	$CC2E5	$CC2SY
CA108	T FSB VALUE IS NOT ALLOWED, GENERATION CONFLICT	$CC2E5	$CC2SY
CA109	T PASSWORD SHOULD HAVE BEEN GIVEN	$CC2E5	$CC2SY
CA110	T PASSWORD NOT SUPPORTED AT GENERATION	$CC2E5	$CC2SY
CA111	T GENERAL POLL NOT SUPPORTED BY CCP	$CC2E5	$CC2BT
CA115	W DUPLICATE 'ATTRID' VALUE GIVEN, STATEMENT IGNORED	$CC2E6	$CC2TA
CA116	T UPPER CASE REQUIRES TRANSLATE	$CC2E6	$CC2TA
CA117	T BSCA NOT SUPPORTED AT GENERATION, BSCA KEYWORD GIVEN	$CC2E6	$CC2TA
CA118	W VARL/SPAN INVALID WITH ITB, ASSUME VARL/ SPAN–NO	$CC2E6	$CC2TA
CA119	T DUPLICATE TERMINAL ADDRESS ON SAME LINE INVALID	$CC2E6	$CC2MT
CA120	T RECL ONLY VALID IF DATAFORM— RECORD IS SPECIFIED	$CC2E6	$CC2TA
CA121	T RECL LARGER THAN BLKL OR INVALID BLOCKING FACTOR	$CC2E6	$CC2TA
CA122	T RECL NOT AN EVEN MULTIPLE OF BLKL	$CC2E6	$CC2TA
CA123	I *** SOURCE MEMBER END ***	$CC2E6	$CCPAS
CA125	T INDEXSEND/IDEXRCV IS VALID ONLY ON SWITCHED LINES	$CC2E6 $CC2E6	$CC2BL $CC2BT
CA126	T POLLLIST AND POLLLOOP ARE VALID ONLY ON CS LINES	$CC2E6 $CC2E6	$CC2BL $CC2ML

Assignment Build Messages (continued)

Message ID	Message Text	Contained In	Issued By
CA127	T TRANSPARENCY INVALID WITH XMCODE–ASCII	$CC2E6	$CC2BT
CA129	T MAXRECL IS LESS THAN MINIMUM REQUIRED	$CC2E7	$CC2ML
CA130	T SOURCE MEMBER NOT FOUND OR NO SOURCE LIBRARY ON PACK	$CC2E7	$CCPAS
CA133	W MAXRECL SIZE WILL NOT SUPPORT ON–LINE TEST	$CC2E7	$CC2ML
CA135	T LINE NUMBER GIVEN ON PREVIOUS STATEMENT	$CC2E7 $CC2E7	$CC2BL $CC2ML
CA136	T LINENUM OR LINE TYPE NOT SUPPORTED AT GENERATION	$CC2E7 $CC2E7	$CC2BL $CC2ML
CA138	T LINE TYPE–CS REQUIRES A POLL LIST	$CC2E7 $CC2E7	$CC2BL $CC2ML
CA140	T INVALID HEXADECIMAL CHARACTER ENTERED	$CC2E7 $CC2E7 $CC2E7	$CC2BL $CC2BT $CC2MT
CA141	T TOO MANY OR TOO FEW CHARACTERS IN VALUE	$CC2E7 $CC2E7	$CC2BL $CC2BT
CA142	T TRANSMISSION CODE NOT SUPPORTED AT GENERATION	$CC2E7 $CC2E7 $CC2E7	$CC2BL $CC2ML $CC2MT
CA160	T TYPE REQUIRES INDEXSEND OR ADDRCHAR/ POLLCHAR	$CC2E7	$CC2MT
CA161	T DEFINED TERMINAL NOT IN POLL LIST	$CC2E7 $CC2E7	$CC2BT $CC2MT
CA162	W TYPE NOT COMMAND CAPABLE, ASSUME COMMAND–NO	$CC2E7	$CC2BT
CA163	T POLL LIST TOO LONG	$CC2E8	$CC2MT
CA164	T VERIFYID REQUIRES INDEXRCV	$CC2E8	$CC2BT
CA165	W VERIFYID IGNORED, INVALID WITH LINE TYPE SPECIFIED	$CC2E8	$CC2BT
CA166	W SPAN VALID ONLY FOR 3735 AND CPU, ASSUME SPAN–NO	$CC2E8	$CC2BT

Assignment Build Messages (continued)

Message ID	Message Text	Contained In	Issued By
CA167	T SPAN/VARL REQUIRES RECL TO BE OMITTED OR EQUAL BLKL	$CC2E8	$CC2TA
CA168	W OFF ACTION APPLIES TO COMMAND TERMINALS ONLY	$CC2E8 $CC2E8	$CC2BT $CC2MT
CA172	T TERMINAL ID GIVEN ON PREVIOUS STATEMENT	$CC2E8 $CC2E8	$CC2BT $CC2MT
CA173	T TERMINAL TYPE NOT SUPPORTED AT GENERATION	$CC2E8 $CC2E8	$CC2BT $CC2MT
CA174	T COMBINATION OF TERMINAL TYPES ON THIS LINE INVALID	$CC2E8 $CC2E8	$CC2BT $CC2MT
CA175	T ATTRIBUTES SET INVALID	$CC2E8 $CC2E8	$CC2BT $CC2MT
CA176	T LINE TYPE CONFLICTS WITH ATTRIBUTES SWITCHED VALUE	$CC2E8 $CC2E8	$CC2BT $CC2MT
CA177	W MAXRECL TOO SMALL, MINIMUM REQUIRED VALUE USED	$CC2E8	$CC2MT
CA178	T PREVIOUS POLL LIST CONTAINS AN INVALID TERMINAL ID	$CC2E9 $CC2E9	$CC2BT $CC2MT
CA179	T SYSTEM STATEMENT SPECIFIED NO SYMBOLIC FILES	$CC2E9	$CC2DF
CA185	T 1050 REQUIRES PINCOMP AND/OR POUTCOMP	$CC2E9	$CC2MT
CA186	T TERMINAL TYPE CONFLICTS WITH LINE TYPE OR XMCODE	$CC2E9	$CC2MT
CA187	T PINCOMP/POUTCOMP VALID ONLY FOR 1050	$CC2E9	$CC2MT
CA188	W TERMINAL ATTRIBUTES CONTAIN BSCA KEYWORDS	$CC2E9	$CC2MT
CA195	T DUPLICATE TERMINAL NAME ENTERED	$CC2E9	$CC2TN
CA196	T TERMID NOT DEFINED ON BSCATERM OR MLTATERM STATEMENT	$CC2E9	$CC2TN

Assignment Build Messages (continued)

Message ID	Message Text	Contained In	Issued By
CA197	T PHONE NUMBER VALID ONLY IF TERMINAL ID IS SPECIFIED	$CC2E9	$CC2TN
CA198	T INCOMP/OUTCOMP/MSTRNAME INVALID WITH TERMINAL TYPE	$CC2E9	$CC2TN
CA200	T MASTER NAME REQUIRES INCOMP OR OUTCOMP	$CC2E9	$CC2TN
CA202	T PHONE NUMBER IS REQUIRED FOR AUTO OR MANUAL CALL	$CC2E9	$CC2TN
CA203	T MASTER NAME NOT PREVIOUSLY DEFINED	$CC2EA	$CC2TN
CA209	W MSTRINDX INVALID WITH 5445, ASSUME NO MSTRINDX	$CC2EA	$CC2DF
CA210	T DEVICE NOT SUPPORTED AT GENERATION	$CC2EA	$CC2DF
CA212	T KEYL AND KEYPOS REQUIRED FOR INDEXED FILES	$CC2EA	$CC2DF
CA213	T KEYL, KEYPOS, AND MASTER INDEX REQUIRES ORG-I	$CC2EA	$CC2DF
CA214	T DISKFILE NAME NOT PREVIOUSLY DEFINED	$CC2EA	$CC2DF
CA216	T SYMBOLIC NAMES CANNOT REFERENCE BOTH 5444 AND 5445 FILES	$CC2EA	$CC2DF
CA217	T DISKFILE CONTAINS DUPLICATE ENTRIES	$CC2EA	$CC2DF
CA218	T INVALID FILE NAME – $CCPFILE	$CC2EA	$CC2DF
CA219	T DUPLICATE FILE NAME ENTERED	$CC2EA	$CC2DF
CA220	W MSTRINDX KEYWORD CONFLICT, ASSUME NO MASTER INDEX	$CC2EA	$CC2DF
CA221	T INCONSISTENT DISKFILE ATTRIBUTES	$CC2EA	$CC2DF
CA230	W SORT/PGMDATA CONFLICT, ASSUME PGMDATA-YES	$CC2EB	$CC2PG
CA231	W GENERATED CCP DOES NOT SUPPORT UPDATE FILE SHARING	$CC2EB	$CC2PG
CA232	T TRANSLATION OF LINE CODE REQUIRED BY GENERATION	$CC2EB	$CC2MT
CA233	T DFF SUPPORT WITH ASCII CODE REQUIRES TRANSLAT–YES	$CC2EB	$CC2BT
CA234	W SQB OMITTED, CCP GENERATED TO SHARE FILES	$CC2EB	$CC2SY
CA235	T PROGRAM NAME PREVIOUSLY GIVEN	$CC2EB	$CC2PG

Assignment Build Messages (continued)

Message ID	Message Text	Contained In	Issued By
CA236	W MRTMAX/PGMDATA CONFLICT, ASSUME PGMDATA–YES	$CC2EB	$CC2PG
CA237	W REUSABLE/NEVEREND CONFLICT, ASSUME NEVEREND–NO	$CC2EB	$CC2PG
CA238	W REUSABLE/LANGUAGE CONFLICT, ASSUME REUSABLE–NO	$CC2EB	$CC2PG
CA239	T PRINTER, MFCU, 3741, OR RP1442 NOT SUPPORTED AT GENERATION	$CC2EB $CC2EB	$CC2SY $CC2PG
CA240	T TERMINAL NAME NOT DEFINED ON TERMNAME STATEMENT	$CC2EB	$CC2PG
CA241	W POLLLOOP VALUE TOO LARGE, ASSUME VALUE OF 3	$CC2EB	$CC2BT
CA242	T FILE ACCESS INCONSISTENT WITH DISKFILE STATEMENT	$CC2EB	$CC2PG
CA243	T FILE ACCESS DOES NOT MEET CCP DISK/SORT REQUIREMENTS	$CC1EB	$CC2PG
CA244	W UNREFERENCED FILE STATEMENTS IGNORED	$CC2EC	$CC2HK
CA245	W SHR INCONSISTENT WITH ACCESS, ASSUME NO SHR	$CC2EC	$CC2PG
CA246	T FILE NAME NOT PREVIOUSLY DEFINED	$CC2EC	$CC2PG
CA247	T LANGUAGE SPECIFIED NOT SUPPORTED AT GENERATION	$CC2EC	$CC2PG
CA248	W SHAREDIO NOT VALID FOR LANGUAGE, ASSUME SHAREDIO–NO	$CC2EC	$CC2PG
CA249	T FILE ACCESS INCONSISTENT WITH PREVIOUS ACCESS	$CC2EC	$CC2PG
CA250	W POSSIBLE FILE ACCESS CONFLICT	$CC2EC	$CC2PG
CA251	T 3270 FORMATTING KEYWORD CONFLICT	$CC2EC $CC2EC	$CC2SY $CC2PG
CA252	UPDATE ACCESS REQUIRES AT LEAST ONE SQB	$CC2EC	$CC2PG
CA253	W MASTER INDEX SIZE TOO SMALL, ASSUME NO MASTER INDEX	$CC2EC	$CC2DF

Assignment Build Messages (continued)

Message ID	Message Text	Contained In	Issued By
CA254	T INVALID PROGRAM NAME	$CC2EC	$CC2PG
CA255	T MRTMAX OR REUSABLE PROHIBITS USE OF SYMBOLIC FILES	$CC2EC	$CC2PG

Assignment List Messages

Message ID	Message Text	Contained In	Issued By
CL001	T $CCPFILE NOT INITIALIZED	$CCPAL	$CCPAL
CL002	W DUPLICATE KEYWORD	$CCPAL	$CCPAL
CL003	W INVALID KEYWORD	$CCPAL	$CCPAL
CL004	W INVALID DELIMITER	$CCPAL	$CCPAL
CL005	W INVALID KEYWORD PARAMETER	$CCPAL	$CCPAL
CL006	W INVALID IDENTIFIER IN COL'S 1-3	$CCPAL	$CCPAL
CL007	W INVALID STATEMENT IDENTIFIER	$CCPAL	$CCPAL
CL008	W SET ID NOT FOUND	$CCPAL	$CCPAL
CL009	W SET HAS NO REQUEST COUNT	$CCPAL	$CCPAL
CL010	T CCP BEING RUN IN OTHER LEVEL	$CCPAL	$CCPAL
CL011	I THIS $CCPFILE CONTAINS NO SETS	$CCPAL	$CCPAL

User Security Routine Messages

Message ID	Message Text	Contained In	Issued By
AU010	PERMANENT DISK ERROR	$CCPAU	$CCPAU
AU011	$CCPAU TERMINATED	$CCPAU	$CCPAU
AU012	$CCPAU PROGRAM FINISHED	$CCPAU	$CCPAU
AU013	EXTENT ERROR ON $CCPFILE	$CCPAU	$CCPAU
AU014	INVALID DATA CARD FROM SYSIN	$CCPAU	$CCPAU

User Security Routine Messages (continued)

Message ID	Message Text	Contained In	Issued By
AU015	ERROR WHILE READING FROM SYSIN	$CCPAU	$CCPAU
AU016	WARNING – SECURITY DATA EXCEEDS ALLOWABLE LENGTH	$CCPAU	$CCPAU
AU017	WARNING – DATA BETWEEN COUNT & DATA PORTION OF INPUT RECORD	$CCPAU	$CCPAU
AU018	WARNING – INPUT DATA RECORD LESS THAN SPECIFIED BY COUNT FIELD	$CCPAU	$CCPAU
AU019	WARNING – INPUT DATA RECORD EXCEEDS LENGTH IN COUNT FIELD	$CCPAU	$CCPAU
AU020	WARNING – NON NUMERIC DATA FOUND IN DATA RECORD	$CCPAU	$CCPAU
AU021	WARNING – COUNT FIELD VALUE IS ODD NUMBER, MUST BE EVEN	$CCPAU	$CCPAU
AU022	WARNING – TOTAL SECURITY DATA LESS THAN CCP GENERATION SPECIFICATION	$CCPAU	$CCPAU
AU023	SECURITY DATA MODULE CONTENTS	$CCPAU	$CCPAU
AU024	$CCPAU PROGRAM CANCELLED	$CCPAU	$CCPAU
AU025	WARNING – SECURITY DATA MODULE SIZE UNEQUAL TO CCP GENERATION SPECIFICATION	$CCPAU	$CCPAU
AU026	WARNING – DATA FIELD IS TOO LONG FOR THIS RECORD TYPE	$CCPAU	$CCPAU
AU027	WARNING – DATA VALUE TOO LARGE FOR THIS RECORD TYPE	$CCPAU	$CCPAU

Display Format Generator Messages

Message ID	Message Text	Contained In	Issued By
CG001	T INVALID CONTROL STATEMENT	$CC2CF	$CC2CF
CG002	T INVALID DISPLAY NAME	$CC2CF	$CC2CF
CG003	W INVALID CURSOR/FIELD NAME	$CC2CF	$CC2CF

Display Format Generator Messages (continued)

Message ID	Message Text	Contained In	Issued By
CG004	T INVALID DISPLAY SIZE	$CC2CF	$CC2CF
CG005	W INVALID CLEAR DISPLAY OPTION	$CC2CF	$CC2CF
CG006	W INVALID UNIT SPECIFIED	$CC2CF	$CC2CF
CG007	W INVALID PRINTER CONTROL ENTRY	$CC2CF	$CC2CF
CG008	T INVALID CONTINUATION	$CC2CF	$CC2CF
CG009	T INVALID FIELD NAME	$CC2CF	$CC2CF
CG010	T DUPLICATE FIELD NAME	$CC2CF	$CC2CF
CG011	T INVALID FIELD LOCATION	$CC2CF	$CC2CF
CG012	T INVALID FIELD LENGTH	$CC2CF	$CC2CF
CG013	W INVALID AUTOSKIP OPTION	$CC2CF	$CC2CF
CG014	T INVALID DATA SOURCE OPTION	$CC2CF	$CC2CF
CG015	T INVALID TYPE OPTION	$CC2CF	$CC2CF
CG016	T INSUFFICIENT SPACE BETWEEN FIELDS	$CC2CF	$CC2CF
CG017	T INSUFFICIENT SPACE FOR LEADING NULLS	$CC2CF	$CC2CF
CG018	T SPD FIELD NOT ALL ON ONE LINE	$CC2CF	$CC2CF
CG019	T FIELD OVERFLOWS DISPLAY	$CC2CF	$CC2CF
CG020	T LAST POSITION ON DISPLAY NOT AVAILABLE	$CC2CF	$CC2CF
CG021	W CURSOR FIELD NOT DEFINED	$CC2CF	$CC2CF
CG022	T TOO MANY FIELDS	$CC2CF	$CC2CF
CG023	W CURSOR FIELD IS PROTECTED	$CC2CF	$CC2CF
CG024	T ENTRIES UNDER MORE THAN ONE FIELD CLASS	$CC2CF	$CC2CF
CG025	T INVALID FIELD DEFINITION STATEMENT	$CC2CF	$CC2CF
CG026	T TOO MANY FIELDS ON A LINE WITH AN SPD FIELD	$CC2CF	$CC2CF
CG027	I DISPLAY NOT GENERATED DUE TO ERROR	$CC2CF	$CC2CF

Display Format Generator Messages (continued)

Message ID	Message Text	Contained In	Issued By
CG028	T FIELD IN DISPLAY WHICH IS INVALID WITH SPD ATTENTION FIELD	$CC2CF	$CC2CF
CG029	W TWO DEVICES, OR DEVICE AND WCC, SPECIFIED	$CC2CF	$CC2CF
CG030	T NO FIELDS DEFINED FOR DISPLAY	$CC2CF	$CC2CF
CG031	I WARNING ERROR DURING DISPLAY GENERATION	$CC2CP	$CC2CP
CG032	T ERROR OCCURRED DURING FILE FUILD	$CC2CP	$CC2CP
CG033	T INSUFFICIENT CONTINUATIONS FOR DATA DATA LENGTH	$CC2CF	$CC2CF
CG034	W DUPPING WITH NL/EM SPECIFIED ON CONTROL CARD	$CC2CR	$CC2CR
CG035	T UNSUCCESSFUL $SOURCE ACCESS	$CC2CF	$CC2CR
CG036	T DUPPING FIELDS ON FIRST LINE	$CC2CF	$CC2CR

Display Format Test Routine Messages

Message ID	Message Text	Contained In	Issued By
DT010	W INVALID OPTION ENTERED	$CCPDT	$CCPDT
DT011	W BLANK NOT FOUND BETWEEN PARAMETERS	$CCPDT	$CCPDT
DT012	I REVERSE INTERRUPT WAS RECEIVED	$CCPDT	$CCPDT
DT013	W STATION DID NOT RESPOND TO POLLING AND ADDRESSING	$CCPDT	$CCPDT
DT014	I REMOTE STATION DOES NOT RESPOND TO ATTEMPTED DATA TRANSFER	$CCPDT	$CCPDT
DT015	I INCORRECT DATA RECEIVED	$CCPDT	$CCPDT
DT016	W SENSE/STATUS BYTES INDICATE _____ UNIT SPECIFY TRANSMISSION CHECK COMMAND REJECT INTERVENTION REQUIRED EQUIPMENT CHECK DATA CHECK CONTROL CHECK OPERATION CHECK	$CCPDT	$CCPDT
DT017	W FORMAT NAME MUST BEGIN WITH $Z	$CCPDT	$CCPDT

Display Format Test Routine Messages (continued)

Message ID	Message Text	Contained In	Issued By
DT018	I PROGRAM INTERVENTION CAUSED BY _____ CLEAR KEY ENTER KEY TEST REQ KEY SELECTER PEN CARD READER PF1-PF12 KEY PA1-PA3 KEY		
DT019	W FORMAT WAS NOT ON PROGRAM PACK	$CCPDT	$CCPDT
DT020	W INVALID FORMAT NAME	$CCPDT	$CCPDT

Printer Format Generator Messages

Message ID	Message Text	Contained In	Issued By
CP001	T INVALID CONTROL STATEMENT	$CC2CG	$CC2CG
CP002	T INVALID PRINTER NAME	$CC2CG	$CC2CG
CP003	T INVALID PRINTER SIZE	$CC2CG	$CC2CG
CP004	T INVALID LINE LENGTH	$CC2CG	$CC2CG
CP005	T INVALID NUMBER OF LINES PER PAGE	$CC2CG	$CC2CG
CP006	T INVALID PLATEN LENGTH	$CC2CG	$CC2CG
CP007	T INVALID CONTINUATION	$CC2CG	$CC2CG
CP008	T INVALID DISK STORAGE UNIT	$CC2CG	$CC2CG
CP009	W DUPLICATE FIELD NAME	$CC2CG	$CC2CG
CP010	T INVALID FIELD LOCATION	$CC2CG	$CC2CG
CP011	T INVALID FIELD LENGTH	$CC2CG	$CC2CG
CP012	W INVALID FIELD NAME	$CC2CG	$CC2CG
CP013	T INVALID DATA SOURCE OPTION	$CC2CG	$CC2CG
CP014	T INVALID MULTIPLE PAGES ENTRY	$CC2CG	$CC2CG
CP015	T INSUFFICIENT SPACE BETWEEN FIELDS	$CC2CG	$CC2CG
CP016	T INVALID VERTICAL FORMS FEED ENTRY	$CC2CG	$CC2CG

Printer Format Generator Messages (continued)

Message ID	Message Text	Contained In	Issued By
CP017	T FIELD OVERFLOWS DISPLAY	$CC2CG	$CC2CG
CP018	T DATA IN RESERVED AREA AFTER CONTINUATION	$CC2CG	$CC2CG
CP019	T INVALID REPEAT LAST CHARACTER ENTRY	$CC2CG	$CC2CG
CP020	T TOO MANY FIELDS	$CC2CG	$CC2CG
CP021	T INVALID FIELD DEFINITION STATEMENT	$CC2CG	$CC2CG
CP022	I DISPLAY NOT GENERATED DUE TO ERROR	$CC2CG	$CC2CG
CP023	T NO FIELDS DEFINED FOR DISPLAY	$CC2CG	$CC2CG
CP024	T INSUFFICIENT CONTINUATIONS FOR DATA LENGTH	$CC2CG	$CC2CG
CP025	T FIELD OVERFLOWS LINE	$CC2CG	$CC2CG
CP026	W DATA IN RESERVED AREA	$CC2CG	$CC2CG
CP027	T REPEAT LAST CHARACTER WITH EXECUTION DATA	$CC2CG	$CC2CG
CP028	T INVALID ENTRY FOR FORMS FEED STATEMENT	$CC2CG	$CC2CG
CP029	T @@@FF WITHOUT MULTIPLE PAGES INDICATED	$CC2CG	$CC2CG
CP030	T LINE 1 POS 1 NOT VALID AFTER FORMS FEED	$CC2CG	$CC2CG
CP031	T UNSUCCESSFUL $SOURCE ACCESS	$CC2CG	$CC2CS
CP032	T INVALID DUPLICATION STATEMENT	$CC2CG	$CC2CS
CP033	T DATA IN RESERVED AREA ON @@@DP STATEMENT	$CC2CG	$CC2CS
CP034	T INVALID KATAKANA OPTION ENTRY	$CC2CG	$CC2CG
CP035	T INVALID ENTRY ON PRINT/NO-PRINT OPTION	$CC2CG	$CC2CG
CP036	I WARNING ERROR DURING PRINTER FORMAT GENERATION	$CC2CQ	$CC2CQ
CP037	T ERROR OCCURRED DURING FILE BUILD	$CC2CQ	$CC2CQ

Message ID	Message Text	Contained In	Issued By
SU001	SYSTEM/3 CCP	$CCP	$CCP
SU002	$CCPFILE ON UNIT **	$CCP	$CCP
SU003	DEFAULT SET = *	$CCP	$CCP
SU004	SPECIFY UNIT FOR $CCPFILE	$CCP	$CCP
SU009	*ERROR* INVALID RESPONSE	$CCP	$CCP
SU010	*ERROR* OTHER PROGRAM LEVEL ACTIVE	$CCP	$CCP
SU011	ANY SPECIFICATIONS?	$CCP	$CCP
SU012	*ERROR* INCOMPATIBLE USE OF DUAL PROGRAMMING FEATURE	$CCP	$CCP
SU013	*ERROR* $CCPFILE INFORMATION INVALID	$CCP	$CCP
SU014	INITIALIZE $CCPFILE WITH PROGRAM $CC1BF AND ENTER ASSIGNMENT SETS	$CCP	$CCP
SU015	*ERROR* NO ASSIGNMENT SETS IN $CCPFILE	$CCP	$CCP
SU020	DEFAULT SET NOT ASSIGNED	$CCP	$CCP

Startup Messages (continued)

Message ID	Message Text	Contained In	Issued By
SU025	ANY TEMPORARY ASSIGNMENT CHANGES?	$CCP	$CCP
SU035	UNLESS CHANGED, SET '*' USED	$CCP	$CCP
SU040	SELECT SET ID FROM LIST:	$CCP	$CCP
SU045	ANY CHANGES TO SET '*'?	$CCP	$CCP
SU055	ASSIGNED MAXIMUM CONCURRENT USER PROGRAMS = *	$CCP	$CCP
SU060	TO CHANGE, ENTER NEW VALUE	$CCP	$CCP
SU065	NON-ZERO DIGIT REQUIRED	$CCP	$CCP
SU070	ASSIGNMENT SPECIFICATION EXCEEDED	$CCP	$CCP
SU071	INITIALIZING CCP	$CCP	$CCP
SU075	ASSIGNED PASSWORD = ******	$CCP	$CCP
SU090	MINIMUM DYNAMIC T/P BUFFER AREA = ***** BYTES	$CCP	$CCP
SU095	MINIMUM USER PROGRAM AREA = **.**K	$CCP	$CCP
SU099	*ERROR* INVALID SYNTAX	$CCP	$CCP
SU100	TO CHANGE MINIMUM T/P BUFFER AREA, ENTER VALUE (BYTES)	$CCP	$CCP
SU101	MINIMUM T/P BUFFER = ***** BYTES	$CCP	$CCP
SU105	TO CHANGE MINIMUM USER PROGRAM AREA, ENTER VALUE (NN.NNK)	$CCP	$CCP
SU110	*ERROR* INVALID VALUE	$CCP	$CCP
SU111	PREVIOUSLY SPECIFIED	$CCP	$CCP
SU115	REQUESTED SPACE UNAVAILABLE	$CCP	$CCP
SU119	*ERROR* MODULE ****** NOT FOUND	$CCP	$CCP
SU125	*ERROR* INVALID KEYWORD	$CCP	$CCP
SU126	*ERROR* INVALID KEYWORD FOR CCP CONFIGURATION	$CCP	$CCP
SU130	*ERROR* INVALID DISK UNIT	$CCP	$CCP

Startup Messages (continued)

Message ID	Message Text	Contained In	Issued By
SU425	*ERROR* ALL PROGRAMS NOW SUPPRESSED	$CC3FS	$CC3FS
SU430	PROGRAM '******' NOW SUPPRESSED	$CC3FS	$CC3FS
SU435	SYMFILE '********' NOW SUPPRESSED	$CC3FS	$CC3FS
SU440	*ERROR INVALID SYNTAX $CC3FS	$CC3FS	$CC3FS
SU449	CCP TERMINATED	$CC3FS	$CC3FS
SU450	PROGRAM '******' NOT COMPILED/LINK-EDITED ON MODEL 12, SUPPRESSED	$CC3IP	$CC3IP
SU451	PROGRAM '******' NOT FOUND, SUPPRESSED	$CC3IP	$CC3IP
SU452	PROGRAM '******' SUPPRESSED BECAUSE TEMPORARY	$CC3IP	$CC3IP
SU453	*ERROR* ALL PROGRAMS NOW SUPPRESSED	$CC3IP	$CC3IP
SU459	*ERROR* VALID $$MCRI NOT FOUND	$CC3TB	$CC3TB
SU461	*ERROR* CORE EXHAUSTED WHILE BUILDING T/P CONTROL BLOCKS	$CC3TB	$CC3TB
SU463	*ERROR* INVALID BSCA OCL STATEMENT	$CC3TB	$CC3TB
SU541	CORE EXHAUSTED WHILE BUILDING TERMINAL UNIT *ERROR* BLOCKS	$CC3TC	$CC3TC
SU543	*ERROR* CORE EXHAUSTED WHILE BUILDING TERMNAME'S	$CC3TC	$CC3TC
SU547	TERMNAME '******' NOW UNASSIGNED	$CC3TC	$CC3TC
SU555	*** MORE TRACKS NEEDED IN $CCPFILE	$CC3LD	$CC3LD
SU557	*ERROR* EXHAUSTED CORE IN ATTEMPTING TO:	$CC3LD	$CC3LD
SU560	LOAD $CC4	$CC3LD	$CC3LD
SU565	*ERROR* MODULE ****** NOT FOUND	$CC3LD	$CC3LD
SU570	*ERROR* INVALID CCP BRANCH INSTRUCTION IN CONSOLE INTERRUPT HANDLER	$CC3LD	$CC3LD
SU575	*ERROR* INSUFFICIENT DISK CORE DUMP SPACE	$CC3LD	$CC3LD
SU580	*ERROR* INSUFFICIENT DISK TRACE SPACE	$CC3LD	$CC3LD
SU585	ALLOCATE PROGRAM REQUEST COUNT TABLE	$CC3LD	$CC3LD
SU590	LOAD CCP TRACE MODULE	$CC3LD	$CC3LD

Startup Messages (continued)

Message ID	Message Text	Contained In	Issued By
SU595	ALLOCATE CCP TRACE TABLE	$CC3LD	$CC3LD
SU600	LOAD SERVICE AID MODULE	$CC3LD	$CC3LD
SU605	LOAD BSCA TRACE MODULE	$CC3LD	$CC3LD
SU610	LOAD MLTA TRACE MODULE	$CC3LD	$CC3LD
SU651	CORE EXHAUSTED WHILE LOADING TERMINAL ATTRIBUTE TABLE	$CC3TA	$CC3TA
SU653	*ERROR* MODULE ****** NOT FOUND	$CC3TA	$CC3TA
SU661	*ERROR* CORE EXHAUSTED WHILE BUILDING CONTENTS DIRECTORY ENTRIES	$CC3CE	$CC3CE
SU671	*ERROR* CORE EXHAUSTED WHILE BUILDING PCT INDEX	$CC3PX	$CC3PX
SU681	*ERROR* CORE EXHAUSTED WHILE BUILDING USER TASK CONTROL BLOCKS	$CC3UB	$CC3UB
SU682	*ERROR* $$SLP1 NOT FOUND ON THE SYSTEM PACK	$CC3UB	$CC3UB
SU691	*ERROR* CORE EXHAUSTED WHILE BUILDING SYMBOLIC FILE SPECIFICATION BLOCKS	$CC3FB	$CC3FB
SU701	*ERROR* CORE EXHAUSTED WHILE BUILDING SECTOR ENQUEUE BLOCKS	$CC3QB	$CC3QB
SU710	*ERROR* MULTIVOLUME FILES NOT SUPPORTED, FILENAME: ********	$CC3DL	$CC3DL
SU715	*ERROR* MISSING OCL, FILENAME: ********	$CC3DL	$CC3DL
SU725	*ERROR* CORE EXHAUSTED WHILE BUILDING SHORT DTFs	$CC3DL	$CC3DL
SU727	CORE EXHAUSTED WHILE BUILDING SYMBOLIC FILE REFERENCE POINTERS	$CC3DL	$CC3DL
SU730	*ERROR* INVALID FILENAME OR LABLE: ********	$CC3DL	$CC3DL
SU735	*ERROR* DUPLICATE FILENAME: ********	$CC3DL	$CC3DL
SU751	PREVIOUS ERRORS, NO DISKFILES OPENED	$CC3DF	$CC3DF
SU753	OPENING DISKFILES	$CC3DF	$CC3DF
SU759	*ERROR* CORE EXHAUSTED WHILE BUILDING MASTER INDEXES	$CC3DF	$CC3DF

Startup Messages (continued)

Message ID	Message Text	Contained In	Issued By
SU760	*ERROR* $CCPLOG FILE IS NOT ON SPECIFIED UNIT	$CC3DF	$CC3DF
SU901	*ERROR* TRANSIENT $CC4** NOT FOUND	$CC3EJ	$CC3EJ
SU915	DYNAMIC T/P BUFFER AREA = ***** BYTES	$CC3EJ	$CC3EJ
SU916	USER PROGRAM AREA = **.**K	$CC3EJ	$CC3EJ
SU917	UNALLOCATED CORE = **.**K	$CC3EJ	$CC3EJ
SU918	SPECIFY ANY EXTRA USER PROGRAM AREA (NN.NNK)	$CC3EJ	$CC3EJ
SU921	*ERROR* INVALID SYNTAX	$CC3EJ	$CC3EJ
SU925	*ERROR* INVALID RESPONSE	$CC3EJ	$CC3EJ
SU940	SPECIFY ANY OFFLINE TERMINAL ID	$CC3EJ	$CC3EJ
SU941	TWO NON-BLANK CHARACTERS REQUIRED	$CC3EJ	$CC3EJ
SU942	TERMINAL ALREADY OFFLINE	$CC3EJ	$CC3EJ
SU945	TERMINAL ID NOT FOUND	$CC3EJ	$CC3EJ
SU951	*ERROR* UNABLE TO OPEN MLTA ADAPTER	$CC3EJ	$CC3EJ
SU952	*ERROR* UNABLE TO OPEN MLTALINE–*, CODE–**	$CC3EF	$CC3EJ
SU953	CCP TERMINATED	$CC3EJ	$CC3EJ
SU997	OPENING COMMUNICATION LINES	$CC3EJ	$CC3EJ
SU999	ERROR DURING STARTUP, CCP TERMINATED	$CC3EJ	$CC3EJ
NONE	ANY REPS FOR THIS RUN?	$CC3CX	$CC3CX
NONE	READY SYSIN DEVICE	$CC3CX	$CC3CX
NONE	SYSIN READ ERROR – PRESS END WHEN READY TO RETRY	$CC3CX	$CC3CX
NONE	NOT REP CARD – PRESS END WHEN READY TO RETRY	$CC3CX	$CC3CX
NONE	REP CARD EMPTY – PRESS END WHEN READY TO RETRY	$CC3CX	$CC3CX
NONE	BAD ADDRESS – PRESS END WHEN READY TO RETRY	$CC3CX	$CC3CX
NONE	NO PATCH VALUES – PRESS END WHEN READY TO RETRY	$CC3CX	$CC3CX

Startup Messages (continued)

Message ID	Message Text	Contained In	Issued By
NONE	ODD NUMBER HEX DIGITS – PRESS END TO RETRY	$CC3CX	$CC3CX
NONE	BAD HEX DIGIT IN PATCH – PRESS END TO RETRY	$CC3CX	$CC3CX
NONE	DELIMITER ERROR – PRESS END TO RETRY	$CC3CX	$CC3CX

System Operator Messages

Message ID	Message Text	Contained In	Issued By
000	CCP STARTED	$CC4SU	$CC4SU
022	REJECTED–PGM–REQUEST PENDING/ACTIVE	$CC4QQ	$CC4QQ
060	REJECTED–NO BUFFER	$CC4K7	$CC4K7
061	CONSOLE NOT AVAILABLE FOR REQUESTS	$CC4EA	$CC4R1
066	I/O ERROR ON PCT FILE–PROGNAME 'XXXXXX'	$CC4R8	$CC4R8
093	REJECTED–PGM–REQUEST PENDING/ACTIVE	$CC4ED	$CC4FL
180	OK	$CC4EU	$CC4AP
181	CCP ALREADY HAS DEVICE	$CC4EU	$CC4AL
182	OTHER LEVEL ALREADY HAS DEVICE	$CC4EU	$CC4AL
183	DEVICE NOT DEFINED TO CCP	$CC4EU	$CC4AL
184	NOT DPF SYSTEM	$CC4EU	$CC4AL
185	DEVICE IN USE BY CCP	$CC4EU	$CC4AL
186	SYNTAX ERROR	$CC4EU	$CC4AL
187	SCHEDULER INTERLOCK–RETRY	$CC4EU	$CC4AL
188	DEVICE NOT AVAILABLE	$CC4EU	$CC4AN
189	ALLOCATE REQUEST REJECTED–DISK ERROR	$CC4EU $CC4EU	$CC4AN $CC4AP
211	NO REPLY PENDING	$CC4EF	$CC4C2
220	AWAITING REPLIES	$CC4C2	$CC4C2

System Operator Messages (continued)

Message ID	Message Text	Contained In	Issued By
221	TASK ID=N	$CC4C2	$CC4C2
240	SYMBOLIC TERMINAL NAME 'PHYSICAL ID'	$CC4C3	$CC4C3
241	SYNTAX ERROR	$CC4EF	$CC4C3
242	INVALID SYMBOLIC NAME	$CC4EF	$CC4C3
261	INVALID OPERAND	$CC4EF	$CC4C4
270	TERMINAL–'XX/TERMNAME – 'XXXXXX' TASK=# ONLINE/OFFLINE MODE– CMND/CMDI/DATA/ INIT/STBY REQ–Y/N	$CC4C4 $CC4C8	$CC4C4 $CC4C8
280	TSK PGMNAM #CORE #T #F #U STATUS	$CC4C5	$CC4C5
281	TSK 'NAME' #CORE #T #F #U ACTIVE/SUSPEND/ W–XXXX *ALLOCATION*	$CC4C5	$CC4C9
290	TASK(S) NOT ACTIVE	$CC4EG $CC4EG	$CC4C5 $CC4C7
291	SYNTAX ERROR	$CC4EG	$CC4C7
292	INVALID TASK ID	$CC4EG	$CC4C7
293	TASK IN ALLOCATION	$CC4EG	$CC4C7
295	TASK RESOURCES	$CC4C7	$CC4C7
299	LAST TASK – 'TASK ID'	$CC4C7	$CC4C7
300	QUEUE DISPLAY	$CC4C6	$CC4C6
301	NO QUEUED PGM REQUEST	$CC4EF	$CC4C6
302	'SYMBOLIC TERMINAL NAME' 'PROGRAM NAME'	$CC4C6	$CC4C6
318	TERMINAL ALREADY ONLINE	$CC4E2	$CC4VA
319	TERMINAL STILL BEING VARIED OFFLINE	$CC4E2	$CC4VA
320	VARIED ONLINE AS 'SYMBOLIC TERMINAL NAME'	$CC4VB	$CC4VB
321	TERMINAL BEING VARIED OFFLINE	$CC4E3 $CC4E3	$CC4VA $CC4VB
322	TERMINAL VARIED OFFLINE–NO PRIMARY NAME	$CC4E3	$CC4VB

System Operator Messages (continued)

Message ID	Message Text	Contained In	Issued By
323	TERMINAL VARIED OFFLINE–PRIMARY NAME REASSIGNED	$CC4E3	$CC4VB
324	SYNTAX ERROR	$CC4E2	$CC4VA
325	SYMBOLIC NAME NOT ASSIGNED TO TERMINAL	$CC4E2	$CC4VA
326	UNKNOWN SYMBOLIC NAME	$CC4E2	$CC4VA
327	INVALID TERMINAL ID	$CC4E2	$CC4VA
329	INVALID/MISSING OPERAND	$CC4E2	$CC4VA
330	NOT CURRENT SYMBOLIC NAME	$CC4E2	$CC4VA
331	CAN'T VARY SUB-TERMINAL	$CC4E2	$CC4VA
332	CAN'T VARY CONSOLE	$CC4E2	$CC4VA
333	TP LINE NOT OPEN	$CC4E2	$CC4VA
334	TERMINAL HAS NO SYMBOLIC NAME	$CC4E2	$CC4VA
335	TERMINAL IN ONLINE TEST	$CC4E3	$CC4VB
336	TERMINAL BELONGS TO USER PROGRAM	$CC4E3	$CC4VB
338	TERMINAL ALREADY OFFLINE	$CC4E3	$CC4VB
340	MESSAGE SENT	$CC4EH	$CC4CG
341	SYNTAX ERROR	$CC4EH	$CC4CG
342	INVALID TERMINAL ID	$CC4EH	$CC4CG
343	UNKNOWN SYMBOLIC NAME	$CC4EH	$CC4CG
344	NAME UNASSIGNED	$CC4EH	$CC4CG
345	NOT A COMMAND TERMINAL	$CC4EH	$CC4CG
346	TERMINAL OFFLINE	$CC4EH	$CC4CG
347	TERMINAL BELONGS TO USER PROGRAM	$CC4EH	$CC4CG
348	MESSAGE TOO LONG	$CC4EH	$CC4CG
350	TERMINAL IN ERROR RECOVERY	$CC4EH	$CC4CG

System Operator Messages (continued)

Message ID	Message Text	Contained In	Issued By
352	TERMINAL IN TEST	$CC4EH	$CC4CG
353	LINE DISCONNECTED	$CC4EH	$CC4CG
360	NAME ASSIGNED	$CC4E4	$CC4AT
361	SYNTAX ERROR	$CC4E4 $CC4E4	$CC4AS $CC4AT
362	INVALID/MISSING OPERAND	$CC4E4 $CC4E4	$CC4AS $CC4AT
363	NAME NOT KNOWN	$CC4E4	$CC4AS
364	CAN'T ASSIGN SUB-TERMINAL NAME	$CC4E4	$CC4AS
365	CAN'T ASSIGN CONSOLE	$CC4E4	$CC4AS
366	NAME CURRENTLY ACTIVE	$CC4E4	$CC4AS
367	NAME IN USE	$CC4E4	$CC4AS
369	INVALID TERMINAL ID	$CC4E4	$CC4AS
370	NAME UNAVAILABLE	$CC4E4	$CC4AS
371	NAME TOO LONG	$CC4E4	$CC4AS
372	PHYSICAL CHARACTERISTICS DON'T MATCH	$CC4E5	$CC4AU
373	INVALID PHONE INDEX #	$CC4E5	$CC4AT
374	PHONE INDEX # INVALID FOR NONSWITCHED TERMINAL	$CC4E5	$CC4AT
375	NAME ASSIGNED 'XX' NAME IS NOW XXXXXX	$CC4AT	$CC4AT
378	START TEST ON 'XX' (— — LOOP)	$CC4MZ $CC4TZ	$CC4MZ $CC4TZ
379	STOP LOOP TEST ON 'XX'	$CC4MZ	$CC4MZ
381	TEST REQ. ALREADY ACTIVE	$CC4EL	$CC4MZ
382	STOP INVALID NOW	$CC4EL	$CC4MZ
383	INVALID TEST NUMBER	$CC4EL	$CC4MZ
384	INVALID COUNT VALUE	$CC4EL	$CC4TZ

System Operator Messages (continued)

Message ID	Message Text	Contained In	Issued By
385	TERMINAL NOT OUTPUT CAPABLE	$CC4EL	$CC4TS
386	TERMINAL OFFLINE	$CC4EL	$CC4TS
387	TERMINAL IN DATA MODE	$CC4EL	$CC4MZ
388	SYMBOLIC NAME NOT FOUND	$CC4EL	$CC4TS
389	INVALID TERMINAL ID	$CC4EL	$CC4TS
390	'XX' ONLINE TEST ENDED	$CC4T2	$CC4T2
391	SYNTAX ERROR	$CC4EL $CC4EL $CC4EL	$CC4TS $CC4TZ $CC4MZ
392	'XX' ONLINE TEST NOT STARTED	$CC4MT	$CC4MT
393	CANNOT START OLT, LINE CLOSED	$CC4EL	$CC4MZ
394	TEST MSG. EXCEEDS 60 CHARACTERS	$CC4EL	$CC4TZ
395	SWITCHED LINE NOT CONNECTED	$CC4EE	$CC4TS
396	SELECTED TERMINAL NOT A CPU	$CC4	$CC4TZ
397	NAME UNASSIGNED	$CC4EE	$CC4TS
400	ERP ACCEPTED	$CC4RP	$CC4RP
401	SYNTAX ERROR	$CC4E1	$CC4RP
402	INVALID TERMINAL ID	$CC4E1	$CC4RP
403	TERMINAL NOT IN ERROR RECOVERY	$CC4E1	$CC4RP
404	INVLAID OPERAND	$CC4E1	$CC4RP
410	(INPUT/OUTPUT) ERROR ON 'XX'	$CC4MA	$CC4MA
411	(INPUT/OUTPUT) ERROR–NN ON 'XX'	$CC4BL $CC4BL $CC4BL	$CC4BE $CC4BA $CC4B5
413	OUTPUT TO 'XX' IN CCP ERROR RECOVERY IGNORED	$CC4MP	$CC4MP

System Operator Messages (continued)

Message ID	Message Text	Contained In	Issued By
414	'XX' SIGNED OFF BY CCP AS YYYYYY	$CC4MF	$CC4MF
415	SWITCHED LINE MN DISCONNECTED–I/O PURGED	$CC4MB	$CC4MB
416	'XX' IN CCP ERROR RECOVERY	$CC4MD	$CC4MD
417	'XX' IN CCP ERROR RECOVERY–BYPASSED	$CC4MD	$CC4MD
418	TP LINE MN CLOSED	$CC4ME	$CC4ME
419	'XX' VARIED OFFLINE	$CC4ME	$CC4ME
420	TRACE OK	$CC4EF	$CC4TE
421	TRACE NOT IN SYSTEM	$CC4EF	$CC4TE
422	INVALID OPERAND	$CC4EF	$CC4TE
423	TRACE NOT ON	$CC4EF	$CC4TE
425	REJECTED–DISK TRACE I/O ERROR	$CC4EF	$CC4TE
440	OK	$CC4EP	$CC4SS
441	INVALID OPERAND	$CC4EP	$CC4SS
442	INVALID TASK–ID + PROGRAM NAME	$CC4EP	$CC4SS
443	TASK ALREADY SUSPENDED	$CC4EP	$CC4SS
460	OK	$CC4EP $CC4EP	$CC4RE $CC4RF
461	SYNTAX ERROR	$CC4EP	$CC4RF
462	COMMANDS NOT SUSPENDED	$CC4EP	$CC4RE
463	INVALID TASK ID	$CC4EP	$CC4RE
464	INIT REJECTED, USERS IN EFFECT	$CC4EP	$CC4RE
465	INVALID PROGRAM NAME	$CC4EP	$CC4RE
466	SUSPEND USERS NOT IN EFFECT	$CC4EP	$CC4RE
467	SUSPEND INIT NOT IN EFFECT	$CC4EP	$CC4RE

System Operator Messages (continued)

Message ID	Message Text	Contained In	Issued By
468	PROGRAM TERMINATING	$CC4EP	$CC4RE
480	CANCEL OK	$CC4CX	$CC4CX
481	INVALID CANCEL REQUEST	$CC4EC	$CC4CJ
482	PGM–NAME TOO LONG	$CC4EC	$CC4CJ
483	TASK–ID AND/OR PGM–NAME WRONG	$CC4EC	$CC4CJ
484	INVALID SYNTAX	$CC4EC	$CC4CJ
485	INVALID USER TASK ID	$CC4EC	$CC4CJ
486	TASK NOT ACTIVE	$CC4EC	$CC4CJ
487	CAN'T CANCEL PGM IN ALLOCATION	$CC4EC	$CC4CJ
500	SHUTDOWN ACCEPTED	$CC4EF	$CC4SH
501	PRIOR SHUTDOWN REQUESTED	$CC4EF	$CC4SH
502	CANCEL/SHUTDOWN STARTED	$CC4EJ	$CC4EJ
503	CCP CANCEL/SHUTDOWN COMPLETED	$CC5SH	$CC5SH
504	DISK ERROR OCCURRED WHILE READING PCTs	$CC5SH	$CC5SH
505	DISK ERROR OCCURRED WHILE WRITING PCTs	$CC5SH	$CC5SH
506	DISK ERROR OCCURRED WHILE WRITING FINAL DISK TRACE ENTRY	$CC5SH	$CC5SH
507	ERROR OCCURRED WHILE CLOSING FILE ********	$CC5SH	$CC5SH
508	SHUTDOWN REJECTED – LOOPING TEST ON '**'	$CC4SH	$CC4SH
520	'XX' SIGNED OFF – (HOLD/DROP) NAME IS XXXXXX	$CC40G	$CC40G
521	'XX' RELEASED AND SIGNED OFF NAME IS XXXXXX	$CC4RY	$CC4RY
522	'XX' NAME IS NOW XXXXXX	$CC4NM	$CC4NM
524	FOR PGM–XXXXXX (TASK–N): LINE (B/M) N DIAL #NNNNN	$CC4WD	$CC4WD
525	FOR PGM–XXXXXX (TASK–N): LINE (M/B) N WAITING TO RECEIVE CALL	$CC4WD	$CC4WD

System Operator Messages (continued)

Message ID	Message Text	Contained In	Issued By
526	XXXXXX TASK – N CODE – NN (REQR – 'XX' DUMP# – N)	$CC4TW	$CC4TW
527	'XX' SIGN ON (ATTEMPT/OK)	$CC4SO	$CC4SO
528	DFF ERROR IN DATA AT POSITION ****	$CC4DD	$CC4DD
529	TERMINAL RELEASE: XXXXXX ('XX')	$CC4NC	$CC4NC
530	** CCP CNCL** CMP–XX (DUMP# –N)	$CC4TW	$CC4TW
533	POLL LOOP COUNT EXCEEDED ON BSCA LINE X	$CC4BL	$CC4BE
534	INVALID TEST REQUEST FOR 'XX'	$CC4BL	$CC4BE
900	FROM 'XX': TEST OF MESSAGE	$CC4MG	$CC4MG
951	INPUT TOO LONG	$CC4K7	$CC4K7
952	NO TASK X AWAITING REPLY	$CC4K7	$CC4K7
990	Text based on issuer of log request.		$CC4H1 $CC4H2
991	Text based on issuer of log request.		$CC4H1 $CC4H2
992	Text based on issuer of log request.		$CC4H1
993	Text based on issuer of log request.		$CC4H1 $CC4H2
	ABRT	$CC4KA	$CC4KA
	CNCL	$CC4K7	$CC4K7
	FORM	$CC4K8	$CC4K8
	I/O ATTN	$EURI	$CC4KB

Terminal Operator Messages

Message ID	Message Text	Contained In	Issued By
A01	SIGNED ON – PROCEED	$CC4SO	$CC4SO
A02	MESSAGE SENT	$CC4EF	$CC4EF
A03	Q STATUS – PROCEED	$CC4QQ	$CC4QQ

Terminal Operator Messages (continued)

Message ID	Message Text	Contained In	Issued By
A04	NOQ STATUS – PROCEED	$CC4QQ	$CC4QQ
A05	ACCEPTED – PROCEED	$CC4ED	$CC4NM
A06	ACCEPTED – PROCEED	$CC4ED	$CC4F2
A07	ENTER /MSG, /RELEASE, OR /RUN	$CC4DM	$CC4DM
A08	RELEASED – PROCEED	$CC4NC	$CC4NC
A09	RESUME PROGRAM	$CC4RN	$CC4RN
A10	SIGNED OFF – (HOLD/DROP) TERMINAL	$CC4OG	$CC4OG
A11	CLEAR	$CC4BO	$CC4CM
E01	INVALID SYNTAX	$CC4EC	$CC4PC
E02	OPERAND MISSING	$CC4SO $CC4EC	$CC4SO $CC4PC
E03	ALREADY SIGNED ON – PROCEED	$CC4SO	$CC4SO
E04	PASSWORD ERROR	$CC4SO	$CC4SO
E05	THAT COMMAND NOT VALID NOW	$CC4QQ	$CC4QQ
E06	MESSAGE TOO LONG	$CC4EF	$CC4MG
E07	THAT COMMAND NOT VALID NOW	$CC4EC	$CC4CN
E08	INVALID SIGN OFF OPTION	$CC4EC	$CC4OF
E09	NOT SIGNED ON	$CC4EC	$CC4OF
E10	THAT COMMAND NOT VALID NOW	$CC4EC	$CC4OF
E11	INVALID PROGNAME	$CC4EA	$CC4R3
E12	INPUT DATA NOT ALLOWED	$CC4EA	$CC4R3
E13	PROGNAME TOO LONG	$CC4EA	$CC4R1
E14	THAT COMMAND NOT VALID NOW	$CC4EA	$CC4R1
E15	ENTER /FILE THEN PROG REQ	$CC4EB	$CC4R4
E16	THAT COMMAND NOT VALID NOW	$CC4RN	$CC4RN

Terminal Operator Messages (continued)

Message ID	Message Text	Contained In	Issued By
E17	FILE NAME TOO LONG	$CC4ED	$CC4FL
E18	THAT COMMAND NOT VALID NOW	$CC4ED	$CC4FL
E19	INVALID SYNTAX	$CC4ED	$CC4FL
E20	FILENAME ******** UNKNOWN	$CC4ED	$CC4ED
E21	THIS FILE COMBINATION NOT VALID	$CC4ED	$CC4FL
E22	THAT COMMAND NOT VALID NOW	$CC4ED	$CC4NM
E23	NAME TOO LONG	$CC4ED	$CC4NM
E24	UNKNOWN TERMINAL NAME	$CC4ED	$CC4NM
E25	UNASSIGNED TERMINAL NAME	$CC4ED	$CC4NM
E26	NAME NOT IN FILE TABLE	$CC4ED	$CC4F2
E27	INPUT TOO LONG	$CC4EE	$CC4PC
E32	PRUF INPUT TO NON-PRUF PROGRAM	$CC4EA $CC4R3	$CC4R1 $CC4RC
R01	COMMANDS SUSPENDED – WAIT	$CC4EE	$CC4PC
R02	COMMANDS RESUMED – REENTER	$CC4RF	$CC4RF
R03	REQUESTED PROG SUSPENDED	$CC4EA	$CC4R1
R04	REQUESTED PROG BUSY	$CC4EA	$CC4R1
R05	PROG NO LONGER AVAILABLE	$CC4EA	$CC4R1
R06	SYSTEM BUSY – RETRY OR /Q	$CC4EA	$CC4R2
R07	SYSTEM ERROR – RETRY	$CC4EA	$CC4R2
R08	REQUESTED PROG UNAVAILABLE	$CC4EA	$CC4R3
R09	UNIT RECORD DEVICE UNAVAILABLE	$CC4EB	$CC4R5
R10	LOGICAL TERMINAL UNAVAILABLE	$CC4EB	$CC4R5
R11	DISK FILE NO LONGER AVAILABLE	$CC4EB	$CC4R4
R12	DISK FILE NO LONGER AVAILABLE	$CC4EB	$CC4R4
R13	CORE UNAVAILABLE	$CC4EB	$CC4R7

Terminal Operator Messages (continued)

Message ID	Message Text	Contained In	Issued By
R14	CORE TEMP UNAVAILABLE	$CC4EB	$CC4R7
R15	REQUIRED TERMINAL UNAVAILABLE	$CC4EB	$CC4R5
R16	DISK FILE TEMP UNAVAILABLE	$CC4EB	$CC4R4
R17	FILE TABLE FULL	$CC4ED	$CC4F2
S01	PROG REL – PROCEED	$CC4RX	$CC4RX
S02	PROG REL – SHUTDOWN	$CC4RX	$CC4RX
S03	PROG END – PROCEED	$CC4TY $CC4CX	$CC4TY $CC4CX
S04	SHUTDOWN	$CC4SH $CC4EE	$CC4SH $CC4PC
S05	PROG END – SHUTDOWN	$CC4TY	$CC4TY
S06	PROG CANCELLED – SHUTDOWN	$CC4TY	$CC4TY
S07	NOW ONLINE – NAMED ******	$CC4VB	$CC4VB
S08	NOW OFFLINE	$CC4VB	$CC4VB
S09	PROG CANCELLED – PROCEED	$CC4TY	$CC4TY
S10	USED AS INPUT FROM SYSTEM OPERATOR		$CC4CG
S11	INPUT IGNORED – PROCEED	$CC4EE	$CC4PF
S12	TP ERROR ON INPUT	$CC4RP	$CC4RP

CCP Dump Messages

Message ID	Message Text	Contained In	Issued By
DP010	PERMANENT DISK ERROR	$CCPDD	$CCPDD
CP011	$CCPDD TERMINATED	$CCPDD	$CCPDD
DP012	DUMP NUMBER TOO LARGE	$CCPDD	$CCPDD
DP013	CORE DUMP N FINISHED	$CCPDD	$CCPDD

CCP Dump Messages (continued)

Message ID	Message Text	Contained In	Issued By
DP014	TRACE TABLE DUMP FINISHED	$CCPDD	$CCPDD
DP015	ALL CORE DUMPS FINISHED	$CCPDD	$CCPDD
DP016	INVALID DUMP CONTROL RECORD	$CCPDD	$CCPDD
DP017	INVALID $CCPFILE CHECK CHARACTERS	$CCPDD	$CCPDD
DP018	EXTENT ERROR ON $CCPFILE	$CCPDD	$CCPDD
DP019	$CCPDD PROGRAM CANCELLED	$CCPDD	$CCPDD
DP020	ERROR WHILE READING FROM SYSIN	$CCPDD	$CCPDD
DP021	SELECTED DUMP DATA INVALID, NOT FROM MOST RECENT RUN OF CCP	$CCPDD	$CCPDD
DP022	LOWER CORE VALUE TOO LARGE FOR THIS SYSTEM	$CCPDD	$CCPDD
DP023	$CCPDD PROGRAM FINISHED	$CCPDD	$CCPDD
DP024	NO DISK TRACE TABLE	$CCPDD	$CCPDD
DP025	NO CORE DUMPS PRESENT	$CCPDD	$CCPDD

Installation Verification Program (CCPIVP) Messages

Note: Indicates variables either keyed from the console or supplied by CCPIVP.

Message ID	Message Text	Contained In	Issued By
EJNMSG	PROGRAM DONE. SCRATCH ********. PERFORM OLT.		CCPIVP
EJNMSG	PROGRAM DONE.ERRORS ENCOUNTERED.	CCPIVP	CCPIVP
EMSG1	PLEASE ENTER MM/DD/YY.	CCPIVP	CCPIVP
EMSG2	DATA ENTERED NOT 03 CHARS LONG OR /*. PLEASE TRY AGAIN.	CCPIVP	CCPIVP
EMSG3	DISK FILE NEVER OPENED. GOING TO EJ ROUTINE.	CCPIVP	CCPIVP
EMSG4	PRINT ROUTINE NOT LINK EDITED. GOING TO EJ ROUTINE.	CCPIVP	CCPIVP

Installation Verification Program (CCPIVP) Messages (continued)

Message ID	Message Text	Contained In	Issued By
MESG1	PLEASE ENTER 03 CHARS OF DATA. THIS DATA WILL BECOME A 5444 CONSECUTIVE DISK RECORD OF THE FORM ***–MM/DD/YY–NNN. TO CLOSE FILE ENTER /*.	CCPIVP	CCPIVP
MESG2	DISK FILE CLOSED. ENTER P IF $$LPRT WAS LINK EDITED WITH THIS PROGRAM. DISK FILE WILL BE DUMPED TO THE PRINTER. ANY OTHER CHARS WILL CAUSE THE DISK FILE TO BE DUMPED TO THE CONSOLE	CCPIVP	CCPIVP
MSGRY1	TNAME–******, UNXPCTD RET CODE ****, ENTER TA OR **	CCPIVP	CCPIVP
MSG1CD	PLEASE ENTER 03 CHARS OF DATA. TO CLOSE ENTER /*	CCPIVP	CCPIVP
RQSMSG	THIS COPY OF CCPIVP REQUESTED BY ******.	CCPIVP	CCPIVP

DISPLAY FORMAT TEST ROUTINE DIAGNOSTIC MESSAGES

During execution, the display format test routine ($CCPDT) diagnoses error conditions and logs diagnostic messages on the system printer. The messages consist of a message number followed by a severity code and message text. The severity codes are:

W - *Warning* — If the input data received from the SYSIN device is not in the correct form, a message indicating why is printed. Also, if the format is not on the program pack, a warning message is printed.

I - *Information* — Informs the user of why requests were rejected, and the type of attention identification (AID) received from the terminal.

DT010 W INVALID OPTION ENTERED

Explanation - The options that can be entered are A, B, C, D, and E.
System Action - $CCPDT returns to SYSIN to read another option.

DT011 W BLANK NOT FOUND BETWEEN PARAMETERS

Explanation - A blank must be between each parameter.
System Action - $CCPDT returns to SYSIN to read another option.

DT012 I REVERSE INTERRUPT WAS RECEIVED

Explanation - The terminal received a reverse interrupt in response to polling and addressing.
System Action - The $CCPDT reads the status of the display and prints it on the system printer; then retries the request.

DT013 W STATION DID NOT RESPOND TO POLLING AND ADDRESSING

Explanation - The wrong poll and address characters were entered.
System Action - $CCPDT returns to SYSIN to read another option.

DT014 I REMOTE STATION DOES NOT RESPOND TO ATTEMPTED DATA TRANSFER

Explanation - Remote station responded to polling and addressing but the request for data transfer is not accepted (possible hardware problem).
System Action - $CCPDT returns to SYSIN to read another option.

DT015 I INCORRECT DATA RECEIVED

Explanation - Data check occurred during transfer of data (possible modem of line problem).
System Action - $CCPDT returns to SYSIN to read another option.

DT016 I SENSE/STATUS BYTES INDICATE_____

Explanation - This message is printed on the system printer when a request is issued to the 3270 and it is locked with status information. The type of status is put in the blank line. The types are: UNIT SPECIFY, TRANSMISSION CHECK, COMMAND REJECT, INTERVENTION REQUIRED, EQUIPMENT CHECK, DATA CHECK, CONTROL CHECK, and OPERATION CHECK.
System Action - $CCPDT returns to SYSIN to read another option.

Note: For a description of the status information, refer to the *IBM 3270 Information Display System Component Description,* GA27-2749.

DT017 W FORMAT NAME MUST BEGIN WITH $Z

Explanation - The first two characters of the format name must begin with $Z.
System Action - $CCPDT returns to SYSIN to read another option.

DT018 I PROGRAM INTERVENTION CAUSED BY _____

Explanation - Data is entered at the terminal and a program intervention is caused by a PF key, a PA key, the CLEAR key, and the ENTER key. The blank line is filled when a program intervention is received.

System Action - $CCPDT returns to SYSIN to read another option.

DT019 W FORMAT NAME WAS NOT ON PROGRAM PACK

Explanation - The format must be on the same pack from which $CCPDT is loaded.

System Action - $CCPDT returns to SYSIN to read another option.

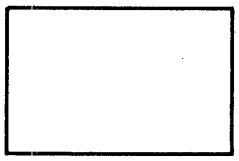
DT020 W INVALID FORMAT NAME

Explanation - The format name must be at least three characters and not more than six characters long.

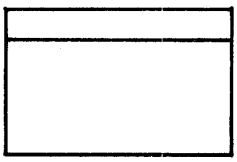
System Action - $CCPDT returns to SYSIN to read another option.

FLOWCHARTS

The following symbols are used in this manual:

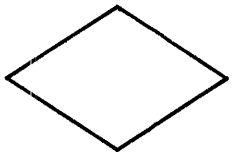


Processing

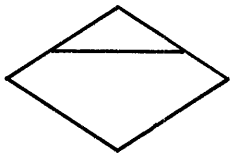


Striped Processing

Predefined Processing

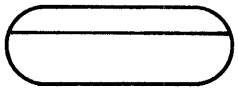


Decision

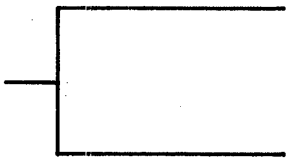


Striped Decision

Entry/Terminal



Striped Entry/Terminal



Comment

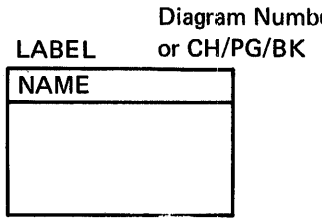
On-Page Connector

Off-Page Connector

Most of the symbols are self-explanatory, but some need more explanation.

1. The striped blocks (striped processing, striped decision, and striped entry/terminal) indicate an exit to a routine flowcharted elsewhere in the manual.

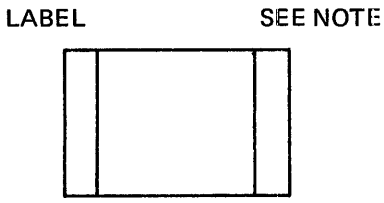
   Example:



   CH/PG/BK indicates the chart ID, page, and block identification of the flowcharted routine.

2. The predefined processing block indicates a routine flowcharted and/or described in another logic manual.

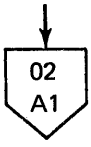
   Example:



   Note: See *logic manual title and order number*.

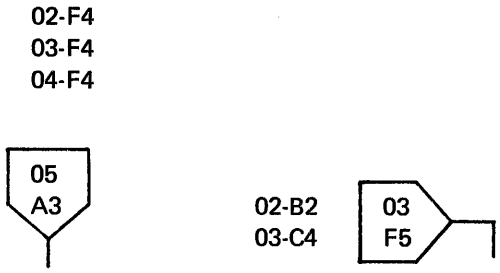
3. Off-page connectors are used to reference between different pages of the same chart ID. Off-page connectors leaving a page contain the page number and block number of their destination.

   Example:



   Off-page connectors entering a page contain the page number and block number of their origin. If the entry point referenced by the off-page connector is referenced from more than one origin, all origins are given. The origins are listed in alphameric order with the last reference contained within the block.

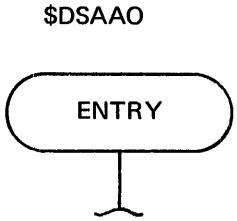
   Example:



   On-page connectors contain the location of a block on the same page. On-page connectors always contain the location of the destination block.

4. The label in the upper lefthand corner just above the entry symbol is the entry point in the listing for that part of the program.

   Example:



HIPO

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

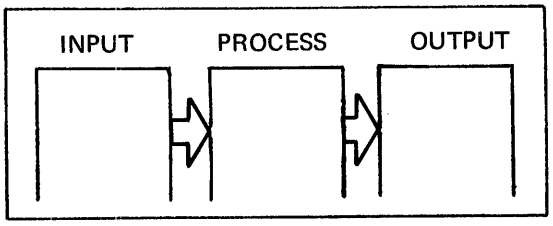
The main objective of this method of diagramming is to improve communication procedures and techniques through the effective use of functional operation diagrams.

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

Though the method of operation diagrams in this manual are not all HIPO diagrams, HIPO techniques are used in the program organization sections to describe many of the CCP modules. These low level HIPO diagrams contain unit level information (that level of information reflecting the actual workings of the code). Each low level diagram is arranged to best show:

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

Stated graphically:



The diagrams contain as few words as possible. There are two reasons for this:

- When the picture becomes cluttered with text, it loses some value as a recall mechanism.
- The degree of difficulty of maintaining the diagrams increases with increased number of words in the picture area.

In the diagrams, functions are related by cross references to the module(s) supporting those functions. The addition of cross reference creates a functional mapping through the implementation.

To summarize graphic and information content:

Boxes used to:

- Localize inputs, process, outputs.
- Isolate related sub-processes within the processing area.
- Localize extended descriptions.

Arrows used to:

- Lead the reader through the process. ➡
- Show data flow. ⟹
- Pointer ⟶ or ⟷

Where more than one page reference is given, the major reference is first.

This page intentionally left blank.