Xerox Real-Time Batch Monitor (RBM)

Xerox 530 and Sigma 2/3 Computers

Technical Manual

90 11 53F

Xerox Corporation
701 South Aviation Boulevard
El Segundo, California 90245
213 679-4511

XEROX

Xerox Real-Time Batch Monitor (RBM)

Xerox 530 and Sigma 2/3 Computers

Technical Manual

90 11 53F

February 1975

Price: $8.75

Printed in U.S.A.

REVISION

This publication, 90 11 53F, is a revision of the Xerox Real-Time Batch Monitor (RBM)/Technical Manual for Xerox 530 and Sigma 2/3 computers, 90 11 53E (dated October 1973). The changes made to the text are for the G00 version of RBM. All changes in the text from that of the previous manual are indicated by a vertical line in the margin of the page.

RELATED PUBLICATIONS

Title	Publication No.
Xerox 530 Computer/Reference Manual	90 19 60
Xerox Sigma 2 Computer/Reference Manual	90 09 64
Xerox Sigma 3 Computer/Reference Manual	90 15 92
Xerox Availability Features/Reference Manual	90 30 54
Xerox Real-Time Batch Monitor (RBM)/RT, BP Reference Manual	90 10 37
Xerox Real-Time Batch Monitor (RBM)/OPS Reference Manual	90 15 55
Xerox Real-Time Batch Monitor (RBM)/SM Reference Manual	90 30 36
Xerox Real-Time Batch Monitor (RBM)/User's Guide	90 17 85

Manual Content Codes: BP - batch processing, LN - language, OPS - operations, RP - remote processing, RT - real-time, SM - system management, TS - time-sharing, UT - utilities.

The specifications of the software system described in this publication are subject to change without notice. The availability or performance of some features may depend on a specific configuration of equipment such as additional tape units or larger memory. Customers should consult their Xerox sales representative for details.

CONTENTS

APPENDIXES

FIGURES

TABLES

PREFACE

The primary purpose of this manual is to provide a guide for better comprehension of the Monitor listings supplied with the Xerox 530 and Sigma 2/3 Real-Time Batch Monitor operating system. The manual is intended for users who require an in-depth knowledge of the structure and internal functions of the system for maintenance purposes.

It is assumed that the reader is familiar with the RBM Reference Manual (90 10 37) and that more detailed information about the various program elements will be obtained from the listings.

Since this manual and the Monitor listings are complementary, it is recommended that the listings be readily available when referencing the manual.

1. INTRODUCTION

Priority Interrupts

Under RBM, both Monitor and user real-time tasks must be connected to a specific, unique hardware priority level. Each task operates at the priority level of its corresponding hardware interrupt. There is no software scheduling of tasks, except for two special cases:

1. The background has no specific hardware interrupt, but operates as though it were connected to the lowest priority interrupt, below all hardware priority levels.

2. The RBM Control Task controls its subtasks on a software-priority basis. The Control Task must be connected to the lowest priority hardware interrupt in the system. Each subtask priority corresponds to a bit in a special core location (R:RBM). When this bit is set to 1, the subtask is active or waiting; when the bit is reset to 0, the subtask is inactive. Thus, RBM tasks and subtasks can set the appropriate bit (or bits) in this status word and trigger the RBM Control Task interrupt with a special write-direct code to provide simple, responsive, and ordered processing of other related subtasks. Since the priority level of the RBM Control Task is lower than all real-time levels and higher than the background, it can provide simple and direct control of all operator communication and all batch background processing.

Figures 1 through 7 show the flow of control through the various RBM tasks, and the sum of all these tasks can be considered to be RBM. Each task is a closed loop in terms of execution; however, a task may pass information to other tasks or may trigger other tasks to ensure that RBM functions take place at the appropriate priority level. The accounting routine for RBM is shown in Figure 5.

Monitor Service Routines

Each Monitor service routine operates as a closed subroutine when called by a task to aid in carrying out task functions. All Monitor service routines can be reentered, can be interrupted during execution at any time by a higher priority task, can perform a service for the higher priority task, and can continue execution of the interrupted task when the higher task releases control.

To achieve reentrancy, certain Monitor service routines require temporary storage. This storage space is provided by the user task's dynamic temporary (temp) stack. The Monitor service routines will reserve the amount of temporary space required by a call to the Monitor service routine M:RES, which will reserve the requested space and return with the base register set to the first word of the reserved temporary space. Since there is a three-cell overhead, all routines requiring temporary space must request the amount actually required plus three. The first three words reserved are used as follows:

1. Word 0 is a flag used by M:POP to determine whether dynamic storage or static storage is used. It is also used by the Monitor routine Q:ROC to control the loading and unloading of nonresident Monitor service routines.

2. Word 1 is set to the previous contents of the base register.

3. Word 2 is set to the return address, initially set by M:RES to M:ABORT.

Thus, Monitor service routines using temp storage do not modify the first two words of the temp space reserved. The appropriate return address must be stored in the third word prior to calling the service routine M:POP to release the temporary space and exit to the address given in the third word.

Entry to all Monitor service routines from the background is through a vector of addresses in the zero table, so that validity of background service calls can be established. All Monitor service routines must verify that requested background operations will not modify protected memory or use any devices reserved for the foreground. Return addresses are verified by the Protection Task when the original entry is made, and the background is aborted if the return is not set to valid background space.

Machine Fault Interrupt Task
Model 530

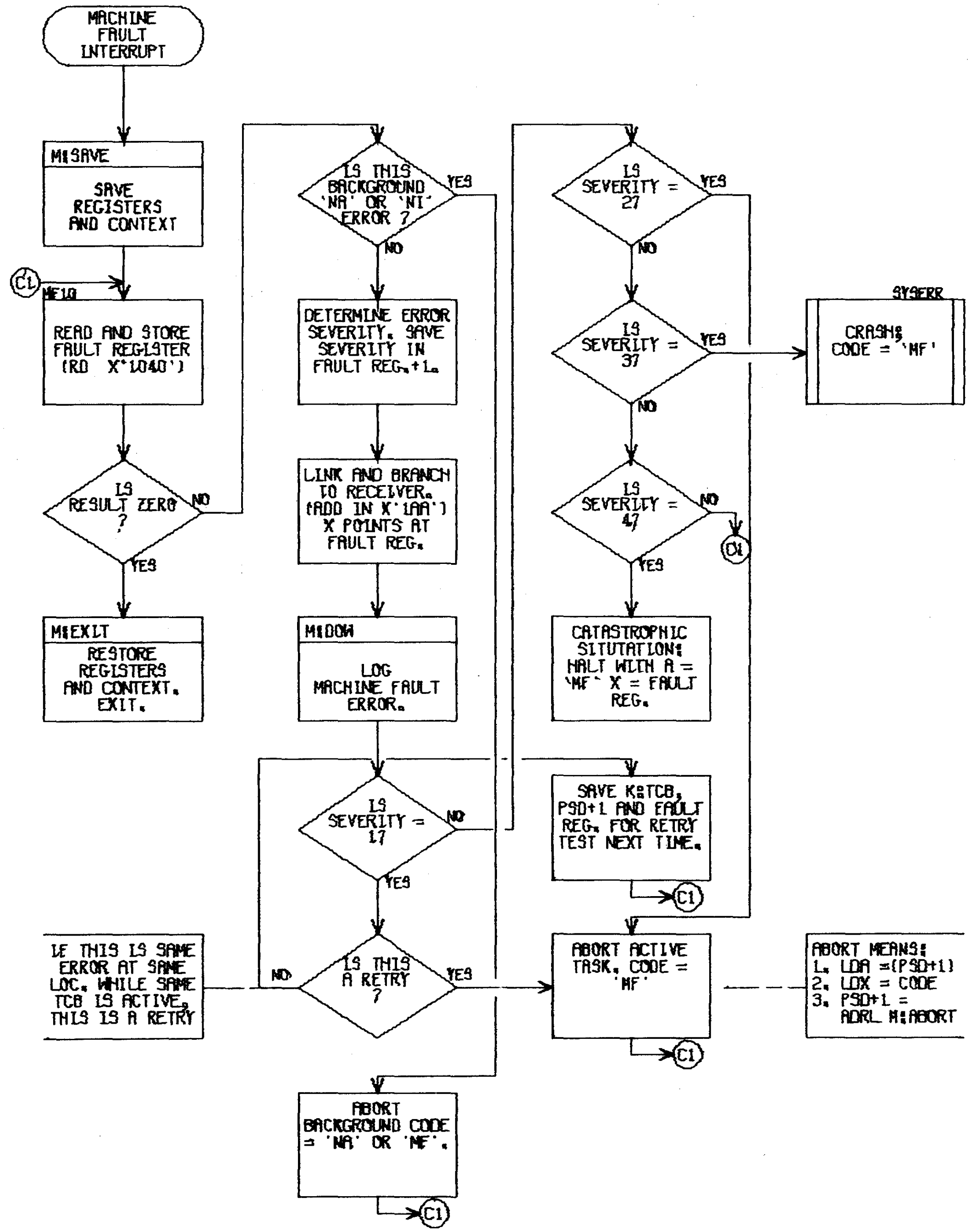

MACHINE FAULT INTERRUPT
M:SAVE
SAVE REGISTERS AND CONTEXT
C1
MF10
READ AND STORE FAULT REGISTER (RD X'1040')
IS RESULT ZERO ?
NO
YES
M:EXIT
RESTORE REGISTERS AND CONTEXT. EXIT.
IS THIS BACKGROUND 'NA' OR 'NI' ERROR ?
YES
NO
DETERMINE ERROR SEVERITY. SAVE SEVERITY IN FAULT REG.+1.
LINK AND BRANCH TO RECEIVER. (ADD IN X'1AA') X POINTS AT FAULT REG.
M:DOW
LOG MACHINE FAULT ERROR.
IS SEVERITY = 17
NO
YES
IF THIS IS SAME ERROR AT SAME LOC. WHILE SAME TCB IS ACTIVE, THIS IS A RETRY
NO
IS THIS A RETRY ?
YES
ABORT BACKGROUND CODE = 'NA' OR 'MF'.
C1
IS SEVERITY = 27
YES
NO
IS SEVERITY = 37
YES
NO
SYSERR
CRASH: CODE = 'MF'
IS SEVERITY = 47
NO
C1
YES
CATASTROPHIC SITUTATION: HALT WITH A = 'MF' X = FAULT REG.
SAVE K:TCB, PSD+1 AND FAULT REG. FOR RETRY TEST NEXT TIME.
C1
ABORT ACTIVE TASK. CODE = 'MF'
C1
ABORT MEANS:
1. LDA =(PSD+1)
2. LDX = CODE
3. PSD+1 = ADRL M:ABORT

Figure 1. RBM Machine Fault Task Flow

Machine Fault Interrupt Task
Sigma 3

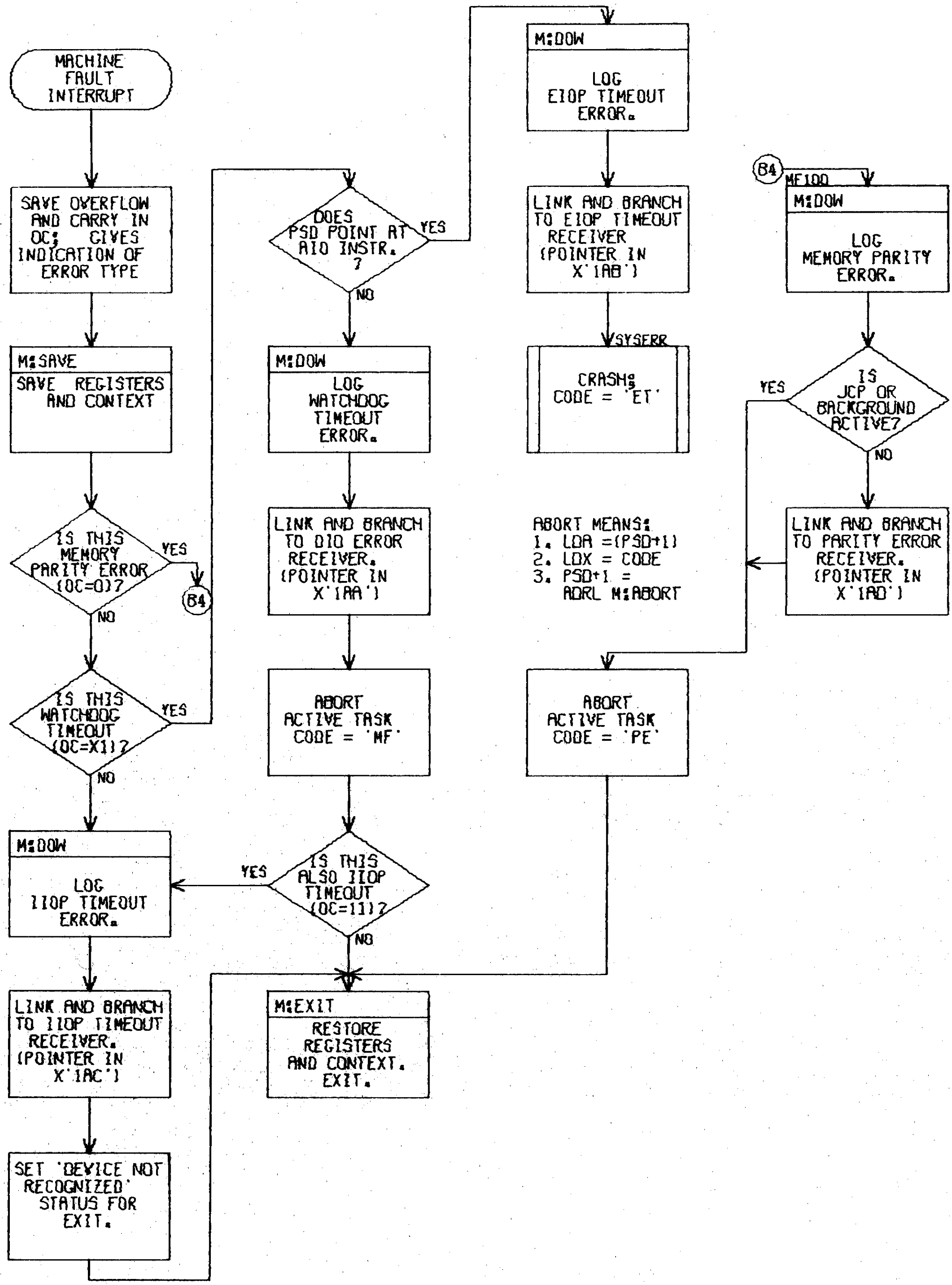

MACHINE FAULT INTERRUPT
SAVE OVERFLOW AND CARRY IN OC; GIVES INDICATION OF ERROR TYPE
M:SAVE
SAVE REGISTERS AND CONTEXT
IS THIS MEMORY PARITY ERROR (OC=0)?
YES
B4
NO
IS THIS WATCHDOG TIMEOUT (OC=X1)?
YES
NO
M:DOW
LOG IIOP TIMEOUT ERROR.
LINK AND BRANCH TO IIOP TIMEOUT RECEIVER. (POINTER IN X'1AC')
SET 'DEVICE NOT RECOGNIZED' STATUS FOR EXIT.
DOES PSD POINT AT AIO INSTR. ?
YES
NO
M:DOW
LOG WATCHDOG TIMEOUT ERROR.
LINK AND BRANCH TO DIO ERROR RECEIVER. (POINTER IN X'1AA')
ABORT ACTIVE TASK CODE = 'MF'
IS THIS ALSO IIOP TIMEOUT (OC=11)?
YES
NO
M:EXIT
RESTORE REGISTERS AND CONTEXT. EXIT.
M:DOW
LOG EIOP TIMEOUT ERROR.
LINK AND BRANCH TO EIOP TIMEOUT RECEIVER (POINTER IN X'1AB')
SYSERR
CRASH; CODE = 'ET'
ABORT MEANS:
1. LDA =(PSD+1)
2. LDX = CODE
3. PSD+1 = ADRL M:ABORT
ABORT ACTIVE TASK CODE = 'PE'
B4
MF100
M:DOW
LOG MEMORY PARITY ERROR.
IS JCP OR BACKGROUND ACTIVE?
YES
NO
LINK AND BRANCH TO PARITY ERROR RECEIVER. (POINTER IN X'1AD')

Figure 1. RBM Machine Fault Task Flow (cont.)

Machine Fault Interrupt Task
Sigma 2

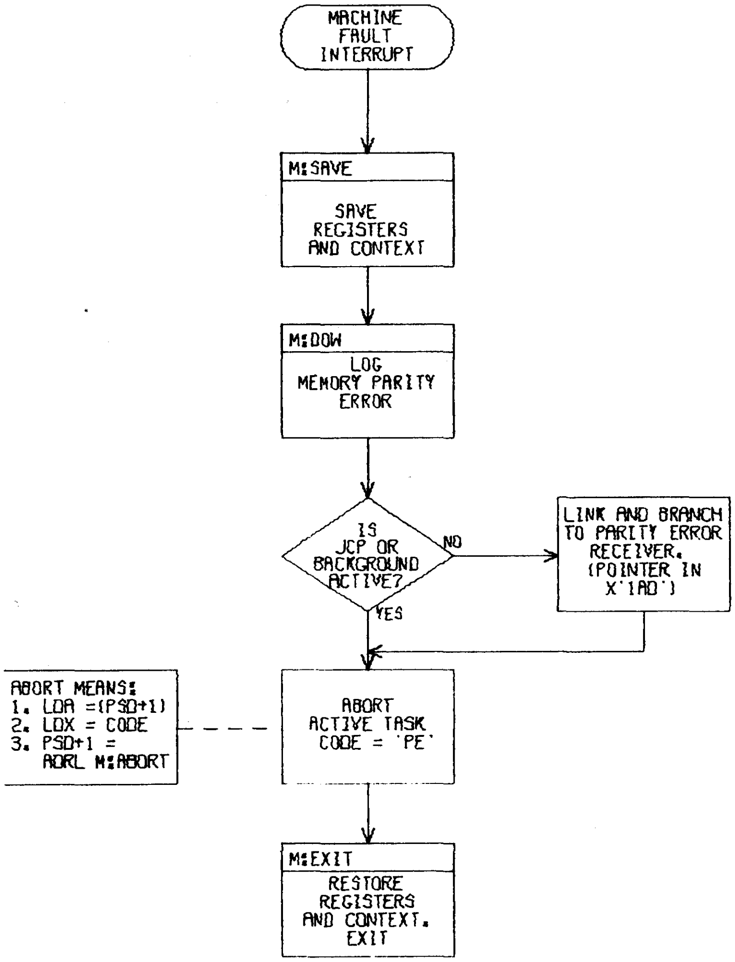
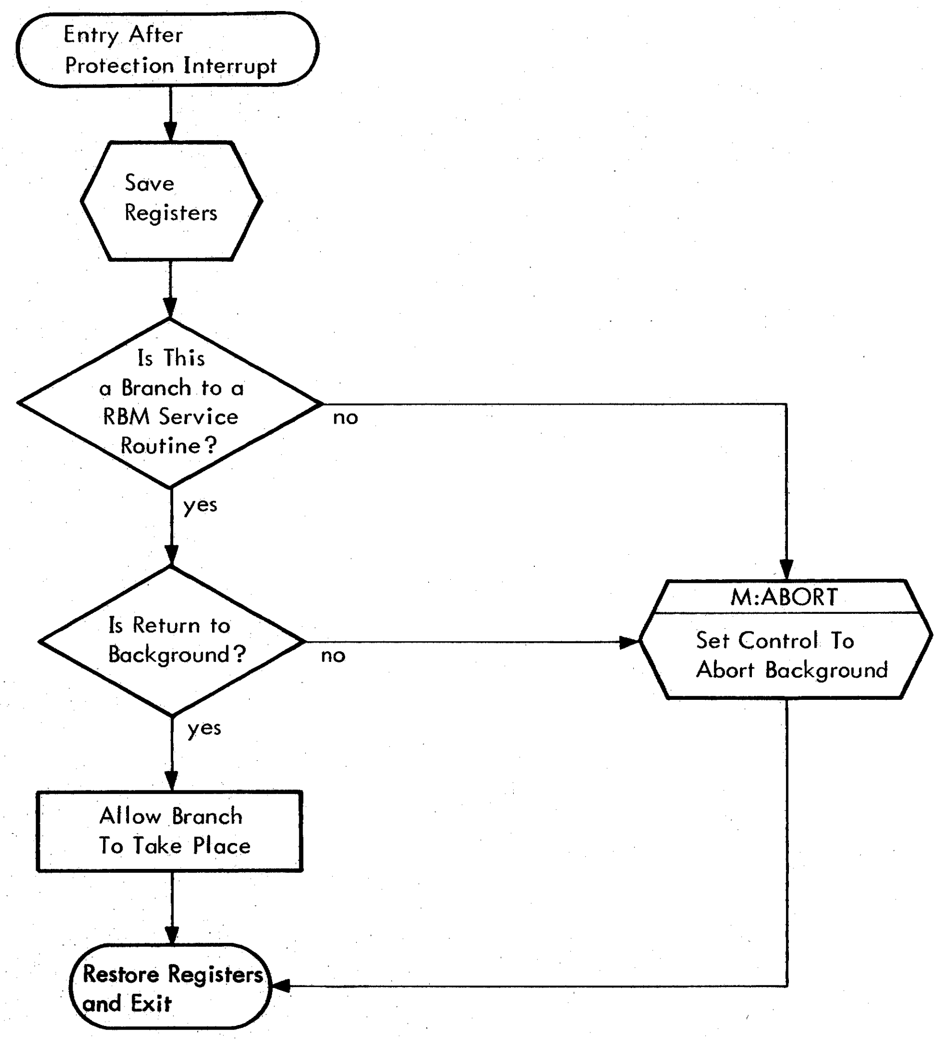

MACHINE FAULT INTERRUPT
M:SAVE
SAVE REGISTERS AND CONTEXT
M:DOW
LOG MEMORY PARITY ERROR
IS JCP OR BACKGROUND ACTIVE?
NO
YES
LINK AND BRANCH TO PARITY ERROR RECEIVER. (POINTER IN X'1AD')
ABORT MEANS:
1. LDA =(PSD+1)
2. LDX = CODE
3. PSD+1 = ADRL M:ABORT
ABORT ACTIVE TASK CODE = 'PE'
M:EXIT
RESTORE REGISTERS AND CONTEXT. EXIT

Figure 1. RBM Machine Fault Task Flow (cont.)


Entry After Protection Interrupt
Save Registers
Is This a Branch to a RBM Service Routine?
no
yes
Is Return to Background?
no
yes
M:ABORT
Set Control To Abort Background
Allow Branch To Take Place
Restore Registers and Exit

Note: Foreground cannot cause a protection violation.

Figure 2. RBM Protection Task Flow

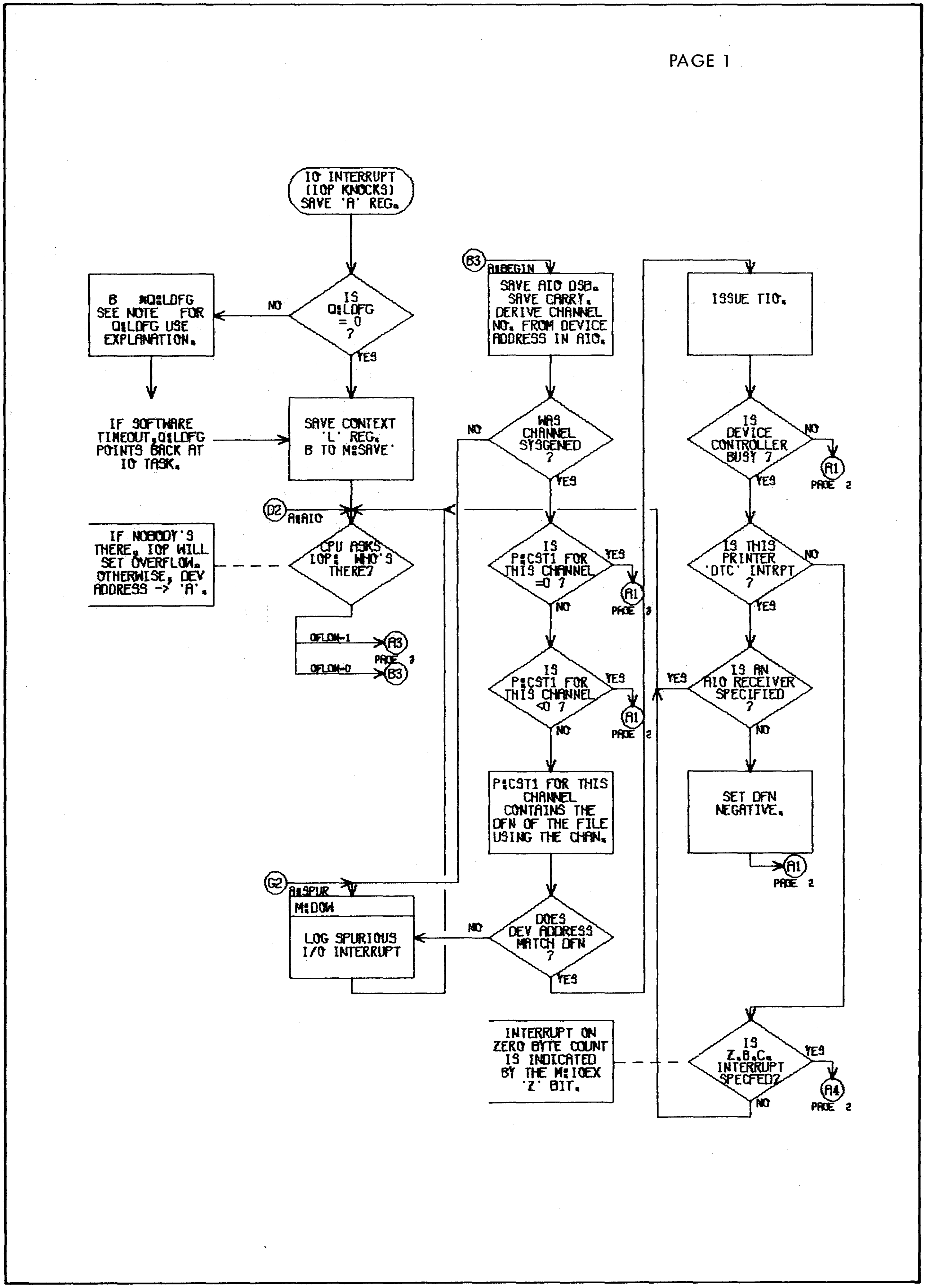

PAGE 1
IO INTERRUPT (IOP KNOCKS) SAVE 'A' REG.
IS Q:LDFG = 0 ?
NO
YES
B *Q:LDFG SEE NOTE FOR Q:LDFG USE EXPLANATION.
IF SOFTWARE TIMEOUT, Q:LDFG POINTS BACK AT IO TASK.
SAVE CONTEXT 'L' REG. B TO M:SAVE'
D2
A:AIO
IF NOBODY'S THERE, IOP WILL SET OVERFLOW. OTHERWISE, DEV ADDRESS -> 'A'.
CPU ASKS IOP: WHO'S THERE?
OFLOW-1
A3
PAGE 3
OFLOW-0
B3
B3
A:BEGIN
SAVE AIO DSB. SAVE CARRY. DERIVE CHANNEL NO. FROM DEVICE ADDRESS IN AIO.
WAS CHANNEL SYSGENED ?
NO
YES
IS P:CST1 FOR THIS CHANNEL =0 ?
YES
A1
PAGE 3
NO
IS P:CST1 FOR THIS CHANNEL <0 ?
YES
A1
PAGE 2
NO
P:CST1 FOR THIS CHANNEL CONTAINS THE DFN OF THE FILE USING THE CHAN.
G2
A:SPUR
M:DOW
LOG SPURIOUS I/O INTERRUPT
NO
DOES DEV ADDRESS MATCH DFN ?
YES
ISSUE TIO.
IS DEVICE CONTROLLER BUSY ?
NO
A1
PAGE 2
YES
IS THIS PRINTER 'DTC' INTRPT ?
NO
YES
YES
IS AN AIO RECEIVER SPECIFIED ?
NO
SET DFN NEGATIVE.
A1
PAGE 2
INTERRUPT ON ZERO BYTE COUNT IS INDICATED BY THE M:IOEX 'Z' BIT.
IS Z.B.C. INTERRUPT SPECFED?
YES
A4
PAGE 2
NO

Figure 3. RBM Input/Output Task Flow

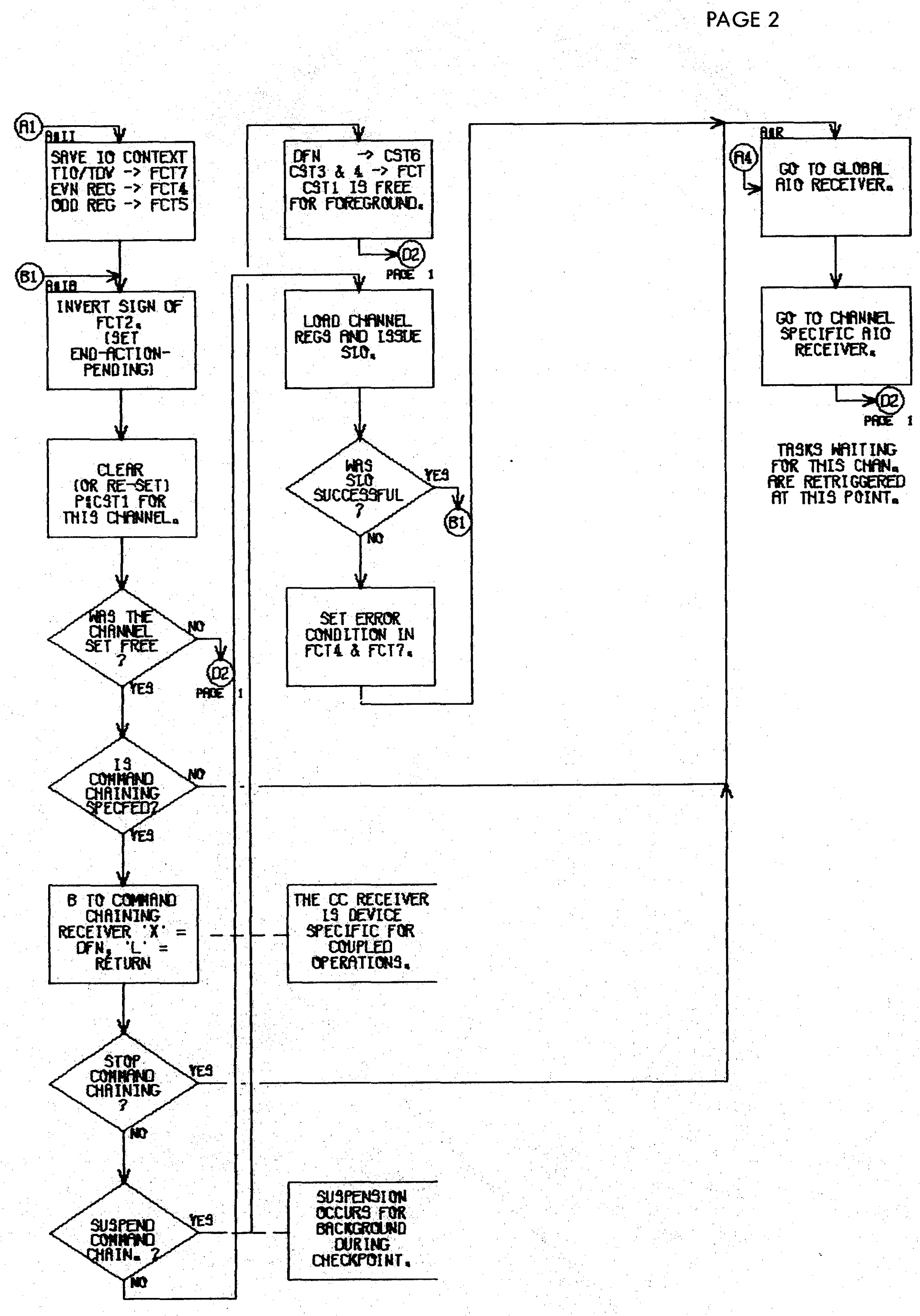
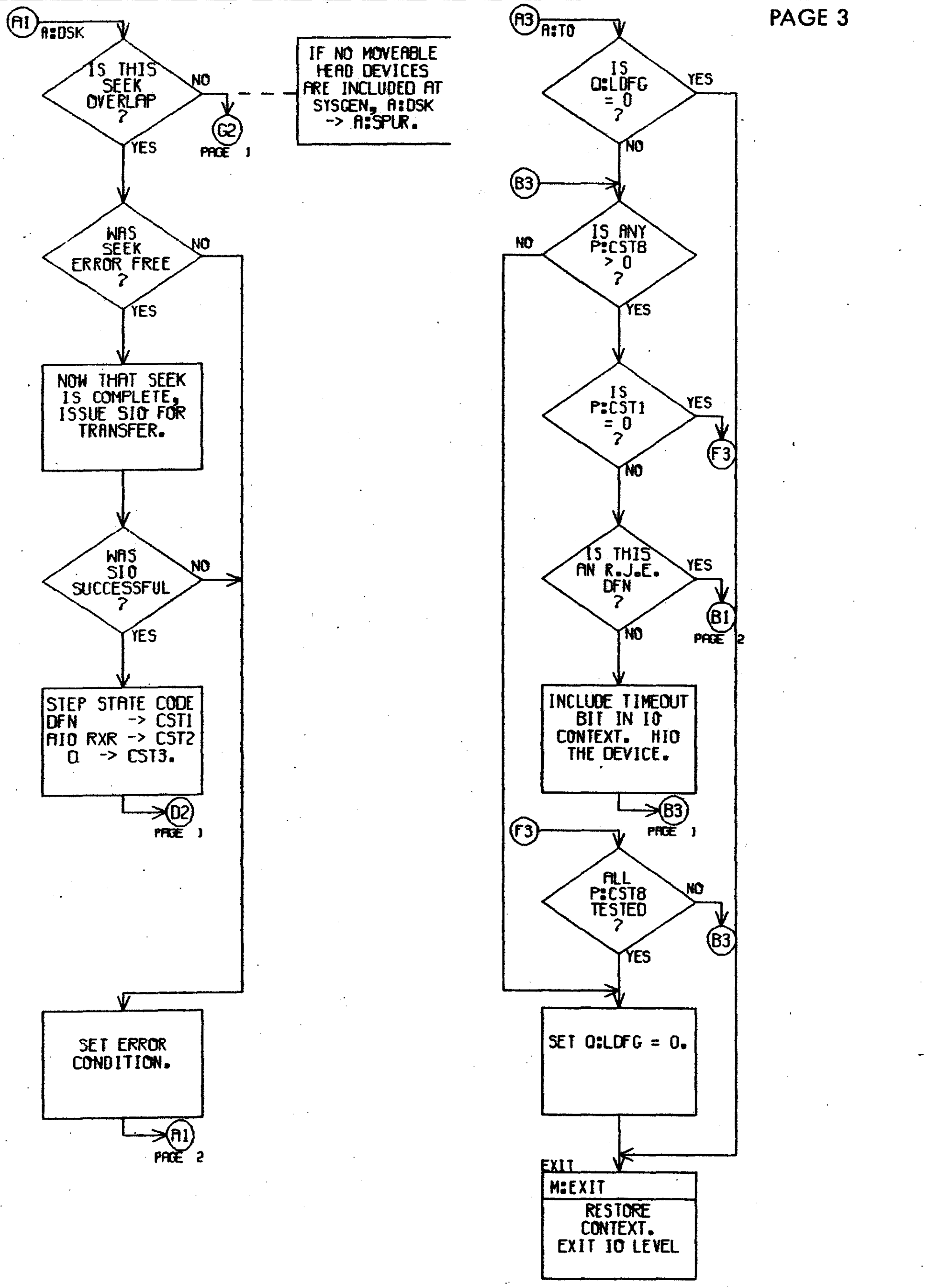

PAGE 2
A1
A:II
SAVE IO CONTEXT TIO/TDV -> FCT7 EVN REG -> FCT4 ODD REG -> FCT5
B1
A:IB
INVERT SIGN OF FCT2. (SET END-ACTION-PENDING)
CLEAR (OR RE-SET) P:CST1 FOR THIS CHANNEL.
WAS THE CHANNEL SET FREE ?
NO
YES
D2
PAGE 1
IS COMMAND CHAINING SPECFED?
NO
YES
B TO COMMAND CHAINING RECEIVER 'X' = DFN, 'L' = RETURN
THE CC RECEIVER IS DEVICE SPECIFIC FOR COUPLED OPERATIONS.
STOP COMMAND CHAINING ?
YES
NO
SUSPEND COMMAND CHAIN. ?
YES
NO
SUSPENSION OCCURS FOR BACKGROUND DURING CHECKPOINT.
DFN -> CST6 CST3 & 4 -> FCT CST1 IS FREE FOR FOREGROUND.
D2
PAGE 1
LOAD CHANNEL REGS AND ISSUE SIO.
WAS SIO SUCCESSFUL ?
YES
B1
NO
SET ERROR CONDITION IN FCT4 & FCT7.
A:R
A4
GO TO GLOBAL AIO RECEIVER.
GO TO CHANNEL SPECIFIC AIO RECEIVER.
D2
PAGE 1
TASKS WAITING FOR THIS CHAN. ARE RETRIGGERED AT THIS POINT.

Figure 3. RBM Input/Output Task Flow (cont.)



Note: Q:LDFG is a convention which allows code to be executed at the I/O interrupt level. This is necessary for non-reentrant code, such as the "load channel registers, set channel status tables, issue SIO" sequence found in Q:LOADC and the RAD handler. These routines inhibit interrupts, set Q:LDFG to the address of the non-reentrant code, trigger the I/O interrupt level and then uninhibit interrupts. The non-reentrant code then resets Q:LDFG, performs the prescribed functions and exits the I/O level.

This convention precludes tasks which run at an interrupt level higher than the I/O level from using Monitor I/O (nearly all Monitor Service Routines use Monitor I/O).

This restriction can be overcome by setting Q:NLDFG to some nonzero value. However, other implications of doing Monitor I/O at a level higher than the I/O level make this a very esoteric feature. For example, guaranteed response time will increase to about 700 to 800 µsec.

Figure 3. RBM Input/Output Task Flow (cont.)

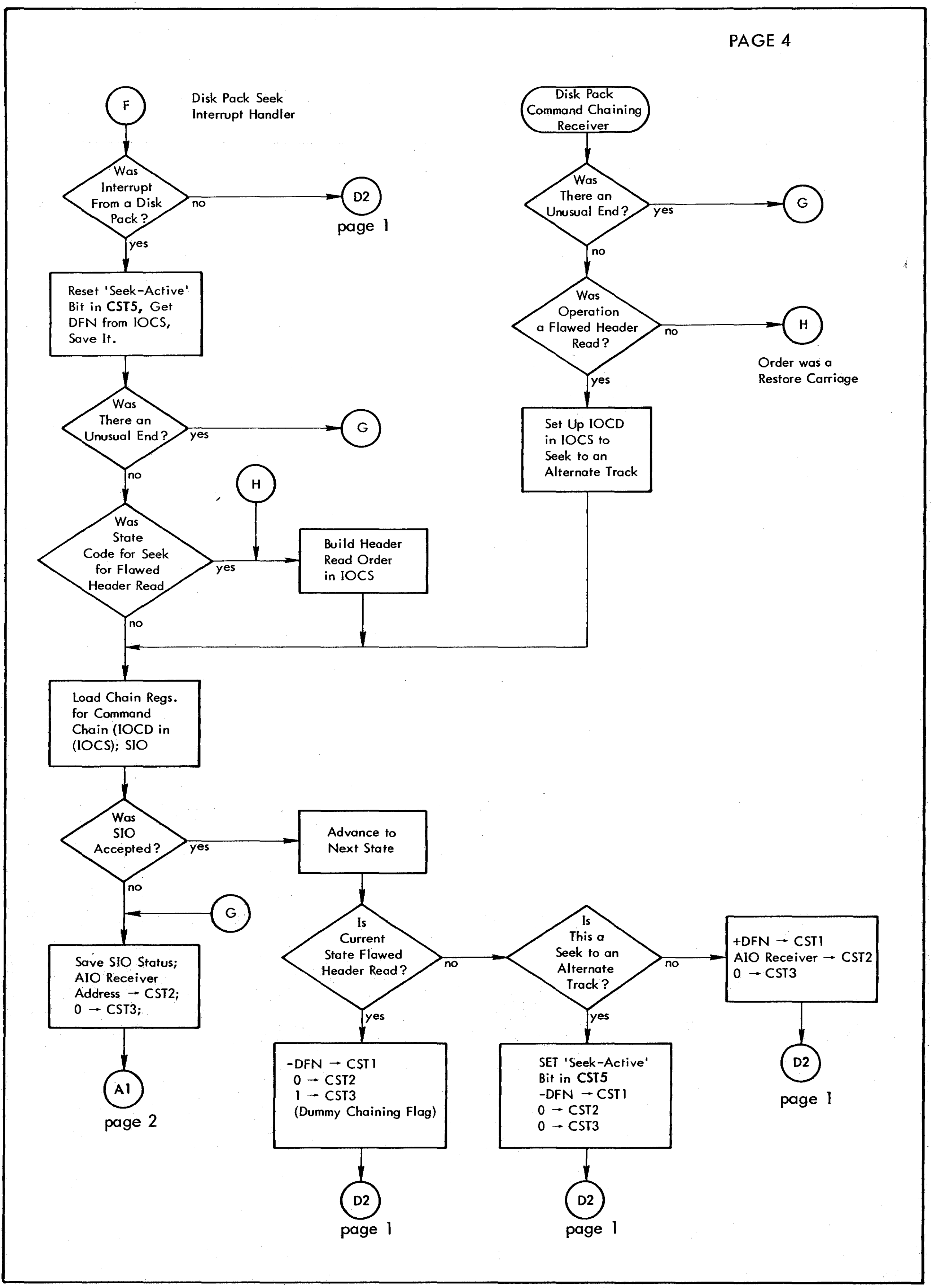

PAGE 4
F
Disk Pack Seek Interrupt Handler
Was Interrupt From a Disk Pack?
no
D2
page 1
yes
Reset 'Seek-Active' Bit in CST5, Get DFN from IOCS, Save It.
Was There an Unusual End?
yes
G
no
H
Was State Code for Seek for Flawed Header Read
yes
Build Header Read Order in IOCS
no
Load Chain Regs. for Command Chain (IOCD in (IOCS); SIO
Was SIO Accepted?
yes
Advance to Next State
no
G
Save SIO Status; AIO Receiver Address → CST2; 0 → CST3;
A1
page 2
Is Current State Flawed Header Read?
no
yes
-DFN → CST1
0 → CST2
1 → CST3
(Dummy Chaining Flag)
D2
page 1
Is This a Seek to an Alternate Track?
no
yes
SET 'Seek-Active' Bit in CST5
-DFN → CST1
0 → CST2
0 → CST3
D2
page 1
+DFN → CST1
AIO Receiver → CST2
0 → CST3
D2
page 1
Disk Pack Command Chaining Receiver
Was There an Unusual End?
yes
G
no
Was Operation a Flawed Header Read?
no
H
Order was a Restore Carriage
yes
Set Up IOCD in IOCS to Seek to an Alternate Track

Figure 3. RBM Input/Output Task Flow (cont.)

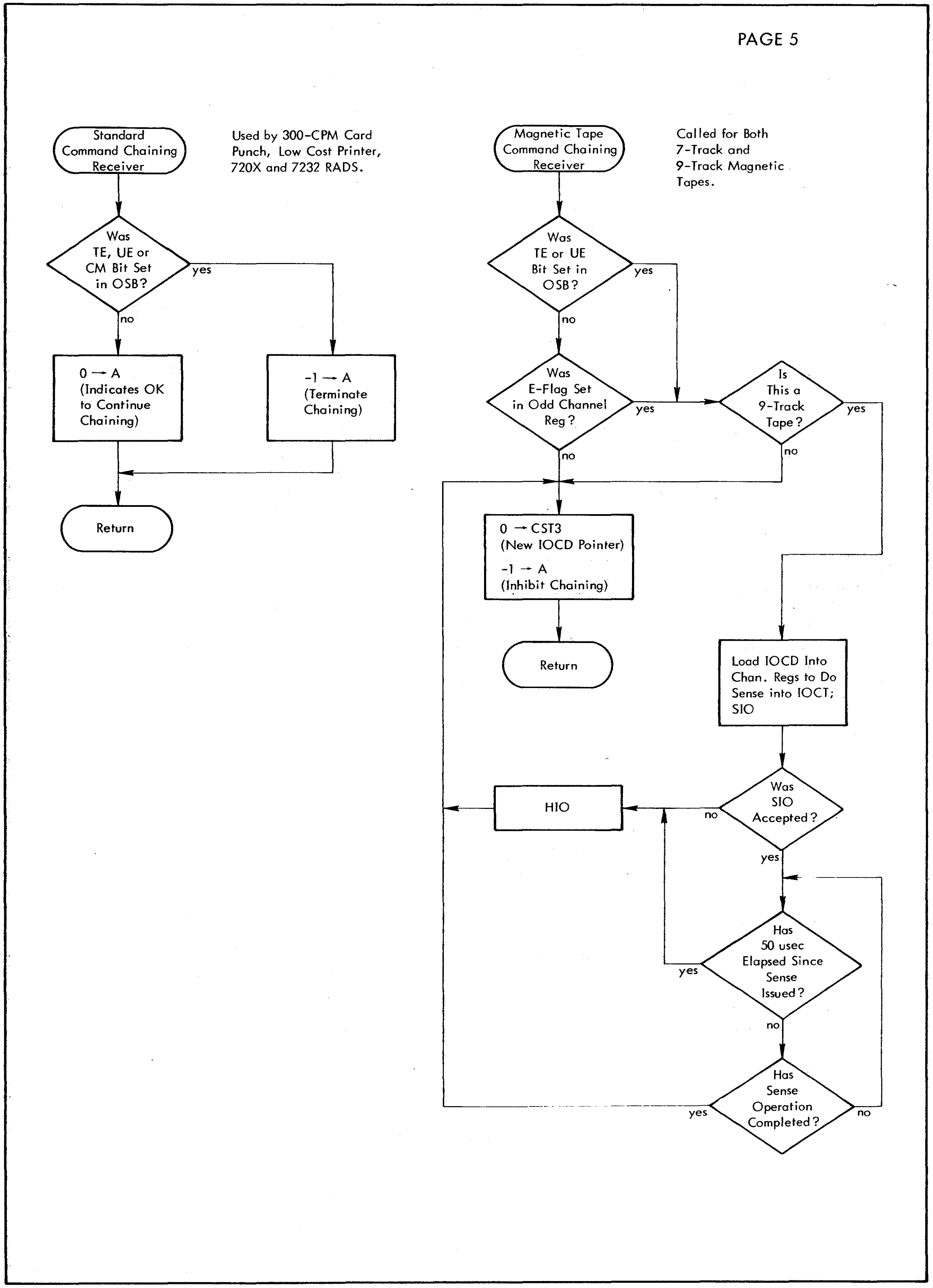

PAGE 5
Standard Command Chaining Receiver
Used by 300-CPM Card Punch, Low Cost Printer, 720X and 7232 RADS.
Was TE, UE or CM Bit Set in OSB?
yes
no
0 → A (Indicates OK to Continue Chaining)
-1 → A (Terminate Chaining)
Return
Magnetic Tape Command Chaining Receiver
Called for Both 7-Track and 9-Track Magnetic Tapes.
Was TE or UE Bit Set in OSB?
yes
no
Was E-Flag Set in Odd Channel Reg ?
yes
no
Is This a 9-Track Tape ?
yes
no
0 → CST3 (New IOCD Pointer)
-1 → A (Inhibit Chaining)
Return
Load IOCD Into Chan. Regs to Do Sense into IOCT; SIO
Was SIO Accepted ?
no
yes
HIO
Has 50 usec Elapsed Since Sense Issued ?
yes
no
Has Sense Operation Completed ?
yes
no

Figure 3. RBM Input/Output Task Flow (cont.)

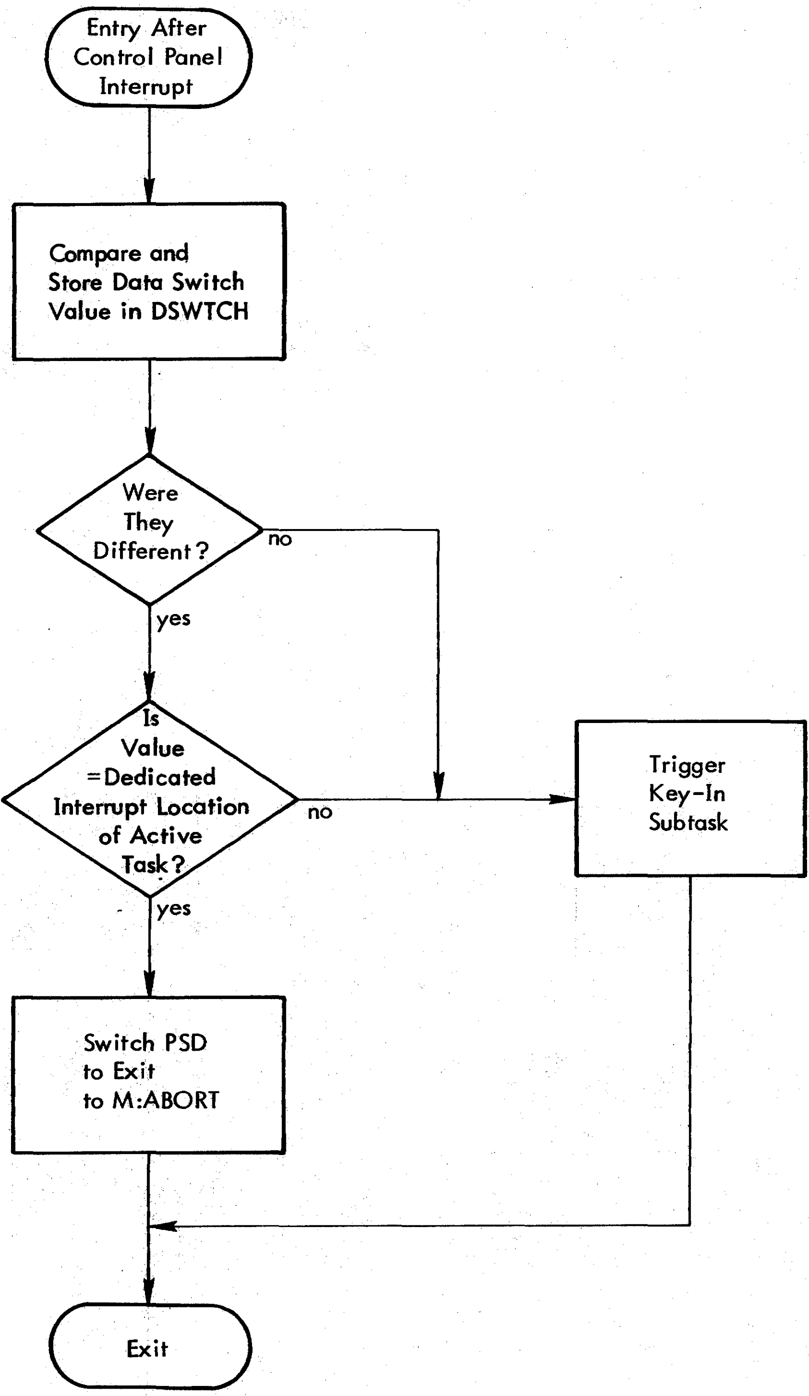

Entry After Control Panel Interrupt
Compare and Store Data Switch Value in DSWTCH
Were They Different?
no
yes
Is Value =Dedicated Interrupt Location of Active Task?
no
yes
Trigger Key-In Subtask
Switch PSD to Exit to M:ABORT
Exit

Figure 4. RBM Control Panel Task Flow

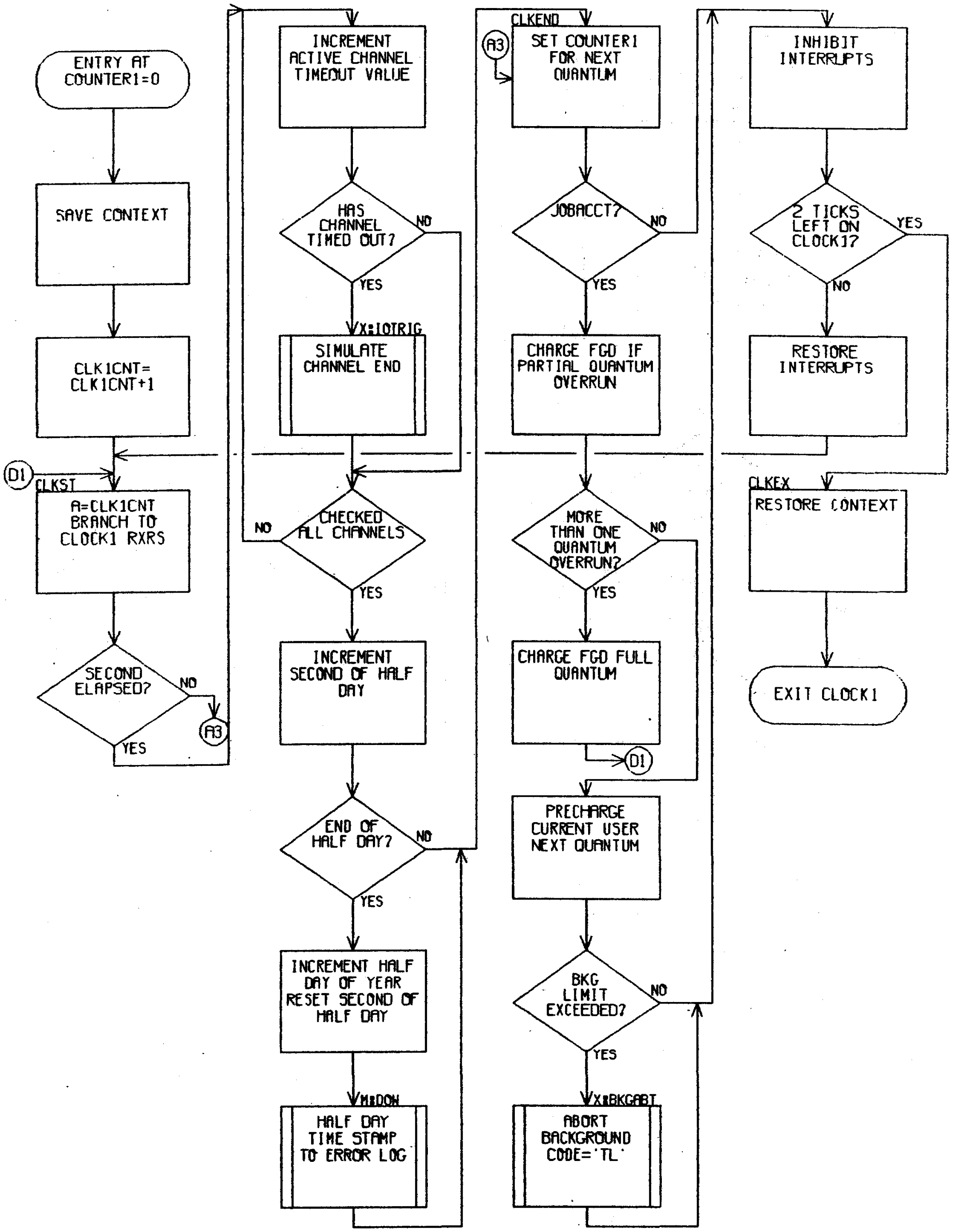


Figure 5. RBM Clock 1 Task Flow

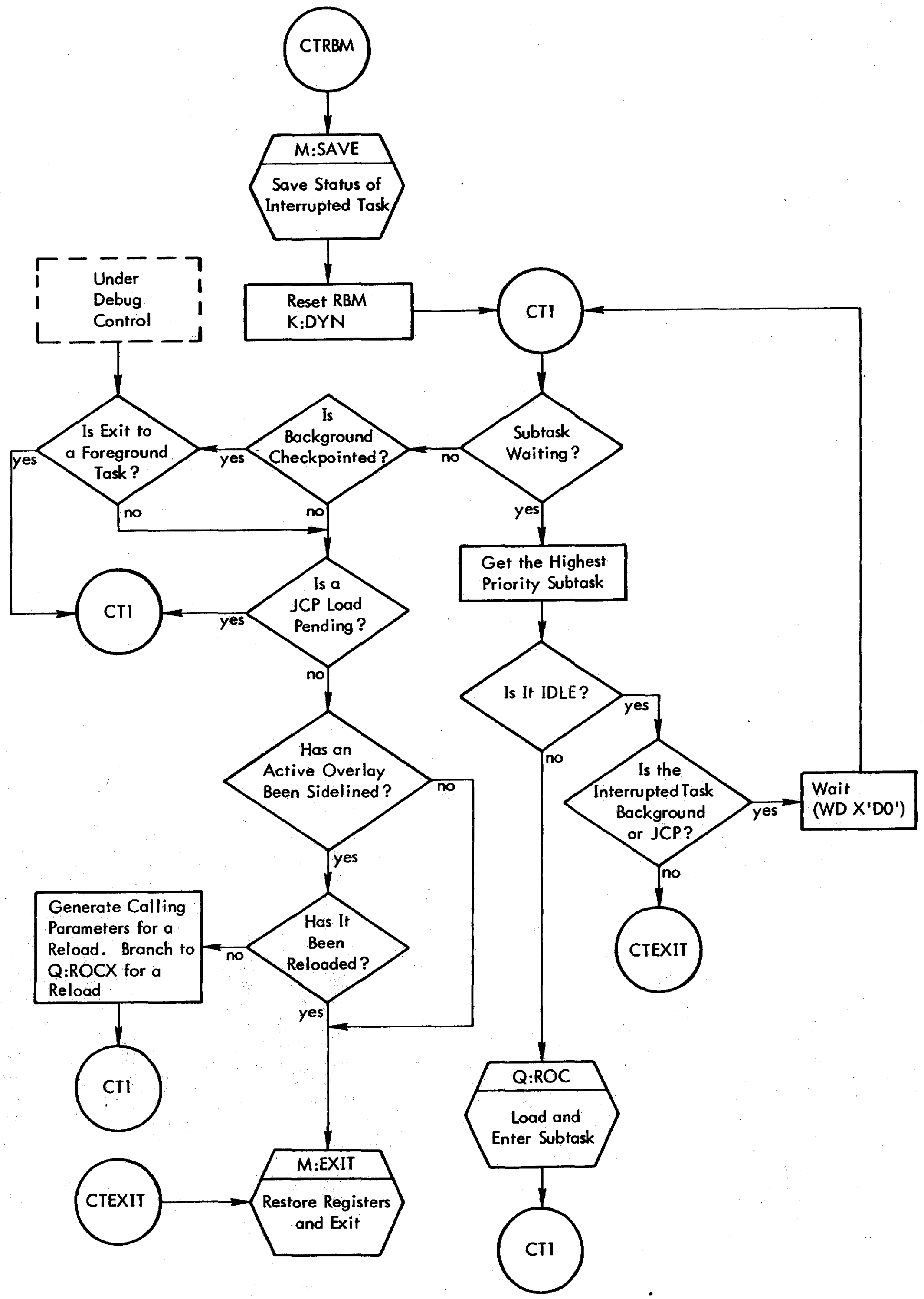

CTRBM
M:SAVE
Save Status of Interrupted Task
Under Debug Control
Reset RBM K:DYN
CT1
Is Exit to a Foreground Task?
yes
no
Is Background Checkpointed?
yes
no
Subtask Waiting?
no
yes
Get the Highest Priority Subtask
CT1
Is a JCP Load Pending?
yes
no
Is It IDLE?
yes
no
Has an Active Overlay Been Sidelined?
no
yes
Is the Interrupted Task Background or JCP?
yes
no
Wait (WD X'D0')
Generate Calling Parameters for a Reload. Branch to Q:ROCX for a Reload
no
Has It Been Reloaded?
yes
CTEXIT
CT1
Q:ROC
Load and Enter Subtask
CTEXIT
M:EXIT
Restore Registers and Exit
CT1

Figure 6. RBM Control Task Flow

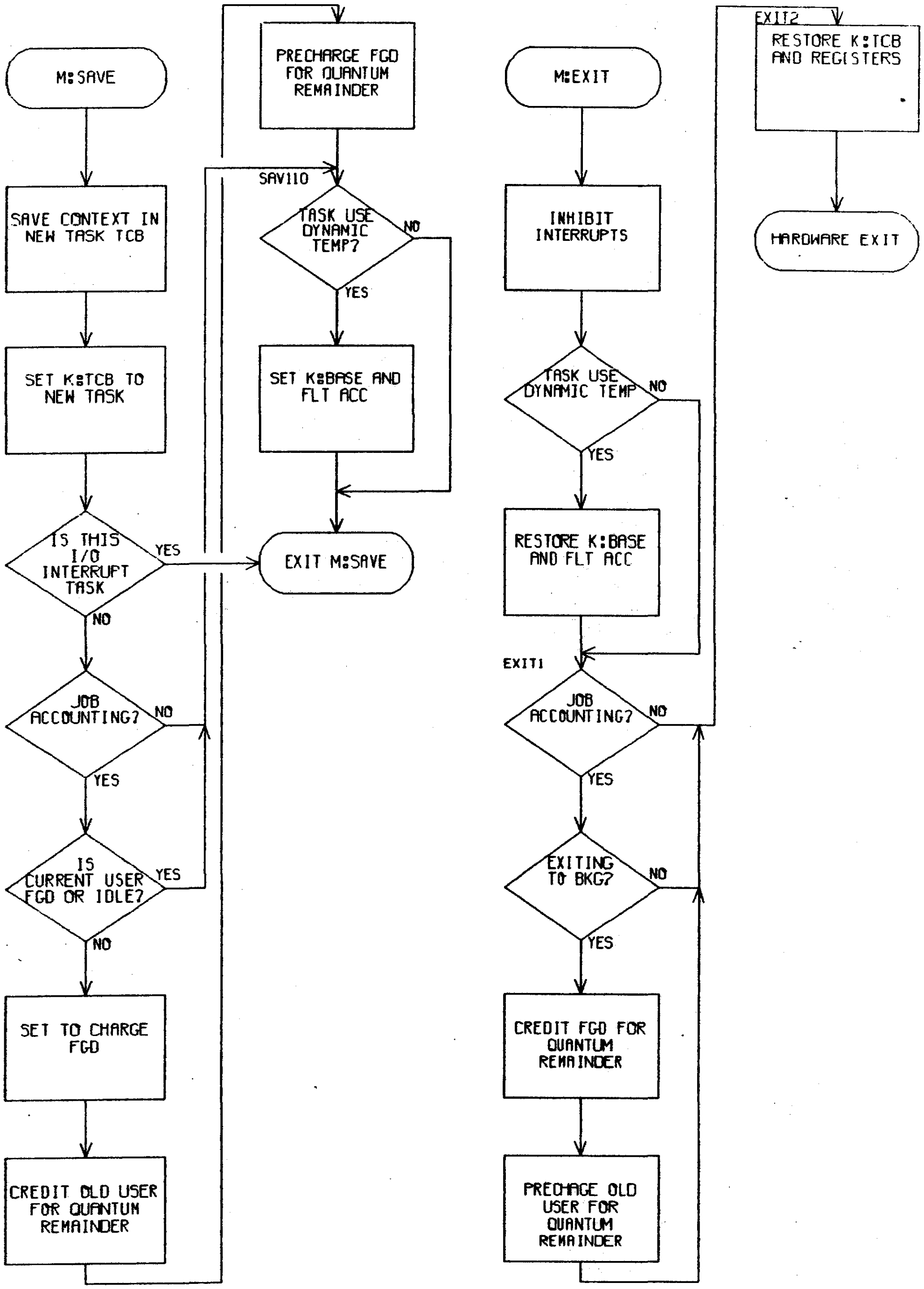

M:SAVE
SAVE CONTEXT IN NEW TASK TCB
SET K:TCB TO NEW TASK
IS THIS I/O INTERRUPT TASK
YES
NO
JOB ACCOUNTING?
NO
YES
IS CURRENT USER FGD OR IDLE?
YES
NO
SET TO CHARGE FGD
CREDIT OLD USER FOR QUANTUM REMAINDER
PRECHARGE FGD FOR QUANTUM REMAINDER
SAV110
TASK USE DYNAMIC TEMP?
NO
YES
SET K:BASE AND FLT ACC
EXIT M:SAVE
M:EXIT
INHIBIT INTERRUPTS
TASK USE DYNAMIC TEMP
NO
YES
RESTORE K:BASE AND FLT ACC
EXIT1
JOB ACCOUNTING?
NO
YES
EXITING TO BKG?
NO
YES
CREDIT FGD FOR QUANTUM REMAINDER
PRECHAGE OLD USER FOR QUANTUM REMAINDER
EXIT2
RESTORE K:TCB AND REGISTERS
HARDWARE EXIT

Figure 7. RBM M:SAVE/M:EXIT Task Flow

Access to the overlaid Monitor service routines is controlled by service routine Q:ROC. By using the temp stack and the overlay control flag K:SEGIN (see Figures 8 and 9), Q:ROC controls the loading and subsequent reloading (if required) of the Monitor overlay area (a 512-word reserve). Normally a call to an overlaid Monitor service routine results in a three-word call to Q:ROC to explicitly load and transfer control to the overlaid routine. The calling sequence to Q:ROC is as follows:

```
RCPYI    P,T        T points to ADRLST
B        Q:ROC
DATA     'IDNN'     ID  = segment ID
                    NN  = temp stack requirement
                          (NN ≥ 12)
```

		0	1 2 … 7	8 … 15
K:SEGIN		A	TASKID	Current Overlay ID
Temp Stack	0	A		Previous Overlay ID
	1	M:RES/M:POP (B)		
	2	Set to (L) by Q:ROC		
	3	X		
	4	A		
	5	E		
	6	Request (overlay ID)		
	7	X'8802'		
	8	Device File Number of 'RM' File		
	9	Overlay Area Address		
	10	Byte Size of Overlay		
	11	Sector Displacement of Overlay		
	11 + R	Additional Temp Space Required by the Overlay		

Notes:

1. Words 3 through 12+R are available for use by the overlay, where R is the number of cells required minus 12. If the result is negative, R is zero.

2. The symbols used in the figure are described below.

K:SEGIN is the overlay control flag and actually resides as the first word of the overlay region. It is composed of the following:

A indicates whether the overlay is active (A = 1) or not active (A = 0).

TASKID is the seven low order bits of the dedicated interrupt location of the task using the overlay.

X, A, and E contain, at exit, the status returned by the routine.

Figure 8. Q:ROC Use of Temp Stack and K:SEGIN

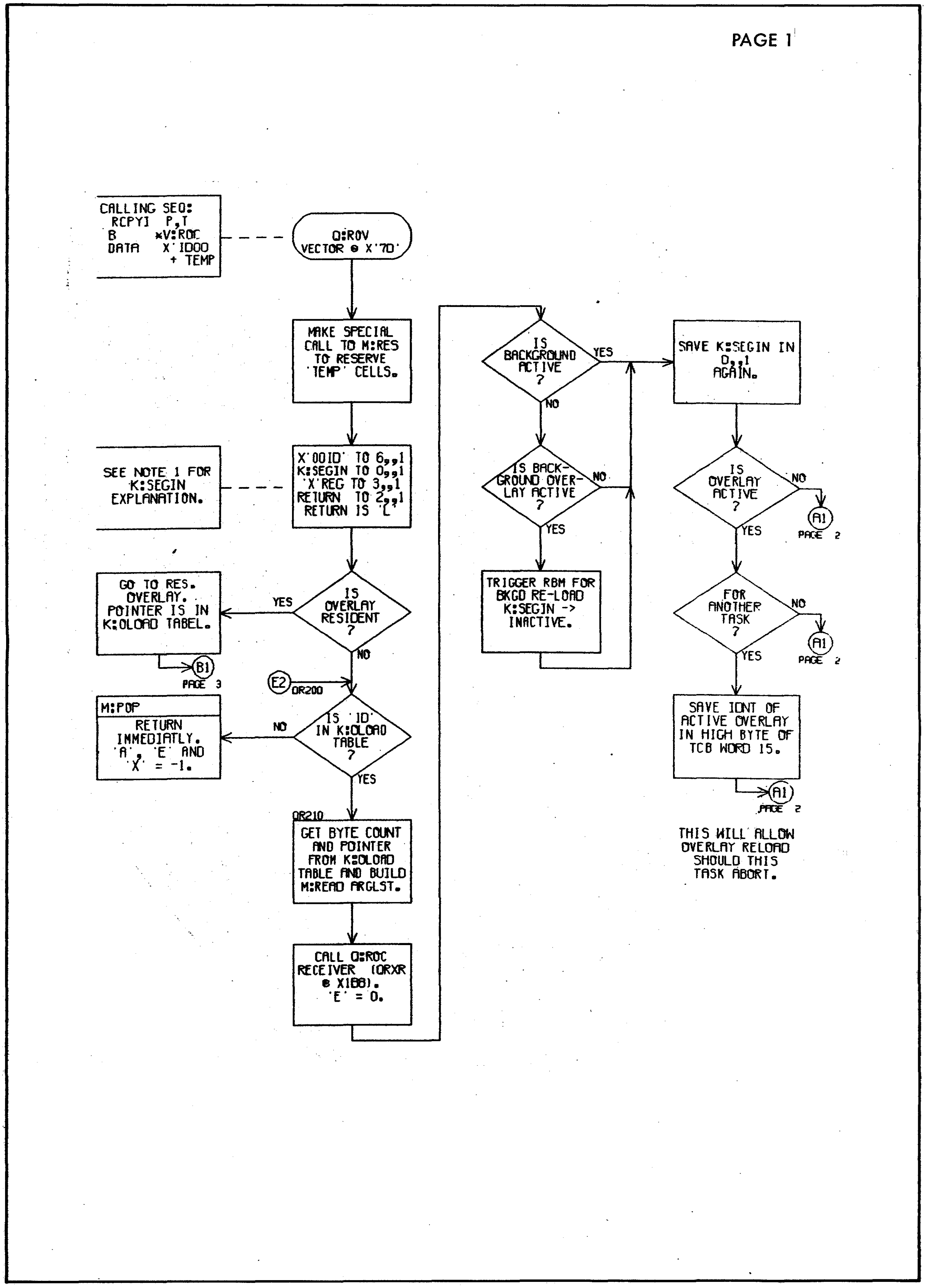

PAGE 1
CALLING SEQ:
RCPYI P,T
B *V:ROC
DATA X'IDOO + TEMP
Q:ROV
VECTOR @ X'70'
MAKE SPECIAL CALL TO M:RES TO RESERVE 'TEMP' CELLS.
SEE NOTE 1 FOR K:SEGIN EXPLANATION.
X'00ID' TO 6,,1
K:SEGIN TO 0,,1
'X'REG TO 3,,1
RETURN TO 2,,1
RETURN IS 'L'
IS OVERLAY RESIDENT ?
YES
NO
GO TO RES. OVERLAY. POINTER IS IN K:OLOAD TABEL.
B1
PAGE 3
E2
OR200
IS 'ID' IN K:OLOAD TABLE ?
NO
YES
M:POP
RETURN IMMEDIATLY. 'A', 'E' AND 'X' = -1.
OR210
GET BYTE COUNT AND POINTER FROM K:OLOAD TABLE AND BUILD M:READ ARGLST.
CALL Q:ROC RECEIVER (ORXR @ X1BB).
'E' = 0.
IS BACKGROUND ACTIVE ?
YES
NO
IS BACK-GROUND OVER-LAY ACTIVE ?
NO
YES
TRIGGER RBM FOR BKGD RE-LOAD K:SEGIN -> INACTIVE.
SAVE K:SEGIN IN 0,,1 AGAIN.
IS OVERLAY ACTIVE ?
NO
A1
PAGE 2
YES
FOR ANOTHER TASK ?
NO
A1
PAGE 2
YES
SAVE IDNT OF ACTIVE OVERLAY IN HIGH BYTE OF TCB WORD 15.
A1
PAGE 2
THIS WILL ALLOW OVERLAY RELOAD SHOULD THIS TASK ABORT.

Figure 9. Q:ROC Flow

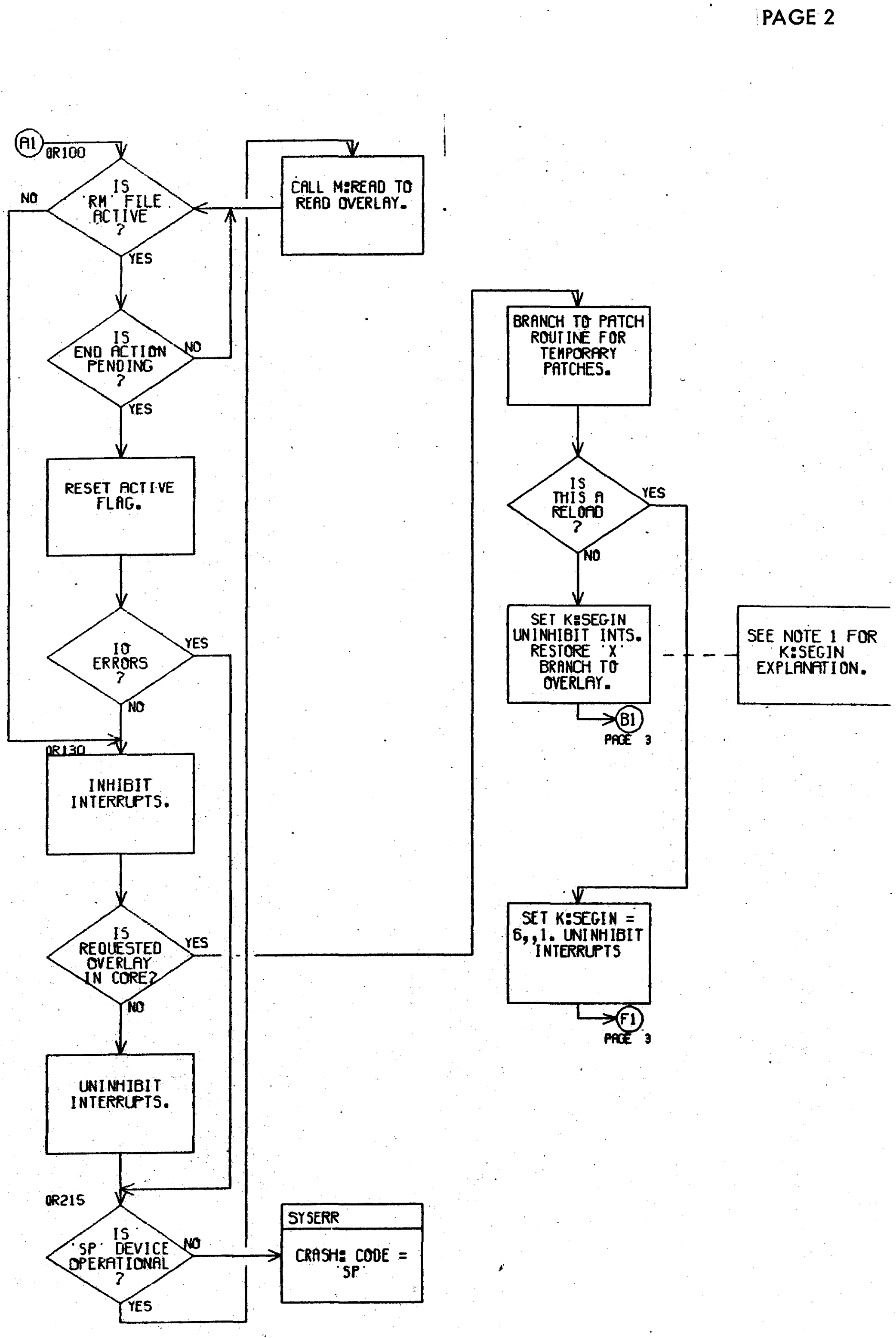
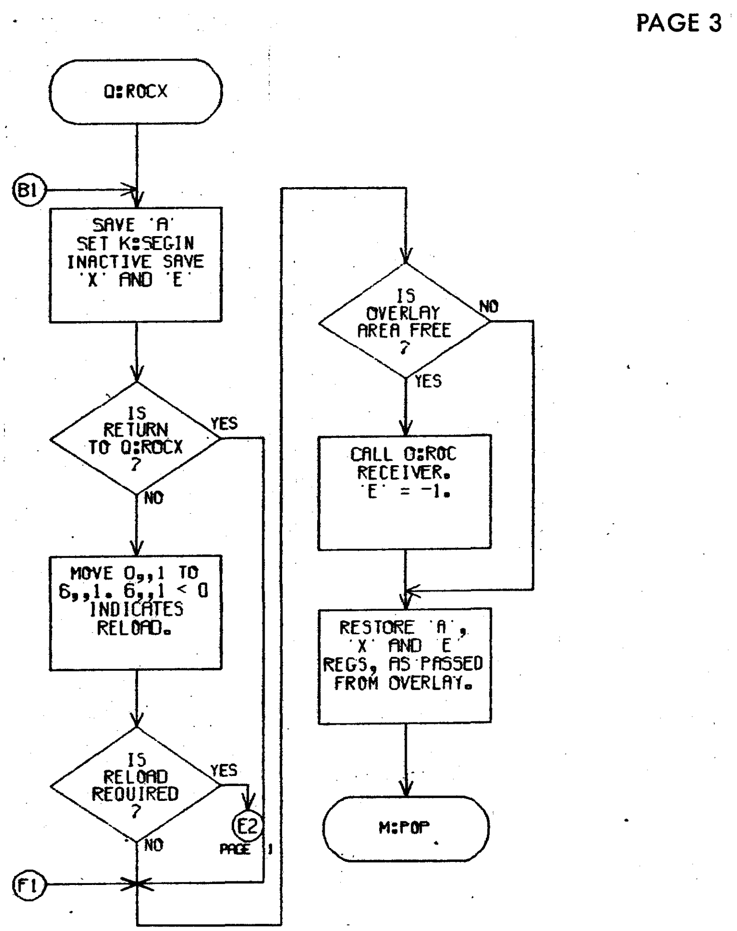

PAGE 2
A1
QR100
IS 'RM' FILE ACTIVE ?
NO
YES
CALL M:READ TO READ OVERLAY.
IS END ACTION PENDING ?
NO
YES
RESET ACTIVE FLAG.
IO ERRORS ?
YES
NO
QR130
INHIBIT INTERRUPTS.
IS REQUESTED OVERLAY IN CORE?
YES
NO
UNINHIBIT INTERRUPTS.
QR215
IS 'SP' DEVICE OPERATIONAL ?
NO
YES
SYSERR
CRASH: CODE = 'SP'
BRANCH TO PATCH ROUTINE FOR TEMPORARY PATCHES.
IS THIS A RELOAD ?
YES
NO
SET K:SEGIN UNINHIBIT INTS. RESTORE 'X' BRANCH TO OVERLAY.
SEE NOTE 1 FOR K:SEGIN EXPLANATION.
B1
PAGE 3
SET K:SEGIN = 6,,1. UNINHIBIT INTERRUPTS
F1
PAGE 3

Figure 9. Q:ROC Flow (cont.)



Note: Q:ROC first reserves NN temporary cells via the monitor service routine M:RES and then saves the current status of the overlay area (indicated by K:SEGIN) into word 0 and the return address into word 2 of the temp stack just reserved. (Q:ROC steps into M:RES with the number of required temp cells in the A register and with the T register less than X'100'. M:RES recognizes a call from Q:ROC by the contents of the Tregister and after reserving the requested number of cells and saving the original contents of the A register, returns directly to Q:ROC.) Q:ROC then sets K:SEGIN to reflect the status of the overlay area. If the overlay is not already resident, Q:ROC loads the requested overlay. At completion of the load, K:SEGIN is set to ID "active", the X register is restored, and control is transferred to the second word of the overlay area. The routine will perform its prescribed function and return to the location indicated by the monitor pointer Q:ROCX.

K:SEGIN, pointed to by location X'7F' is the first cell of the overlay area and contains the following information.

Bit 0	if on, the overlay area is active; if off, the overlay area passively contains the last overlay executed.
Bits 1-7	contain the low order 7 bits of the dedicated interrupt location associated with the task currently (or, most recently) using the overlay area. These 7 bits are zero for Background, JCP or PMD.
Bits 8-15	contain the overlay IDNT.

Each overlay must be assembled with the first cell containing X'FF00' + IDNT.

Figure 9. Q:ROC Flow (cont.)

This address is also contained in the L register at entry to the routine. Q:ROC will initiate a reload operation if the previous overlay status was "active". A call is now made to the Monitor service routine M:POP to release the temporary space and to return to the calling program with the status returned by the overlaid routines in the X, A, and E registers.

A special entry point, Q:ROCC, is used when one overlay calls another. This entry requires that the IDENT of the requested overlay be in 6,, 1.

The minimum temp stack requirement for any overlay is 12 + R. The maximum temp stack requirement is a function of the nested calls to both resident and overlaid Monitor service routines.

The maximum nested temp stack requirement occurs when a call is made to M:SEGLD, data switch #0 is set, and RBM and its overlays are assembled with #TEST = YES.

		Reserved	Total
1.	User calls M:SEGLD	10	10
2.	M:SEGLD calls M:READ to read segment	19	29
3.	M:READ returns to M:SEGLD	-19	10
4.	M:SEGLD calls M:WAIT which transfers to Q:ROC	15	25
5.	Q:ROC calls M:READ to load overlay	19	44
6.	M:READ returns to Q:ROC, which branches to M:WAIT	-19	25
7.	M:WAIT calls M:WRITE to write "BEGIN SEG XX"	19	44
8.	M:WRITE calls Keyboard Edit routine which transfers to Q:ROC	13	57
9.	Q:ROC calls M:READ to load overlay	19	76
10.	M:READ returns to Q:ROC which branches to the Keyboard Edit routine which, in turn, exits, and the temp stack is eventually unwound. The entire process is diagrammed in Figure 10.		

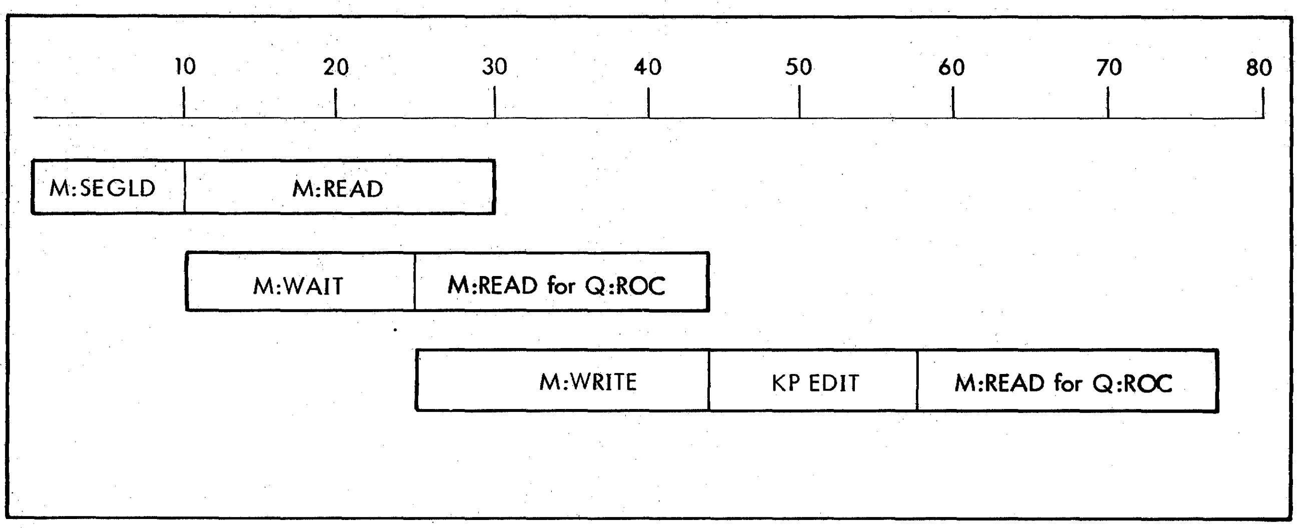


Figure 10. Temp Stack Usage

RBM Initialization and Selection

There are two phases in establishing an RBM tailored to a particular installation:

1. The selection phase, SYSGEN (shown in Figure 11), is performed at least once for each system. SYSGEN selects the options and the peripheral devices to be used by the system and allocates areas on the RAD.

2. The initialization phase, SYSLOAD, loads the RBM overlays onto the RAD and then writes RBM and the RBM symbol table onto the RAD. SYSLOAD also stores information in the RBM RAD bootstrap that enables the bootstrap to load RBM from the RAD. SYSLOAD then exits to the bootstrap.

There are several characteristics common to both phases. Both are nonresident (as shown in Figure 12); that is, when each has performed its function, it can be overwritten by foreground or background programs. Also, both operate as protected programs, and hence the memory protection must be off during these phases because the protection registers are not set up until the RAD bootstrap executes.

There are several functions performed in the selection phase:

1. All I/O tables are set to the installation dependent values.

2. All tables are compacted in low core (either in unused hardware interrupt space or just above resident RBM instructions) to conserve space. Thus, pointers to all I/O tables are dynamically assigned and initialized in the zero table as part of the selection operation.

3. The selection of mandatory and optional resident tasks and routines and nonresident routines is determined by symbols placed in the SYSGEN Symbol Table and in the RBM Overlay Table. The format of the SYSGEN Symbol Table is based on the BCM Linking Loader symbol table.

   The maximum allowable size of an entry is six words (up to eight characters) per symbol. The table is not ordered by the loader during the loading process. Entries are inserted as encountered. The entry size includes the control word.

   word 0

Def	Ref	Dec	0	0	No. of words	Module declaration No.
0	1	2	3	4	5 6 7	8 9 10 11 12 13 14 15

   word 1

For a DEF: Effective address For a REF: Effective address of 1st link in REF chain
0 15

   word 2

1st character of DEF or REF	2nd character
0 7	8 15

   ⋮

   word 5

7th character	8th character
0 7	8 15

   In this item, word 0 is the control word

   where

   bit 0 is set only if the entry is a definition value. During loading, definitions are declared at the beginning of the module but are defined later in the module. The entry is made in the symbol table when the REF is encountered or the DEF is declared. Bit 2 = 1 indicates that the entry is a definition declaration (the symbol has not been defined yet). Bit 0 = 1 indicates that a definition address or value has been inserted in word 1 of the table.

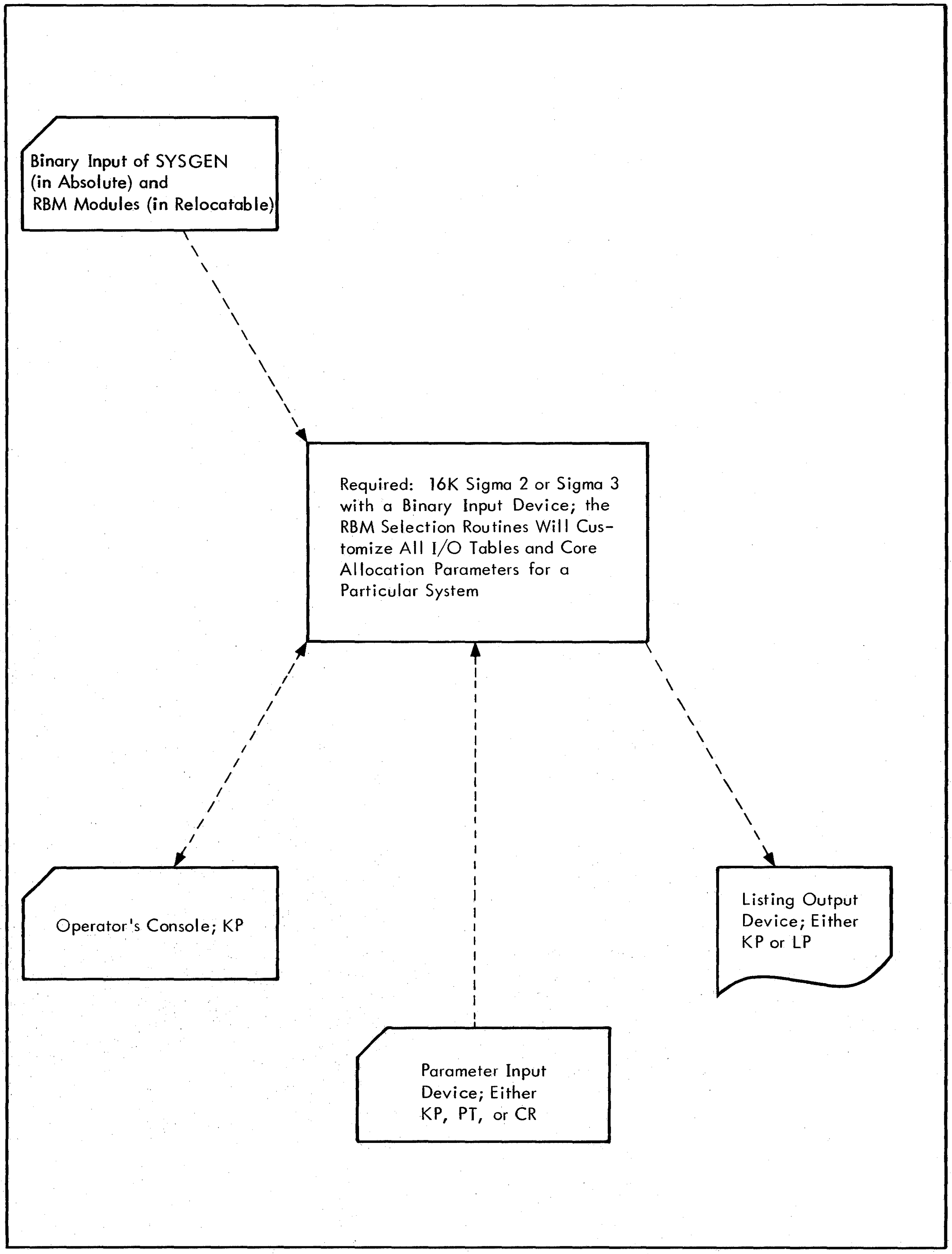

Binary Input of SYSGEN (in Absolute) and RBM Modules (in Relocatable)
Required: 16K Sigma 2 or Sigma 3 with a Binary Input Device; the RBM Selection Routines Will Customize All I/O Tables and Core Allocation Parameters for a Particular System
Operator's Console; KP
Parameter Input Device; Either KP, PT, or CR
Listing Output Device; Either KP or LP

Figure 11. RBM Selection Operation

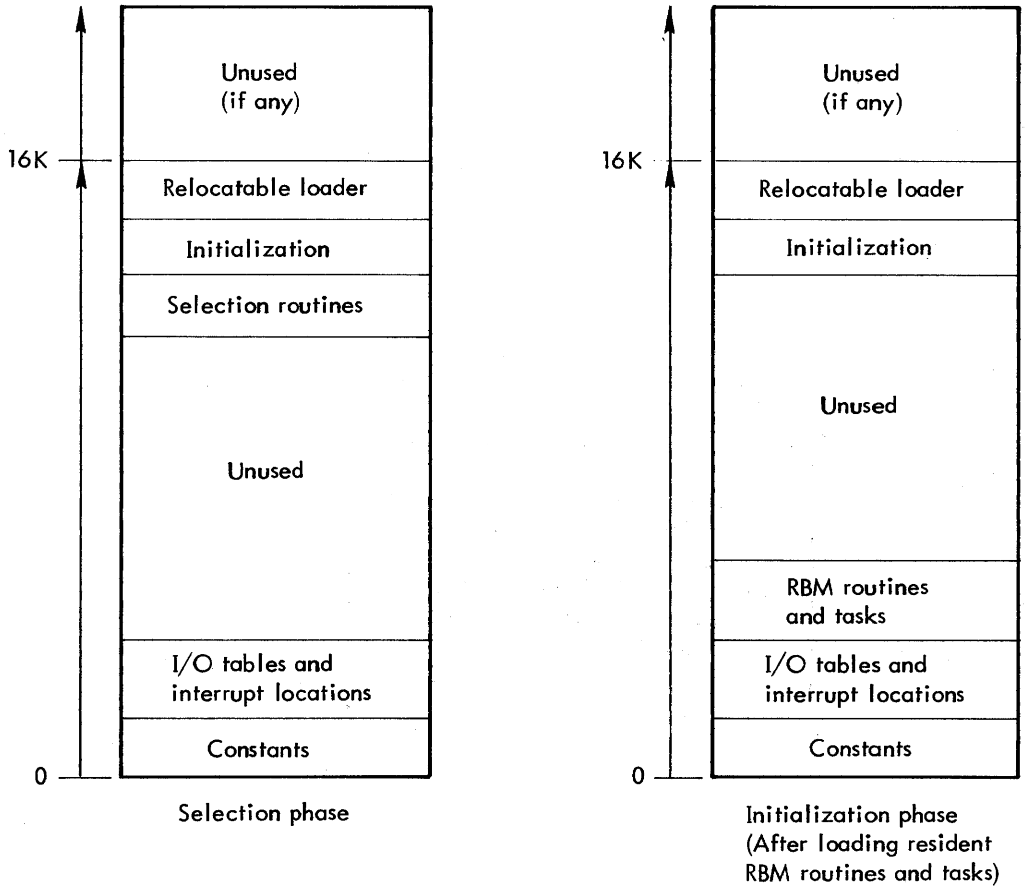


Figure 12. Core Memory Allocation

bit 1 is set if the reference name is encountered prior to encountering a definition value. When bit 0 is set, bit 1 is reset. However, bits 1 and 2 may both be set at the same time if a reference and definition declaration are encountered before the definition value.

bit 2 is set when a declaration is encountered. It is used for flagging (on the map) definitions that are declared but not defined.

bits 5–7 indicate the length of an entry in words (the length can be 3 to 6 words). Trailing blanks in a symbol are suppressed.

bits 8–15 are the declaration number of the entry. As a start item is encountered, the Loader assigns to the module a declaration number between 1 and X'FF' (this includes library programs). The number is used to locate the source of definitions for mapping.

Word 1 is the effective address of the item. If the entry is a definition address the effective address or value of the definition is contained in word 1. If the entry is a reference, the effective address of the first link in the threaded reference list is contained.

Words 2 to 5 contain the EBCDIC representation of the SREF, REF, or DEF. If the symbol entry contains less than eight characters, trailing blank words are suppressed.

The following procedures are available in the SYSGEN source program to simplify management of the symbol and overlay tables.

Procedure — Function

REFSYM — Creates a symbol table entry for an optionally-included module DEF.

Calling sequence: label REFSYM 'c_1c_2', 'c_3c_4', . . .

where c_i are up to eight characters.

STDSYM — Creates a symbol table entry for a required module DEF. These symbol table entries must appear between the labels PST and SYSLOAD.

Calling sequence: STDSYM 'c_1c_2', 'c_3c_4', . . .

where c_i are up to eight characters.

ADDREF — Adds a symbol to the SYSGEN REF stack.

Calling sequence: [label] ADDREF symbol

where 'symbol' is the label of the REFSYM procedure.

If the symbol already exists, the new pointer will be linked to the previous one, in which case carry will be set on return; otherwise, carry is reset on return.

ADDOV — Adds an ident to the OV:LOAD table.

Calling sequence: [label] ADDOV 'id'

where id is a 2-character EBCDIC constant corresponding to the ident of the overlay to be added.

If the overlay ID already exists in the table, the routine will exit without adding it again, in which case carry will be set on return; otherwise, carry will be reset.

4. The resident tasks and routines (optional and mandatory) for monitor services and I/O handling are loaded from the same device used to load SYSGEN. Each routine is assembled as a relocatable module that contains a DEF statement to externally define the module. The module may also contain any (or all) of the following characteristics:

- REF statements for linkage to other modules.
- DEF statements for reference by other modules.
- Initialization code to set pointers in zero table, I/O tables, etc., or to eliminate unnecessary configuration-dependent code. The cell IN:UB1 is set to the initialization start, which may be altered by the initialization routine. Loading of the next module begins at the location in IN:UB1 on return.

Initially, if there is no previous data on the RAD to be saved, the initialization phase writes zeros into the first two sectors of all SYSGEN-defined areas of the RAD. Then the initialization phase loads the RBM overlays from the SYSGEN boot device, writes them on the RAD, and constructs the RBM OV:LOAD table that is used by Q:ROC when the overlays are loaded for execution. Next, RBM and the transfer vector (TVECT) table are written on the RAD. The IOCDs necessary to read RBM from the RAD are calculated and stored in the RAD bootstrap, and the bootstrap is written on the RAD. Finally, the RBM symbol table (used by the Overlay Loader to satisfy external references to Monitor service routines) is written, and the initialization phase reads in the RAD bootstrap and transfers control to it.

Job Control Processor

The Job Control Processor for RBM is herein defined as the routines required to control the operation of a background processor, which includes loading, initializing, and checkpointing and subsequent restarting of a background job. For the purpose of illustration, the Job Control Processor has been divided into three main parts: the RBM Control Task, the RBM subtasks, and the Control Command Interpreter. Each part is described separately, but each part interacts with all of the other parts.

RBM Control Task

The RBM Control Task operates at the lowest hardware interrupt level. When triggered to operate one of the RBM subtasks, the RBM Control Task will scan the RBM Control Task status word (R:RBM) for the highest priority subtask currently requested (see Figure 13).

If the subtask is not already in core, it will be read into the RBM overlay area, and control will be transferred to the subtask at the RBM Control Task level. When its operation is finished, the subtask will clear the respective flag in the RBM Control Task status word and return to the RBM Control Task. This process is repeated until all requests for subtasks have been satisfied.

RBM Subtasks

Currently there are 13 RBM subtasks, each of which is described below in order of priority:

1. S:PARPWR outputs power-on and machine fault alarms (see Figure 14).

2. S:CKPT controls the checkpointing of a background processor.

3. S:REST controls the restarting of a background processor that has been checkpointed.

4. S:LOAD loads root segments and controls the initialization of background processors or foreground program.

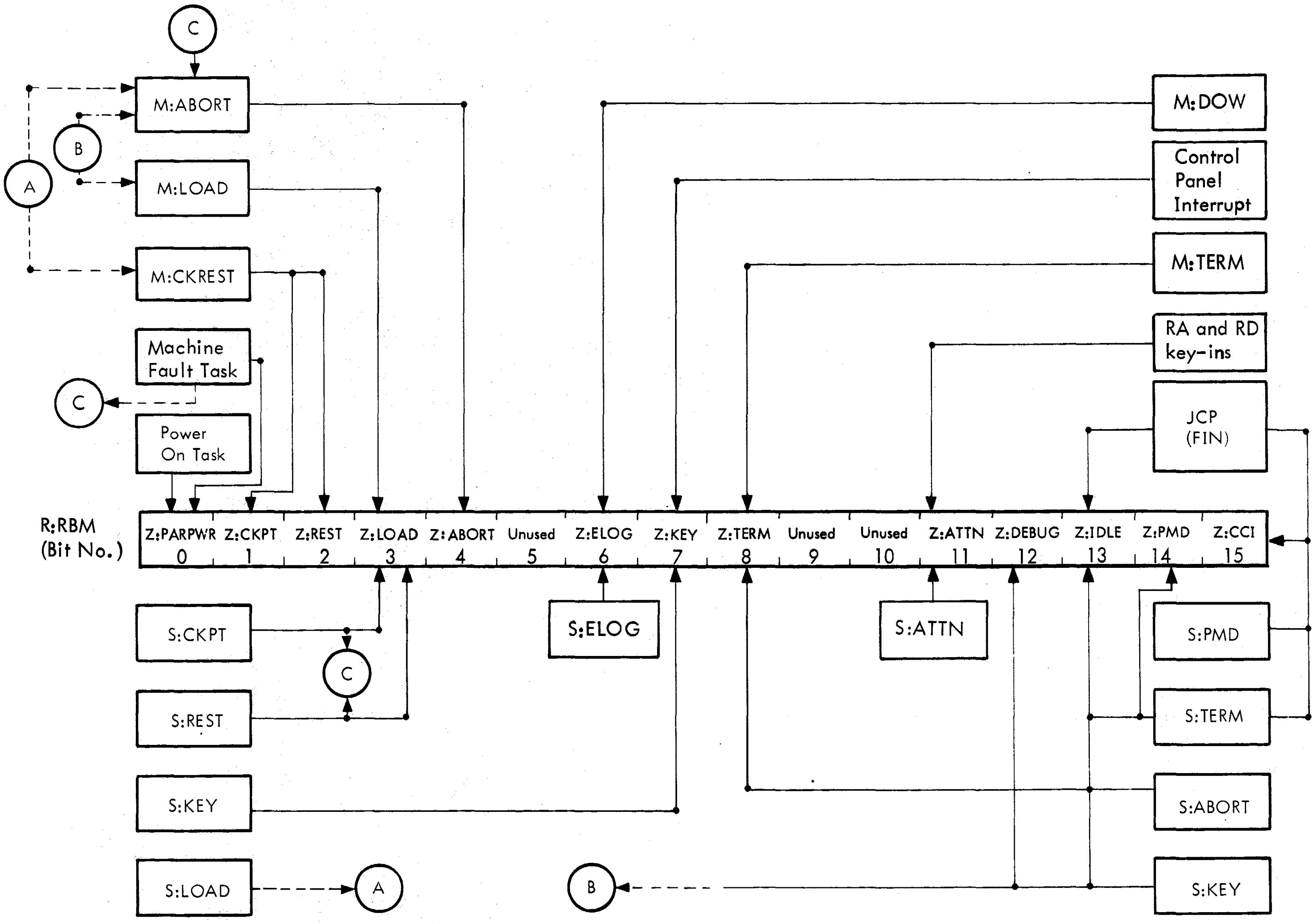
5. S:ABORT controls the aborting of a background processor and also outputs foreground abort messages without affecting background. S:ABORT examines the Job Control Processor status word (R:JCP; see Figure 15) to determine whether or not additional subtasks are required (e.g., S:PMD, S:IDLE).

6. S:ELOG writes error-log entries from memory to the ERRLOG disk file, and tests the file for imminent overflow condition.

7. S:KEY responds to all unsolicited key-ins and is called as a result of a control panel interrupt.

8. S:TERM terminates a background processor after all background input/output is finished.

9. S:ATTN transmits Turn Data Terminal Ready On order code to the keyboard/printer device with AIS code of 2 in FCT2, and sets AIS to 3. If function is being performed for keyboard/printer other than DFN1, foreground attention receiver (ATTNRXR) is entered with the X register containing the keyboard/printer DFN.



Notes: 1. Solid lines show direct access to RBM, and broken lines shown indirect access via subsequent calls.

2. The priority of subtasks is from left to right.

Figure 13. RBM Control Task Status Word (R:RBM)

Power-On Task for Sigma 3 Computers with External Interrupts and Sigma 2 Computers.

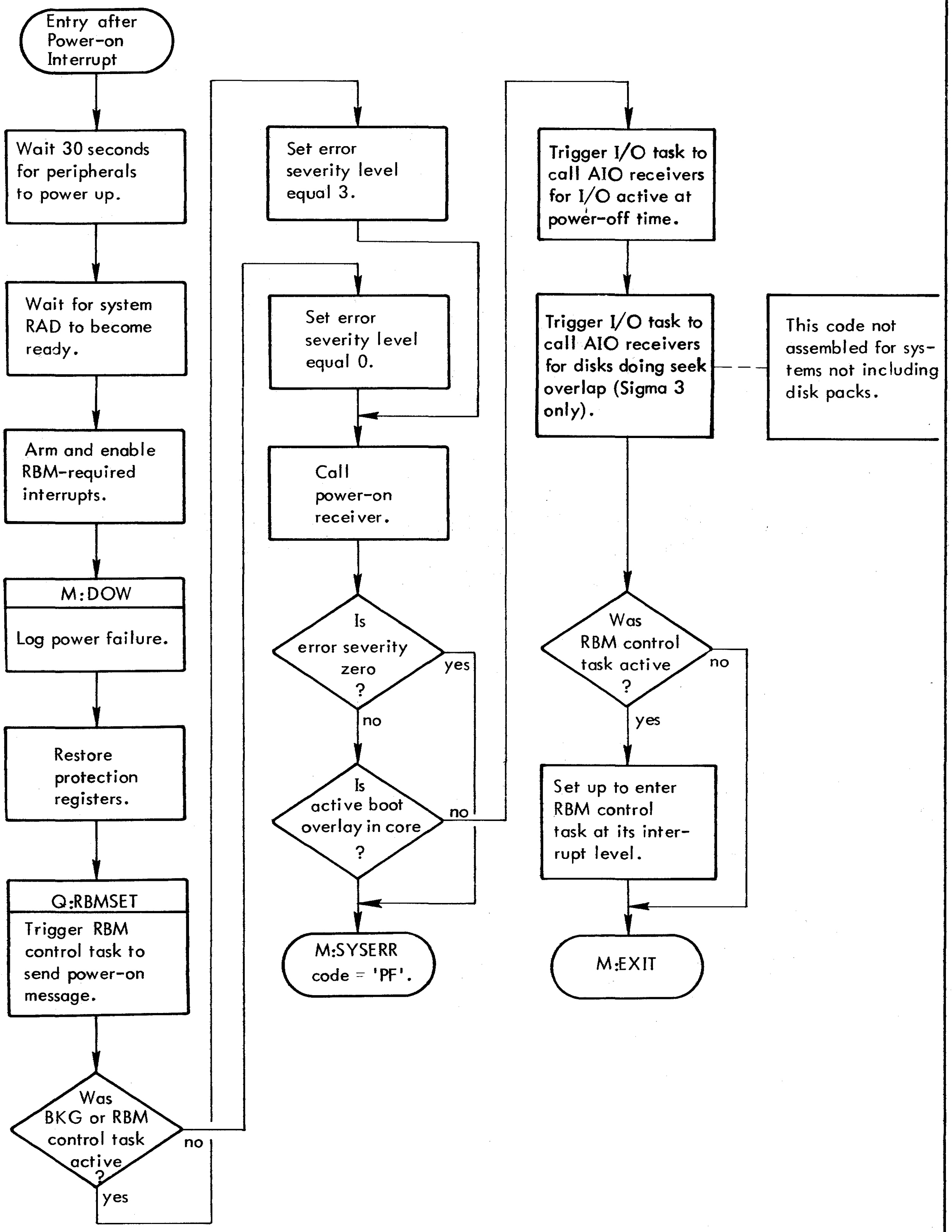

Entry after Power-on Interrupt
Wait 30 seconds for peripherals to power up.
Wait for system RAD to become ready.
Arm and enable RBM-required interrupts.
M:DOW
Log power failure.
Restore protection registers.
Q:RBMSET
Trigger RBM control task to send power-on message.
Was BKG or RBM control task active ?
no
yes
Set error severity level equal 3.
Set error severity level equal 0.
Call power-on receiver.
Is error severity zero ?
yes
no
Is active boot overlay in core ?
no
M:SYSERR code = 'PF'.
Trigger I/O task to call AIO receivers for I/O active at power-off time.
Trigger I/O task to call AIO receivers for disks doing seek overlap (Sigma 3 only).
This code not assembled for systems not including disk packs.
Was RBM control task active ?
no
yes
Set up to enter RBM control task at its interrupt level.
M:EXIT

Figure 14. Power-On/Power-Off Tasks

Power-On Task for Sigma 3 Computers with no External Interrupts and Xerox 530 Computers.

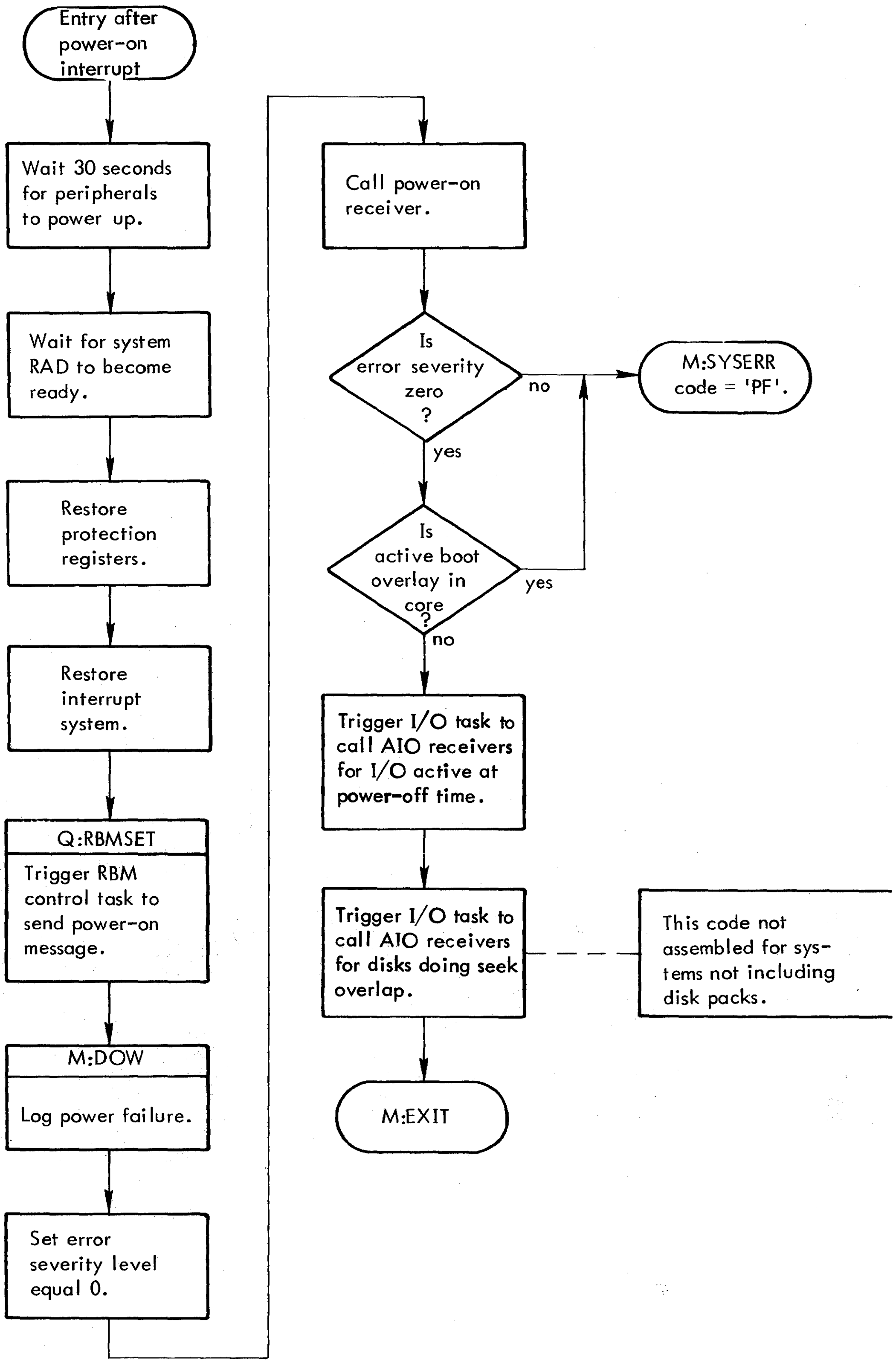
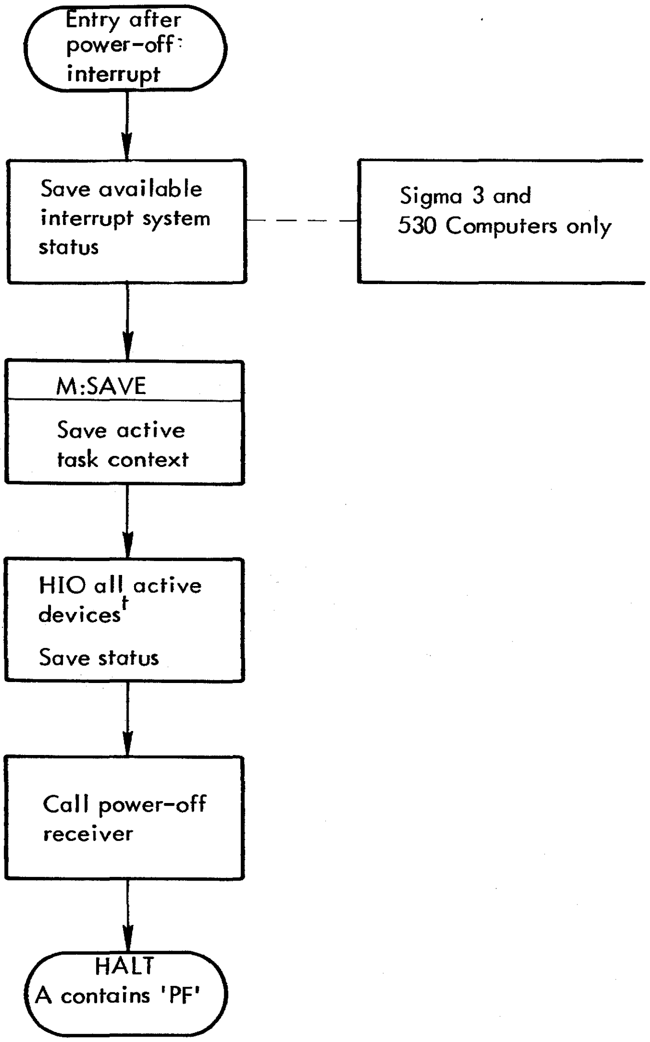

Entry after power-on interrupt
Wait 30 seconds for peripherals to power up.
Wait for system RAD to become ready.
Restore protection registers.
Restore interrupt system.
Q:RBMSET
Trigger RBM control task to send power-on message.
M:DOW
Log power failure.
Set error severity level equal 0.
Call power-on receiver.
Is error severity zero ?
no
yes
M:SYSERR code = 'PF'.
Is active boot overlay in core ?
yes
no
Trigger I/O task to call AIO receivers for I/O active at power-off time.
Trigger I/O task to call AIO receivers for disks doing seek overlap.
This code not assembled for systems not including disk packs.
M:EXIT

Figure 14. Power-On/Power-Off Tasks (cont.)

Power-Off Task for Sigma 2/3 and Xerox 530 Computers


Entry after power-off interrupt
Save available interrupt system status
Sigma 3 and 530 Computers only
M:SAVE
Save active task context
HIO all active devices[t]
Save status
Call power-off receiver
HALT
A contains 'PF'

[t]An assembly switch can be reset which will cause the removal, at assembly time, of the code which HIO's disk packs doing overlapped seek operations for systems not configured with these devices.

Figure 14. Power On/Power Off Tasks (cont.)

R:JCP	Z:JCKPT	CKPT Type	Z:JBKACT	Z:JNRACT	Z:JPMACT	Z:JPMREQ	Z:JATEND	Z:JSKIP	Z:JTEMP	Z:JIDLE	Z:JCCACT	Unused	Z:JSAVCC	Z:JERFIL
Bit No.	0	1 2	3	4	5	6	7	8	9	10	11	12 13	14	15

Z:JCKPT	0 = no checkpoint. 1 = background is checkpointed.
CKPT Type	00 = resident or nonresident foreground. 01 = S:LOAD checkpoint.
Z:JBKACT	0 = not active (RAD not required for checkpoint). 1 = active.
Z:JNRACT	0 = nonresident foreground not active. 1 = nonresident active (prevents any new loads).
Z:JPMACT	0 = postmortem dump not active. 1 = active.
Z:JPMREQ	0 = postmortem dump not requested. 1 = requested.
Z:JATEND	0 = background job not attended. 1 = attended.
Z:JSKIP	0 = not in skip mode. 1 = skip until JOB or FIN.
Z:JTEMP	0 = clear background temp files at end of activity. 1 = retain background temp files until next job.
Z:JIDLE	0 = not in idle. 1 = idle mode.
Z:JCCACT	0 = JCP not active. 1 = JCP active.
Z:JSAVCC	0 = reset 'CC' at !JOB 1 = SAVE 'CC' for 1 !JOB.
Z:JERFIL	0 = No-op. 1 = out error file overflow warning.

Figure 15. JCP Status Word (R:JCP)

10. S:DEBUG transfers control to the Debug executive.

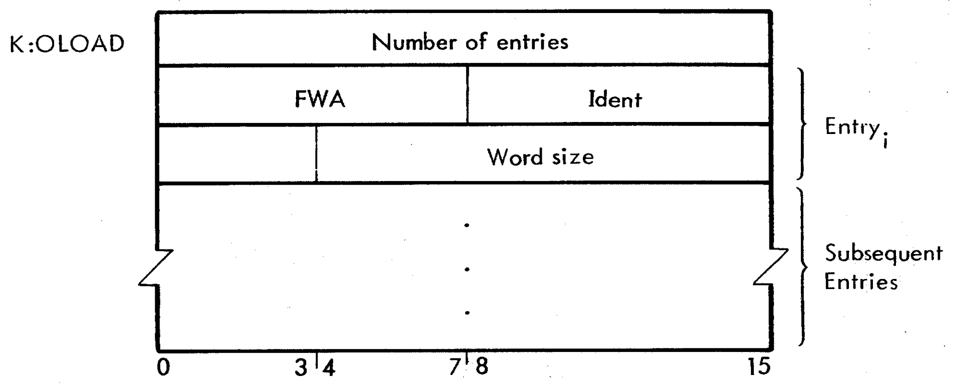
11. S:IDLE is a resident routine that causes the background to go into an idle state until the operator supplies a key-in of S to resume operation.

12. S:PMD controls the operation of the postmortem dump by loading and transferring control to the postmortem dump overlay.

13. S:CCI initiates the loading of the control command interpreter and extends memory protection to the background.

RBM Overlay Table

The format of the RBM overlay table is shown below:



where FWA is

1. The relative sector number of the start of this overlay, or

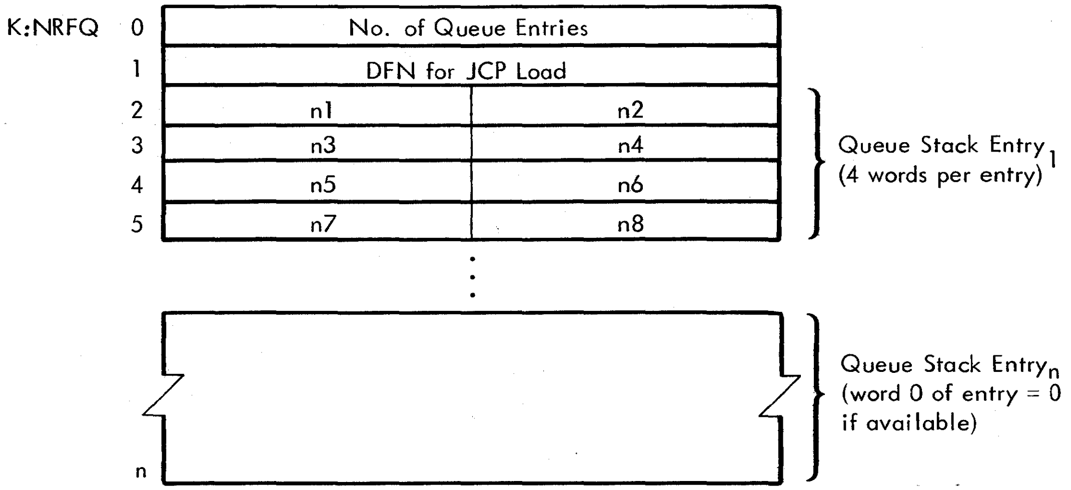
2. A flag (X'FF') which indicates that this overlay permanently resides in resident core.

If FWA is a relative sector number, Q:ROC, which makes extensive use of the K:OLOAD table, will add FWA to the contents of absolute core location X'1B5' (ROVBIAS) and use the result as the disk record displacement for the IO operation to read in the overlay specified by Ident. The byte count for this operation is Word Size times two.

If, on the other hand, FWA is X'FF', Word Size is actually a 16-bit value and indicates the core-resident starting address of the overlay.

Nonresident Foreground Queue Stack

The format of the nonresident foreground queue stack is shown below.



K:NRFQ	0	No. of Queue Entries		
	1	DFN for JCP Load		
	2	n1	n2	Queue Stack Entry$_1$ (4 words per entry)
	3	n3	n4	
	4	n5	n6	
	5	n7	n8	
	⋮			
	n			Queue Stack Entry$_n$ (word 0 of entry = 0 if available)

where n1 through n8 represent the program name.

The size of the nonresident foreground queue stack is 4n + 3, where n is the number of queue entries allocated at SYSGEN (n is always greater than 0). A request is inserted into the queue stack by storing the program name (as defined in the User Processor Dictionary) into the first available entry. As each entry is loaded, the queue stack entries are pushed up and the last entry is set to "empty" by zeroing out the first word of that entry. Word 1 is a special entry for the Job Control Processor; whenever a load is requested by the Job Control Processor, the device-file number associated with the load is stored into word 1 of the queue stack.

Control Command Interpreter

The Control Command Interpreter (CCI) is a routine that operates in the background area to read and process control commands (see Figure 16). It operates at the background priority level, with memory protection extended to the background. When CCI is active, the task control block pointer K:TCB will be set to K:BACKP + 1. This process identifies CCI as a foreground task.

The Control Command Interpreter will process all the Monitor control commands (see the Sigma 2/3 RBM Reference Manual for a description of the control commands); will copy relocatable binary decks onto the GO file; and, via processor control commands, will cause the loading of system or user processors.

See Figure 17 for an illustration of loading processors from JCP, and Figure 18 for queue stack loading of foreground processors.

Background Termination Procedures

There are two ways for a background job to terminate: (1) normal termination, with a call to M:TERM, and (2) abnormal termination, with a call to M:ABORT.

On termination, the RBM subtask S:TERM allows all input/output for the background to run to completion. If an SIO has been issued on a background device and if that device is in manual when the termination is attempted, the input/output is aborted by an HIO operation and the input/output status tables are cleared. This procedure prevents an uncompleted or incorrect operation in one job from affecting the following job. A postmortem dump will be performed if appropriate.

A postmortem dump is initiated by a PMD command following the JOB command. The Job Control Processor builds the postmortem dump table from the parameters on the PMD card (see Figure 19). Whenever background terminates, this table is used to determine whether or not a dump must be performed and what areas of memory are to be dumped.

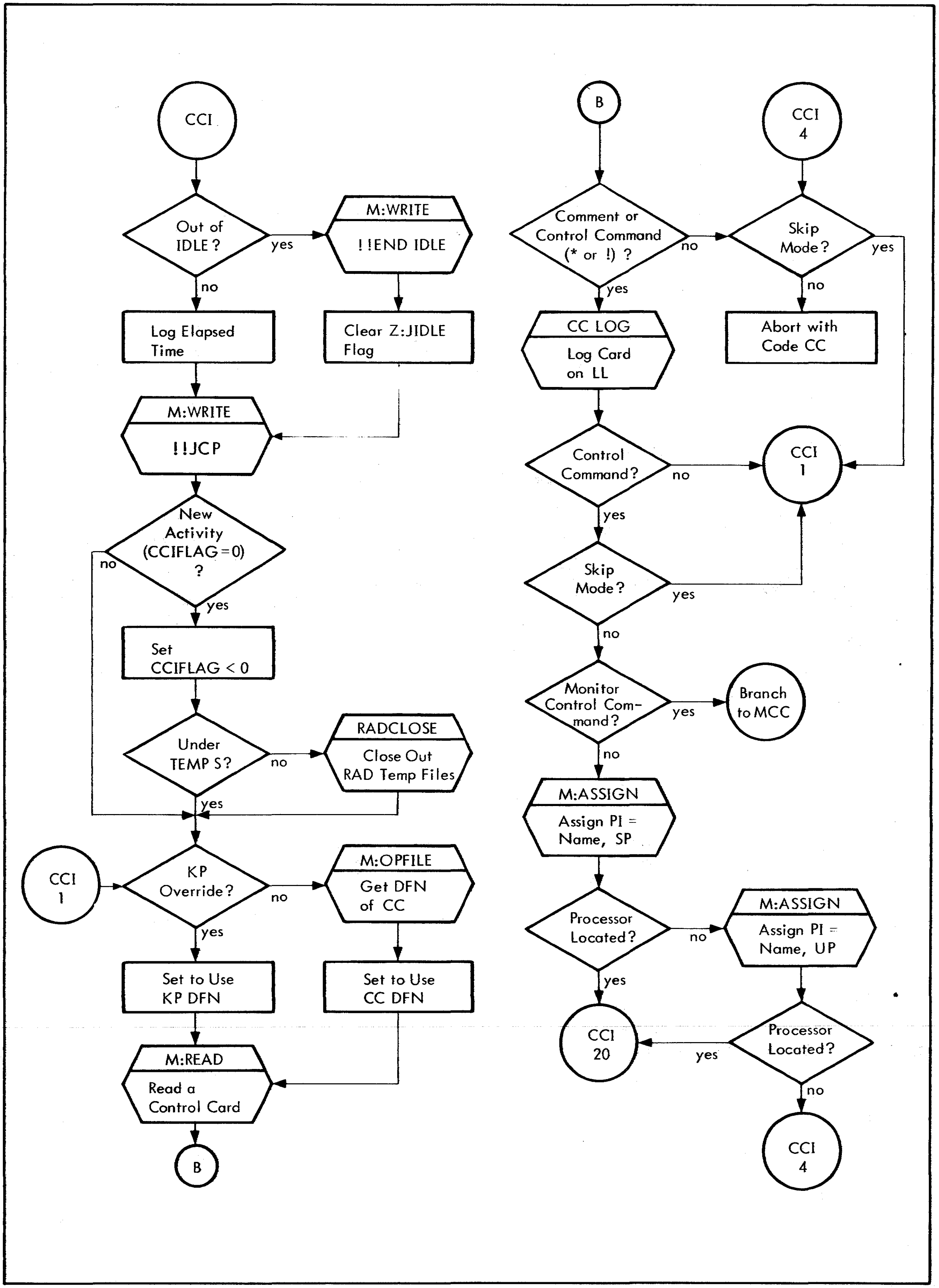
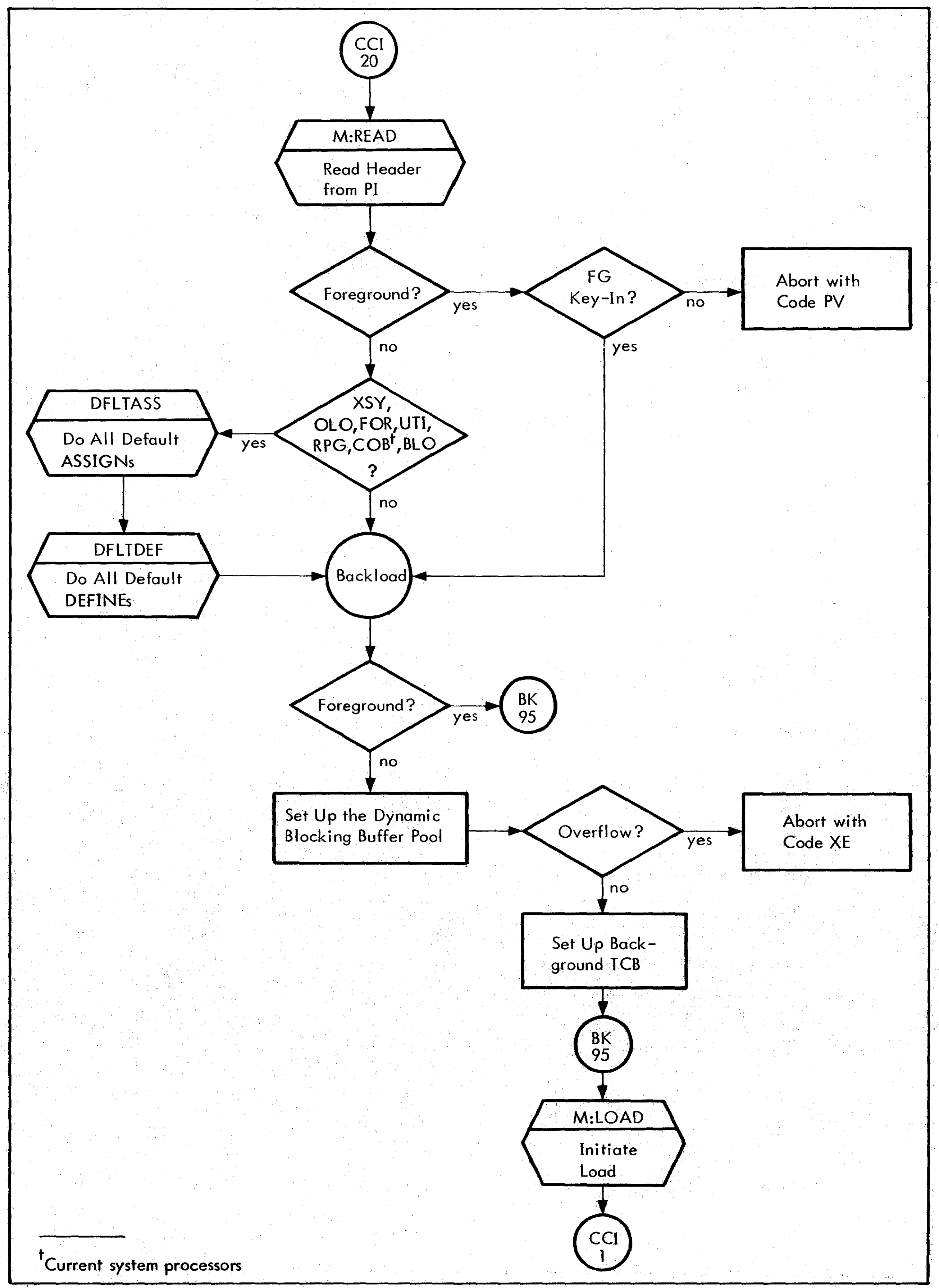


Figure 16. Job Control Processor



[†] Current system processors

Figure 16. Job Control Processor (cont.)

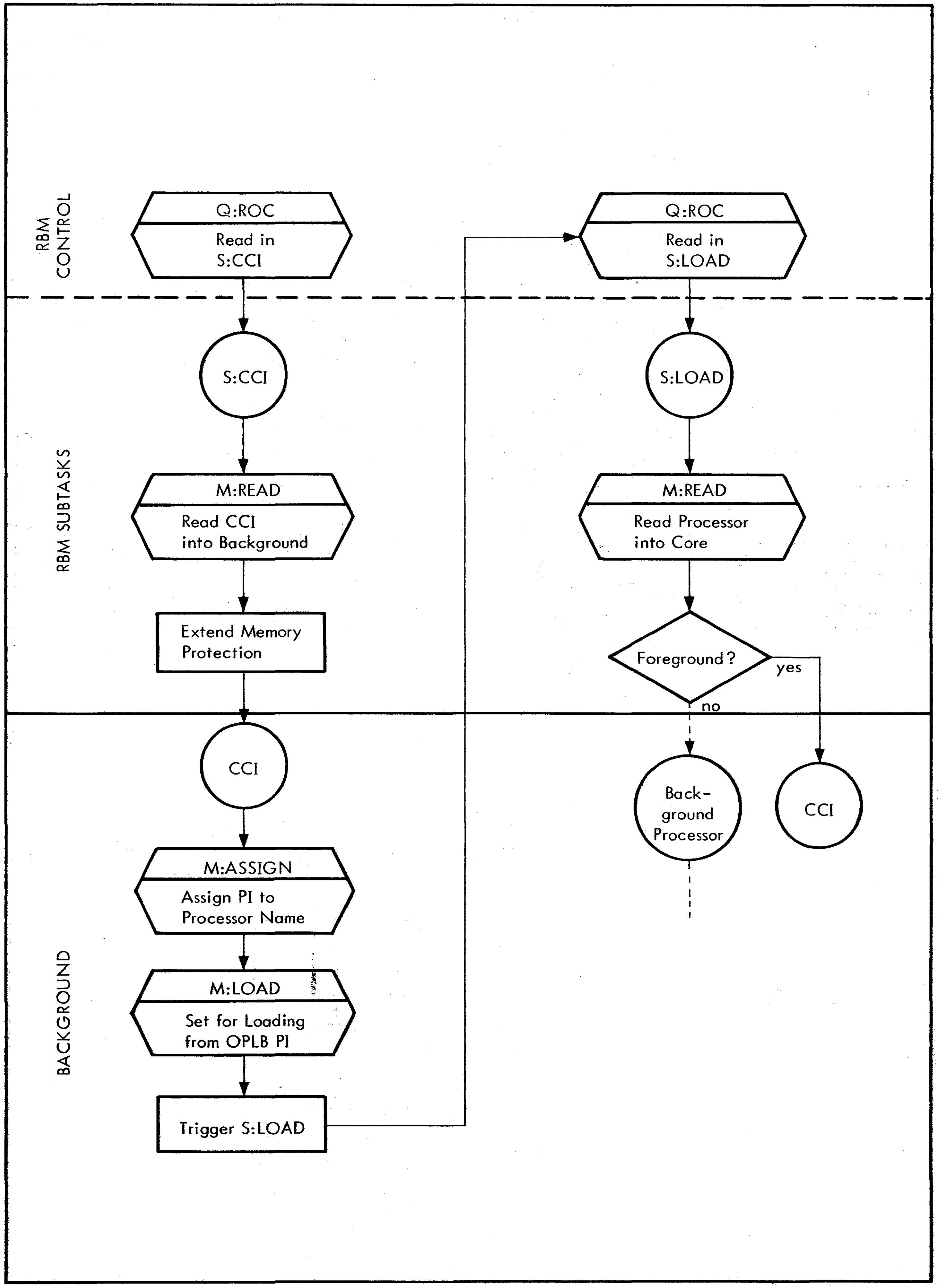


Figure 17. Loading Processors from JCP

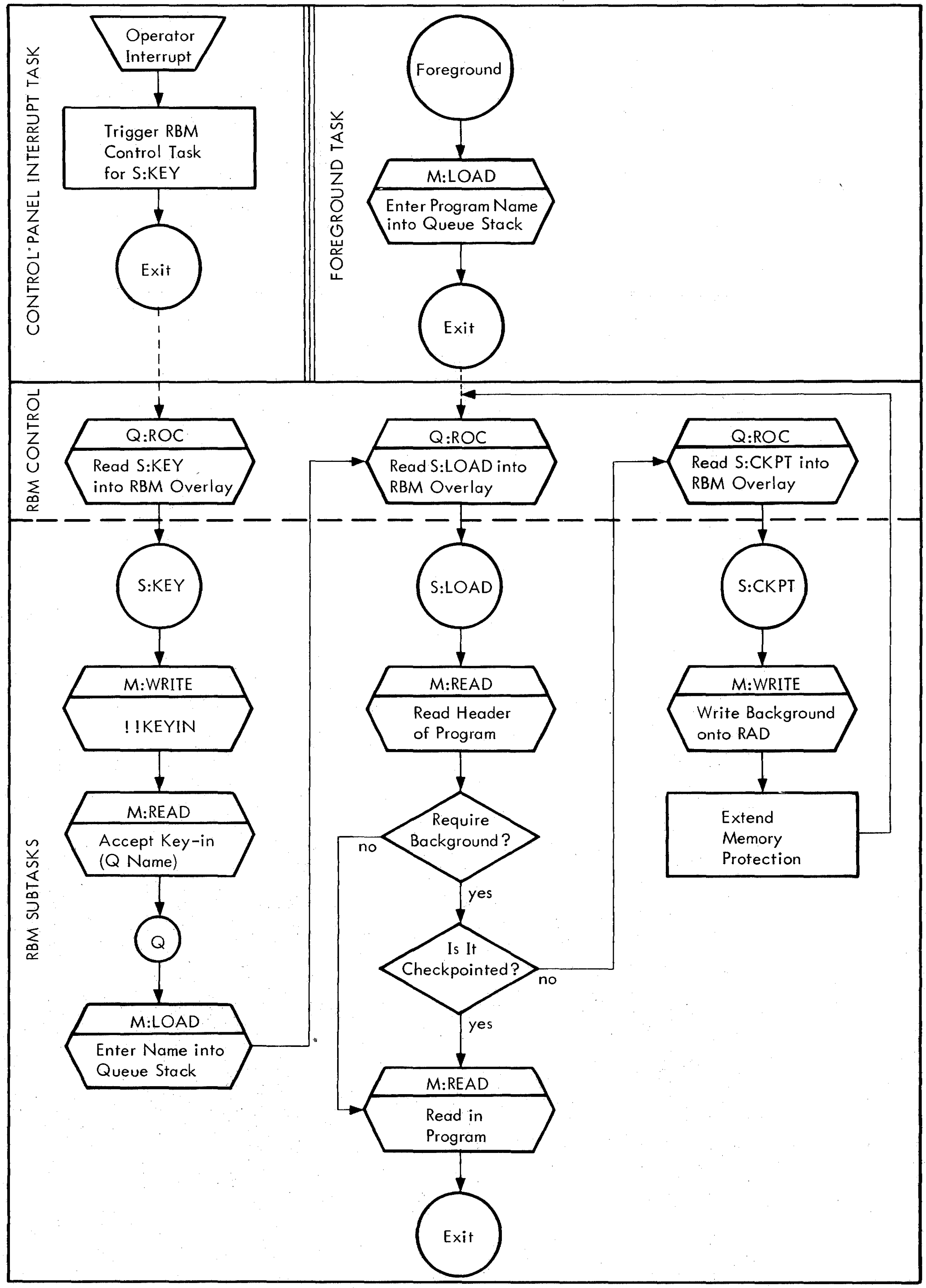
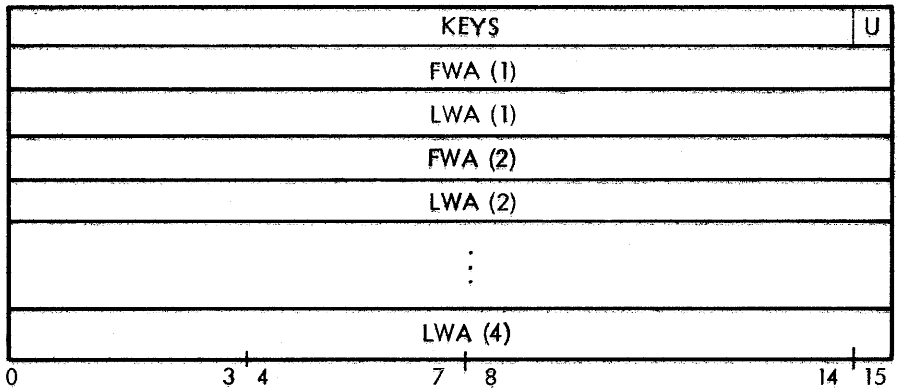


Figure 18. Queue Stack Loading of Foreground Processors



KEYS is a series of 3 bit codes interpreted as follows (each code specifies a complete dump entry):

000 no more dumps

001 hexadecimal

010 hexadecimal/EBCDIC

100 integer

110 mnemonic

U is set to 1 for an unconditional dump request (i.e., PMD will occur regardless of the method if background termination).

FWA is the first word address to be dumped.

LWA is the last word address to be dumped.

Figure 19. Postmortem Dump Table

If an unconditional postmortem dump was requested, it will occur after either type of background termination. If a conditional postmortem dump was requested, it will occur only after an abnormal termination.

Figure 20 illustrates the operation of the RBM Control Task following a request for a postmortem dump.

RBM Accounting

When the JOBACCT option is specified at SYSGEN, accounting services are provided for controlling the execution time of a background job and for maintaining a log of background jobs.

The accounting functions are controlled via the RBM accounting table (see Figure 21). The current date and time of day are stored and maintained in the first two words of the accounting table. The date is stored as the year bias from the most recent leap year (e.g., year-68) and as the half-day of year. The time is stored as the second of the half-day, minus 43,200, and is incremented once each second by the counter-one-equals-zero routine. The counter-one-equals-zero routine will reset this value to -43,200 at the end of the half-day and will increment the half-day of year. The Clock 1 routine will provide watchdog services on background execution time.

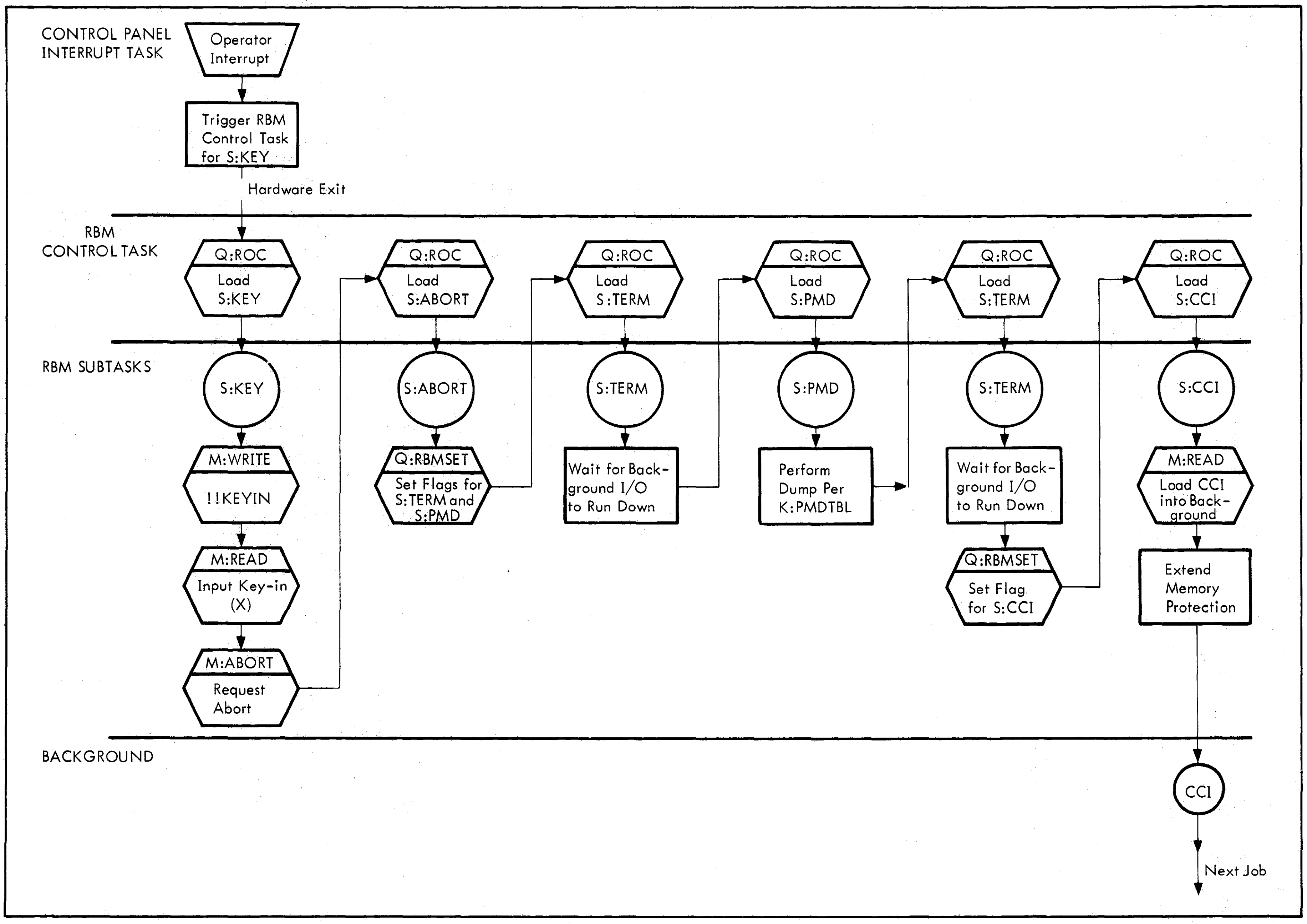
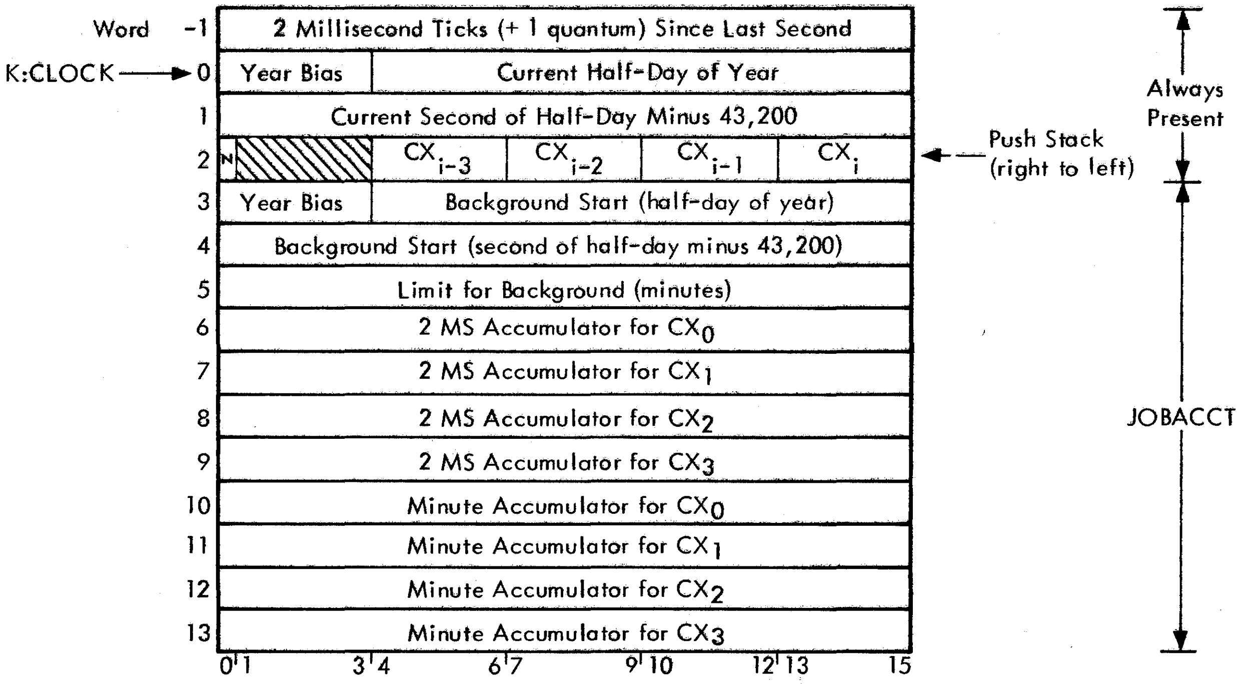


Figure 20. Operator Abort with Postmortem Dump



Word	0	1–3	4–6	7–9	10–12	13–15
-1	2 Millisecond Ticks (+ 1 quantum) Since Last Second					
0	Year Bias		Current Half-Day of Year			
1	Current Second of Half-Day Minus 43,200					
2	N	▨	CX_{i-3}	CX_{i-2}	CX_{i-1}	CX_i
3	Year Bias		Background Start (half-day of year)			
4	Background Start (second of half-day minus 43,200)					
5	Limit for Background (minutes)					
6	2 MS Accumulator for CX_0					
7	2 MS Accumulator for CX_1					
8	2 MS Accumulator for CX_2					
9	2 MS Accumulator for CX_3					
10	Minute Accumulator for CX_0					
11	Minute Accumulator for CX_1					
12	Minute Accumulator for CX_2					
13	Minute Accumulator for CX_3					

Notes: 1. The abbreviations used in this table are described below:

K:CLOCK is a pointer in the zero table.

Year Bias is the value to be added to 1972 to determine year.

N is a flag to indicate job accounting (N = 1 for no job accounting).

CX_i indicates current charge index.

CX_{i-1}, CX_{i-2}, CX_{i-3} indicate previous charge index.

CX_0 foreground execution plus I/O wait.

CX_1 reserved for background I/O wait.

CX_2 background execution.

CX_3 idle (W, FIN, M:WAIT, PAUSE).

2. The number 43200 is a constant (there are 43,200 seconds in a half day).

3. Word-1 when added to Counter 1 will yield 2 ms ticks since last second.

Figure 21. RBM Accounting Table

When a JOB control command is encountered, an entry is made in the RBM accounting file (RBMAL,SD) (see Figure 22). At this time the entry will contain the start time of the job, the user name, and the accounting number as specified on the JOB command. The start time is also recorded in the RBM accounting table. If a LIMIT command is encountered, the execution limit (expressed in seconds) is stored in the RBM accounting table and will be used by the RBM accounting routine to control job execution time.

All time available for use by the background is charged to the entry just created. On encountering a new JOB command or a FIN command, the entry is updated on the RAD to reflect the accumulated execution time.

RBM Accounting File (RBMAL)

The RBM accounting file is a blocked random RAD file that is allocated at system initialization time. It is long enough to contain approximately 75 entries and resides in the System Data area. Each entry or record within the file is submitted as shown in Figure 22.

A special case is made for the IDLE account. The IDLE account will occupy the first entry in the accounting file. Entries n1 through n12 will be blank, entries a3 and a4 will contain the record displacement to the current accounting file entry, and entries a5 and a6 will be used to expand the elapsed time to a double precision value.

All non-IDLE entries, and the elapsed time given in words 10 and 11 of the IDLE entry, will be reset when the accounting file is cleared by the PURGE command specifying the clear option.

Word				
0	Year Bias	Half-Day of Year		Date and Time of Start of Job
1	Second of Half-Day Minus 43,200			
2	n1		n2	
3	.		.	
4	.		.	
5	.		.	
6	.		.	
7	n11		n12	
8	a1		a2	
9	a3		a4	
10	a5		a6	
11	Elapsed Time (seconds)			
	0 3	4 7	8 15	

Note: The abbreviations in this accounting file are described as follows:

Year Bias is the value to be added to 1972 to determine year.

Half-Day of Year is the date at the start of the job.

Second of Half-Day is the time at the start of the job.

n1 - n12 represents the name given on the JOB card, expressed as 12 EBCDIC characters.

a1 - a6 represents the account number given on the JOB card, expressed as 6 EBCDIC characters.

Elapsed Time represents the total background execution time. This value is not set until the next !JOB or !FIN card is encountered.

Figure 22. RBM Accounting File (RBMAL)

2. INPUT/OUTPUT PROCEDURES

Protection

All input/output tables are in protected memory, and all foreground and RBM devices are flagged as protected input/output. Consequently, all background input/output requests are checked for validity before operation is permitted; the check includes both device name and device address and data address. Since any number of devices can be specified for an installation at system generation time, the user has complete control over all input/output protection in the system.

Input/Output Priority

All input/output is initiated at the priority level of the requesting task by calls to the appropriate RBM service routines. No queuing of requests, either on a device or a channel basis, is performed. Thus, up to the point just prior to issuing the SIO, a higher level task can interrupt and seize control of a channel or device from a lower level task. Since device-file numbers are unique to a task, any end action for an input/output operation is remembered for the initiating task until that task has a chance to process it. This implies that all channel end information is saved in the device-file tables rather than in the channel registers. Thus, real-time tasks always have control of the order of input/output operations.

The Monitor does not explicitly know the priority level associated with a given request, but the method of interrupt control guarantees responsiveness to the higher priority tasks. By initiating I/O at the I/O interrupt priority level and following the ground rule that a task with a priority level higher than I/O may not use Monitor I/O, RBM prevents having one I/O request partially initiated on a particular device and then having a higher priority task interrupt with a request for the same device. Without these "inhibits", a device shared by many tasks (e.g., the keyboard/printer) could become "locked", with an operation partially begun by a low-priority task and with a wait loop in a high-priority task that could never be satisfied.

To further solve this problem, all input/output initiated by RBM uses interrupts at device and channel end. When an input/output interrupt is received by the RBM input/output interrupt task, all pertinent status is saved and the channel and device are released for subsequent use by another task. The I/O interrupt is higher than all interrupts that can use RBM I/O services; hence RBM can always release a channel as soon as the actual input/output is complete. Real-time tasks with a priority higher than the I/O level must perform their own input/output without using I/O interrupt control.

Asynchronous Operation

Since Sigma 2/3 can simultaneously operate up to 28 I/O devices on separate I/O channels, RBM must provide for buffering operations. The no-wait options in all I/O requests and the AIO receiver option for foreground can be used to simply and efficiently control buffering operations on several channels concurrently. To reduce system overhead, RBM does not attempt any buffering for the user, but assumes that the user knows better than the Monitor the operations that should be buffered. Hence the user can always control which operations are to be buffered.

Error Recovery

All error recovery is performed at the initiating task level rather than at the I/O interrupt level. Each call to RBM can control whether standard error checking is to be attempted by RBM or whether the user is to perform his own. RBM will retry all operations a given number of times (depending on the device), providing that automatic retry is possible for the device involved and that the user has specified standard error recovery in his calling sequence. RBM makes no attempt to provide "bandwidth" control over I/O operations; thus systems with high-speed devices must control these devices through the external I/O processor. (If a data rate error occurs, it will be treated by RBM like a parity error.) No error recovery will be performed until a "check" operation is requested if the input/output was initiated with a no-wait option. If the background task terminates or aborts without a check request, no error recovery will be attempted. (An initiate and wait is the same as a check request, but may be used in place of a check request.)

Any error message is output to the operator on device file number 1 without using a specific device-file entry, since no entry may be free. The status at channel end is not saved for these messages.

Command Chaining

Command chaining in 530 RBM is a software convention that parallels the command chaining in the Sigma 5/7 hardware. It is used only by RBM and is not available to the user except by M:IOEX. Command chaining is used by RBM to control the unbuffered card punch and low-cost line print (but could be easily adapted to control only unbuffered device). In addition, command chaining is used to obtain sense information from the 9-track tapes and to check for errors on the RAD. It also controls the input/output of information from the keyboard/printer and paper tape to provide flexible editing on a character basis and to simulate fixed-length records on nonrecord equipment by a character-by-character analysis. The pre-setup and the post-setup (e.g., adding or deleting trailing blanks to the keyboard/printer or paper tape in EBCDIC mode) is performed at the level of the initiating task. On input, however, the characters are scanned for editing codes or termination codes at the I/O interrupt level by the keyboard/printer command chaining receiver.

Use of a command chain allows output or input to be performed without requiring the IOCDs to be at the end of the data buffer (e.g., when a NEW LINE code is issued by RBM at the end of each line of output to the keyboard/ printer) and without modifying the user's buffer or moving the entire buffer to the Monitor area. This use is similar to the performance of the Sigma 5/7 with a separate stack of IOCDs. The format of the command chaining operation is illustrated in Figure 23.

When RBM loads the channel registers, the loading routine checks the E flag in the second word of the first IOCD. If the flag is set, RBM will then clear the flag before loading the actual hardware registers, but will pick up the word following the IOCD as a pointer to the next IOCD in the command chain.

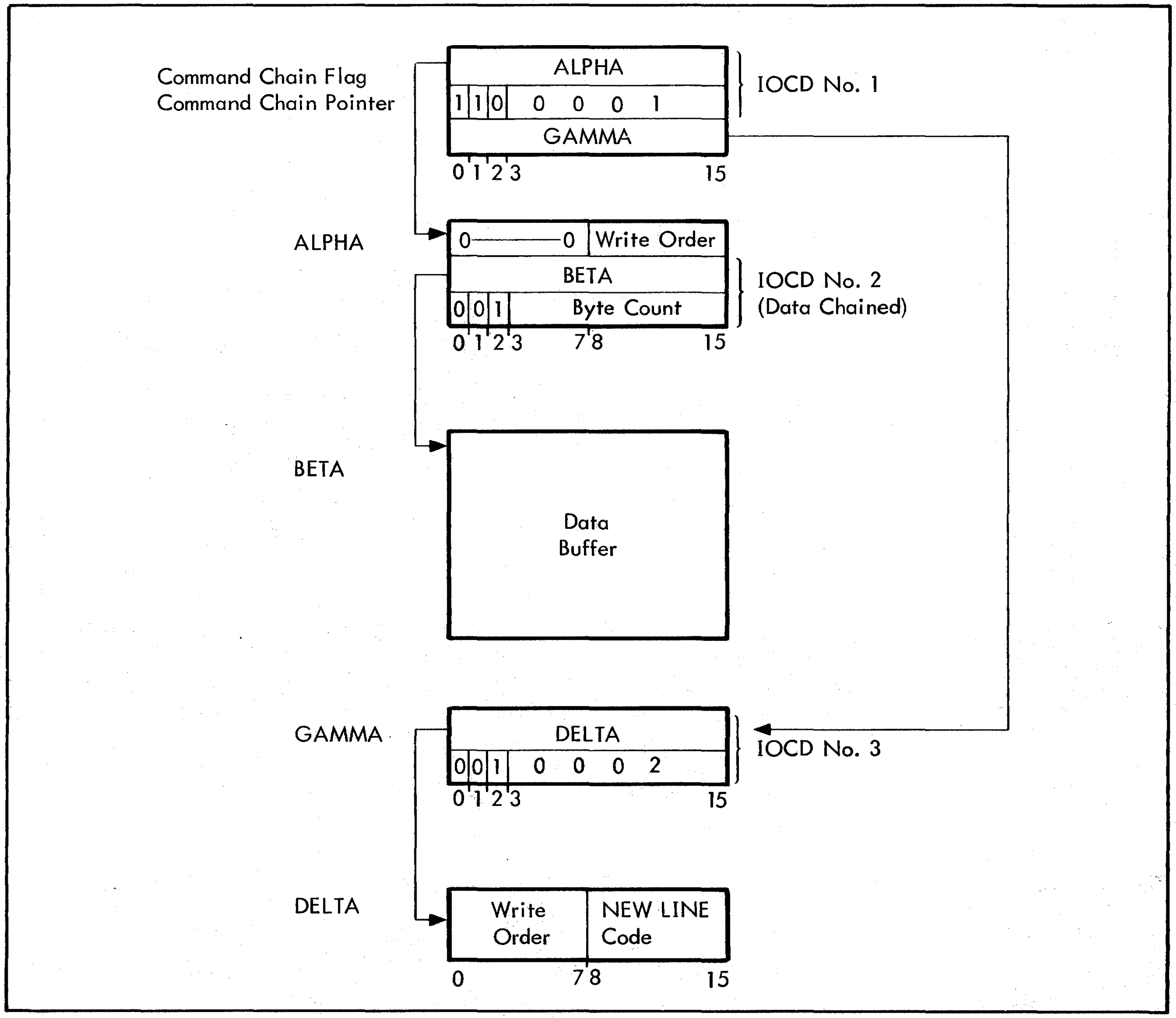


Figure 23. Illustration of Command Chaining

Command chaining requires a new SIO to be issued by the I/O interrupt task when channel end occurs on the first IOCD and when all data chaining is complete. When the chaining modifier or an unusual end is encountered, command chaining ceases. (The command chaining receiver for the keyboard/printer signals the end of the command chaining when a NEW LINE code is encountered in read automatic.)

Command chaining allows RBM to perform editing on a character-by-character basis for low-priority requesting tasks (e.g., the background) even though high-priority tasks are active, since the editing is performed at the I/O task level. Thus operator input to the background on the keyboard/printer can never exceed the ability of the system to respond, even though the requesting task is waiting for a higher priority task to finish. However, a real-time task at a level higher than I/O can seriously affect an I/O operation where command chaining is used. If this task operates for too long (or if interrupts are locked out for too long), the I/O interrupt task will be delayed and a data overrun may occur. For this reason, any real-time task with higher priority than the I/O group must operate for a very short time.

Device-Independent Input/Output

RBM can make many standard operations completely device-independent by (1) using the routines M:READ and M:WRITE to set up I/O requests on a functional rather than a specific basis (see Figure 24), and (2) using device type tables (see Figure 25). Special device-dependent editing routines are called by the general M:READ/M:WRITE routines without the user's knowledge. The structure can also be expanded to new devices with similar characteristics. Device type tables are used by M:READ, M:WRITE, and M:CTRL to set up standard IOCDs and standard order bytes; this frees the user from the work and minimizes duplication of Monitor routines when several I/O devices are included in a system. The device type tables are assembled into RBM and then are compressed and relocated by the selection routines to use only the devices referenced. The general routine M:IOEX does not use the device type tables and thus permits the user to operate with nonstandard requests on standard or nonstandard I/O devices, providing that the devices are compatible with the Sigma I/O interface.

M:CTRL

M:CTRL provides a device-independent positioning capability for magnetic tapes and disk files. An M:CTRL service call exercised on other devices or files will receive a status return of "operation not meaningful".

M:CTRL may be used with either "WAIT" or "NO-WAIT" for completion. Calls to M:CTRL may or may not result in physical transfers. If a "NO-WAIT" call is performed and no physical transfer occurs (e.g., a record backspace for a magnetic tape already at load point), a return will be made with the X-register set to -1 to indicate that the AIO receiver will not be entered. The "check" should be performed immediately. The same condition occurs for both magnetic tape and disk files (although I/O for disk files is actually performed with a "WAIT").

End-action status will be deferred until a subsequent "check" operation is performed. M:CTRL status returns are identical to those for M:READ/M:WRITE. Specifically the X-register will be set to a -1 (when the A-register is equal to zero) on return from "NO-WAIT" calls. This indicates that the AIO receiver will not be entered.

The M:CTRL entry point is the same as for M:READ/M:WRITE. However, M:READ/M:WRITE will bypass the device status check, pseudo order byte test, and validity test on buffer address and byte count, and will proceed directly to the M:CTRL overlay. The magnetic tape overlay will establish the IOCT and temp stack for Q:LOADC. It will then POP to M:READ/M:WRITE to call Q:LOADC.

For "NO-WAIT" M:CTRL calls, a check (order=4) must be performed. This order is treated in the same way as a M:READ/M:WRITE "check" request except that the appropriate device post-I/O editor will be entered. This facilitates EOF and BOT tests.

Channel Time Limits

When Clock 1 is reserved at SYSGEN for RBM accounting, I/O operations initiated by M:READ and M:WRITE are subject to channel time limits. The actual time limit depends on the device and is assembled into RBM. When Q:LOADC (i.e., the channel register loading routine) is called by M:READ or M:WRITE, the two's complement of

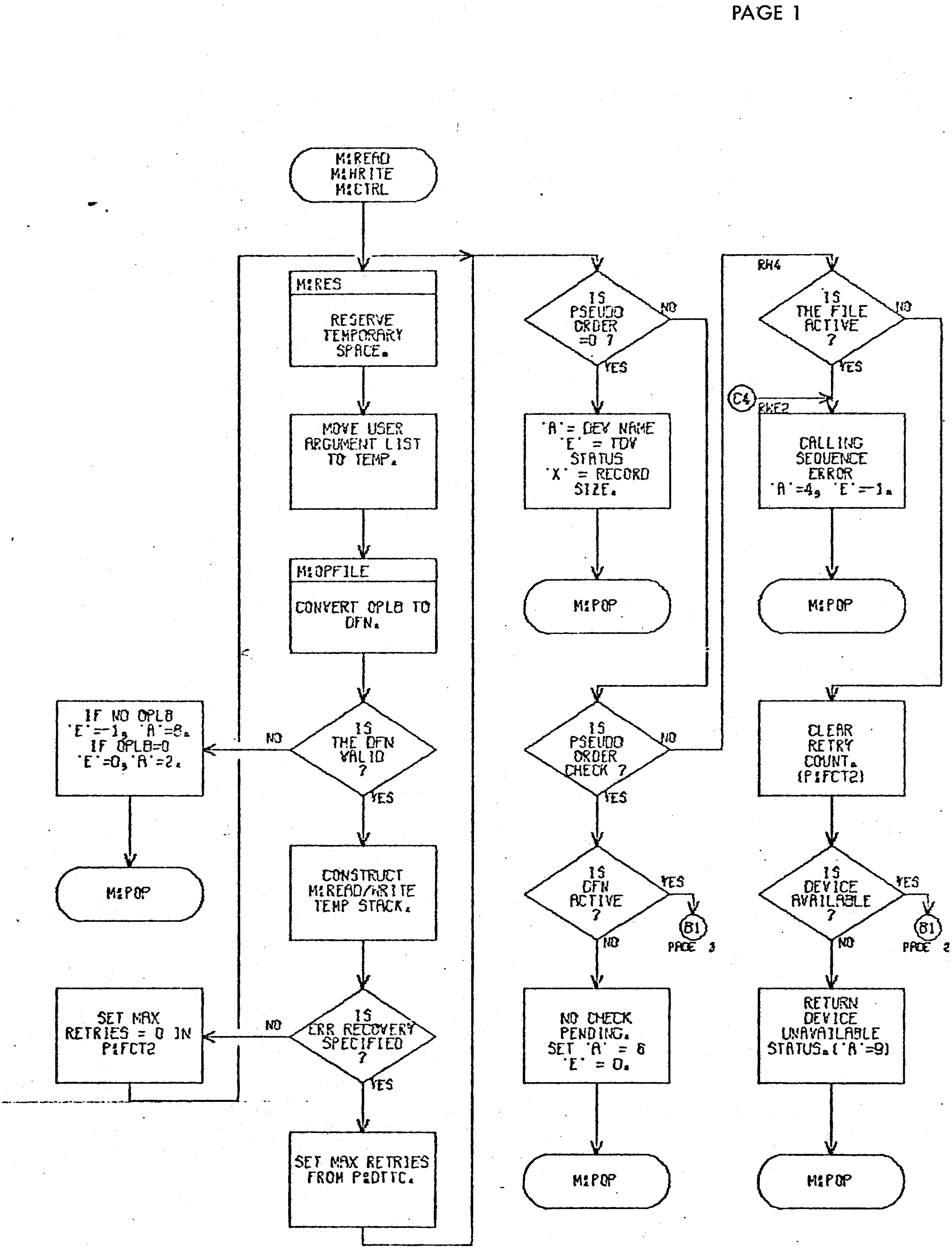

PAGE 1
M:READ
M:WRITE
M:CTRL
M:RES
RESERVE TEMPORARY SPACE.
MOVE USER ARGUMENT LIST TO TEMP.
M:OPFILE
CONVERT OPLB TO DFN.
IS THE DFN VALID ?
NO
YES
IF NO OPLB 'E'=-1, 'A'=8. IF OPLB=0 'E'=0,'A'=2.
M:POP
CONSTRUCT M:READ/WRITE TEMP STACK.
IS ERR RECOVERY SPECIFIED ?
NO
YES
SET MAX RETRIES = 0 IN P:FCT2
SET MAX RETRIES FROM P:DTTC.
IS PSEUDO ORDER =0 ?
NO
YES
'A' = DEV NAME 'E' = TDV STATUS 'X' = RECORD SIZE.
M:POP
IS PSEUDO ORDER CHECK ?
NO
YES
IS CFN ACTIVE ?
YES
NO
B1
PAGE 3
NO CHECK PENDING. SET 'A' = 6 'E' = 0.
M:POP
RW4
IS THE FILE ACTIVE ?
NO
YES
C4
RWE2
CALLING SEQUENCE ERROR 'A'=4, 'E'=-1.
M:POP
CLEAR RETRY COUNT. (P:FCT2)
IS DEVICE AVAILABLE ?
YES
NO
B1
PAGE 2
RETURN DEVICE UNAVAILABLE STATUS. ('A'=9)
M:POP

Figure 24. M:READ/M:WRITE Flow

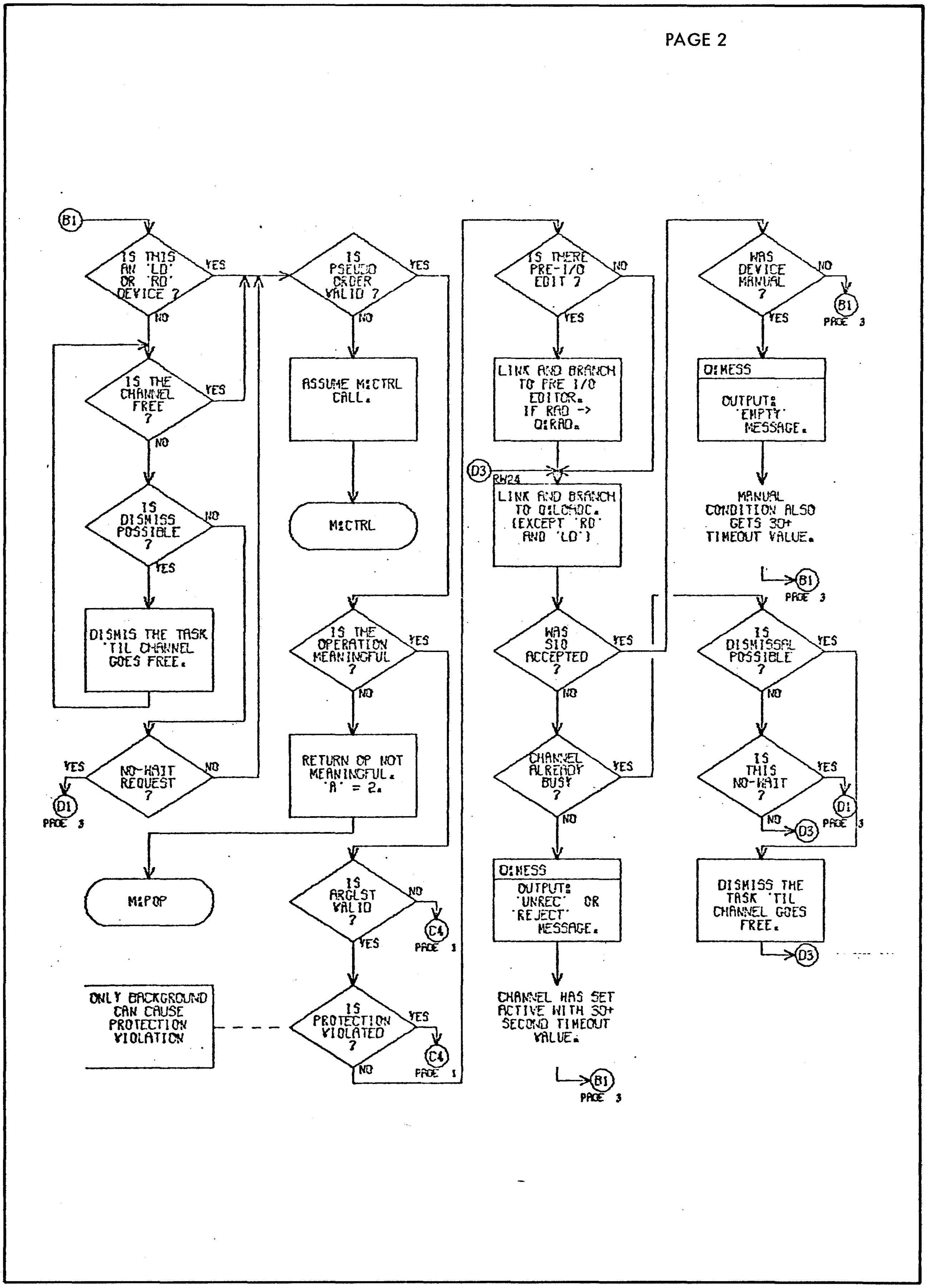

PAGE 2
B1
IS THIS AN 'LD' OR 'RD' DEVICE ?
YES
NO
IS THE CHANNEL FREE ?
YES
NO
IS DISMISS POSSIBLE ?
NO
YES
DISMIS THE TASK 'TIL CHANNEL GOES FREE.
NO-WAIT REQUEST ?
YES
NO
D1
PAGE 3
M:POP
ONLY BACKGROUND CAN CAUSE PROTECTION VIOLATION
IS PSEUDO ORDER VALID ?
YES
NO
ASSUME M:CTRL CALL.
M:CTRL
IS THE OPERATION MEANINGFUL ?
YES
NO
RETURN OP NOT MEANINGFUL. 'A' = 2.
IS ARGLST VALID ?
NO
YES
C4
PAGE 3
IS PROTECTION VIOLATED ?
YES
NO
C4
PAGE 3
IS THERE PRE-I/O EDIT ?
NO
YES
LINK AND BRANCH TO PRE I/O EDITOR. IF RAD -> O:RAD.
D3
RW24
LINK AND BRANCH TO O:LCHDC. (EXCEPT 'RD' AND 'LD')
WAS SIO ACCEPTED ?
YES
NO
CHANNEL ALREADY BUSY ?
YES
NO
O:MESS
OUTPUT: 'UNREC' OR 'REJECT' MESSAGE.
CHANNEL HAS SET ACTIVE WITH 30+ SECOND TIMEOUT VALUE.
B1
PAGE 3
WAS DEVICE MANUAL ?
NO
YES
B1
PAGE 3
O:MESS
OUTPUT: 'EMPTY' MESSAGE.
MANUAL CONDITION ALSO GETS 30+ TIMEOUT VALUE.
B1
PAGE 3
IS DISMISSAL POSSIBLE ?
YES
NO
IS THIS NO-WAIT ?
YES
NO
D1
PAGE 3
D3
DISMISS THE TASK 'TIL CHANNEL GOES FREE.
D3

Figure 24. M:READ/M:WRITE Flow (cont.)

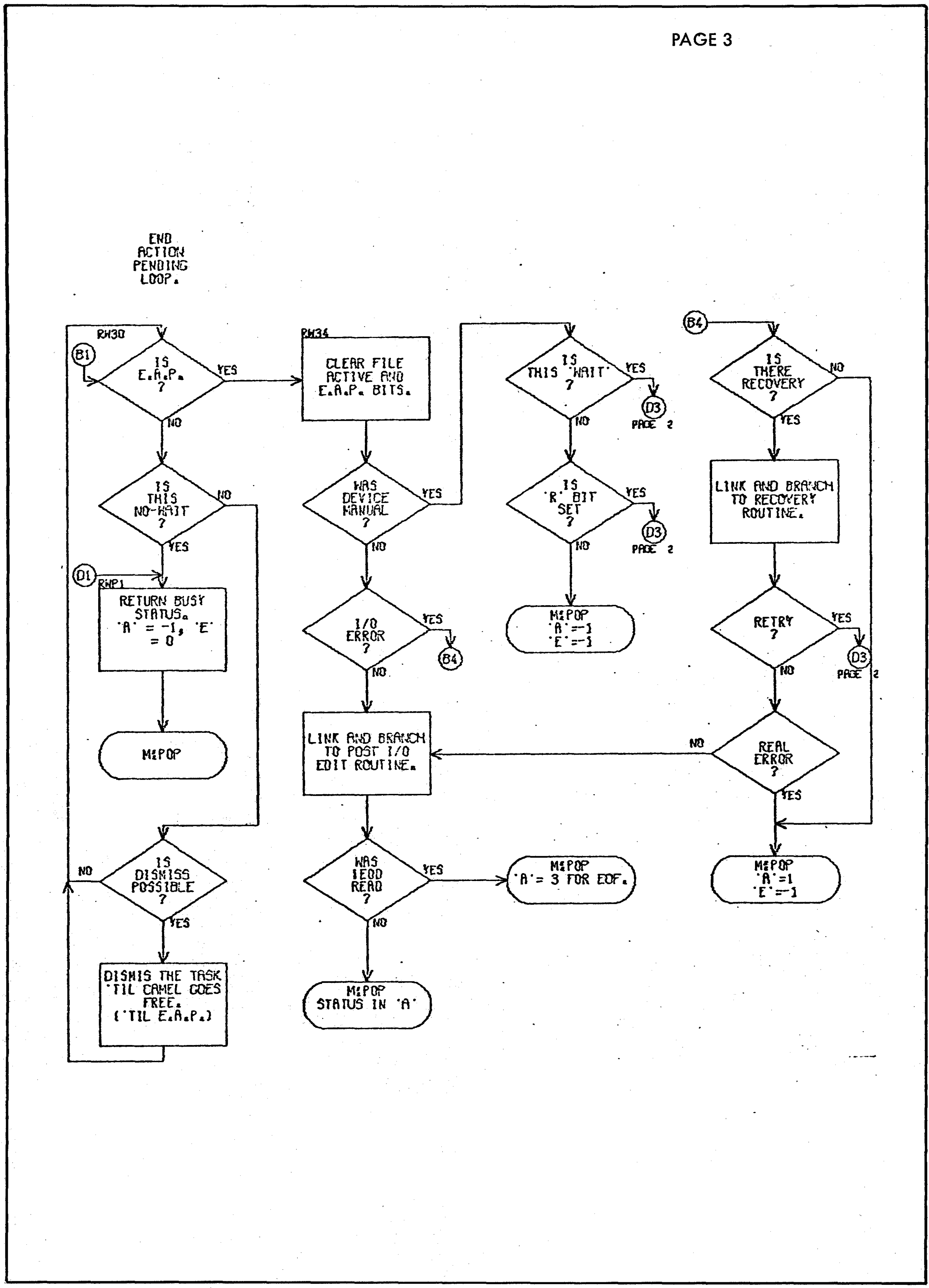
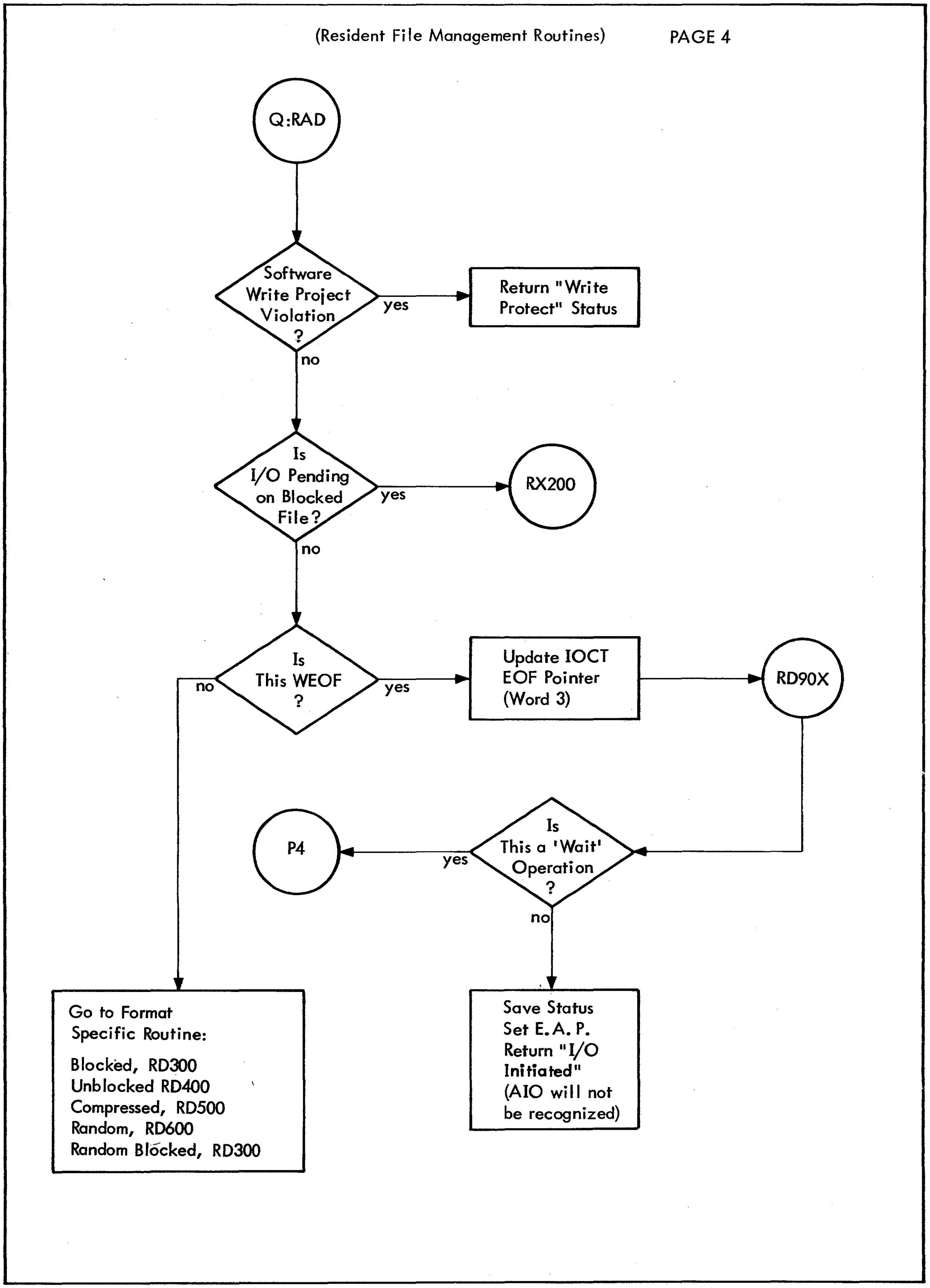


Figure 24. M:READ/M:WRITE Flow (cont.)



(Resident File Management Routines)
PAGE 4
Q:RAD
Software Write Project Violation ?
yes
Return "Write Protect" Status
no
Is I/O Pending on Blocked File?
yes
RX200
no
Is This WEOF ?
yes
Update IOCT EOF Pointer (Word 3)
RD90X
no
Is This a 'Wait' Operation ?
yes
P4
no
Go to Format Specific Routine:
Blocked, RD300
Unblocked RD400
Compressed, RD500
Random, RD600
Random Blocked, RD300
Save Status
Set E.A.P.
Return "I/O Initiated"
(AIO will not be recognized)

Figure 24. M:READ/M:WRITE Flow (cont.)

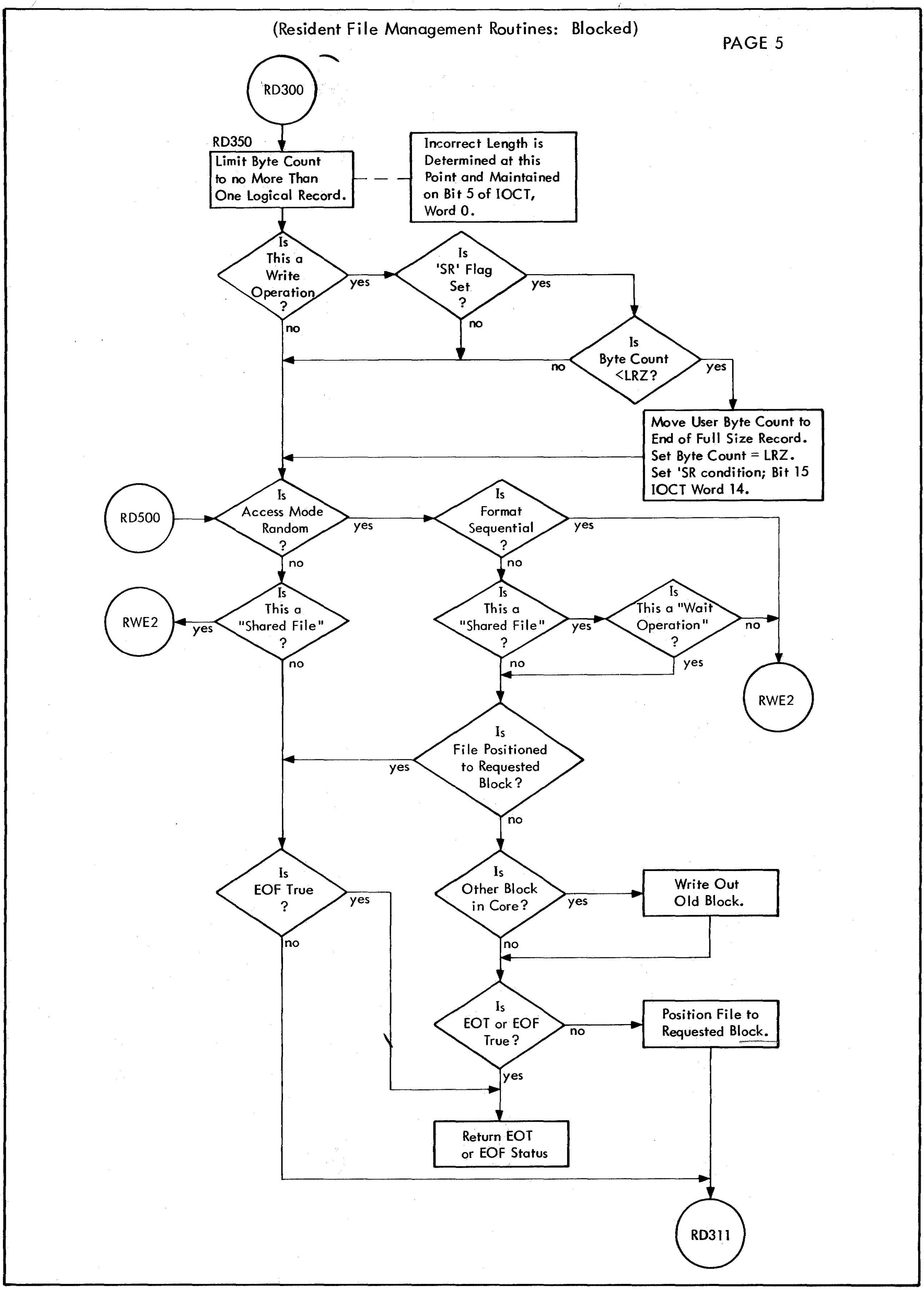

(Resident File Management Routines: Blocked)
PAGE 5
RD300
RD350
Limit Byte Count to no More Than One Logical Record.
Incorrect Length is Determined at this Point and Maintained on Bit 5 of IOCT, Word 0.
Is This a Write Operation ?
yes
no
Is 'SR' Flag Set ?
yes
no
Is Byte Count <LRZ?
no
yes
Move User Byte Count to End of Full Size Record. Set Byte Count = LRZ. Set 'SR condition; Bit 15 IOCT Word 14.
RD500
Is Access Mode Random ?
yes
no
Is Format Sequential ?
yes
no
RWE2
Is This a "Shared File" ?
yes
no
Is This a "Shared File" ?
yes
no
Is This a "Wait Operation" ?
no
yes
RWE2
Is File Positioned to Requested Block?
yes
no
Is EOF True ?
yes
no
Is Other Block in Core?
yes
no
Write Out Old Block.
Is EOT or EOF True?
no
yes
Position File to Requested Block.
Return EOT or EOF Status
RD311

Figure 24. M:READ/M:WRITE Flow (cont.)

(Resident File Management Routines: Blocked - cont.)

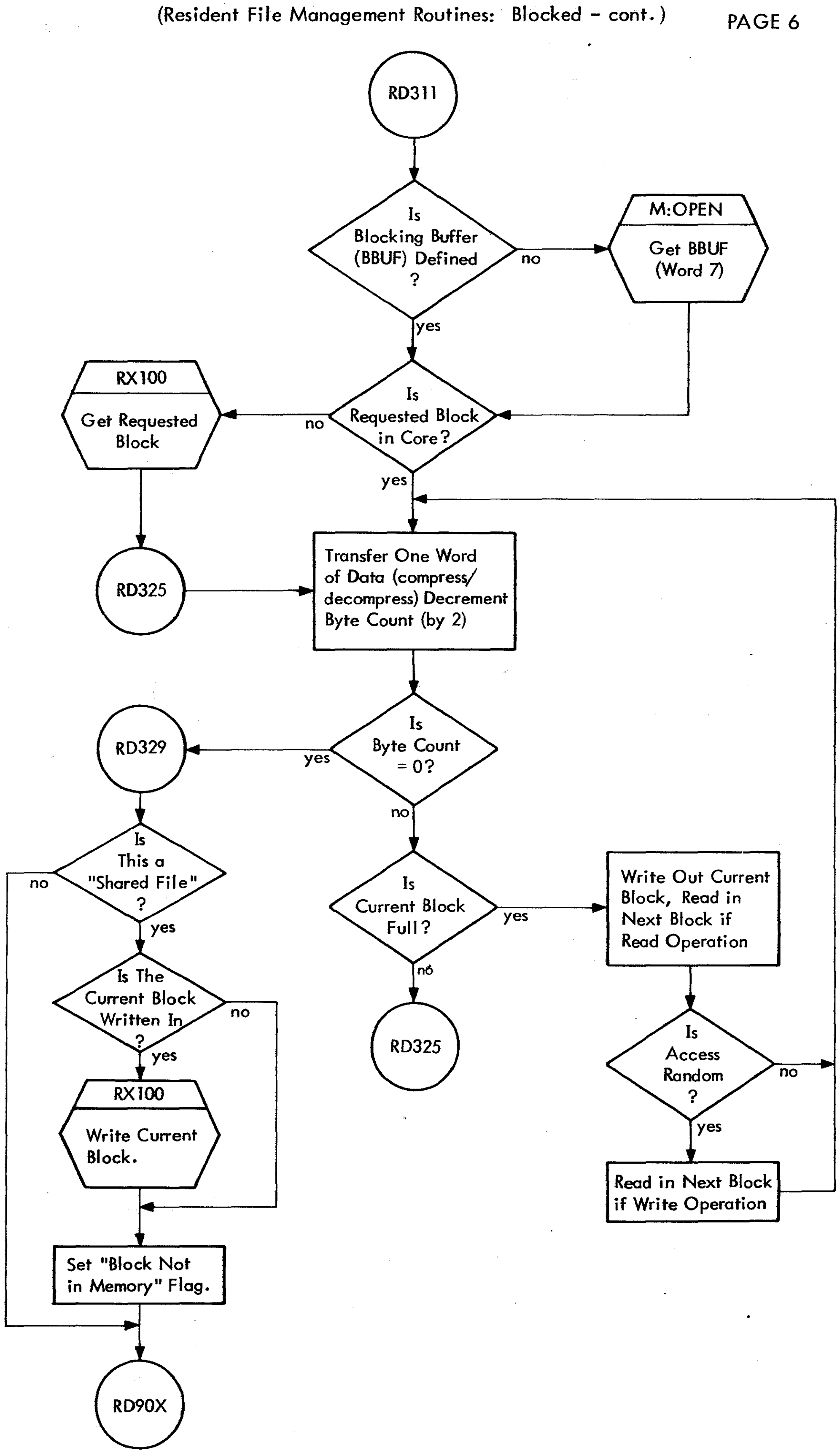

PAGE 6
RD311
Is Blocking Buffer (BBUF) Defined ?
no
M:OPEN
Get BBUF (Word 7)
yes
RX100
Get Requested Block
no
Is Requested Block in Core?
yes
RD325
Transfer One Word of Data (compress/ decompress) Decrement Byte Count (by 2)
RD329
yes
Is Byte Count = 0?
no
Is This a "Shared File" ?
no
yes
Is Current Block Full?
yes
Write Out Current Block, Read in Next Block if Read Operation
Is The Current Block Written In ?
no
yes
nó
RD325
Is Access Random ?
no
yes
RX100
Write Current Block.
Read in Next Block if Write Operation
Set "Block Not in Memory" Flag.
RD90X

Figure 24. M:READ/M:WRITE Flow (cont.)

(Resident File Management Routines: Unblocked)

PAGE 7

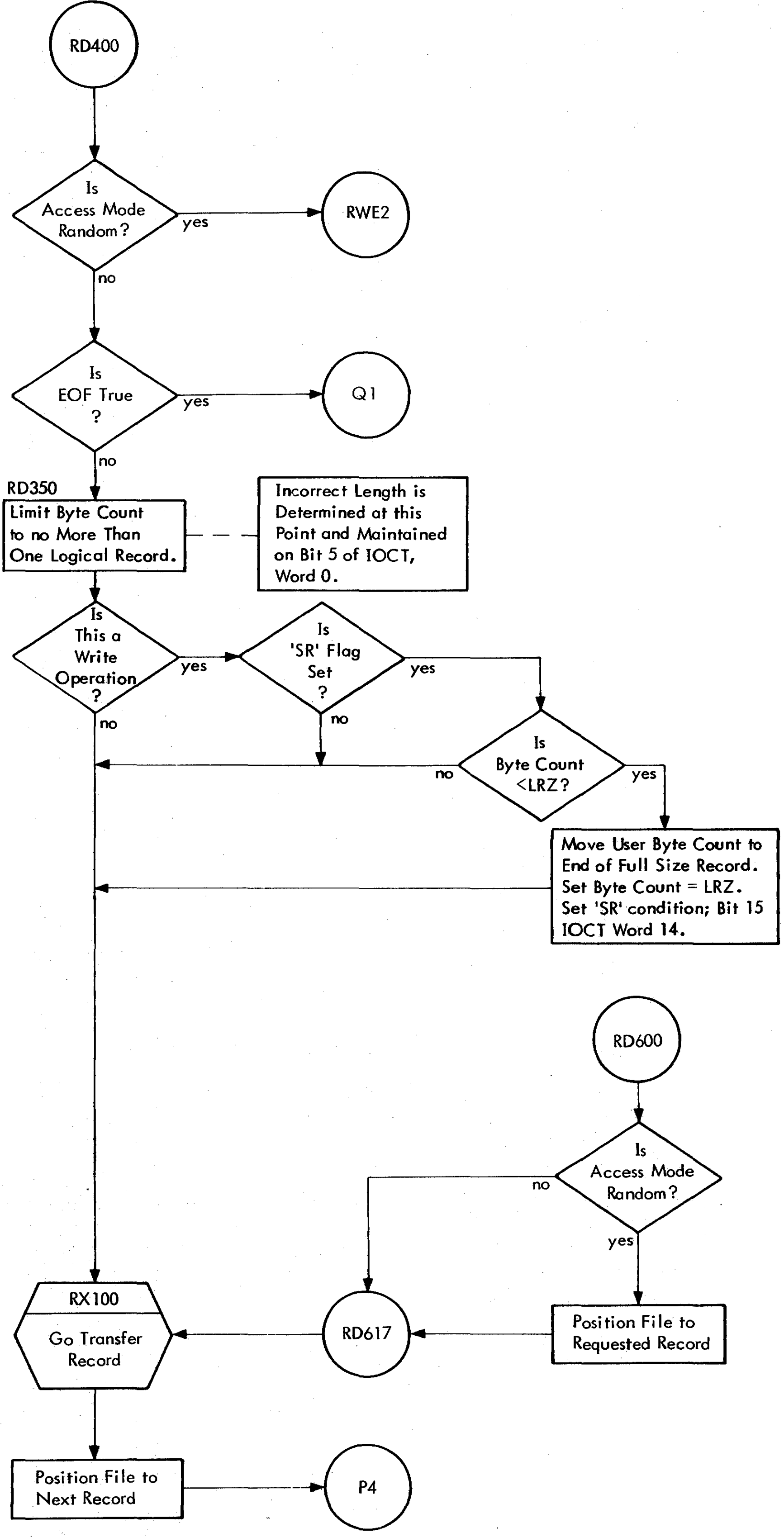

RD400
Is Access Mode Random?
yes
RWE2
no
Is EOF True ?
yes
Q1
no
RD350
Limit Byte Count to no More Than One Logical Record.
Incorrect Length is Determined at this Point and Maintained on Bit 5 of IOCT, Word 0.
Is This a Write Operation ?
yes
Is 'SR' Flag Set ?
yes
no
no
Is Byte Count <LRZ?
no
yes
Move User Byte Count to End of Full Size Record. Set Byte Count = LRZ. Set 'SR' condition; Bit 15 IOCT Word 14.
RD600
Is Access Mode Random?
no
yes
Position File to Requested Record
RD617
RX100
Go Transfer Record
Position File to Next Record
P4

Figure 24. M:READ/M:WRITE Flow (cont.)

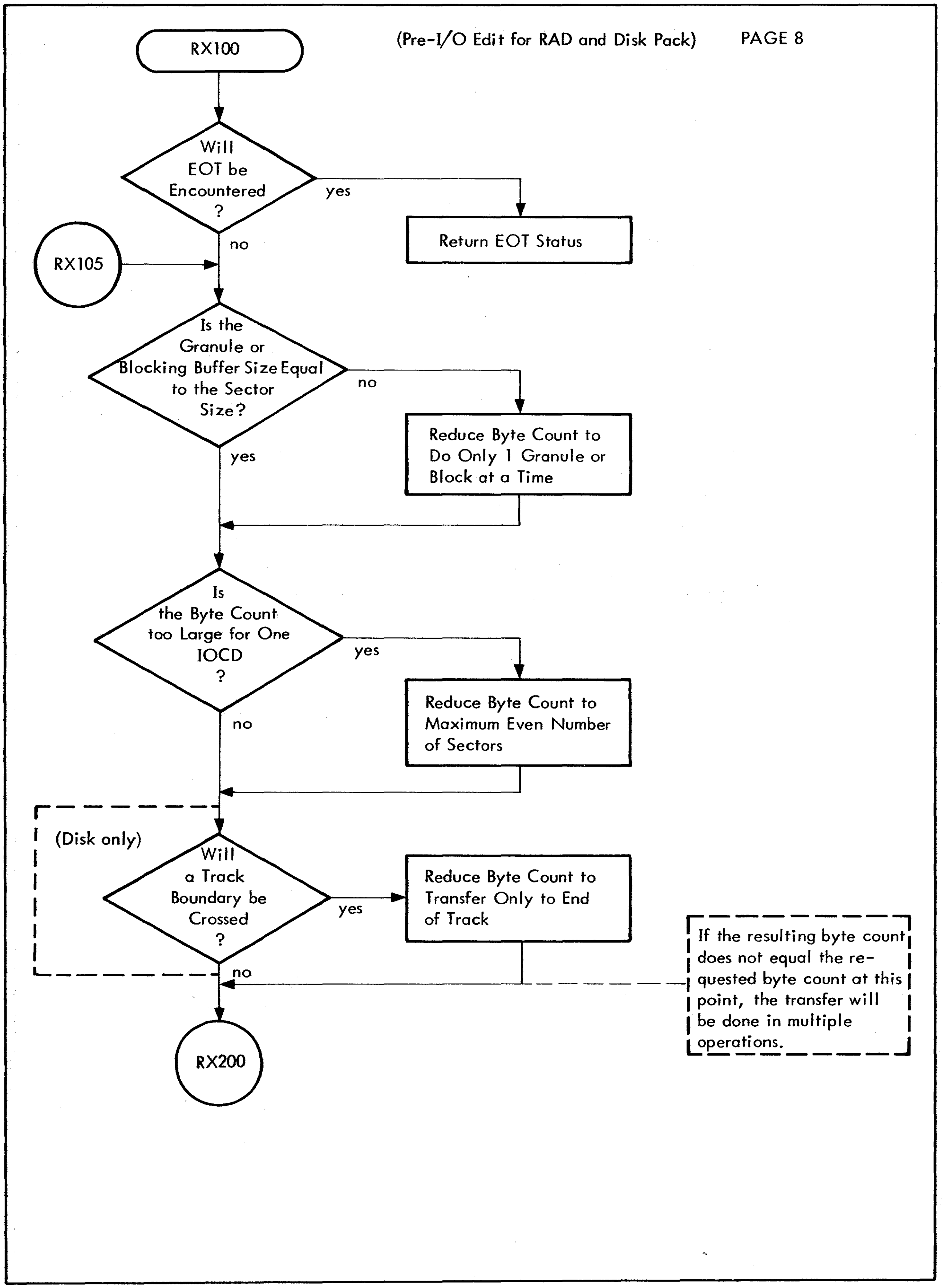

RX100
(Pre-I/O Edit for RAD and Disk Pack)
PAGE 8
Will EOT be Encountered ?
yes
Return EOT Status
no
RX105
Is the Granule or Blocking Buffer Size Equal to the Sector Size?
no
Reduce Byte Count to Do Only 1 Granule or Block at a Time
yes
Is the Byte Count too Large for One IOCD ?
yes
Reduce Byte Count to Maximum Even Number of Sectors
no
(Disk only)
Will a Track Boundary be Crossed ?
yes
Reduce Byte Count to Transfer Only to End of Track
no
If the resulting byte count does not equal the requested byte count at this point, the transfer will be done in multiple operations.
RX200

Figure 24. M:READ/M:WRITE Flow (cont.)

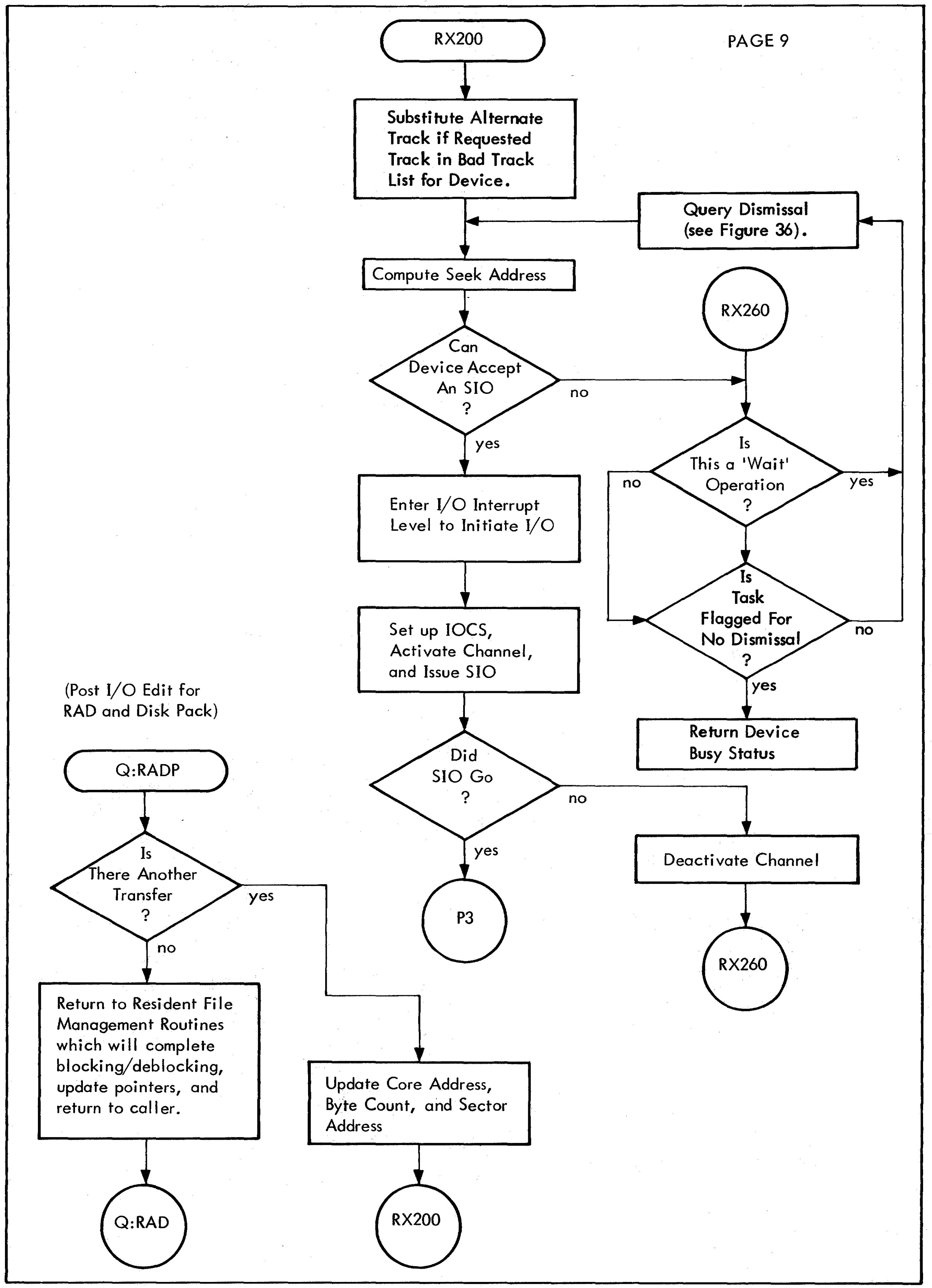

RX200
PAGE 9
Substitute Alternate Track if Requested Track in Bad Track List for Device.
Query Dismissal (see Figure 36).
Compute Seek Address
RX260
Can Device Accept An SIO ?
no
yes
Is This a 'Wait' Operation ?
no
yes
Enter I/O Interrupt Level to Initiate I/O
Is Task Flagged For No Dismissal ?
no
yes
Set up IOCS, Activate Channel, and Issue SIO
(Post I/O Edit for RAD and Disk Pack)
Return Device Busy Status
Q:RADP
Did SIO Go ?
no
yes
Deactivate Channel
Is There Another Transfer ?
yes
no
P3
RX260
Return to Resident File Management Routines which will complete blocking/deblocking, update pointers, and return to caller.
Update Core Address, Byte Count, and Sector Address
Q:RAD
RX200

Figure 24. M:READ/M:WRITE Flow (cont.)

ABS Loc (Hex.)	Pointer	Contents of Tables Pointed to	
55	P:DTT =	Length of Device Type Tables	
56	P:DTT1[†]	Device Type Name for Error Messages	
57	P:DTT2	Standard Record Size (bytes)	
58	P:DTT3	Read Automatic Order Byte	Write EBCDIC Order Byte
59	P:DTT4	Read Binary Order Byte	Write Binary Order Byte
		If not rotating memory:	
5A	P:DTT5	Read Backward Order Byte	Write EOF Order Byte
		If rotating memory:	
		Number of Sectors per Track	
		If not rotating memory:	
5B	P:DTT6	Pre-I/O Editor Address (if applicable; zero means none)	
		If rotating memory:	
		Number of Tracks per Device	Number of Cylinders per Device
		If not rotating memory:	
5C	P:DTT7	Post-I/O Editor Address (if applicable)	
		If rotating memory:	
		Number of Tracks per Cylinder	Number of Alternates per Device
		If not rotating memory:	
5D	P:DTT8	Command Chaining Receiver Address (if applicable)	

[†] For all Logical Device DFNs, the mnemonic 'LD' will be stored in DTT1, while the actual two-character mnemonic used in the SYSGEN deck will be stored in FCT7; e.g., LD, LP, L1 etc.

Figure 25. Device Type Tables

		If rotating memory:		
		Flags (see Note 4)		
		If not rotating memory:		
5E	P:DTT9	Special Error Recovery Address (if applicable)		
		If rotating memory:		
		VTOC Sector		
		If not rotating memory:		
5F	P:DTTA	TDV Manual Read Mask (DSB)	TDV Manual Write Mask (DSB)	
		If rotating memory:		
		Number of Sectors On Device		
60	P:DTTB	Standard IOCD Flags and Byte Count Word, for IOCD No. 1		
61	P:DTTC	I	Max. No. of Retries	IOCT Length
			6	7
62	P:DTTD	Standard User-Byte-Count-Word IOCD Flags		
		If 'PT' device:		
63	P:DTTE	Transfer Rate (always < X'8000')		
		For all other devices:		
		2's Complement of Timeout Value (seconds) -- 0 if 'KP' Device		
70	P:DTTF	Model Number as a Binary Integer		

Notes: 1. Some order bytes are pseudo order bytes and are modified during the pre-I/O process to an actual hardware order byte.

2. This table is indexed by device type, from 1 to n, in the same way as file control tables.

3. I = 1 if command chaining for the device can be interrupted if the device is used by the background and the background has been checkpointed.

Figure 25. Device Type Tables (cont.)

4. Flags in DTT8

Bit	Designation	If Set to 1
0	SOP	Seek overlap okay on devices whose addresses differ only in least significant bit.
1	SO	Seek overlap okay on device type subject to restriction of bit 0.
2	BTL	Alternates indicated for bad tracks by bad track list.
3	FLW	Alternates indicated for bad tracks by flaw marks in headers.
4	RRS	Restore required for seek.
5	RRH	Restore requires header read.
6	TWS	Two word seek address.
7	CSD	Collect sense data.
8	SST	Single track transfers.

Figure 25. Device Type Tables (cont.)

the device time limit is stored into the channel status table entry, indexed through P:CST8. Once each second, this value is incremented by the counter-one-equals-zero routine, and if the time limit is exceeded, an HIO is issued to the offending device and the status and end-action pending flags are stored in the associated file control table entry. The associated channel then is made available for use.

Operational Labels

Many references to I/O devices are on a logical rather than physical basis, and the operational label tables are designed to permit this logical referencing. There are two such tables, one for background and one for foreground. When RBM operates at a hardware priority level, the foreground operational labels are used; when RBM operates at the background level (below all hardware interrupts), the background table is used. The structure of the table is shown in Figures 26 and 27. Note that entries in the tables are indexed with a negative value in the index register to facilitate searching with a BXNC instruction. Thus, the pointers point to one location beyond the actual tables. The pointers P:BOL1 and P:BOL2 are in the zero table to facilitate RBM referencing. The item P:BOL contains the negative length of the table, to be loaded into the X register before the search. All other I/O tables use indexing and zero table pointers but are indexed forward.

Channel Status Tables

The channel status table is a convenient method of controlling channel activity. Since there is no "test channel" I/O command, RBM will maintain status for all device controllers on each channel. The following items are also included in the channel status tables since there are no hardware registers for these items: the AIO receiver, the command chaining receiver, and the command chaining pointer. The channel status tables are created at RBM selection and are cleared with each initialization process.

The channel status tables for a given channel are accessed via a double index, once with a device's actual hardware channel number into the index table (CST0) to the channel status table and then, with the attained value, into the channel status tables. For example, where B:CHAN is the actual channel number,

```
LDX      B:CHAN
LDX      *P:CST0, 1
LDA      *P:CST(X), 1
```

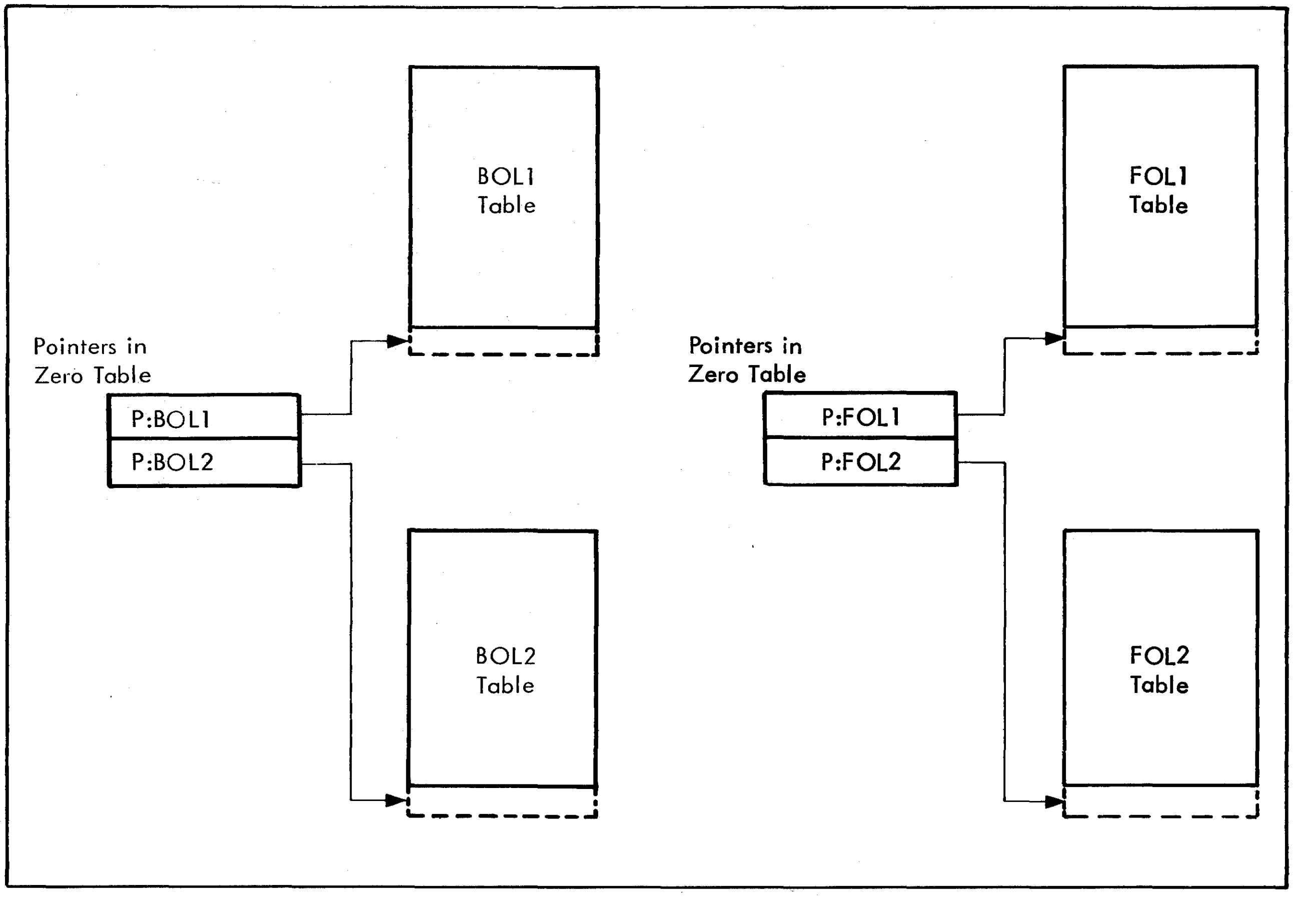
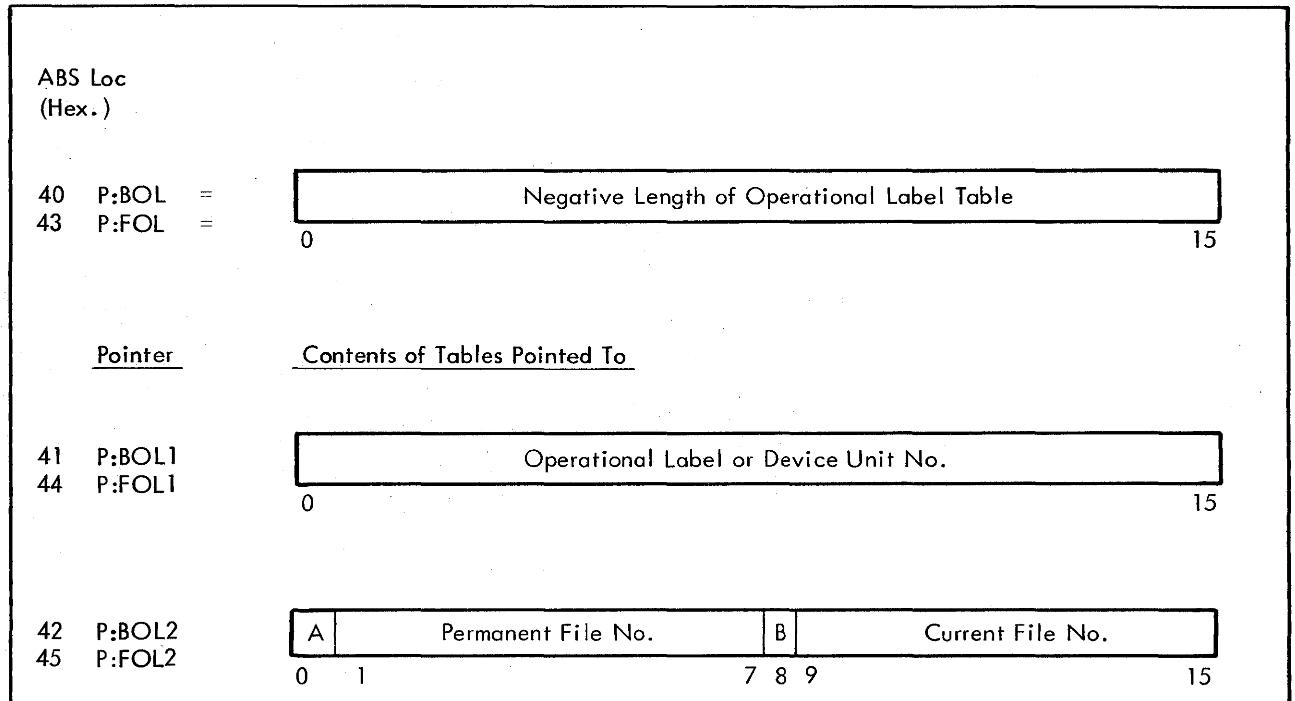


Figure 26. Operational Label Table Pointers



ABS Loc
(Hex.)

40 P:BOL =
43 P:FOL =

Negative Length of Operational Label Table
0 15

Pointer

Contents of Tables Pointed To

41 P:BOL1
44 P:FOL1

Operational Label or Device Unit No.
0 15

42 P:BOL2
45 P:FOL2

A	Permanent File No.	B	Current File No.
0	1 7	8	9 15

Notes: 1. Appropriate BOL1 and BOL2 table entries are obtained by an indexed search operation on a background operational label. Appropriate FOL1 and FOL2 table entries are obtained by an indexed search operation on a foreground operational label.

Figure 27. Operational Label Table

Notes: (cont.)

2. Blank (i.e., zero) entries are used for temporary user assignments.
3. A permanent file is indicated by bit 0 of word 2 being set to 1.
4. Foreground operational labels are similar to background operational labels.
5. If A = 1, this oplb has a standard or permanent file assignment; if A = 0, it does not.
6. B is set to 1 whenever the oplb is assigned to a file by a call to M:ASSIGN or M:DEFINE.

Figure 27. Operational Label Table (cont.)

P:CST0 points at the CST0 (the index table to the channel status table), and the A register contains the value in the Xth channel status table.

Care must be taken in deriving a device's actual channel number from its device number, since multiunit devices on the external IOP will have the same apparent channel number as a device on one of the first eight channels of the internal IOP. To resolve this ambiguity, a nine-word table with a zero table pointer (R:IOP) is used. The apparent channel number of a multiunit device is used as an index into this table. If the result is X'20', the multiunit device is on the external IOP and its true channel number is the apparent channel number plus 12. Otherwise, the value in this table is eight, which indicates that the device is on an internal IOP and that the apparent and actual device channel numbers are the same.

RBM will support one multiunit controller on each of the internal IOP channels 0 through 7 and the external IOP channels X'C' through X'13', and up to four single-unit controllers on each of the remaining channels.

Note: In contrast to any other RBM tables, the channel status tables are indexed forward by the value attained in P:CST0, beginning with zero and ending with the value in P:CST. The channel status pointers and tables are illustrated in Figures 28 and 29.

File Control Tables

A number of central tables are used to preserve the information needed for maintaining reentrant, asynchronous operation, and multiple tasks per device. To facilitate referencing and searching, the central file tables are organized as shown in Figures 30 and 31.

The I/O control tables (see Figures 32, 33, and 34) are designed to control the IOCDs (which must be contiguous) created by the RBM routines.

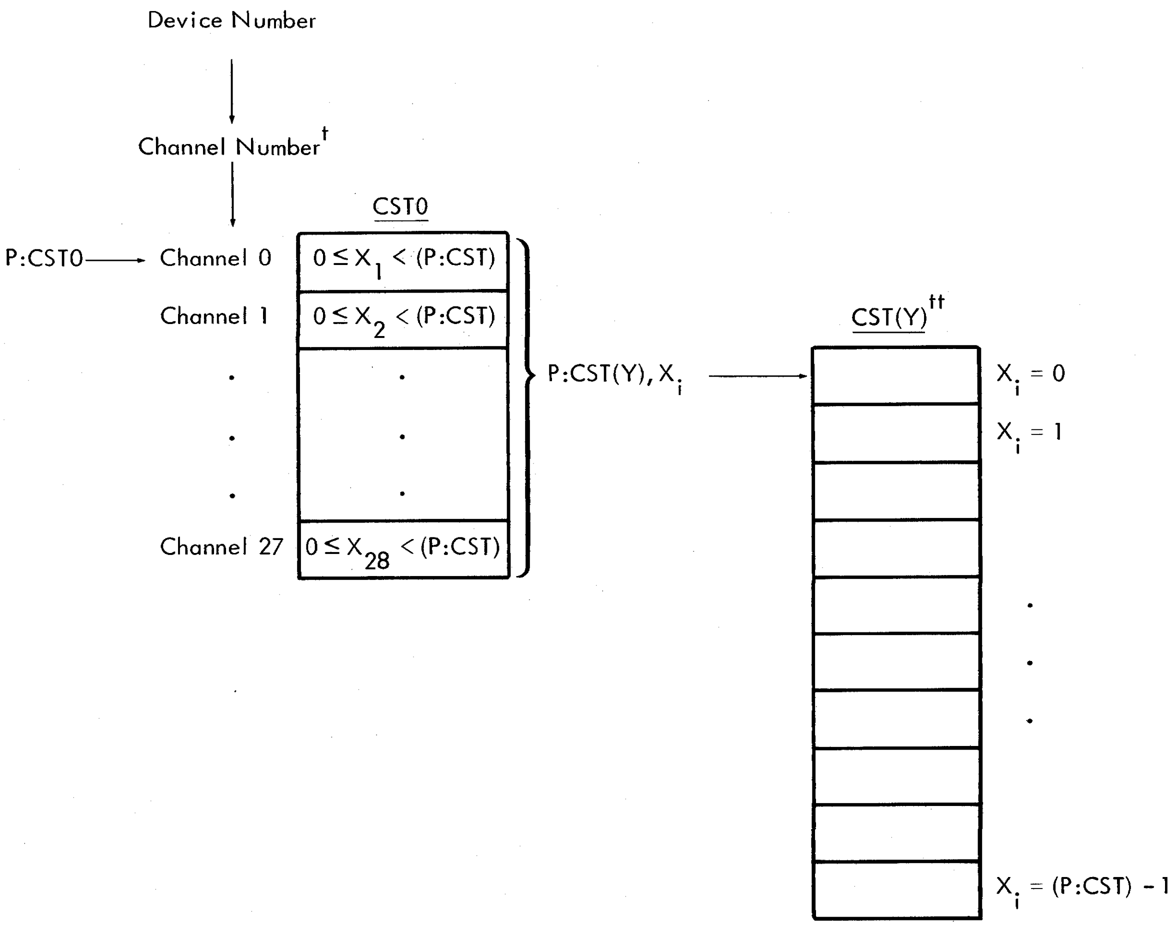
Disk Pack Seek Overlap

Seek overlap is achieved in RBM by setting the channel status indicator (C:CST1) active for only those operations that actually utilize the channel. On the 7242 device, head movement after the seek address is received does not require channel activity, the channel is not set active and other disk devices may use the channel to initiate seek or data transfer operations.

During head movement, the device itself is busy and this status is indicated by the "busy" bit in P:CST5. During other phases of disk pack I/O, the entire channel is active. A flow diagram of the disk pack I/O is given in Figure 35.

Task Dismissal on Wait I/O

The task dismissal feature, a SYSGEN option, allows foreground tasks to be automatically dismissed by RBM if they elect to wait for I/O completion. Dismissal is to the next lower priority ready task, providing a further overlap of CPU execution and I/O processing to the enhancement of low priority throughput. This task is DISABLED for interrupts while dismissed. The feature is controllable on a task basis and on a system basis. This feature can significantly increase total system throughput. The flow of the Dismissal routine is shown in Figure 36.



where

CST0 is a table, 28 words long, one word for each possible channel. If a channel is undefined, $X_i = -1$.

P:CST0 is a pointer in the zero table at entry 0 of CST0.

P:CST contains the number of defined channels.

CST(Y) $(1 \leq Y \leq 8)$ is the actual channel status tables, 1 through 8, as used in Figure 29.

P:CST(Y) is a pointer in the zero table at entry 0 of CST(Y).

[t] As normally derived from the device number, except add 12 for a multiunit device on an external IOP.

[tt] In this example, 10 unique channels have been defined at SYSGEN (i.e., (P:CST) = 10).

Figure 28. Channel Status Table Structure

ABS Loc (Hex.)		
19C	P:CST =	Number of I/O Channels

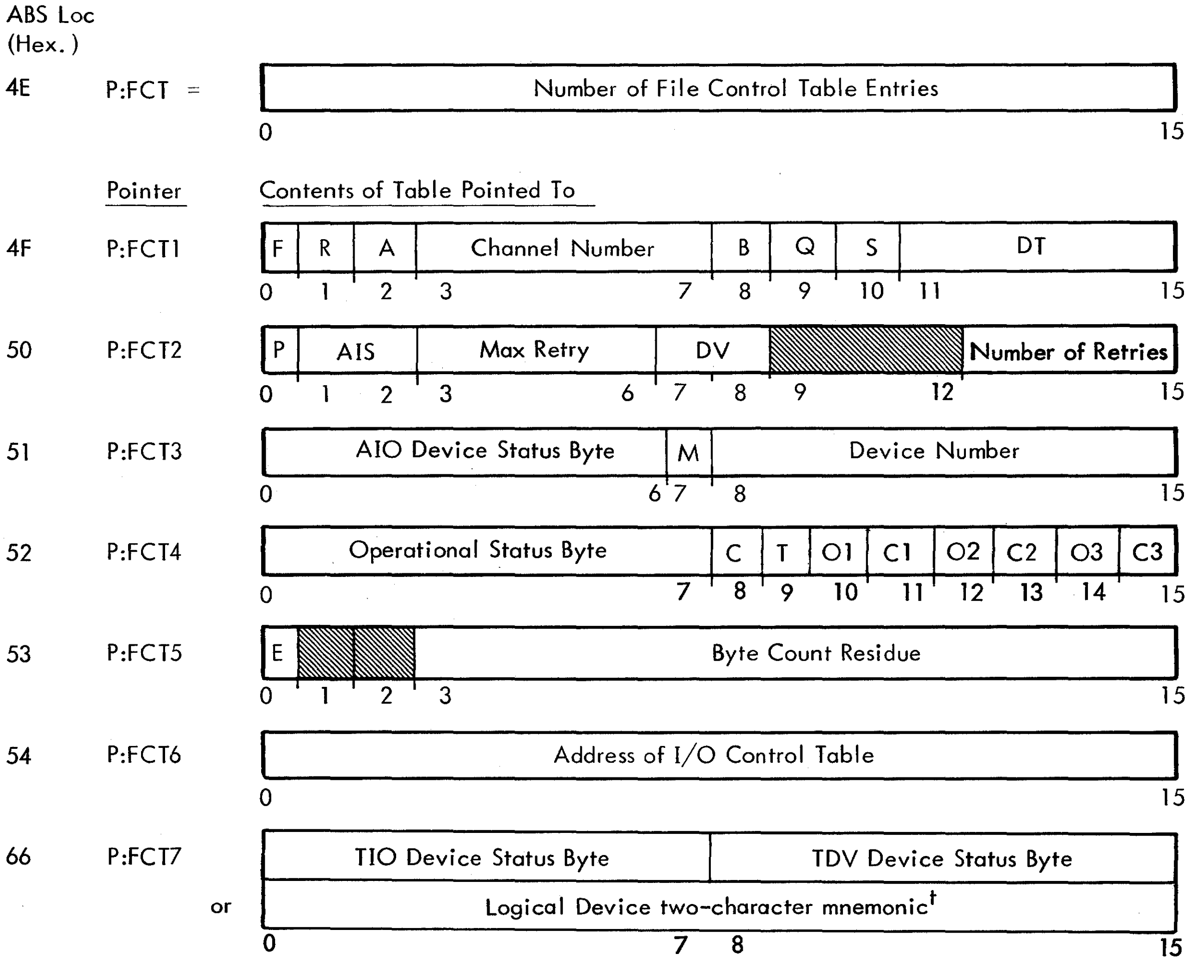
	Pointer	Contents of Table Pointed To
46	P:CST0	Channel Index Value (28 words long)
47	P:CST1	Active Device-File Number
48	P:CST2	Response Control Word (AIO)
49	P:CST3	Command Chaining Pointer
4A	P:CST4	Command Response Word
4B	P:CST5	Busy Bits
4C	P:CST6	P:CST1 If CKPT Suspension
4D	P:CST7	(Currently Unused)
6D	P:CST8	Channel Time Limit
		0 15

Notes:

1. This table is indexed by channel status table index value, from 0 to P:CST - 1.
2. CST1 = 0 if channel is inactive.
3. CST2 = address of AIO receiver. (Zero means none.)
4. CST3 = address of IOCD for 2nd transfer. (Zero means none.)
5. CST4 = address of command chaining receiver. (Zero means none.)
6. P:CST5 contains a busy bit for each disk device attached to the channel. This bit indicates that a seek is in progress. For example, on the Sigma 3, if there is a seek in progress on device E1, bit 1 of P:CST5 is set for channel 6.
7. CST6 holds the contents of CST1 when background command chaining is suspended because of checkpoint.
8. CST8 is set by Q:LOADC to -N, where N represents the allowable channel time limit for this device. Once per second this value is incremented by the counter-one-equals-zero routine, and if the time limit has been exceeded (i.e., the count reaches zero) an HIO is sent to the offending device and unusual end condition flags are set in the associated file control table. In addition, if the operation specified an AIO Receiver, the receiver will be entered at this time.
9. The following is an example of testing channel activity.

```
LDX    B:CHAN        (channel number)
LDX    *P:CST0, 1    (CST Index Value)
LDA    *P:CST1, 1    (A = 0, inactive)
```

Figure 29. Channel Status Table



Notes: 1. This table is indexed by device-file number.

2. The abbreviations used in this table are explained below:

F indicates whether or not the file is active (F = 1, if active file; F = 0, if inactive file). F is set by Q:LOADC or Q:RADLIO and is reset by M:READ/M:WRITE/M:CTRL/M:IOEX. (Refer to RAD I/O routines within the Monitor for mnemonics of the form Q:RAD.)

R indicates whether or not the file is RAD (R = 1, if RAD file; R = 0, if non-RAD file). R is set by SYSGEN.

A indicates whether or not the device-file number has been assigned to a RAD file (A = 1, if assigned to RAD file; A = 0; if not assigned to RAD file). A is set by M:ASSIGN/M:DEFINE and is reset by M:CLOSE. A is only meaningful if R = 1.

B indicates whether file is background (B = 1), foreground (B = 0), or RBM (B = 0). B is set by SYSGEN.

[†] The two-character mnemonic used in the SYSGEN deck to define the Logical Device DFN is stored in P:FCT7 for that DFN; e.g., LP, L1, or even LD. However, all Logical Device DFNs will have the mnemonic 'LD' stored in P:DTT1.

Figure 30. File Control Table

Q indicates whether or not the AIO receiver is operating when data chaining on zero byte count (Q = 1, if yes; Q = 0, if no). This is used only for M:IOEX.

S indicates that the file may be processed through a "shared" blocking buffer if such is warranted by the Task Control Block.

DT is the five-bit Device Type Table Index (DTTX). DT is set by SYSGEN for non-RAD files and by M:ASSIGN/M:DEFINE for disk files.

P indicates the presence of end action (P = 1, if end action pending on current I/O operation; P = 0, if no end action). P is set by I/O interrupt task or Q:RADLIO and is reset by M:READ/M:WRITE/M:CTRL/M:IOEX.

AIS is the "attention interrupt status" used to control use of the remote terminal ring response interrupt. The possible states are:

0 disarmed – any ring interrupt is ignored.

1 armed – a ring interrupt causes bit 15 of R:RBM (X:ATTN) to be set, the RBM control task to be triggered, and AIS to be advanced to 2.

2 waiting – RBM control subtask S:ATTN transmits order to turn data terminal ready signal on, links to foreground receiver ATTNRXR if other than DFN1, and AIS is advanced to 3.

3 active – remote connection has been made. If TDV indicates loss of "carrier detect" status, S:ATTN will transmit order to disarm ring indicator interrupt and will transmit order to turn data terminal ready signal off.

Max Retry is the maximum number of retries to attempt on transmission errors. Max retry is device specific, but is set to zero if no error recovery is specified in the M:READ/M:WRITE argument list.

DV is used to indicate whether a device is; available to Background and Foreground (00), reserved to foreground (01), reserved to a specific foreground task (10), or down (11). I/O may not be performed on a down device unless bit 7 of the request order word is a 1; otherwise, device-unavailable status is returned. Similarly, I/O may not be performed on an "up" device unless bit 7 of the request order word is a zero. If DV = 11 and bit 7 of the request order word is one, the background program may use a foreground DFN.

Number of Retries is the number of retries attempted on the current I/O operation. Number of retries is set/reset by M:READ/M:WRITE.

AIO Device Status Byte is the byte returned from the device when an AIO instruction is executed. Device status byte is set by the I/O interrupt task or Q:RADLIO, Q:RADLWP, Q:RADBOT, Q:RADEOF, or Q:RADEOT.

M is set to 1 if the device was in manual mode, or nonoperational when the SIO was issued. This bit is maintained with the "AIO" DSB (Device Status Byte) if the device times out.

Figure 30. File Control Table (cont.)

Device Number is the hexadecimal number assigned to a peripheral device. Device number is set by SYSGEN for non-RAD files and by M:ASSIGN/M:DEFINE for RAD files.

Operational Status Byte is the byte returned from the device at the conclusion of an I/O operation (i.e., channel end). This is set by the I/O interrupt task or Q:RADLIO.

C indicates whether bits 8–15 of the even I/O channel register were all zeros (C = 0 if yes; C = 1 if no).

T indicates whether the last I/O operation on the device was timed out (T = 1 if yes; T = 0 if no).

O1 AIO overflow indicator.[t]

C1 AIO carry indicator.[t]

O2 TIO overflow indicator.[t]

C2 TIO carry indicator.[t]

O3 TDV overflow indicator.[t]

C3 TDV carry indicator.[t]

E indicates if there are parity errors (E = 1, if there are parity errors on the write operation, or memory parity or bad punches on a read operation; E = 0, if there are no parity errors). E is set by the I/O interrupt task.

Byte Count Residue is the number of bytes not transferred in the I/O operation. This is set by the I/O interrupt task. Note: Bits 1 and 2 of FCT5 will reflect the settings of the data chain and interrupt flags by the last I/O operation on the channel.

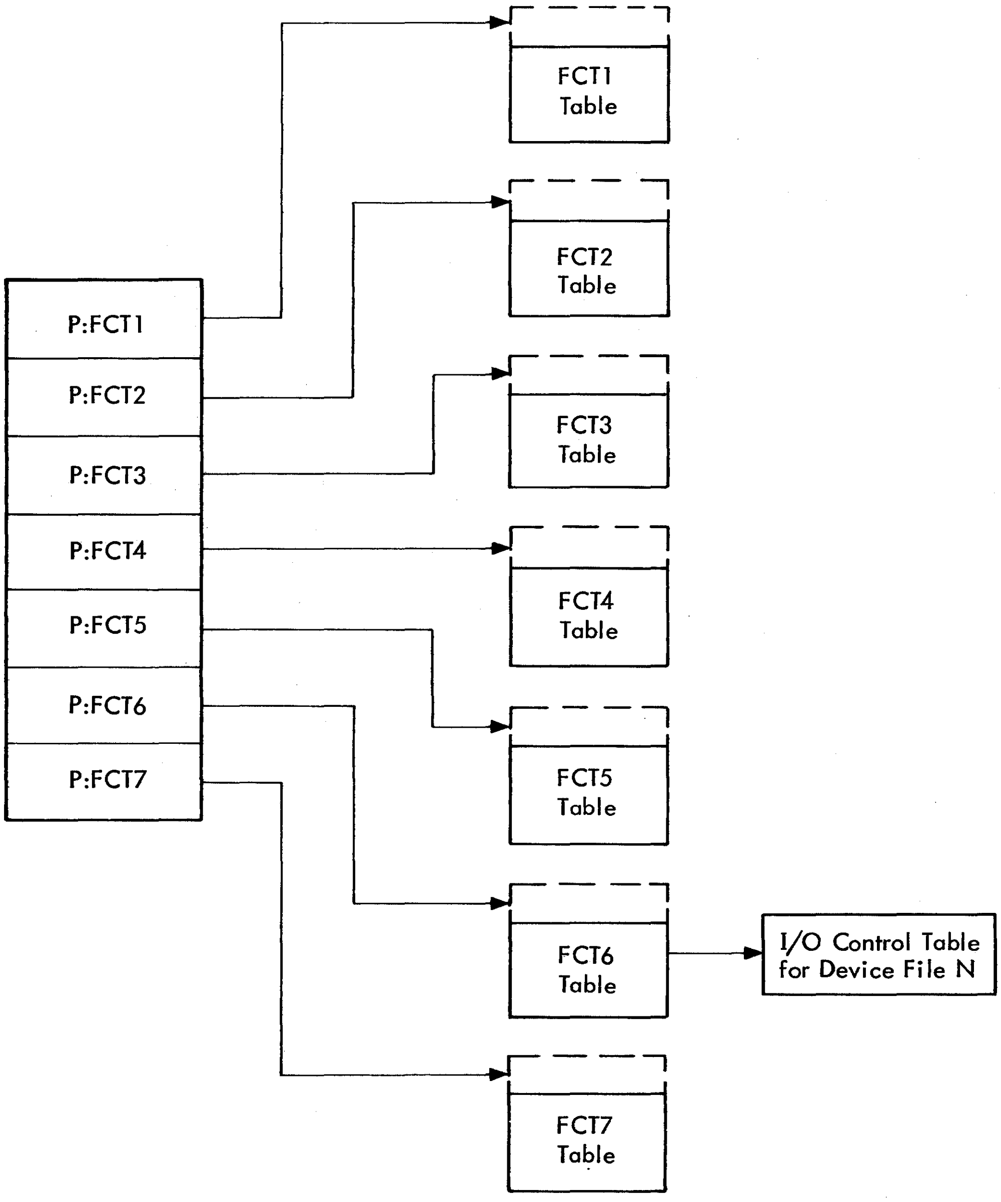
Address of I/O Control Table (IOCT) is the core address of the IOCT entry associated with this file. This is set by SYSGEN.

TIO and TDV Device Status Bytes are the status bytes returned from the device when TIO and TDV instructions are executed at I/O completion time.

When a No-Wait M:CTRL operation is performed for RAD files TIO DSB = FF.

[t] Overflow and carry status at completion of the last I/O operation.

Figure 30. File Control Table (cont.)



Note: The following is used as an index into the I/O control tables:

```
LDX      B:IOEX2        (device-file number)
LDX      *P:FCT6,1
```

For example, LDA 2,1

Figure 31. Storage Allocation of File Control Tables

The general setup after M:READ/M:WRITE is as follows:

NO COMMAND CHAINING

Address	Contents	
0, 1	$+2	
1, 1	(*P:DTTB)	
2, 1	0	Order Byte
3, 1	User Buffer Address	
4, 1	(*P:DTTD) + BC	
	0 7	8 15

where BC is the user byte count.

COMMAND CHAINING

Address	Contents	
0, 1	$+3	
1, 1	(*P:DTTB)	
2, 1	0	
3, 1	0	Order Byte
4, 1	User Buffer Address	
5, 1	(*P:DTTD) + BC	
	0 7	8 15

After pre-I/O edit, the setup of the tables depends on the device type:

LINE PRINTER (3451, 7440, 7441, 7445)
Format Byte = Ax, Bx, Dx or Ex (Format, then print)

Address	Contents
0, 1	$+3
1, 1	A002
2, 1	$+2
3, 1	0300 + Format Byte
4, 1	$+2
5, 1	4002
6, 1	4560
7, 1	User Text Address
8, 1	X'2000' + (User byte count −1)
	0 15

LINE PRINTER (3451, 7440, 7441, 7445)
Format Byte = 8x or 9x (Print, then format)

Address	Contents
0, 1	$+3
1, 1	X'C002'
2, 1	$+4
3, 1	X'0560'
4, 1	User Text Address
5, 1	X'2000' + (User Byte Count −1)
6, 1	$+2
7, 1	X'2002'
8, 1	X'4300' + Format Byte
	0 15

LINE PRINTER (3451, 7440, 7441, 7445)
Format Byte = 60, Cx, Fx

Address	Contents
0, 1	$+2
1, 1	X'4002'
2, 1	X'4500' + Format Byte
3, 1	User Text Address
4, 1	X'2000' + (User Byte Count −1)
	0 15

LINE PRINTER (3451, 7440, 7441, 7445)
User Byte Count = 0 or 1 (Format Only)

Address	Contents
0, 1	$+2
1, 1	X'2002'
2, 1	X'4300' + Format Byte
	0 15

LINE PRINTER (7446, 346x)
Format Byte = 60, Ax, Bx, Cx, Dx, Ex, Fx

Address	Contents
0, 1	$+2
1, 1	X'4002'
2, 1	X'4500' + Format Byte
3, 1	User Text Address
4, 1	X'2000' + (User Byte Count −1)
	0 15

LINE PRINTER (7446, 346x)
User Byte Count = 0 or 1 (Format Only)

Address	Contents
0, 1	$+2
1, 1	X'2002'
2, 1	X'4300' + Format Byte
	0 15

Figure 32. Non-RAD I/O Control Tables

LINE PRINTER (7446, 346x)
Format Byte = 8x, 9x (Print, then Format)

Address	Contents
0, 1	$+3
1, 1	X'C002'
2, 1	$+4
3, 1	X'0560'
4, 1	User Text Address
5, 1	X'2000' + (User Byte Count -1)
6, 1	$+2
7, 1	X'2002'
8, 1	X'4300' + Format Byte

0 15

LINE PRINTER (7450)
Format Byte = 60, Ax, Cx, Ex, Fx

Address	Contents
0, 1	$+3
1, 1	X'C002'
2, 1	$+4
3, 1	X'0500' + Format Byte
4, 1	User Text Address
5, 1	X'2000' + (User Byte Count -1)
6, 1	$+2
7, 1	X'4002'
8, 1	X'4500' + Format Byte
9, 1	User Text Address
10, 1	X'2000' + (User Byte Count -1)

0 15

LINE PRINTER (7450)
User Byte Count = 0 or 1 (Format Only)

Address	Contents
0, 1	$+2
1, 1	X'2002'
2, 1	X'4300' + Format Byte

0 15

PAPER TAPE (Read Binary)

Address	Contents
0, 1	$+3
1, 1	X'4001'
2, 1	0
3, 1	X'0082'
4, 1	User Buffer Address
5, 1	X'2000' + BC
6, 1	0
7, 1	-1
8, 1	-1

0 15

LINE PRINTER (7450)
Format Byte = Bx or Dx (Format, then Print)

Address	Contents
0, 1	$+3
1, 1	X'A002'
2, 1	$+2
3, 1	X'0300' + Format Byte
4, 1	$+3
5, 1	X'C002'
6, 1	$+4
7, 1	X'0560'
8, 1	User Text Address
9, 1	X'2000' + (User Byte Count -1)
10, 1	$+2
11, 1	X'4002'
12, 1	X'4560'
13, 1	User Text Address
14, 1	X'2000' + (User Byte Count -1)

0 15

LINE PRINTER (7450)
Format Byte = 8x or 9x (Print, then Format)

Address	Contents
0, 1	$+3
1, 1	X'C002'
2, 1	$+4
3, 1	X'0560'
4, 1	User Text Address
5, 1	X'2000' + (User Byte Count -1)
6, 1	$+3
7, 1	X'C002'
8, 1	$+4
9, 1	X'0560'
10, 1	User Text Address
11, 1	X'2000' + (User Byte Count -1)
12, 1	$+2
13, 1	X'2002'
14, 1	X'4300' + Format Byte

0 15

PAPER TAPE (Write Binary)

Address	Contents
0, 1	$+3
1, 1	X'4001'
2, 1	0
3, 1	X'0001'
4, 1	User Buffer Address
5, 1	X'2000' + BC
6, 1	0
7, 1	-1
8, 1	-1

0 15

Figure 32. Non-RAD I/O Control Tables (cont.)

KEYBOARD/PRINTER - Model 7012 (Auto. Input)

Address	Contents	
0, 1	$+3	
1, 1	X'A002'	
2, 1	$-2	
3, 1	X'06'	Data
4, 1	User Buffer Address	
5, 1	BC	
6, 1	Actual Byte Count (0)	
7, 1	X'0082'	
8, 1	0	
	0 7	8 15

KEYBOARD/PRINTER - Model 419x (Auto. Input)

Address	Contents
0, 1	$+3
1, 1	X'A002'
2, 1	$+2
3, 1	X'0500' + PROMPT (cell X'FC')
4, 1 (0,1) ⋮ ⋮	Same as 0 through 7 for 7012 Keyboard/Printer Automatic Input
12,1 (8,1)	$+2
13,1 (9,1)	X'2002'
14,1 (10,1)	X'050D'
	0 15

PAPER TAPE (Read Automatic)

Same as Keyboard/Printer (Automatic Input) except:

Address	Contents
3, 1	X'02' Data (8 - 15)
	0 15

KEYBOARD/PRINTER (Binary Input)

Same as Paper Tape (Write Binary) except:

Address	Contents
3, 1	X'0006'
	0 15

LOGICAL DEVICE

Address	Contents
0, 1	$+2
1, 1	Device Type Table B Entry
2, 1	Pseudo-order Byte
3, 1	User Buffer Address from B:IOCD
4, 1	User Byte Count from B:IOCD + 1
5, 1	Write Order Byte from Last Completed I/O operation
6, 1	0
	0 15

KEYBOARD/PRINTER - Model 7012 (EBCDIC Out.)

Address	Contents
0, 1	$+3
1, 1	X'C001' or X'C002'†
2, 1	$+4
3, 1	X'0005' or X'0505'†
4, 1	User Buffer Address + w
5, 1	X'2000' + BC - TB - 1
6, 1	$+2
7, 1	X'2002'
8, 1	X'0515'
	0 15

where

TB = number of trailing blanks.

w = 1 if byte count is odd.

w = 0 if byte count is even.

KEYBOARD/PRINTER - 419x (EBCDIC Output)

Same as Keyboard/Printer 7012 0 through 8 except:

Address	Contents
7, 1	X'2003'
8, 1	X'0005'
9, 1	X'150D'
	0 15

PAPER TAPE (Write EBCDIC)

Same as Keyboard/Printer (EBCDIC Output) except:

Address	Contents
3, 1	X'0001'
	0 15

KEYBOARD/PRINTER (Binary Output)

Same as Paper Tape (Write Binary) except:

Address	Contents
3, 1	X'0005'
	0 15

CARD READER

Address	Contents
0, 1	$+2
1, 1	X'4001'
2, 1	X'000E' or X'000A'
3, 1	User Buffer Address
4, 1	X'2000' + BC
	0 15

†Use X'C001' and X'0005' if format is single space; use X'C002' and X'0505' if format is double space.

Figure 32. Non-RAD I/O Control Tables (cont.)

MAGNETIC TAPE (3xxx – 9 track)

Address	Contents	
0, 1	$+3	
1, 1	X'C001'	
2, 1	$+4	
3, 1	XX	Order Byte
4, 1	User Buffer Address	
5, 1	X'2000' + BC	
6, 1	$+12	
7, 1	X'2010'	
8, 1	SYSGEN Mode Byte	Assign Mode Byte
9, 1	X'00FF'	
10, 1	0	
11, 1	$+2	
12, 1	28	
13, 1	X'9600'	
14, 1	0	
15, 1	0	
16, 1	0	
17, 1	DFN	I/O Address
18, 1	X'04'	Sense Byte 0
19, 1	Sense Byte 1	Sense Byte 2
: :	: :	: :
25, 1	Sense Byte 13	Sense Byte 14
	0 7	8 15

MAGNETIC TAPE (7xxx – 9 track)

Address	Contents	
0, 1	$+3	
1, 1	X'C001'	
2, 1	$+4	
3, 1	XX	Order Byte
4, 1	User Buffer Address	
5, 1	X'2000' + BC	
6, 1	$+2	
7, 1	X'2002'	
8, 1	X'04'	Sense Byte 0
	0 7	8 15

MAGNETIC TAPE (7xxx – 7 track)

Address	Contents	
0, 1	$+3	
1, 1	X'C001'	
2, 1	0	
3, 1		Order Byte
4, 1	User Buffer Address	
5, 1	X'2000' + BC	
	0 7	8 15

CARD PUNCH[†] (Model 7160)
(300 cards per minute)

Address	Contents
0, 1	$+3
1, 1	X'A000' + 81 (or + 121)
2, 1	$ – 2
3, 1	X'09' or X'0D'
4, 1 : 63, 1	80 or 120 Bytes of Data (current card)
64, 1	$ + 3
65, 1	X'A000' + 81 (or + 121)
66, 1	$ – 2
67, 1	X'19' or X'1D' (Previous Order Byte + X'10')
68, 1 : 127, 1	Previous Card Image
	0 15

CARD PUNCH (Model 7165)
(100 cards per minute)

Address	Contents	
0, 1	$+2	
1, 1	X'4001'	
2, 1	0	Order Byte
3, 1	User Buffer Address	
4, 1	X'2000' + BC	
	0 7	8 15

[†] The card punch table is very long because error recovery on the card punch requires the previous card image.

Figure 32. Non-RAD I/O Control Tables (cont.)

0	File Format Byte	F	W	S	STATE	Order Byte	
1	LRZ logical record size or granule size, in words						
2	BOT						
3	EOF						
4	EOT						
5	PRAD, Pointer to Current RAD Address						
6	Temporary Storage[t]						
7	BBA (blocking buffer address)						
8	BBP						
9	Temporary Storage						
10	Remaining word count						
11	Word count for current I/O						
12	Buffer address for current I/O						
13	Return address to file management routine (-1 for return to M:CTRL)						
14	PLR (bits 0-15) or SR (bit 15)						SR
	0 4	5	6	7	8 10	11 14	15

Note: The abbreviations used in this table are explained below:

File Format Byte is

X	X	X	WP
0	1	2	3 4

where the following values of XXX specify the indicated conditions and modes:

Value	Format	Condition[tt]	Mode
000	Unblocked	N/A	sequential only
001	Blocked	inactive	sequential only
010	Compressed	inactive	sequential only
011	Blocked (Packed)	inactive	random and sequential
100	Random	N/A	random and sequential
101	Blocked	Device access pending	N/A
110	Compressed	Device access pending	N/A
111	Blocked (Packed)	Device access pending	N/A

WP indicates the write-protection status (WP=11, if write is permitted only when K:TCB equals T:RBMTCB, unless SY is keyed in; WP=10, if write is permitted only when K:TCB does not equal K:BACKP, unless SY is keyed in; WP=01, if write is permitted only when K:TCB equals K:BACKP, unless SY is keyed in; WP=00, if there is no write protection).

F if set, indicates incorrect length on the last transfer.

W indicates the status of data in the blocking buffer (W = 1, if data has been written in the blocking buffer that has not been written on the disk; W = 0, if data in the blocking buffer has already been output on the disk).

S indicates whether or not the sector addressed by PRAD is currently in the blocking buffer (S = 1, if it is; S = 0, if it is not).

[t] Record number for blocked and packed file formats.

[tt] See RAD File Management, Chapter 3.

Figure 33. RAD I/O Control Table

STATE

0	SEEK to READ FLAWED HEADER
2	READ FLAWED HEADER
4	SEEK FOR REQUESTED OPERATION
5	RESTORE
6	PERFORM REQUESTED OPERATION
7	HEADER READ FOLLOWING RESTORE

Order Byte is the actual order byte for the last operation.

Logical Record Size is the number of words in a logical record or granule.

BOT is the absolute RAD address of the first sector defined for the file.

EOF is the pointer to the logical file mark. If EOF = -1, a logical file mark has not been written. For Unblocked, Random, or Compressed files, EOF is the absolute RAD address of the logical file mark; otherwise (for Blocked files) EOF is the count of the number of logical records that precede the logical file mark.

EOT is the absolute RAD address of the last sector plus one defined for the file.

PRAD is the pointer to the current absolute RAD address of the file and is initially set to BOT by M:ASSIGN/M:DEFINE. Complications may arise in the unauthorized manipulation of this pointer, especially in a mixed RAD system.

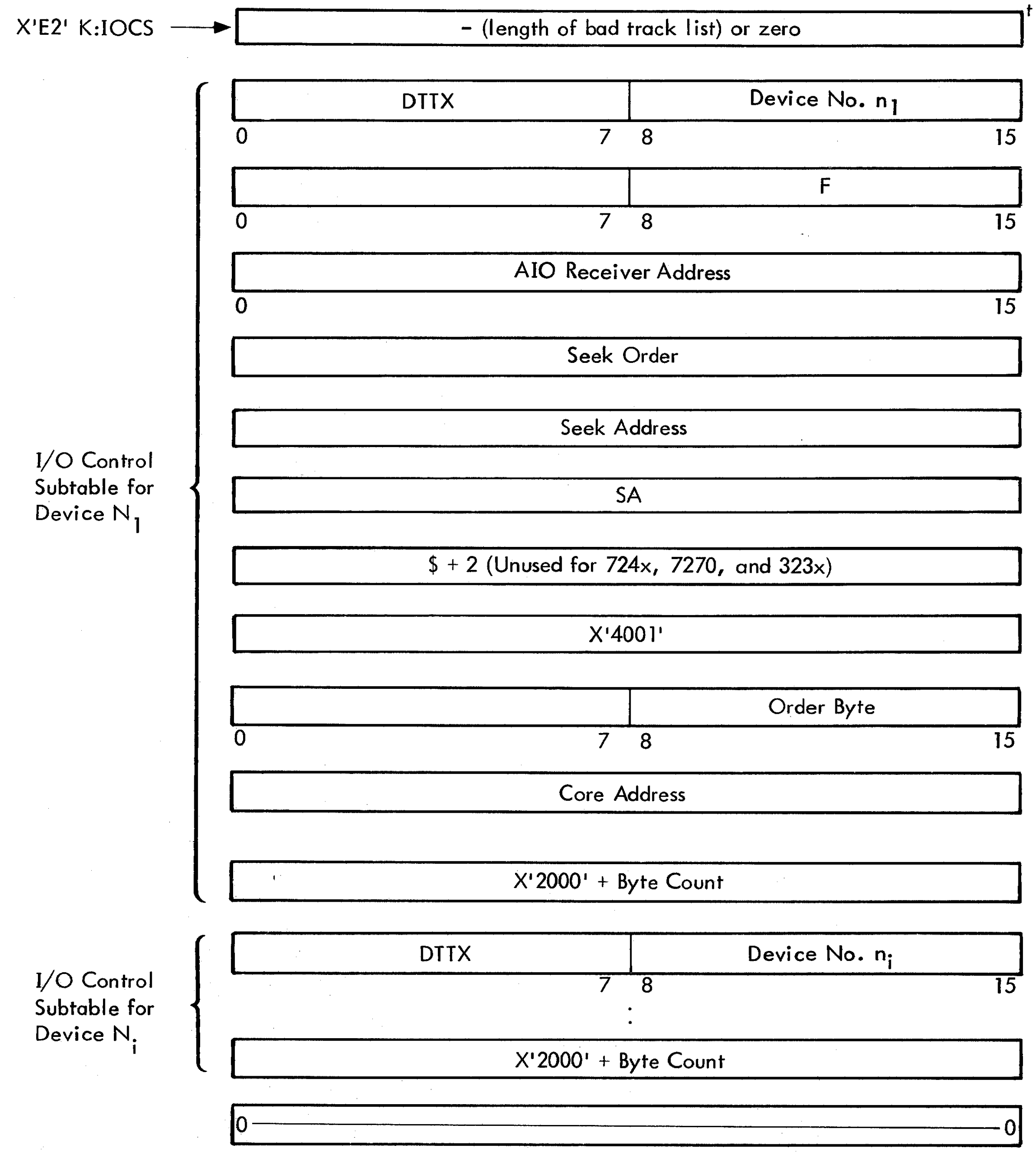
BBA is the core address of the blocking buffer assigned to the file or zero. The pointer is initially set to zero by M:ASSIGN/M:DEFINE.

BBP this pointer contains the address, within the blocking buffer, where the file is currently positioned. The pointer is initially set to BBA by M:ASSIGN/M:DEFINE.

PLR is the address of the FORTRAN-associated variable. Meaningful only for random mode files.

SR is the short record flag (see RAD File Directory).

Figure 33. RAD I/O Control Table (cont.)



where

DTTX is the Device Type Table index.

Device No. n_i is the hardware device address for which this table is dedicated. One table is used for each rotating device. As many as 12 unique rotating devices may be defined.

F is zero for disk devices. For disk devices, F is the Device File Number of the file using the device during the seek operation. At the conclusion of the seek operation, the DFN is moved to the channel status tables.

Seek Order is X'83' for Model 724x, 7270, and 323x disk devices; for all other disk devices use 3.

SA is $+2 for 7202/3/4, 7251/2, and 7232 devices and is loaded into the even channel register during the command chaining procedure. SA is the second 2 bytes of the seek address for all other devices.

[†] See Disk Pack alternate Track Handling Section at the end of this chapter.

Figure 34. Disk I/O Control Subtable

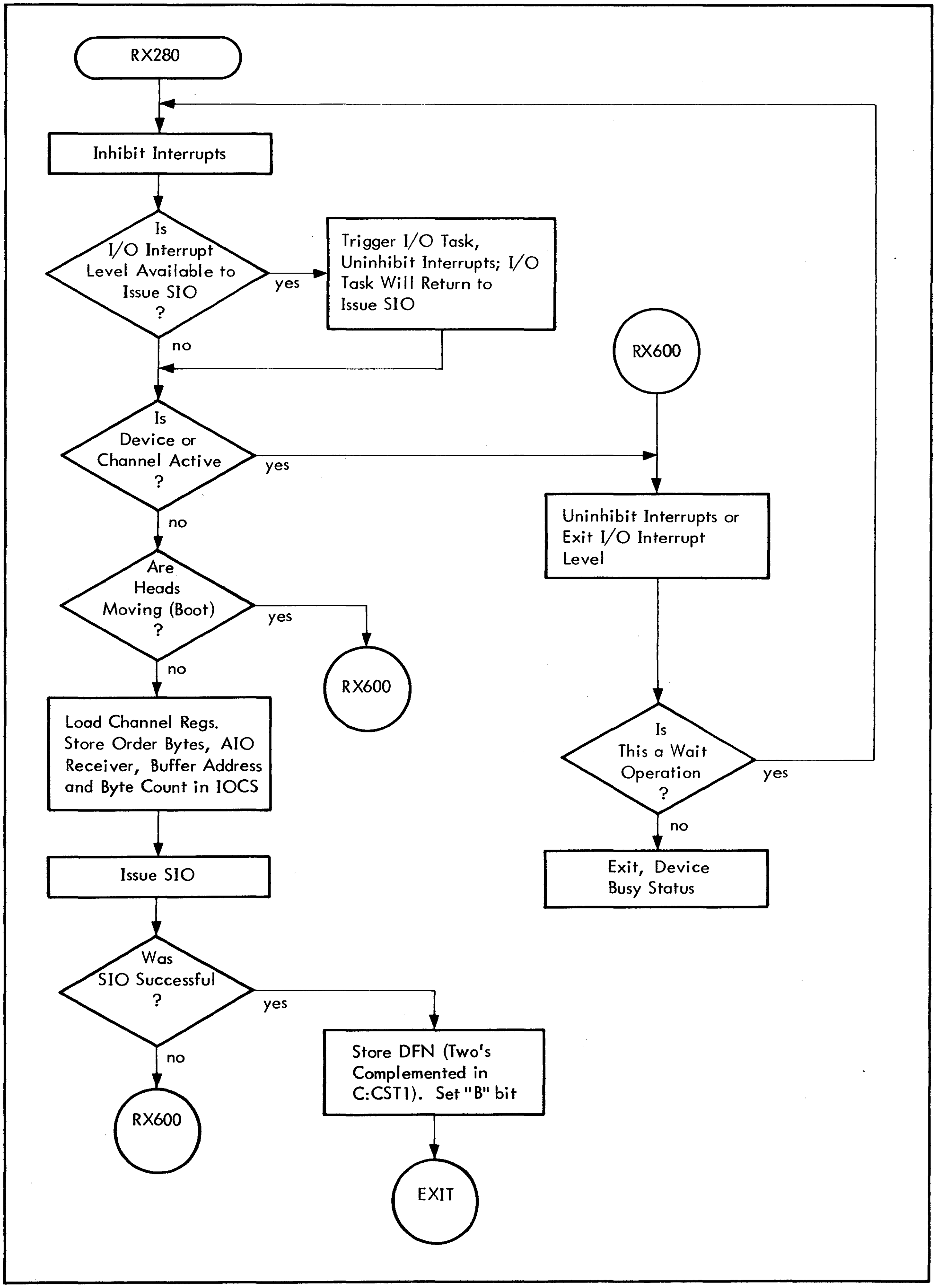
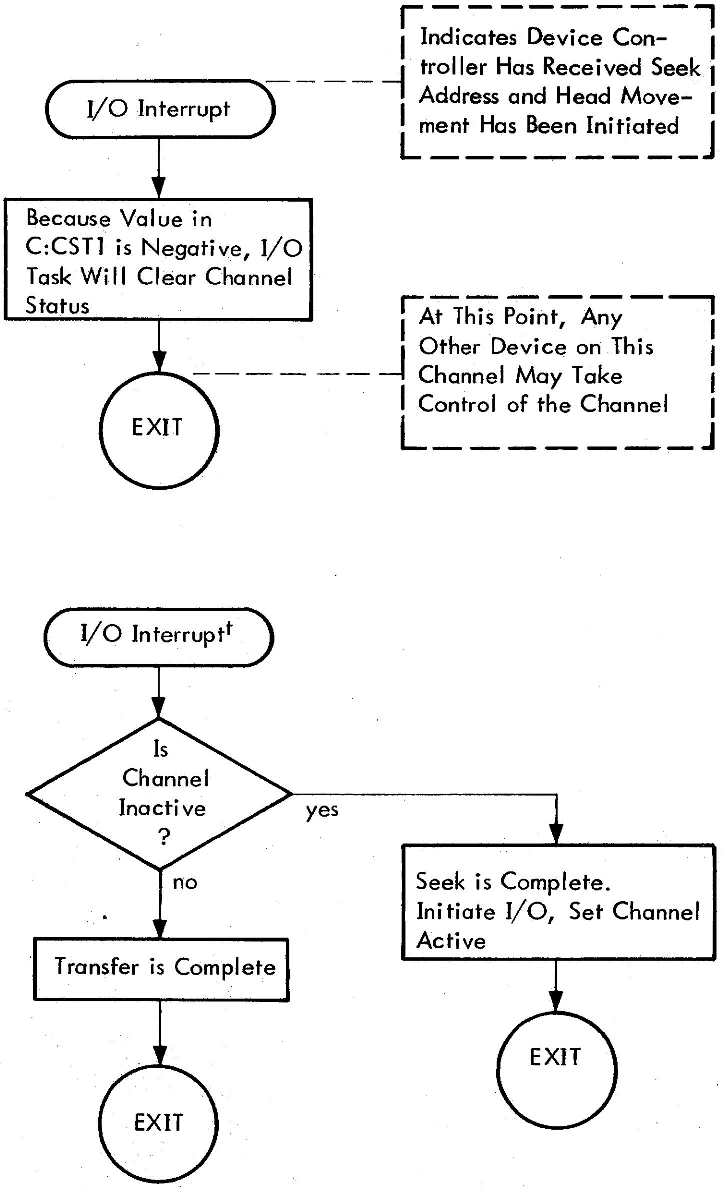

RX280
Inhibit Interrupts
Is I/O Interrupt Level Available to Issue SIO ?
yes
Trigger I/O Task, Uninhibit Interrupts; I/O Task Will Return to Issue SIO
no
Is Device or Channel Active ?
yes
no
Are Heads Moving (Boot) ?
yes
RX600
no
Load Channel Regs. Store Order Bytes, AIO Receiver, Buffer Address and Byte Count in IOCS
Issue SIO
Was SIO Successful ?
yes
no
RX600
Store DFN (Two's Complemented in C:CST1). Set "B" bit
EXIT
RX600
Uninhibit Interrupts or Exit I/O Interrupt Level
Is This a Wait Operation ?
yes
no
Exit, Device Busy Status

Figure 35. Disk Pack Seek Overlap Flow


I/O Interrupt
Indicates Device Controller Has Received Seek Address and Head Movement Has Been Initiated
Because Value in C:CST1 is Negative, I/O Task Will Clear Channel Status
At This Point, Any Other Device on This Channel May Take Control of the Channel
EXIT
I/O Interrupt[t]
Is Channel Inactive ?
yes
no
Seek is Complete. Initiate I/O, Set Channel Active
Transfer is Complete
EXIT
EXIT

[t] This interrupt will occur twice for each transfer on the disk; once when the heads have completed their movement, and once when the transfer is complete.

Figure 35. Disk Pack Seek Overlap Flow (cont.)

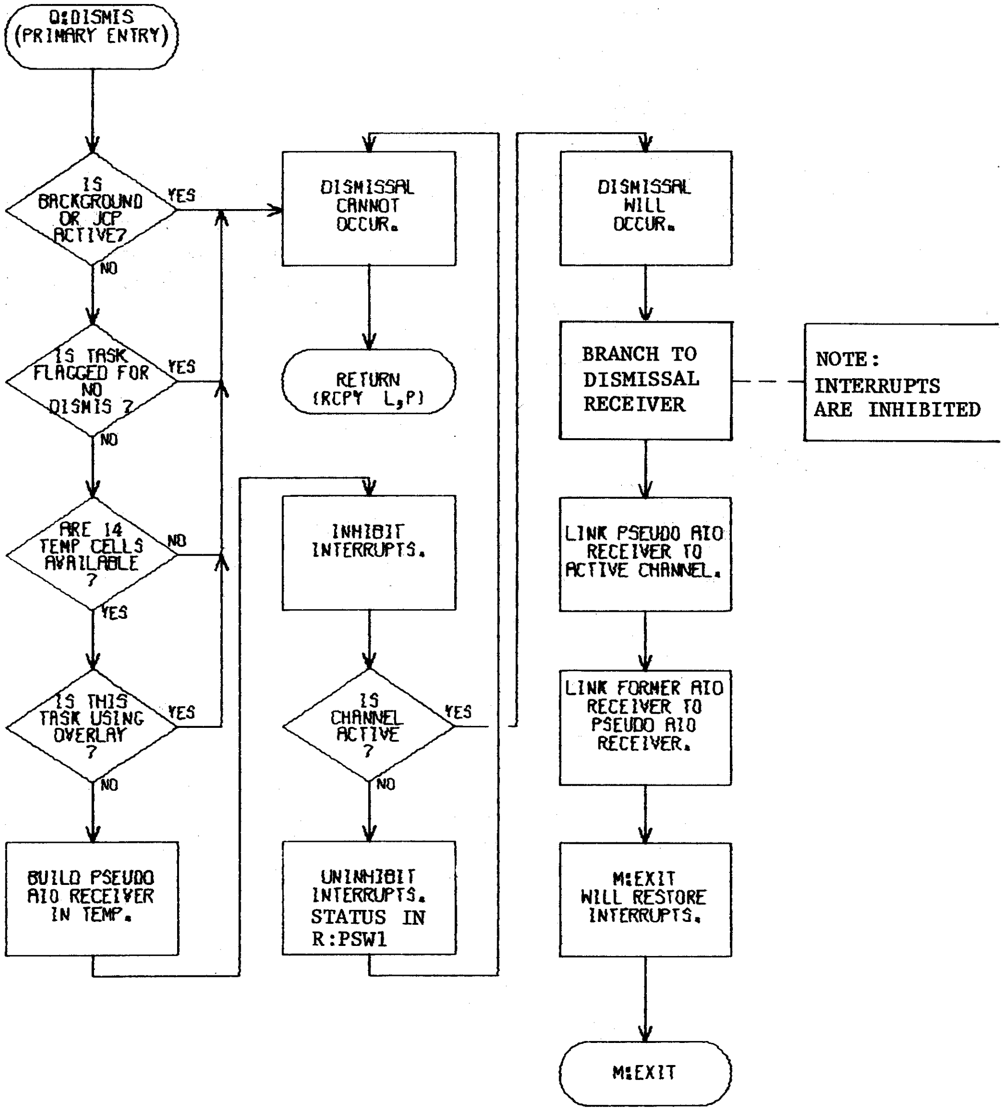

Q:DISMIS (PRIMARY ENTRY)
IS BACKGROUND OR JCP ACTIVE?
YES
NO
IS TASK FLAGGED FOR NO DISMIS ?
YES
NO
ARE 14 TEMP CELLS AVAILABLE ?
NO
YES
IS THIS TASK USING OVERLAY ?
YES
NO
BUILD PSEUDO AIO RECEIVER IN TEMP.
DISMISSAL CANNOT OCCUR.
RETURN (RCPY L,P)
INHIBIT INTERRUPTS.
IS CHANNEL ACTIVE ?
YES
NO
UNINHIBIT INTERRUPTS. STATUS IN R:PSW1
DISMISSAL WILL OCCUR.
BRANCH TO DISMISSAL RECEIVER
NOTE: INTERRUPTS ARE INHIBITED
LINK PSEUDO AIO RECEIVER TO ACTIVE CHANNEL.
LINK FORMER AIO RECEIVER TO PSEUDO AIO RECEIVER.
M:EXIT WILL RESTORE INTERRUPTS.
M:EXIT

Figure 36. Dismissal Routine (Optional)

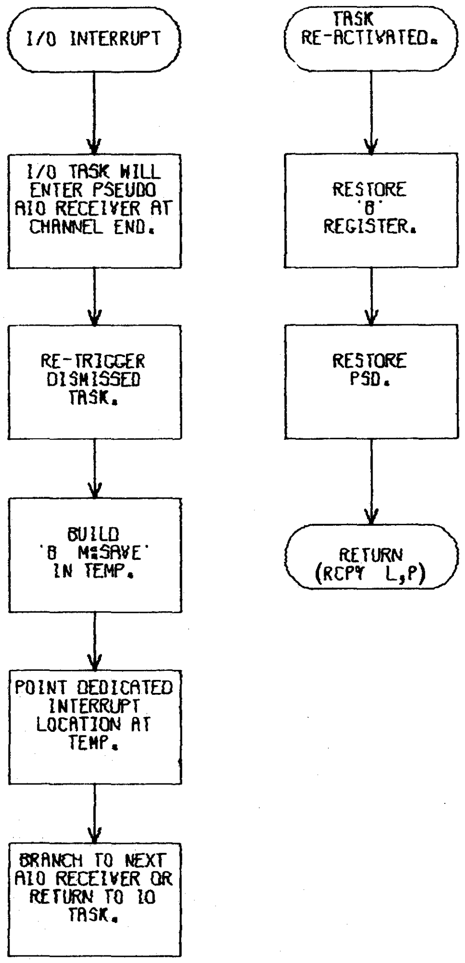

I/O INTERRUPT
I/O TASK WILL ENTER PSEUDO AIO RECEIVER AT CHANNEL END.
RE-TRIGGER DISMISSED TASK.
BUILD 'B M:SAVE' IN TEMP.
POINT DEDICATED INTERRUPT LOCATION AT TEMP.
BRANCH TO NEXT AIO RECEIVER OR RETURN TO IO TASK.
TASK RE-ACTIVATED.
RESTORE 'B' REGISTER.
RESTORE PSD.
RETURN (RCPY L,P)

Figure 36. Dismissal Routine (Optional) (cont.)

The task dismissal feature is used for I/O requests that must wait for access to a channel that is busy; that is, an I/O request to a busy channel is queued, followed by task dismissal. The queue is serviced automatically in priority order on the queued channel upon I/O completion. The task dismissal code is entered by M:READ/M:WRITE at any dismissal opportunity via location V:DISMIS. This location points either to the dismissal routine or to an RCPY L,P instruction (determined at SYSGEN).

Dismissal is disabled during the disk-boot sequence to avoid conflicts resulting from the loading of resident foreground routines. In addition, a 'no dismissal' flag in a task's TCB will inhibit dismissal if desired.

Disk Pack Flawed Track Handling – Models 7242/46

The tracks on cylinder's 200, 201, and 202 of the 7242 disk packs may be used as alternates for faulty tracks on the device. The track substitution is made using RADEDIT !#GDTRACKS command. The !#BDTRACKS command locates the next available alternate track and writes new headers on it and the faulty track, as follows:

Byte	Faulty Track	Alternate Track
0	X'FF'	0
1	0	0
2	Faulty track cylinder	Alternate cylinder
3	Faulty track head	Alternate head
4	Sector	Sector
5	Alternate cylinder	Faulty track cylinder
6	Alternate head	Faulty track head
7	0	0

The !#GDTRACKS command places zero in bytes 0, 5, and 6 of the faulty track headers, and it sets the contents of bytes 5 and 6 of the alternate track header to X'FF'.

Whenever a Read or a Write is issued to the flawed track, a flawed track error condition will be returned. As all Reads and Writes are segmented initially to a maximum transfer size of a single track, there is no difficulty in merely repeating the last transfer with the alternate track instead of the flawed track. The determination that the track returned a "flawed header" status is made by Q:RADE, the RAD error recovery routine.

After the sensibility of a retry is established, Q:RADE returns to RX128 (the point where a simple retry is made) with the state code set to zero, indicating that a seek to the original track is to be followed by a header read.

When the seek is complete, the header read is formatted in the IOCS by the disk pack seek interrupt handler to read the header into the IOCS. The state code is then set to 2, indicating that a seek to the track described by the alternate cylinder and head is to be formatted and issued when the header read is complete (by the disk pack command chaining receiver). The state code is set to 4 at the completion of issuing the seek to the alternate track. The remainder of the operation proceeds as if the seek were to the original track.

Disk Pack Alternate Track Handling – Models 7251/52 and 3231/32/33

The tracks in cylinders 400–407 of the 3232 disk packs and cylinders 200–203 of the disk packs model 7252, may be used as alternates for faulty tracks on the device. A record of faulty tracks is kept on cylinder 0, track 0, sector 2, in a bad track list (BTL). The BTL is created and modified using the RADEDIT !#GDTRACK and !#BDTRACK commands. The M (Mount) keyin will read the BTL into the system tables and the R (Remove "ALL") will delete the BTL for the specified device from the system tables. The bad track list in core precedes the IOCS in memory and is located via K:IOCS which points at the cell following the bad track list. This cell contains – (length of the bad track list).

The format of the BTL in core is as follows:

Word	Bits	Meaning	
0	0-7	n = number of alternates on the device + 2.	
0	8-15	d = device number.	
1	0-15	equals -1 if BTL is in core; equals 0 if not.	
2 to n-1	0-15	Bad track list entry containing	
		Value	Significance
		-1	Corresponding alternate track has never been assigned.
		-2	Corresponding alternate track has been assigned to a bad track and has subsequently been unassigned by a !#GDTRACK command.
		t	Corresponding alternate track has been assigned to replace track t by a !#BDTRACK command.
n	0-15	Same as word 0 for the next device or negative if there are no more devices.	

The bad track list is written on the device without word 0 of the core version.

It is necessary before initializing a disk pack to write the BTL on sector 2 using a !#GDTRACK or !#BDTRACK command. The next step is to key in M dn, BTL in order to read the bad track list into core. Then the !#INITIALIZE command can be used to construct areas on the device.

In bringing up a system, a startup deck is required to initialize the disk packs and their bad track lists. It is most likely that the disk packs which usually are not removed or cannot be removed will have had the BTL in sector 2 destroyed. A record must be kept of the bad tracks for each device.

M:RSVP

The M:RSVP service routine allows foreground tasks to acquire a physical device for exclusive use or for use by only foreground tasks. This is accomplished by flagging the FCTs pointing to that device as "reserved" for foreground use.

A device may be reserved at two different levels. The first level is the same as for pre-G00 RBM reserves. The device is not reserved to a particular task, but only foreground tasks may access it. The second level (new in G00 RBM) is an exclusive reserve. At this level only the task whose dedicated interrupt location (DIL) was active at the time of the reserve call to M:RSVP is allowed to access the device (until such time as that task, M:TERM/M:ABORT, or a "BP" keyin releases it).

When M:RSVP is called for a reserve request and the device has already been reserved, the caller may be queued if table space is available, the request was for exclusive use of the device, and the call specified a Reserve Complete Receiver (RXR).

When M:RSVP is called for a release request, the device is released to the next requestor (the background is the default lowest priority request) in priority order. Release requests always return a status of A=0.

M:RSVP treats reserve and release requests for logical devices as valid, but performs no operation on such requests. The status return is A=0 for both reserve and release of logical devices.

M:RSVP uses a single table with three words per entry (see Figure 37). The size of the table is set by SYSGEN and defaults to 5 entries. If RSVPTABL is set to zero entries at SYSGEN, M:RSVP will be omitted from the system and no table space will be allocated for it. Any calls to M:RSVP when RSVP is not part of the system will receive a status return of A=0 but with X=-1 (see Figure 38). The maximum size of the table is 255 entries. The table contains the DT index and Device Number in word 1 of the entry, flags and the DIL in word 2, and the RXR in word 3.

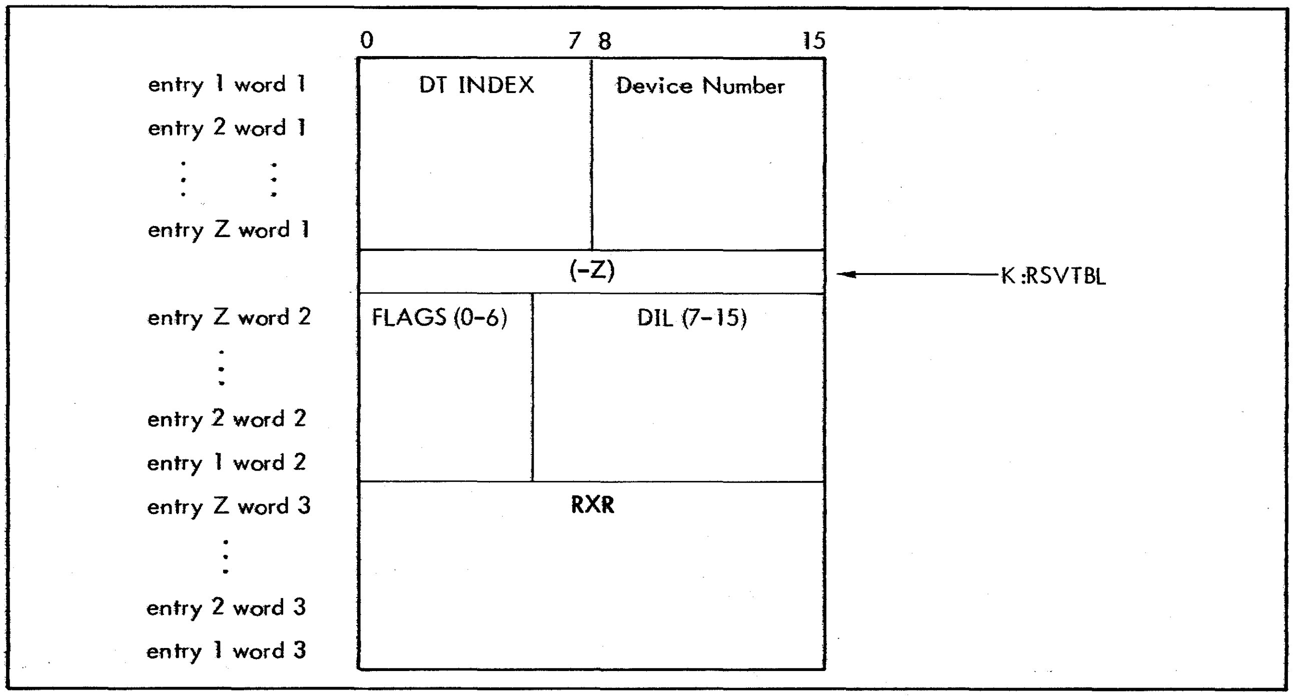


Figure 37. M:RSVP Table Format

Request	DEVICE STATUS (CURRENT)		
	Available	Reserved	
	Background Device	Standard	Exclusive
Reserve "exclusive"	Device becomes reserved exclusive mode, status return "A = 0".	Request is queued, status return "A = 3".	Request is queued, status return "A = 3".
Reserve "standard"	Device becomes reserved "non-exclusive" mode, status return "A = 0".	Device previously reserved, status return "A = -1".	Device previously reserved, status return "A = -1".
Explicit release	Status return "A = 0", no action.	Device is released from current task to the next task and status is returned "A = 0".	If DIL = table DIL, release the device to the next task and return status "A = 0".
BR keyin	"Key error" is output on OC.	Device is released to the next task, status return "A = 0".	Device is released to the next task, status return "A = 0".
Implicit (M:TERM/M:ABORT)	Status return "A = 0".	No action, status return "A = 0".	All devices reserved to the current DIL are released to the next user, return status "A = 0".

Figure 38. M:RSVP Decision Table

3. RAD FILE MANAGEMENT

Overview

As the central storage medium for RBM, the RAD is used for permanent storage of RBM and all related processors and for permanent or temporary storage of users' programs and data. The RBM RAD management scheme is flexible both for rapid storage and retrieval and for easy file maintenance.

The RAD is addressable in physical units called sectors. The most important unit of RAD storage is the file which consists of one or more contiguous sectors treated by RBM as a unit. The RAD file is bounded by the first sector address (BOT) and by the last sector address plus 1 (EOT).

A RAD file becomes accessible to RBM whenever an entry in the file control table is initialized to the boundaries of the file, which usually takes place as one of the functions of an assign or define process. In this sense, a RAD file is similar to a device, and the entire RAD can be considered as many devices.

All RAD files are either blocked or unblocked.

Blocked Files

Blocked files differ from unblocked files in that actual I/O is done in terms of blocks, rather than granules (which are synonymous with logical records) for unblocked files. (On the RAD, a physical record is a sector.)

A block is an increment of RAD space, the size of which is determined at SYSGEN. When a block is in memory, it resides in a blocking buffer.

A block may contain a partial logical record, one logical record, a number of logical records, or a number of logical records and one or two partial records. If a "no-wait" transfer operation references a record which crosses a block boundary and the second block (containing the remainder of the record) cannot be accessed because the RAD device is busy, the file is set to the "I/O Pending" status and "Device Busy" status is returned. The remainder of the record will be transferred on the retry operation.

Blocked files may have any one of the following formats. Only one logical record may be accessed with each call to M:READ/M:WRITE:

Compressed

File format equals 010, must be accessed sequentially. The special codes used for compressed files are as follows:

X'2600' Denoting end of sector.

X'EC00' Denoting end of record.

X'DC00' Denoting compression code, bits 6 through 15 contain the two's complement of the number of words containing X'4040' which are replaced by this code word.

For the sake of C00 capability, the following codes are also recognized.

X'2500'	Denoting end of record.
X'2700'	Denoting compression code. In this case bits 9-15 contain the number of subsequent blank words.

Implications of the above scheme are:

- The maximum record size is 2K-2 bytes.
- Single blanks will not be compressed.
- Only standard EBCDIC data should be compressed.

In addition. each write of a compressed record is followed by an implied EOF mark, which is overwritten on subsequent data writes.

Blocked

File format equals 001, must be accessed sequentially.

Packed

Format equals 011, may be accessed sequentially or randomly. Access mode is determined solely by the "R" bit of the M:READ/M:WRITE argument list (see the RBM/RT,BP Reference Manual, 90 10 37, "Monitor Service Routines" section, M:READ, M:WRITE).

"Packed" files differ from other blocked files in that about twice as many RAD accesses will be required for write operations in the random mode. Therefore, it is more economical to create files sequentially, when possible.

"Shared" Files

RBM permits packed files to use common blocking buffers under certain restrictions.

- The file must be in packed format and may only be accessed randomly.
- The file must only be accessed with the "wait" specification.
- The task using this file must be flagged as possessing "shared" files (via a !$BLOCK card).

- The file must itself be flagged as "shared" via M:ASSIGN or M:DEFINE calls.

- Non-"shared" files may not make use of the "shared" buffer.

If these restrictions are met and a user block is not identified, M:OPEN will allocate the last available blocking buffer for any requests for "shared" files.

In using such a buffer, input records are extracted from a block(s) newly read into the shared buffer regardless of its previous contents, and the buffer is set as clear (W,S = 0 in the IOCT) after each record is extracted. Output will always require the block(s) containing the record in question to be freshly read, updated, and then written to secondary memory before completing the output request and marking the buffer as clear.

Unblocked Files

Unblocked files are unique from blocked in that a RAD transfer is always required and an AIO receiver will always be acknowledged. "I/O Pending" condition does not apply to unblocked files. Unblocked files may have either of the following formats:

> "Unblocked", file format equals 000, may be accessed sequentially only, one record at a time.
>
> "Random:, file format equals 100, may be accessed randomly or sequentially. The access mode is determined by the "R" bit of the M:READ or M:WRITE argument list, as for "Packed" files. "Random" files differ from all other files in that the transfer size is determined by the byte count; multiple records may be accessed with one call to M:READ/M:WRITE.

No transfer initiated by M:READ/M:WRITE for a disk pack will cross a track boundary. The advantages for this restriction are twofold:

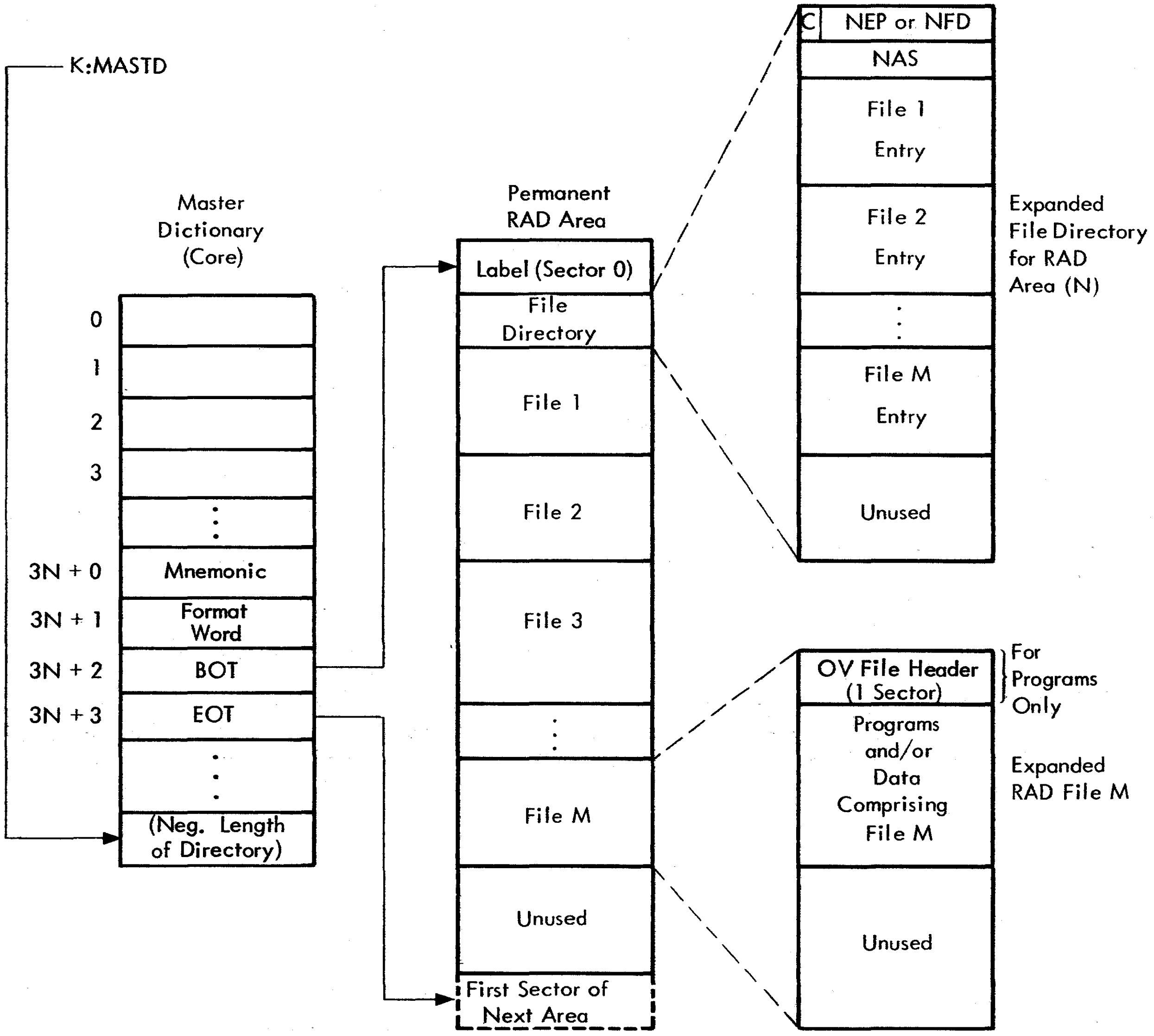
1. Long transfers cannot tie up a device. Foreground response time is improved.

2. Flawed track handling is simplified.

The implication of this restriction is that an AIO Receiver may be entered before the I/O transfer is complete. Secondary transfers will be initialed when the "check" operation is performed. The "check" operation may also specify an AIO Receiver.

At SYSGEN the RAD is divided into large blocks called RAD areas. These areas generally represent functional groupings of files (e.g., RBM and all related processors reside in the system processor area) and are either permanent or temporary. Permanent areas contain a directory of the files within that area, while temporary areas do not. Also while temporary files are created on demand by calls to M:DEFINE and are eliminated by calls to M:CLOSE, permanent files must be created by a separate processor, the RAD Editor.

The boundaries of the RAD areas are contained in the master dictionary in core memory. Since an area is a block of RAD space containing RAD files, it is itself a file. This concept is important in understanding the manipulation of RAD files. To gain access to a permanent file, the directory containing that file's location must be read into core memory by considering the area as a random-access file with BOT and EOT in the master dictionary. Thus the directory of an area is read from sector 1 of the area. Figure 39 shows the relationship between the master



where

mnemonic is the two-character EBCDIC name of the area. The mnemonic may be any two characters except "SK", but is usually one of the following:

SP	UP	BT	Dn
SD	UD	CP	Xn
SL	UL		

where

n is a hexadecimal digit.

If the mnemonic is zero, this entry of the Master Dictionary is not in use.

Figure 39. Permanent RAD File Layout (RAD Area N Containing M Files)

Format Word

DTTX	WP		IOCSB
0 4	5 6	7	8 15

where

DTTX is the index into the Device Type Table for the device containing this area.

WP is the write protect code. (See Figure 33.)

IOCSB is the I/O Control Subtable Bias. When this value is added to the contents of Zero Table cell K:IOCS, it gives the address of the IOCS for the device containing the area.

BOT is the absolute sector address of the beginning of this area.

EOT is the absolute sector address of the end of this area, plus 1.

LABEL always occupies the first sector of each area, including BT and CP, and contains the following information:

Word 0	Area mnemonic in EBCDIC.
Word 1	Bits 0–3 contain the value of the third digit of device model number (e.g., 0 for 7202, 3 for 7232, etc.).
Word 2	BOT
Word 3	EOT

Directory is as shown. All directories may be composed of one or more segments, but each segment is as long as a blocking buffer (K:BLOCK). Therefore, on a 7204 device the label and directory may span relative sector numbers 0 through 3 if K:BLOCK = 512.

Figure 39. Permanent RAD File Layout (RAD Area N Containing M Files) (cont.)

dictionary, the permanent area, and the file within the area. Figures 40 through 43 show functionally the processes executed by the Monitor service routines M:ASSIGN, M:DEFINE, M:OPEN, and M:CLOSE.

RAD File Directory

The first two words of a RAD file directory contain the following:

Word 0	C	NEP or NFD
Word 1	NAS	
	0	15

⋮} First Entry of Directory

where

C indicates the sector in which the directory ends (C = 0, if the directory ends in this RAD sector; C = 1, if the directory continues on another sector).

NEP or NFD depends on the status of C. If C = 0, NEP is the location of the next cell in this RAD sector to be used for future directory entries. If C = 1, NFD is the relative sector address where the directory is continued.

NAS indicates the relative address of the next available sector in this area, and is only meaningful ifC = 0.

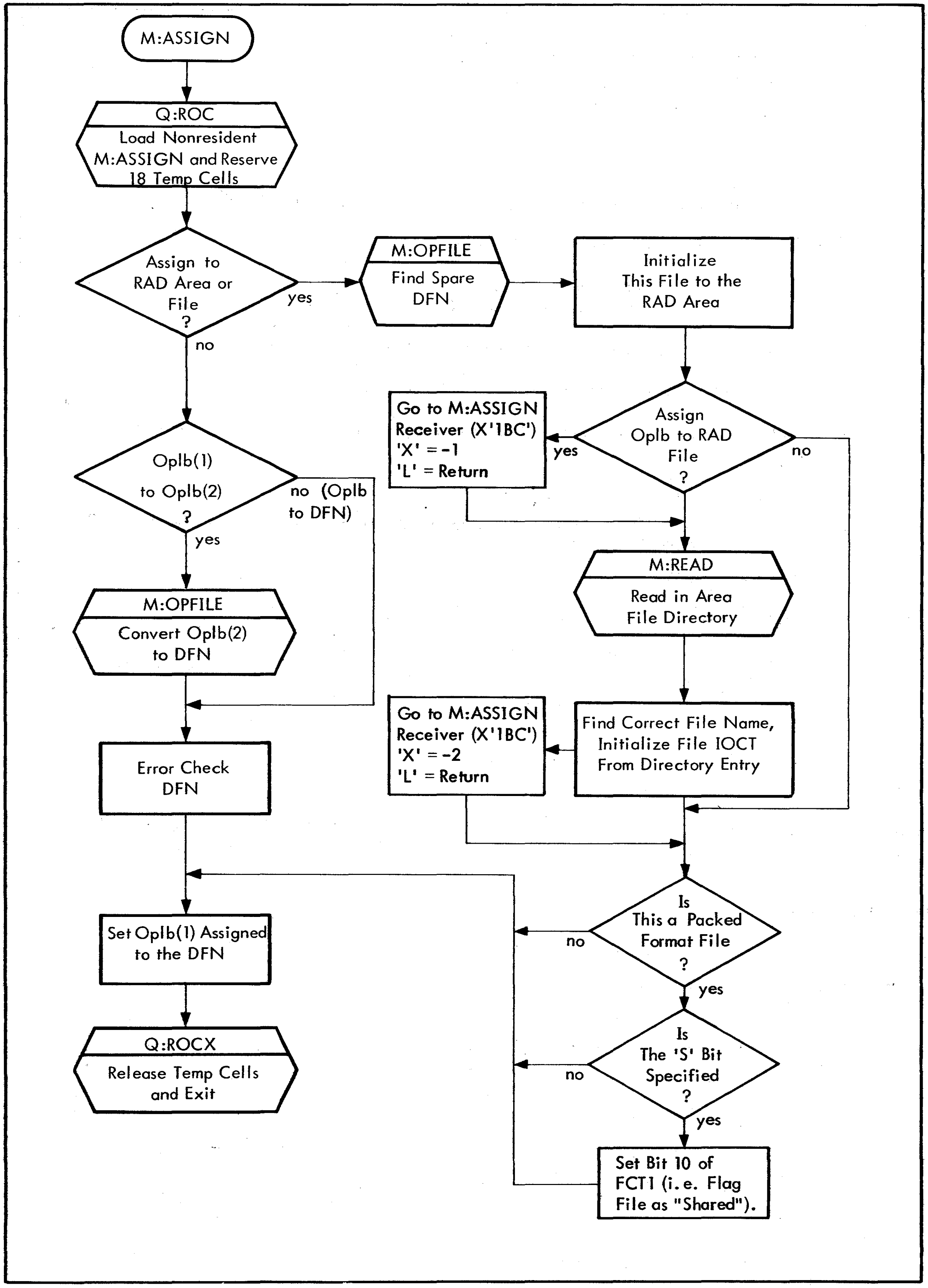

M:ASSIGN
Q:ROC
Load Nonresident M:ASSIGN and Reserve 18 Temp Cells
Assign to RAD Area or File ?
yes
no
M:OPFILE
Find Spare DFN
Initialize This File to the RAD Area
Assign Oplb to RAD File ?
yes
no
Go to M:ASSIGN Receiver (X'1BC') 'X' = -1 'L' = Return
Oplb(1) to Oplb(2) ?
no (Oplb to DFN)
yes
M:READ
Read in Area File Directory
M:OPFILE
Convert Oplb(2) to DFN
Go to M:ASSIGN Receiver (X'1BC') 'X' = -2 'L' = Return
Find Correct File Name, Initialize File IOCT From Directory Entry
Error Check DFN
Is This a Packed Format File ?
no
yes
Set Oplb(1) Assigned to the DFN
Q:ROCX
Release Temp Cells and Exit
Is The 'S' Bit Specified ?
no
yes
Set Bit 10 of FCT1 (i.e. Flag File as "Shared").

Figure 40. Processes Executed by M:ASSIGN

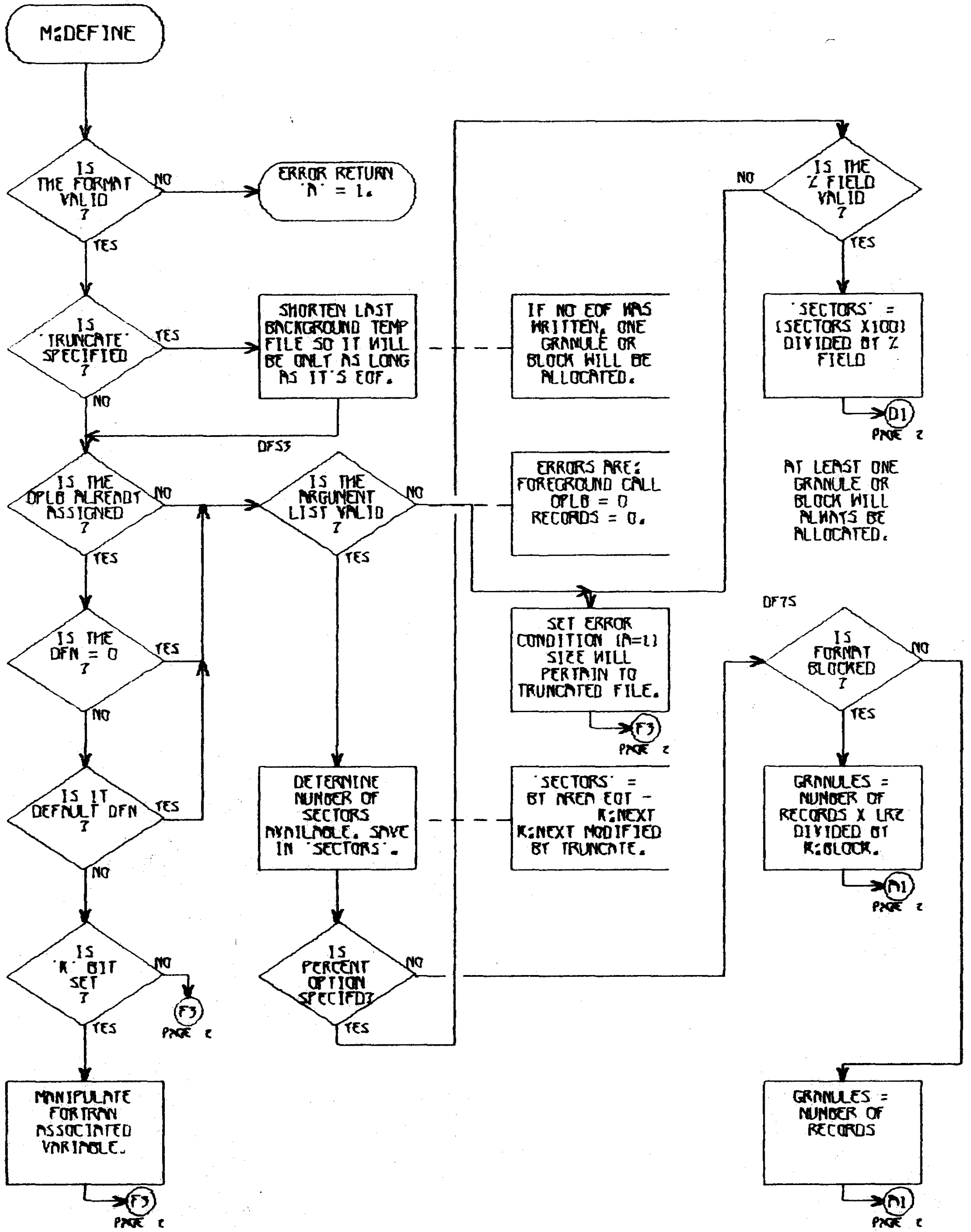

M:DEFINE
IS THE FORMAT VALID ?
NO
ERROR RETURN 'A' = 1.
YES
IS 'TRUNCATE' SPECIFIED ?
YES
SHORTEN LAST BACKGROUND TEMP FILE SO IT WILL BE ONLY AS LONG AS IT'S EOF.
IF NO EOF WAS WRITTEN, ONE GRANULE OR BLOCK WILL BE ALLOCATED.
NO
DFS3
IS THE OPLB ALREADY ASSIGNED ?
NO
IS THE ARGUMENT LIST VALID ?
NO
ERRORS ARE: FOREGROUND CALL OPLB = 0 RECORDS = 0.
YES
YES
IS THE DFN = 0 ?
YES
NO
SET ERROR CONDITION (A=1) SIZE WILL PERTAIN TO TRUNCATED FILE.
F3
PAGE 2
IS IT DEFAULT DFN ?
YES
DETERMINE NUMBER OF SECTORS AVAILABLE. SAVE IN 'SECTORS'.
'SECTORS' = BY AREA EOT - K:NEXT K:NEXT MODIFIED BY TRUNCATE.
NO
IS 'K' BIT SET ?
NO
F3
PAGE 2
YES
IS PERCENT OPTION SPECIFD?
NO
YES
MANIPULATE FORTRAN ASSOCIATED VARIABLE.
F3
PAGE 2
IS THE Z FIELD VALID ?
NO
YES
'SECTORS' = (SECTORS X100) DIVIDED BY Z FIELD
D1
PAGE 2
AT LEAST ONE GRANULE OR BLOCK WILL ALWAYS BE ALLOCATED.
DF7S
IS FORMAT BLOCKED ?
NO
YES
GRANULES = NUMBER OF RECORDS X LRZ DIVIDED BY K:BLOCK.
A1
PAGE 2
GRANULES = NUMBER OF RECORDS
A1
PAGE 2

Figure 41. Processes Executed by M:DEFINE

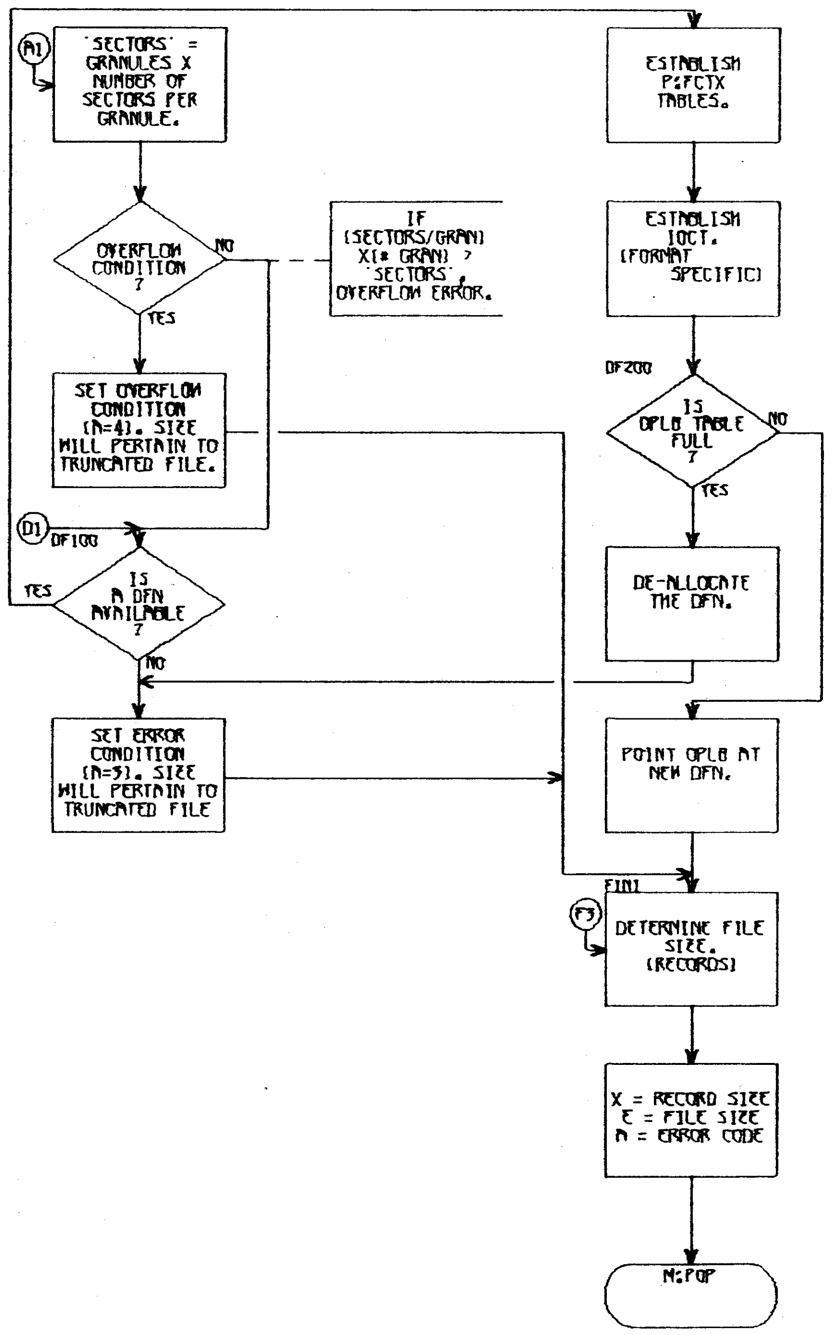

A1
'SECTORS' = GRANULES X NUMBER OF SECTORS PER GRANULE.
OVERFLOW CONDITION ?
NO
YES
IF (SECTORS/GRAN) X(# GRAN) > 'SECTORS', OVERFLOW ERROR.
SET OVERFLOW CONDITION (A=4). SIZE WILL PERTAIN TO TRUNCATED FILE.
D1
DF100
IS A DFN AVAILABLE ?
YES
NO
SET ERROR CONDITION (A=3). SIZE WILL PERTAIN TO TRUNCATED FILE
ESTABLISH P:FCTX TABLES.
ESTABLISH IOCT. (FORMAT SPECIFIC)
DF200
IS OPLB TABLE FULL ?
NO
YES
DE-ALLOCATE THE DFN.
POINT OPLB AT NEW DFN.
FIN1
F3
DETERMINE FILE SIZE. (RECORDS)
X = RECORD SIZE
E = FILE SIZE
A = ERROR CODE
M:POP

Figure 41. Processes Executed by M:DEFINE (cont.)

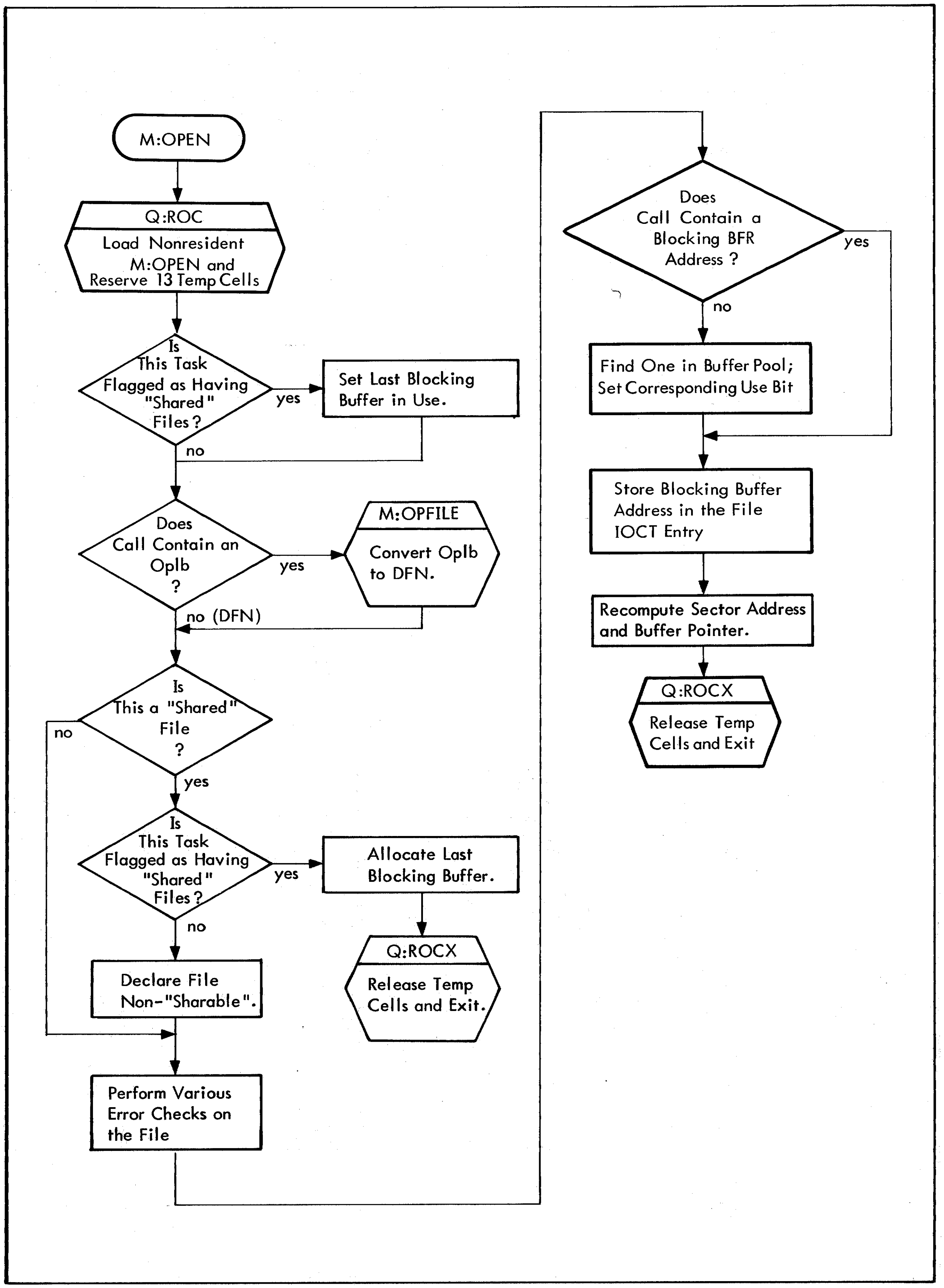

M:OPEN
Q:ROC
Load Nonresident M:OPEN and Reserve 13 Temp Cells
Is This Task Flagged as Having "Shared" Files?
yes
Set Last Blocking Buffer in Use.
no
Does Call Contain an Oplb ?
yes
M:OPFILE
Convert Oplb to DFN.
no (DFN)
Is This a "Shared" File ?
no
yes
Is This Task Flagged as Having "Shared" Files?
yes
Allocate Last Blocking Buffer.
Q:ROCX
Release Temp Cells and Exit.
no
Declare File Non-"Sharable".
Perform Various Error Checks on the File
Does Call Contain a Blocking BFR Address ?
yes
no
Find One in Buffer Pool; Set Corresponding Use Bit
Store Blocking Buffer Address in the File IOCT Entry
Recompute Sector Address and Buffer Pointer.
Q:ROCX
Release Temp Cells and Exit

Figure 42. Processes Executed by M:OPEN

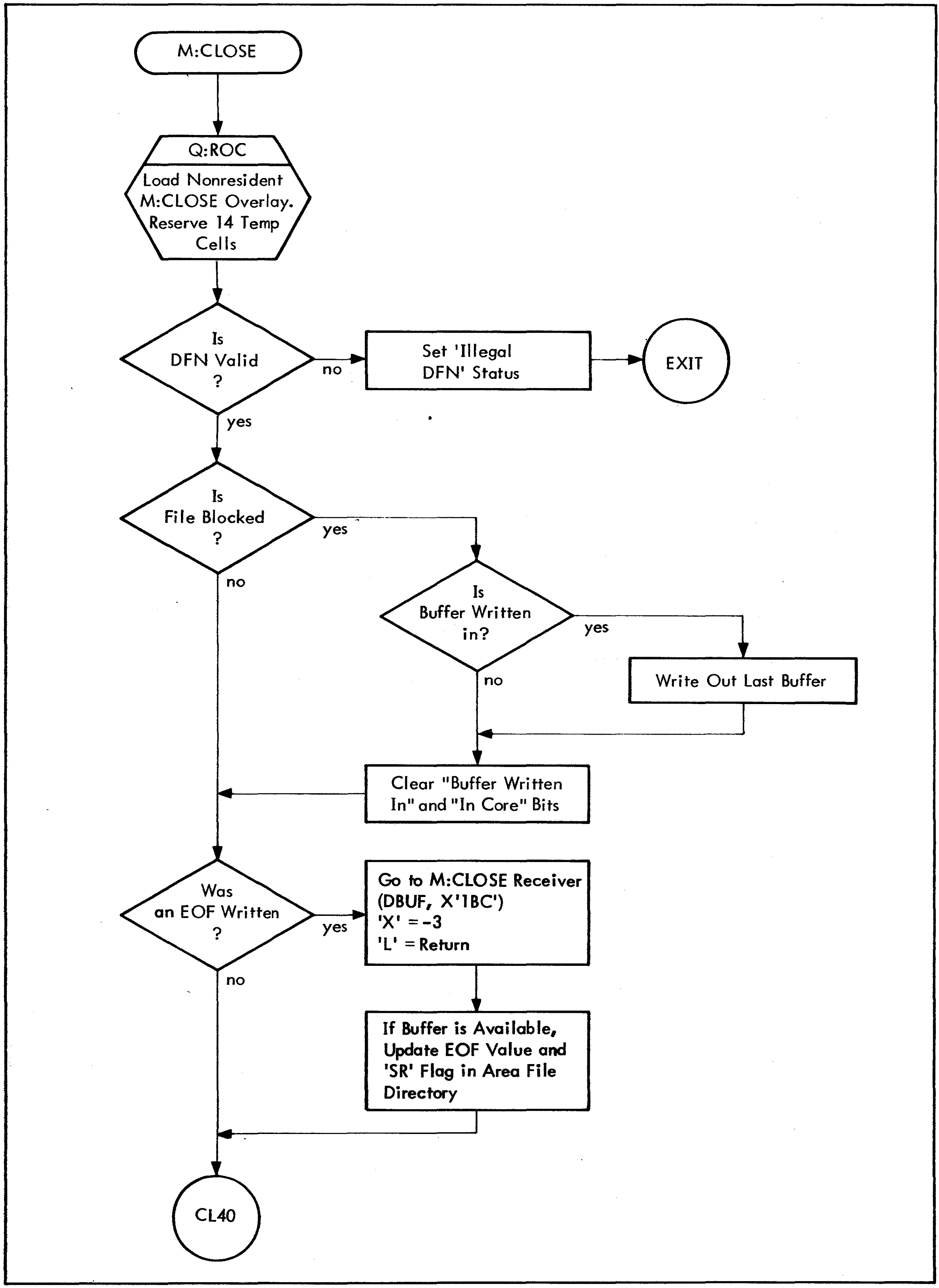

M:CLOSE
Q:ROC
Load Nonresident M:CLOSE Overlay. Reserve 14 Temp Cells
Is DFN Valid ?
no
yes
Set 'Illegal DFN' Status
EXIT
Is File Blocked ?
yes
no
Is Buffer Written in?
yes
no
Write Out Last Buffer
Clear "Buffer Written In" and "In Core" Bits
Was an EOF Written ?
yes
no
Go to M:CLOSE Receiver (DBUF, X'1BC') 'X' = -3 'L' = Return
If Buffer is Available, Update EOF Value and 'SR' Flag in Area File Directory
CL40

Figure 43. Processes Executed by M:CLOSE

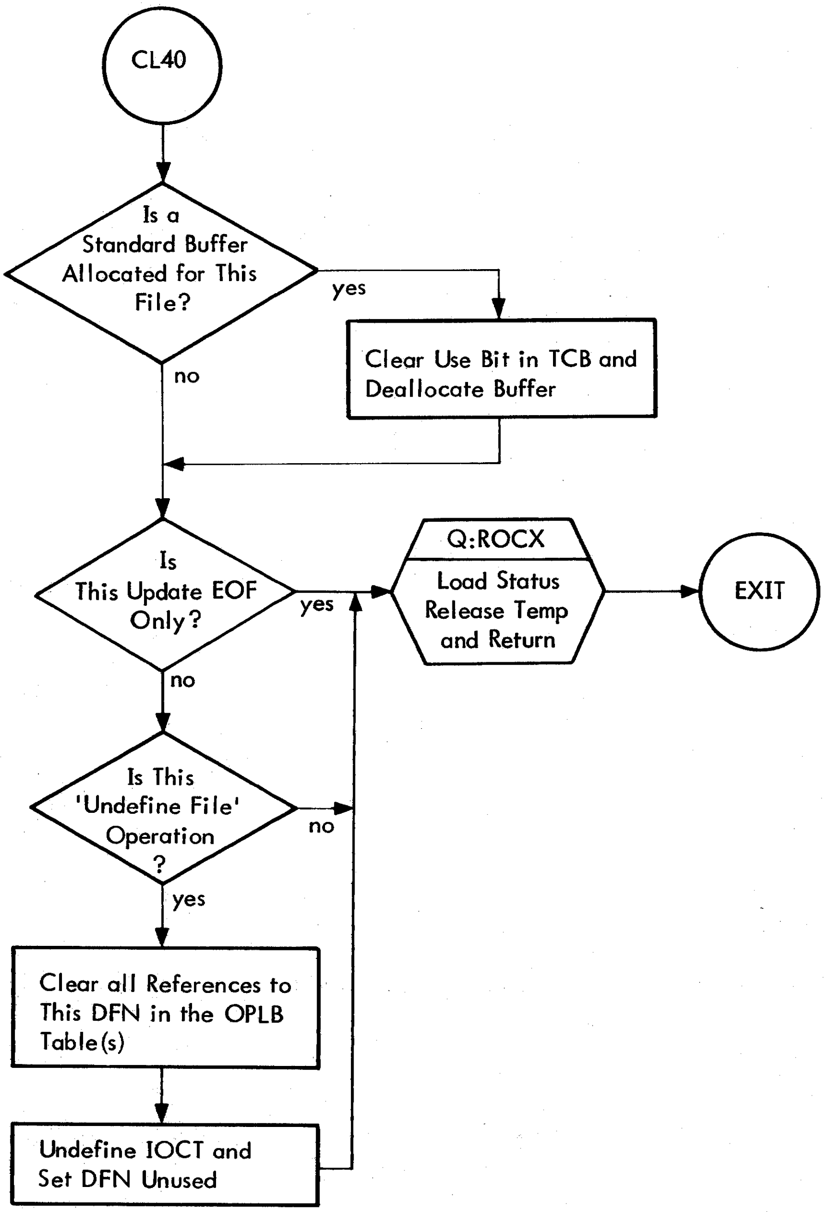

CL40
Is a Standard Buffer Allocated for This File?
yes
no
Clear Use Bit in TCB and Deallocate Buffer
Is This Update EOF Only?
yes
no
Q:ROCX
Load Status Release Temp and Return
EXIT
Is This 'Undefine File' Operation ?
no
yes
Clear all References to This DFN in the OPLB Table(s)
Undefine IOCT and Set DFN Unused

Figure 43. Processes Executed by M:CLOSE (cont.)

Each RAD directory entry has the following format:

0	n1	n2		
1	n3	n4		
2	n5	n6		
3	n7	n8		
4	File Format Byte	RF		SR
5	Logical Record Size			
6	RAD FWA of File			
7	RAD Address of EOF			
8	RAD LWA + 1 of File			
	0 1 2 3 4 5 7	8	9 14	15

where

n1 - n8 is the name of the file in EBCDIC.

File Format Byte (see Figure 33).

RF if set to 1, indicates that this file (foreground) is to be loaded and initialized at boot time.

Logical Record Size is the number of words in a logical record or granule.

SR is only meaningful for 'Blocked' or 'Unblocked' ('B' or 'U') format files and indicates that the final record written in this file (i.e., just before the EOF marker) was written with a byte count less than that specified in word 5, 'Logical Record Size'. If this record is subsequently read with a byte count equal to or greater than the size specified by 'Logical Record Size', incorrect length will be returned and the number of bytes transferred (in the 'X' register) will be the same as that specified in the original, short record written.

This feature is only invoked when the user specifies the 'Short Record' flag in the M:WRITE argument list. Otherwise, 'SR' will be reset on all write operations.

A 'Write-End-of-File' must follow a short record written if the directory is to be updated.

4. OVERLAY LOADERS

Introduction

The following discussion applies to the OLOAD loader. The BLOAD loader is functionally and structurally similar. The differences from OLOAD are detailed at the end of Chapter 4 under the BLOAD heading.

The Overlay Loader consists of a root segment and five overlay segments. Loading of the root segment is initiated by the Job Control Processor upon receiving an !OLOAD control command.

The Overlay Loader performs two mutually exclusive functions:

1. Forms a program (load module) through Loader !$ROOT and !$SEG control commands.
2. Forms a Public Library through Loader !$PUBLIB control commands.

The Loader is assembled as absolute code since it is initially loaded into the system by a separate Loader called the Absolute Loader. However, the Overlay Loader is self-relocating in execution through the use of a base table. The base table, which is pointed to by the B register during execution, is divided into three main areas:

1. Relocatable vector elements (initialized by the root).
2. A common overlay vector area (initialized by each overlay loaded).
3. Remaining pointers, flags, constants, etc. (initialized by the value loaded except where modified by the root or overlays).

An "EQU" list (O:BASE procedure from the S24RBM file) precedes the individual overlay assemblies to define the relative displacements of items in the Root Base Table. Thus, S2 must be assigned to the S24RBM file for successful assembly of the Loader. (S24RBM must have been assembled with switch #OLOAD set YES.)

Loader Structure

The Overlay Loader consists of a root and five segment overlays with the root containing the following elements: OV:LOAD Table, subroutines of common utilization, and the temporary storage space for the monitor service routines. Figure 44 illustrates the Overlay Loader and its parts and Figure 45 shows the format of the OV:LOAD Table.

An OV:LOAD Table is created for the user's program by the Overlay Loader. Information is collected in the Segment Table during PASS1, and the actual OV:LOAD Table inserted in the root segment of the user's program during PASS2.

Each entry is a fixed-length, five-word entry, and the table length = 5n+1, where n is number of segments specified on the !OLOAD control card.

The first overlay initializes the loading process. Symbol table pointers are set and the Permanent Symbol Table is read in. This table consists of LIBSYM, the Public Library definitions, and the RBMSYM Monitor service routine definitions. The first overlay is illustrated in Figure 46.

The next three overlays (2, 3, 4) constitute PASS1 of the loading process. These overlays read control cards, load input modules, and load the required library modules. These overlays are illustrated in Figures 47, 48, and 49 respectively.

The last or fifth; overlay (PASS2) will: satisfy forward references, print any required load map, complete the OV file; or alternatively, create a new Public Library. This overlay is illustrated in Figure 50.

PASS1

The three overlays (2, 3, and 4) of PASS1 are called individually as required. The subfunctions for instruction/data storing (O:STOR) and for Symbol Table insertion (O:INSERT), plus the address lists associated with common read/writes of PASS1 are loaded by overlay 2 and are undisturbed when overlay 3 or 4 is called. The Library Search Criterion Table is used commonly by overlays 3 and 4 and is defined as a leading reserve area of 300 cells in both these segments.

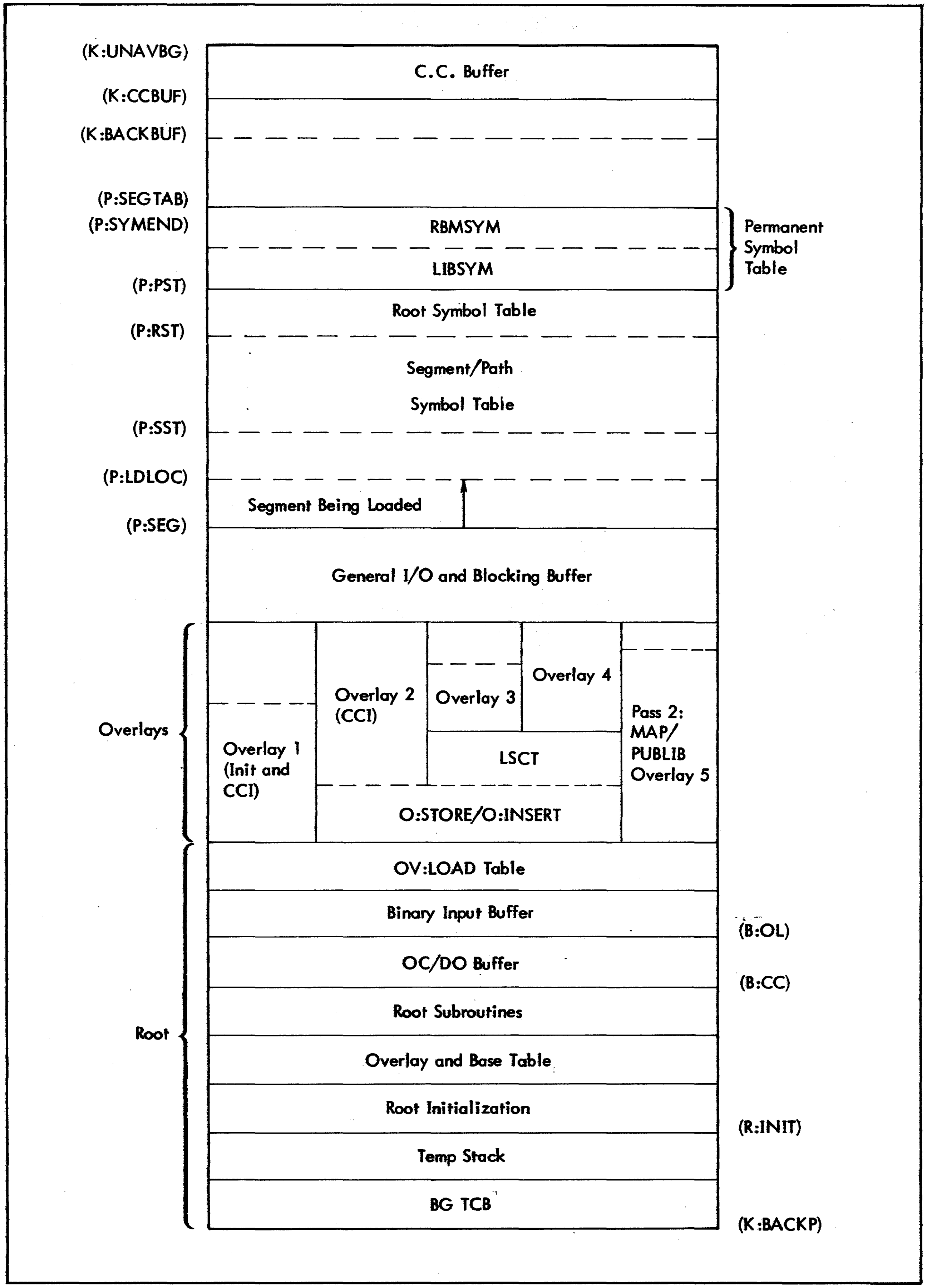
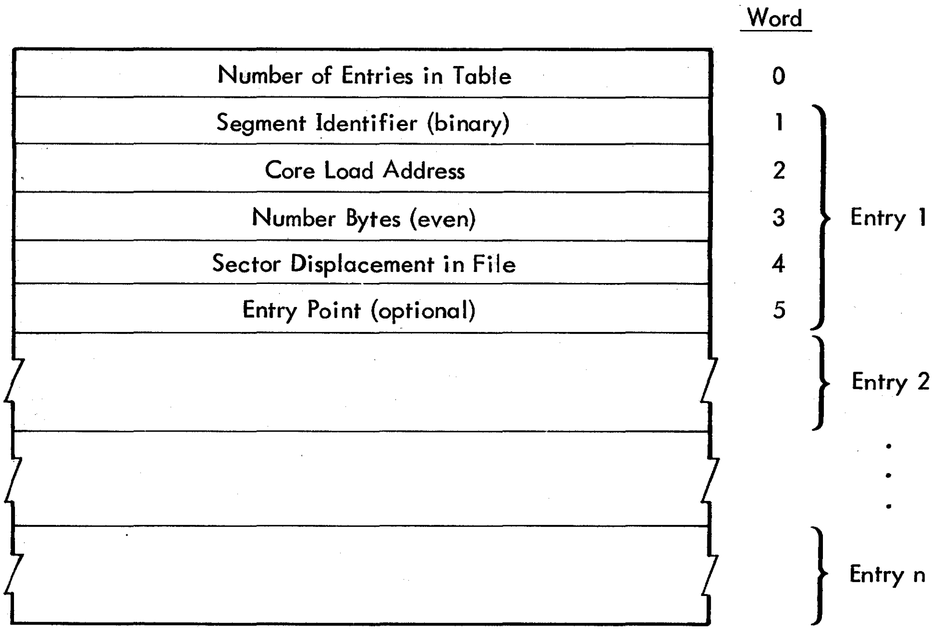


Figure 44. Overlay Loader Core Layout

OV:LOAD



where

Word Description

1 Segment identifier as specified on !$SEG card (1 ≤ N ≤ X'FF').

2 Core load address (address where segment is to be loaded at execution time).

3 Number of bytes in this overlay segment (must be even).

4 Sector displacement of this segment in the OV file (numbering starts at 0). Segments begin on sector boundaries.

5 Entry point, which must be present only if load-and-enter mode is specified in the call to M:SEGLD.

Figure 45. OV:LOAD Table Format

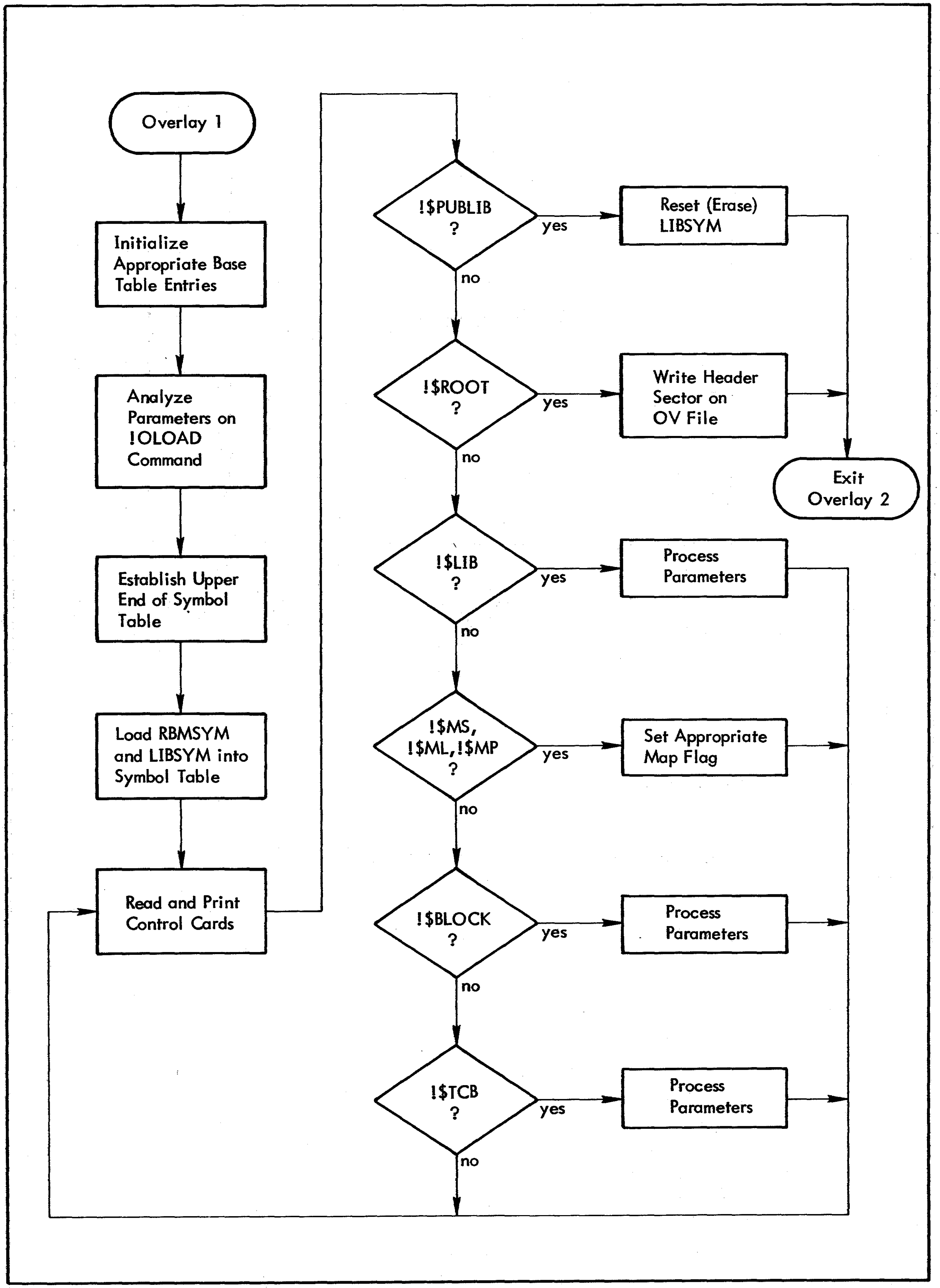

Overlay 1
Initialize Appropriate Base Table Entries
Analyze Parameters on !OLOAD Command
Establish Upper End of Symbol Table
Load RBMSYM and LIBSYM into Symbol Table
Read and Print Control Cards
!$PUBLIB ?
yes
Reset (Erase) LIBSYM
no
!$ROOT ?
yes
Write Header Sector on OV File
Exit Overlay 2
no
!$LIB ?
yes
Process Parameters
no
!$MS, !$ML, !$MP ?
yes
Set Appropriate Map Flag
no
!$BLOCK ?
yes
Process Parameters
no
!$TCB ?
yes
Process Parameters
no

Figure 46. Overlay 1 Structure

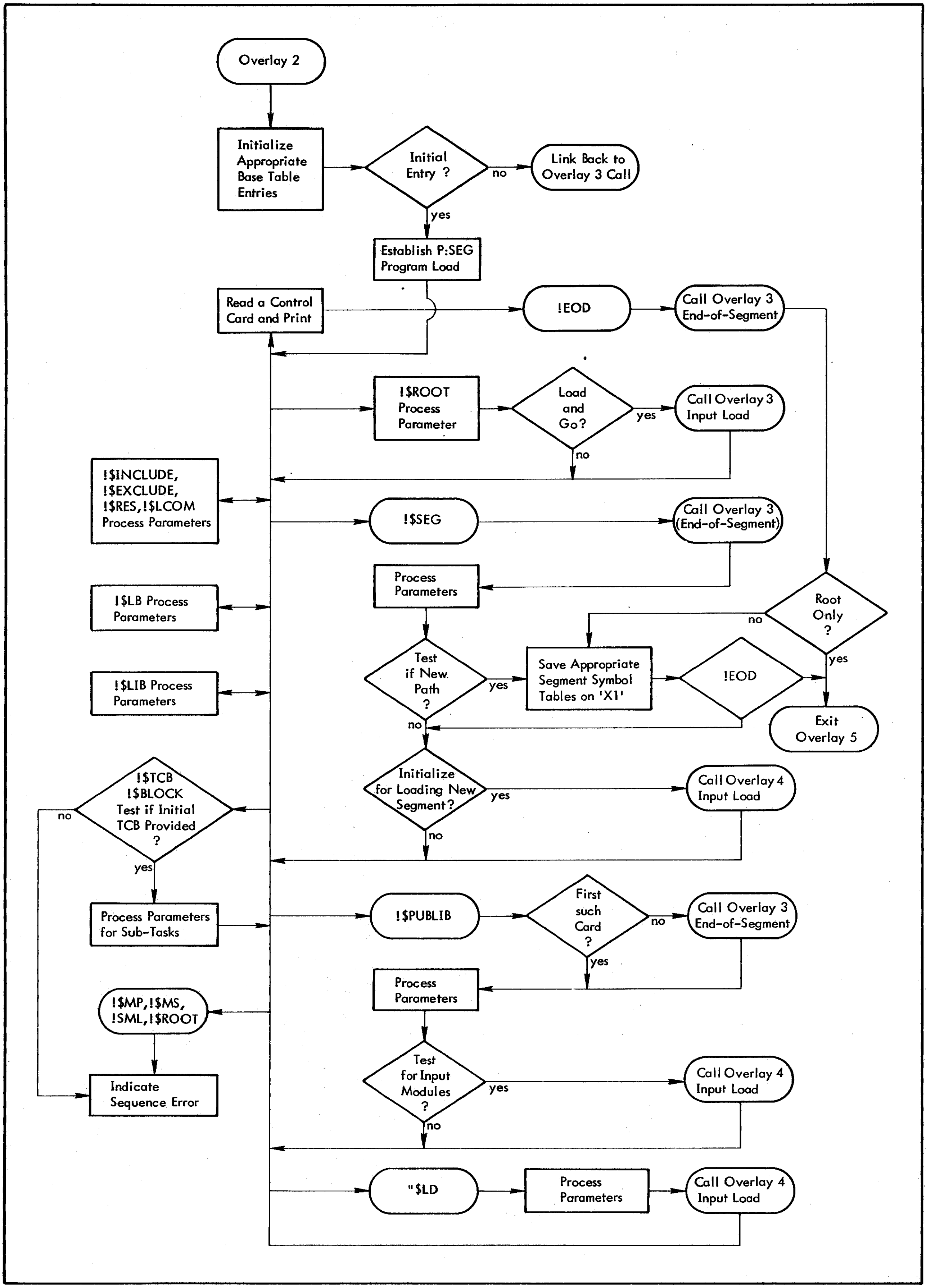

Overlay 2
Initialize Appropriate Base Table Entries
Initial Entry ?
no
Link Back to Overlay 3 Call
yes
Establish P:SEG Program Load
Read a Control Card and Print
!EOD
Call Overlay 3 End-of-Segment
!$ROOT Process Parameter
Load and Go?
yes
Call Overlay 3 Input Load
no
!$INCLUDE, !$EXCLUDE, !$RES, !$LCOM Process Parameters
!$SEG
Call Overlay 3 (End-of-Segment)
Process Parameters
!$LB Process Parameters
Root Only ?
no
yes
!$LIB Process Parameters
Test if New Path ?
yes
Save Appropriate Segment Symbol Tables on 'X1'
!EOD
no
Exit Overlay 5
!$TCB !$BLOCK Test if Initial TCB Provided ?
no
yes
Initialize for Loading New Segment?
yes
Call Overlay 4 Input Load
no
Process Parameters for Sub-Tasks
!$PUBLIB
First such Card ?
no
Call Overlay 3 End-of-Segment
yes
Process Parameters
!$MP, !$MS, !SML, !$ROOT
Indicate Sequence Error
Test for Input Modules ?
yes
Call Overlay 4 Input Load
no
"$LD
Process Parameters
Call Overlay 4 Input Load

Figure 47. Overlay 2 Structure

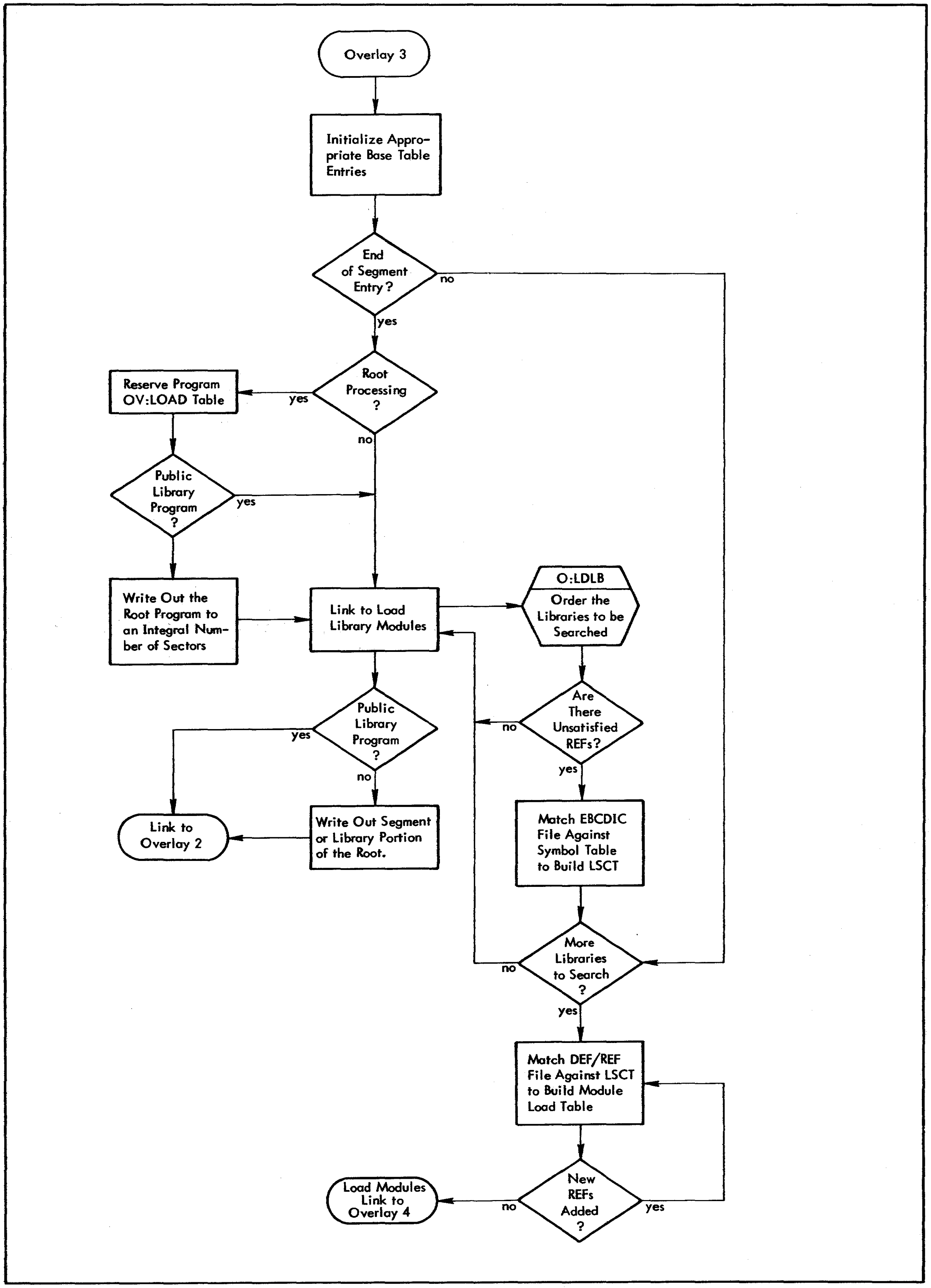

Overlay 3
Initialize Appropriate Base Table Entries
End of Segment Entry?
no
yes
Root Processing ?
yes
no
Reserve Program OV:LOAD Table
Public Library Program ?
yes
Write Out the Root Program to an Integral Number of Sectors
Link to Load Library Modules
O:LDLB
Order the Libraries to be Searched
Are There Unsatisfied REFs?
no
yes
Public Library Program ?
yes
no
Link to Overlay 2
Write Out Segment or Library Portion of the Root.
Match EBCDIC File Against Symbol Table to Build LSCT
More Libraries to Search ?
no
yes
Match DEF/REF File Against LSCT to Build Module Load Table
New REFs Added ?
no
yes
Load Modules Link to Overlay 4

Figure 48. Overlay 3 Structure

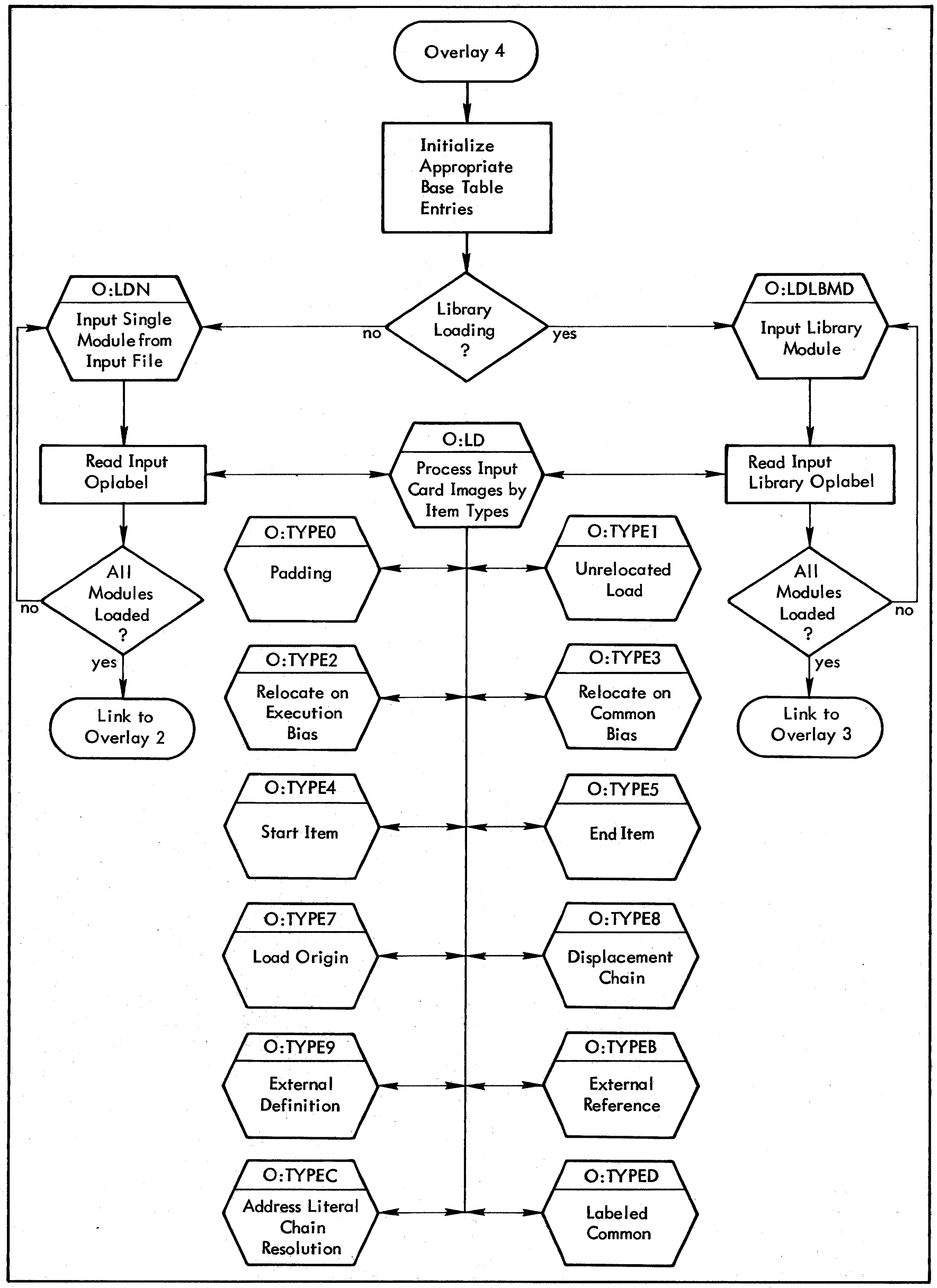
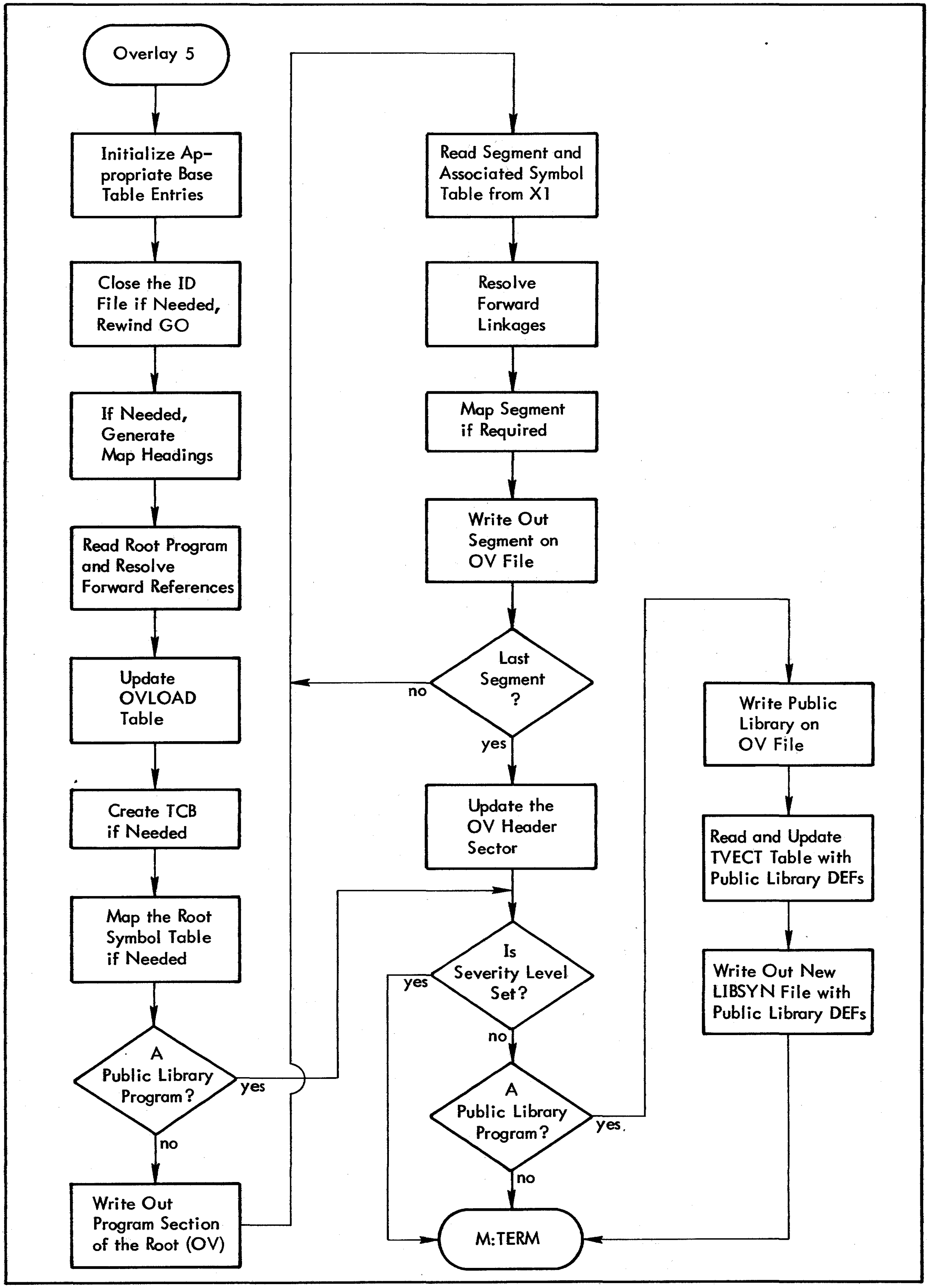


Figure 49. Overlay 4 Structure



Overlay 5
Initialize Appropriate Base Table Entries
Close the ID File if Needed, Rewind GO
If Needed, Generate Map Headings
Read Root Program and Resolve Forward References
Update OVLOAD Table
Create TCB if Needed
Map the Root Symbol Table if Needed
A Public Library Program?
yes
no
Write Out Program Section of the Root (OV)
Read Segment and Associated Symbol Table from X1
Resolve Forward Linkages
Map Segment if Required
Write Out Segment on OV File
Last Segment ?
no
yes
Update the OV Header Sector
Is Severity Level Set?
yes
no
A Public Library Program?
yes
no
M:TERM
Write Public Library on OV File
Read and Update TVECT Table with Public Library DEFs
Write Out New LIBSYN File with Public Library DEFs

Figure 50. Overlay 5 Structure

The primary function of overlay 2 is to read and interpret control commands subsequent to and including !$ROOT or !$PUBLIB commands. These commands include the !$ROOT, !$SEG, !$LD, !$LB, !$INCLUDE, !$EXCLUDE, !$MD, !$TCB, !$BLOCK, and !$PUBLIB control commands. If a set of modules are required as input, overlay 4 is called to perform the load. As a new !$SEG is encountered, optional and default libraries are searched to satisfy the unsatisfied reference. This function is performed by overlay 3, and where library modules are determined to be required, they are loaded by overlay 4.

When a segment is complete it is written on the OV file and the Segment Table is updated. If a new !$SEG card indicates that the last segment written has completed a path, the Symbol Table entries for all segments in the path are written to the X1 file and the Path Segment Table is updated.

The Segment Table is 10(N+1)+1 words in length, where N is the number of segments specified on the !OLOAD card. The Segment Table is illustrated in Figure 51.

The Symbol Table space just purged becomes available for next path. The Symbol Table entry formats are described in Figure 52.

If an !EOD card is encountered in the control command stack, the preceding procedure is followed and overlay 5 (PASS2) is loaded.

PASS2

PASS2 initializes the overlay section of the Base Table. The program section of the root is then read, and forward references into the library section or to higher level segments are satisfied (this is the only segment fractioned in this manner). The Permanent and Root Symbol Tables are mapped. PASS2 then reads each segment into core from OV, reads the appropriate Segment Symbol Table, and satisfies any forward external chain in the segment. Concurrently, the segment map is output on LO and the completed segment is rewritten to OV. When all segment processing has been done, the sector header is reread, updated, and written to OV. OV is closed and normal termination through M:TERM takes place. Header formats are shown in Figures 53 and 54.

If a Public Library is being created, overlay 4 creates a new Public Library on the RAD. The Public Library just loaded is written to the PUBLIB file in the System Processor area. The Monitor Services Transfer Vector (TVECT) file is read from System Processor area, the Public Library section updated, and written to TVECT. A new Public Library Symbol Table is written to LIBSYM file on the System Data area. The new LIBSYM is incompatible with the current in-core Public Library. All files are closed and normal termination through M:TERM takes place.

Loading a User Program Root Segment

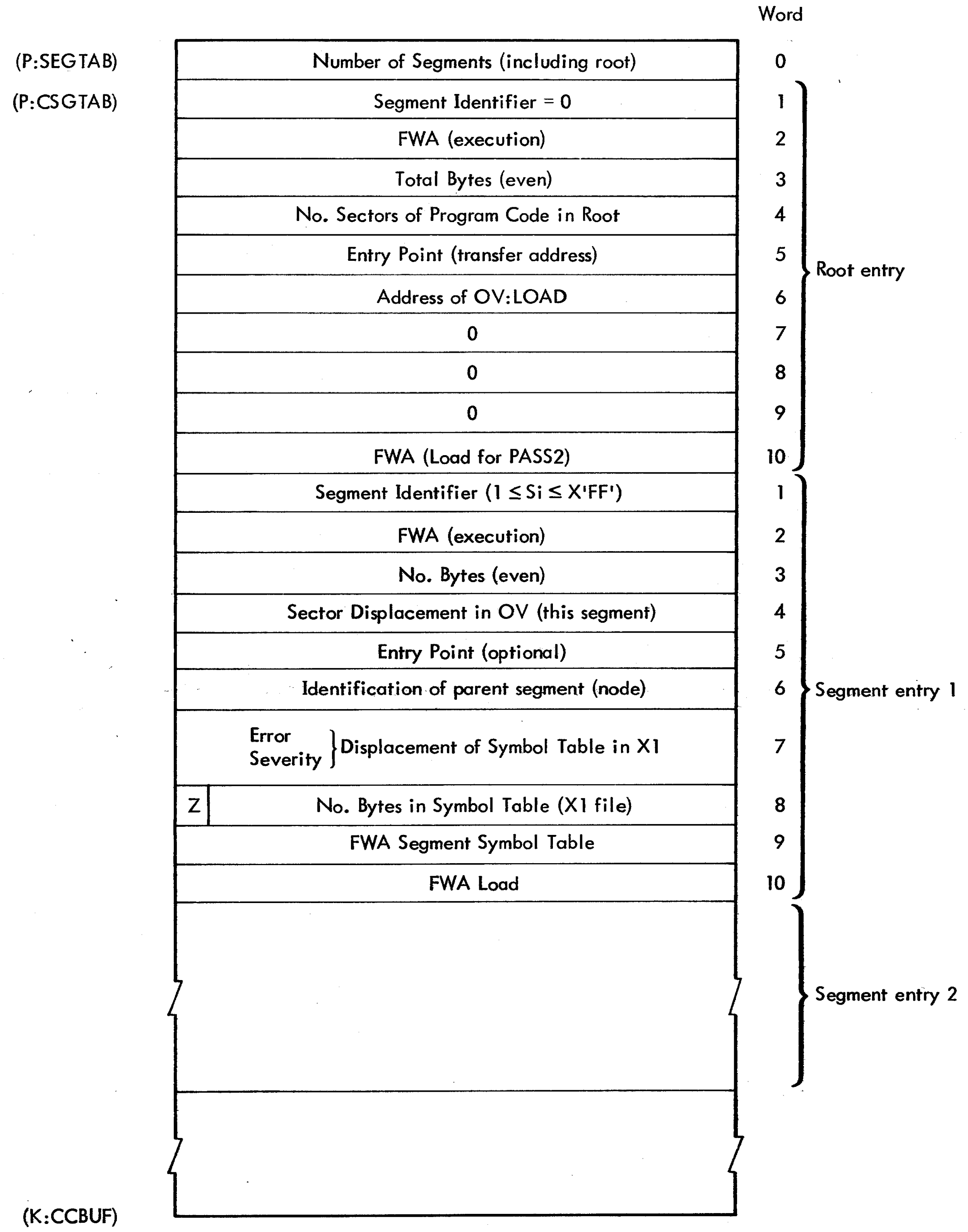
The technique used by the Overlay Loader in loading the root of a user overlay program is different from the loading of any other segment. This is because of the special case where a root and its library subroutines will not fit in core with the Loader and its tables. To allow for such a case, a root is loaded by using the general scheme given below.

Since the temp stack need not occupy core at load time, its presence is indicated by updating the execution location by the appropriate amount. Thus, P:EXORG points to the beginning of the temp stack and P:EXLOC points to the actual origin of the task.

The program section of the root is loaded without resolving any external chains. When a !$SEG card is encountered, an integral number of sectors is written to OV and the OV pointer is updated. The remaining fractional sector of the program section of the root is moved to P:SEG (beginning of the loading area of core), pointers in the segment table updated, and the load bias reset to P:SEG (plus the remainder of the Program segment). The search and loading of the library now commences. It is assumed that all forward and external REFs in the library section will be satisfied and chains resolved during the PASS1 loading of the library subroutines.

At the completion of the library loading, the remaining program portion of the root segment is written out onto the OV file with the required library routines. During PASS2, only the complete program portion of the root segment is read. External reference chains to the library routines are resolved using the Symbol table and rewritten again onto the OV file. The library section of the root is not read during PASS2. Only an integral number of sectors are read and written.

The preceding scheme is followed even in cases where both the program segment and library segment could be contained in core with the Loader.



		Word	
(P:SEGTAB)	Number of Segments (including root)	0	
(P:CSGTAB)	Segment Identifier = 0	1	Root entry
	FWA (execution)	2	
	Total Bytes (even)	3	
	No. Sectors of Program Code in Root	4	
	Entry Point (transfer address)	5	
	Address of OV:LOAD	6	
	0	7	
	0	8	
	0	9	
	FWA (Load for PASS2)	10	
	Segment Identifier (1 ≤ Si ≤ X'FF')	1	Segment entry 1
	FWA (execution)	2	
	No. Bytes (even)	3	
	Sector Displacement in OV (this segment)	4	
	Entry Point (optional)	5	
	Identification of parent segment (node)	6	
	Error Severity } Displacement of Symbol Table in X1	7	
	Z | No. Bytes in Symbol Table (X1 file)	8	
	FWA Segment Symbol Table	9	
	FWA Load	10	
			Segment entry 2
(K:CCBUF)			

where

P:SEGTAB points to word 0 of the table, which contains the number of entries currently in the table. This may be less than the number of entries specified on the !LOAD card. After PASS1 has completed it will specify the actual number of segments loaded (including the root).

Figure 51. Segment Table Format

P:CSGTAB   points to the entry for the current segment being loaded. It is initialized to (P:SEGTAB)+1 for the root and incremented by P:SLEN for each segment entry.

Root Entry

Word   Description

1   Segment identifier of the root (always 0). No segment may have the segment identifier 0.

2   Address to which M:LOAD will read in the root segment.

3   Number of bytes to read (must be even).

4   During PASS1, the OV:LOAD table is assigned as a reserve at the end of the program section of the root, and P:LDLOC and P:EXLOC updated by that amount. The following is applied:

$$\frac{\text{P:LDLOC-P:SEG}}{\text{sector size}} = \text{N (sectors)} + \text{R (words)}$$

N sectors of program code are written to OV starting at sector 1, P:OV is updated to N+1, and N is entered into word 4. R words are moved down to P:SEG and P:LDBIAS and P:LDLOC is set to P:SEG+R. The library section of the root is then loaded. This effectively allows a root up to twice the available load space to be loaded. During PASS2, N+1 sectors of root program code are loaded, and reference chained to the library portion and to other segments resolved. During PASS2 the OV:LOAD table is also completed.

5   Last transfer address encountered in loading the root modules.

6   Load time address of OV:LOAD table in the user's program. The table is completed in PASS2.

7-9   Not used.

10   Load address for N+1 sectors of program code during PASS2 (= P:SEG).

Segment Entry

Word   Description

1-5   Identical to entries in the Root Entry.

6   Corresponds to the Sn parameter on the !$SEG card. If the segment is attached to the root, word 6 is zero.

Figure 51. Segment Table Format (cont.)

7 Error severity (bit 0 = 0 or 1). It is equal to the severity encountered in binary modules forming this segment.

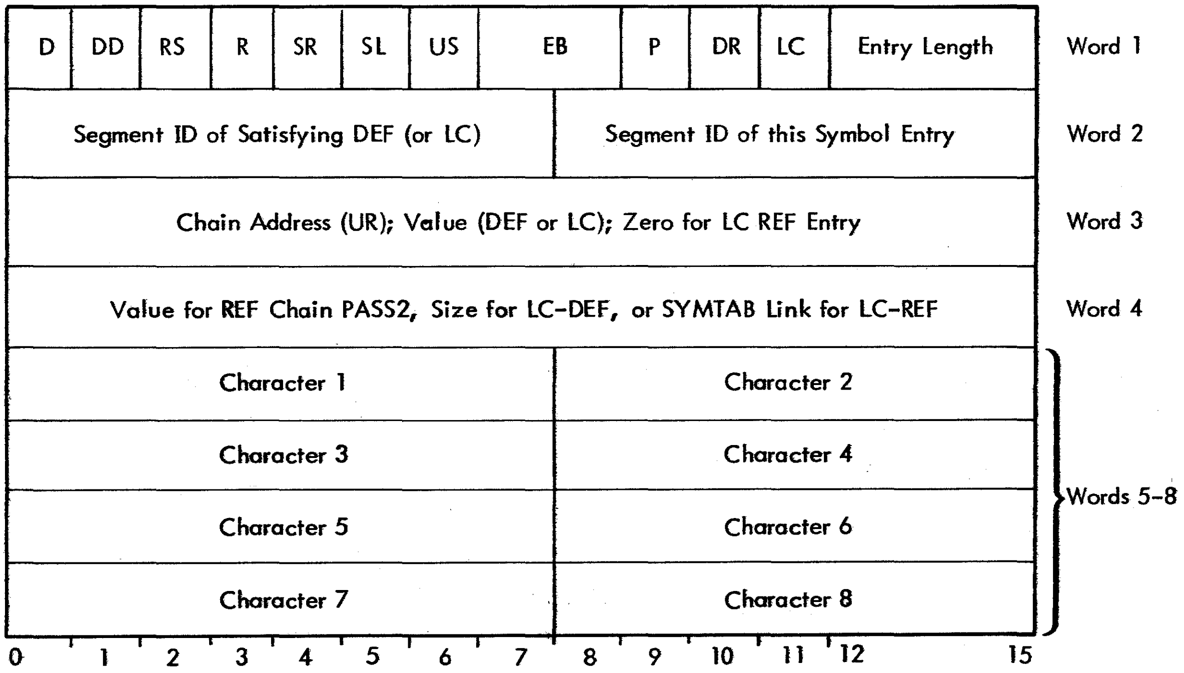
Displacement of the Symbol table in X1 (bits 1-15). As paths are completed and new paths started, symbol tables for each segment are written to X1. They are read during PASS2 to resolve forward references. Word 7 contains the displacement of the Symbol table for this segment in the X1 file. All Symbol tables begin on sector boundaries.

8 Number of bytes in this Symbol table (even number). Z, bit 0, is a flag meaning this Symbol table has been written on the RAD.

9 Location of the Symbol table in core during PASS1. PASS2 does not use this address during loading.

10 Load address for this segment during PASS2. The Loader does not output leading reserves on a segment. Thus word 10 = P:SEG+r, where r is the sum of all reserves in the first module of the segment up to the first loadable data. Note that re-orging data into the leading reserve of a segment will thus cause an SL abort.

Figure 51. Segment Table Format (cont.)



Word	Bits 0-7	Bits 8-15
Word 1	D (0), DD (1), RS (2), R (3), SR (4), SL (5), US (6), EB (7)	EB (8), P (9), DR (10), LC (11), Entry Length (12-15)
Word 2	Segment ID of Satisfying DEF (or LC)	Segment ID of this Symbol Entry
Word 3	Chain Address (UR); Value (DEF or LC); Zero for LC REF Entry	
Word 4	Value for REF Chain PASS2, Size for LC-DEF, or SYMTAB Link for LC-REF	
Words 5-8	Character 1	Character 2
	Character 3	Character 4
	Character 5	Character 6
	Character 7	Character 8

Figure 52. Symbol Table Format

where, if the bit is on

Word	Description

1	D = DEF
	DD = DEF declared
	RS = Satisfied REF
	R = Primary REF
	SR = Secondary REF
	SL = Segment or Library (0 = Segment, 1 = Library)
	US = User or System (0 = System, 1 = User)
	EB = Extended, Basic, or Main Mode (10 = Extended, 01 = Basic, 11 = Main)
	P = Public Library
	DR = Double Reference
	LC = Labeled COMMON
	Entry Length = 5 to 8 (variable)

2	Segment identifier number (on !$SEG card).
3	Chain address for unsatisfied references. If the entry is a primary or secondary reference, this is the last link in the threaded reference chain. If the entry is a DEF, this is the value. If a labeled COMMON entry, this word is the block size value for a defining entry, or zero for a reference entry.
4	If entry is a "satisfied" REF (i.e., a DEF has been found) this is its value to be inserted in the reference chain during PASS2. If an unsatisifed reference (primary or secondary) is meant to be excluded, then word 4 will be a -1. If this is a labeled COMMON entry, word 4 is the defined location address or link to the defining Symbol Table entry.
5-8	Alphanumeric characters of the Symbol Table entry.

Note that if the DEF is in the same segment as the REF, the chain is completed during PASS1 and the reference item is converted to a definition at that time.

References chained from the program section to the library section of the root are completed during PASS1. They are inserted as "satisfied" references.

References from the library section of the root to a segment definition will remain an unsatisfied reference since the library section is not reviewed during PASS2.

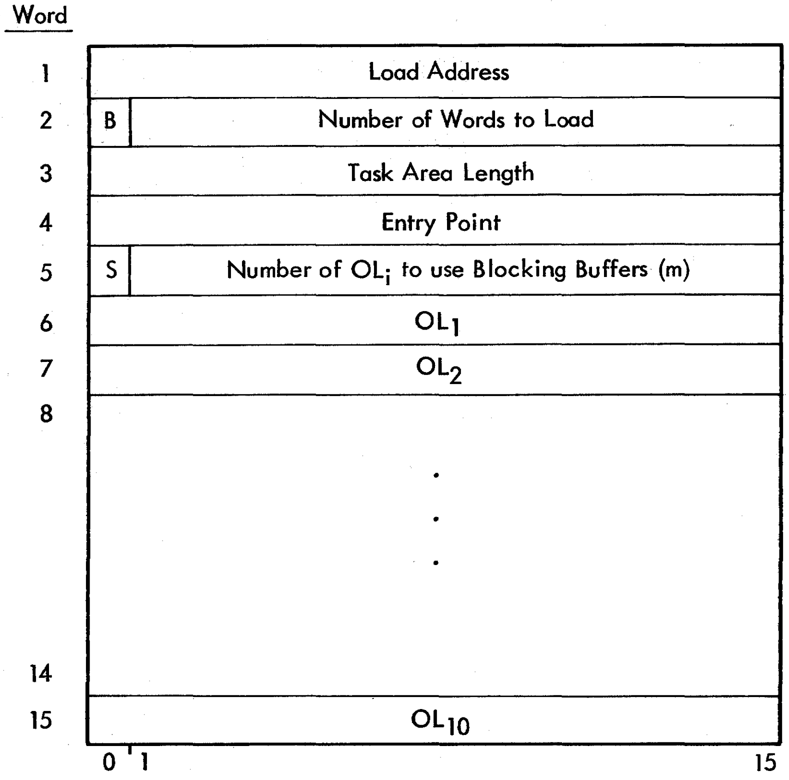
Figure 52. Symbol Table Format (cont.)

Word		
1	Load Address	
2	B	Number of Words to Load
3	Task Area Length	
4	Entry Point	
5	FWA of TEMP Area	
6	LWA+1 of TEMP Area .	
7	LWA+1 of Blocking Buffer Pool	
8	Number of Blocking Buffers Available	
9	S	Number of OL_i to use Blocking Buffers (m)
10	OL_1	
11	OL_2	
12	. . .	
18		
19	OL_{10}	
	0	1 15

where

Word	Description
1	Location at which to begin loading (K:BACKBG).
2	(B = 1) bits 1-15 contain the number of words to load for the root segment.
3	Maximum length needed for largest overlay path (including COMMON).
4	Entry point of the root (must be nonzero). The last transfer address encountered in loading the root.
5,6	Contain words 3 and 4 of TCB.
7,8	Contain words 14 and 15 of the TCB. Word 7 is the value of the COMMON base or end of available memory for this task. Word 8 is calculated by dividing the area between the end of the largest overlay and beginning of COMMON by the sector size.
9	Number of blocking buffers to allow at execution time or, if word 10 is nonzero, the number of operational labels in words 10-19 which may use blocking buffers at execution time. S=1 indicates that block sharing for packed random files is acceptable.
10-19	Two-character EBCDIC operational labels or binary values of F:xxx FORTRAN operational labels.

Figure 53. Background Overlay Task Header



where

Word	Description
1	FWA for loading (also word 0 of TCB).
2	(B = 0) bits 1-15 contain the number of words in the root segment.
3	Largest area this task will require, including reserves and COMMON.
4	If word 4 = 0, M:LOAD arms and enables interrupts. If 4 ≠ 0, this is the entry point to initialization routine in the task that will arm and enable interrupts.
5	Number of blocking buffers to allow at execution time or, if word 6 is nonzero, the number of operational labels to follow in words 6-15 which may require Blocking Buffers at execution time. S=1 indicates that block sharing for packed random files is acceptable.
6-15	Two-character EBCDIC operational labels or binary values of F:xxx FORTRAN operational labels.

Since the TCB for foreground task is output as part of the task, the temp storage limits (words 3 and 4) and the blocking buffer parameters (words 14 and 15) are inserted directly into the TCB before outputting.

Figure 54. Foreground Overlay Task Header

Public Library

Creating the Public Library

In creating the Public Library, the Loader must insert indicators into each definition item in the Permanent Symbol table (LIBSYM) to show whether the routine is in Extended, Basic, or Main mode (see below, "Loading the Public Library"). Since the Loader cannot determine the mode implicitly from the binary module, control information must be input to the Loader through $PUBLIB control commands. By proper use of this command (optionally followed by !$LD, !$LB, and !$INCLUDE commands) a Public Library of any combination can be created. The Loader suppresses the use of the current LIBSYM in creating a new PUBLIB.

To create the Public Library, the Loader selectively loads the routines, concurrently building a Symbol table with bits E, B, or M appropriately set. An !EOD completes the input. Nothing is written on the OV file as loading is completed. Overlay 5 optionally maps the library, writes the Public Library core image onto the System Processor file, creates the Public Library portion of the Monitor Transfer Vector file (TVECT) in the System Processor area by using information from the Symbol table, and writes the new Symbol table into the LIBSYM file of the System Processor area.

Loading the Public Library

Before searching the User or System Library, the Loader will endeavor to satisfy references from the Public Library. Definitions from the Public Library are input as part of the Permanent Symbol table in Overlay 1 from the LIBSYM file in the System Data area of the RAD. Where applicable, the mode (E, B, or M) is checked. If the mode of a matching Public Library definition is incorrect, the appropriate RAD library searches will be made to load the routine of correct mode. A Main mode routine in the Public Library may be utilized in both Extended and Basic library search modes.

System and User Libraries

Library Search Order Tables

The Library Search Order Tables, T:DLIB and T:OLIB contain information as to which libraries are to be searched, and in what order, at any given time in the load process.

T:OLIB, which defines the optional case, is reset to zero (empty) by O:INIT. It can be set to a temporary nonzero value only by a !$LB card. At the completion of the total loading process for that segment, both cells of T:OLIB are reset to zero. T:DLIB defines the default case library search. It is set initially by overlay 1 to the Basic System Library. A !$LB card will not override any values in T:DLIB. However, a !$LIB card sets T:DLIB to the new default case, which remains constant until the termination of the complete load process or until a new !$LIB is encountered.

T:OLIB is always searched first. If the first entry in T:OLIB is empty, the search continues through T:DLIB. If T:OLIB contains entries, T:DLIB is not searched. T:DLIB and T:OLIB are contained in the Root Base table.

Structure of the System/User Library Area is shown in Figure 55.

Input/Output

All I/O is done in initiate-wait mode, using operational labels and invoking RBM error recovery procedures. A generalized I/O check routine, R:CHEKIO, checks the return status and outputs diagnostics for such conditions as end-of-tape, file-protect, etc. Control is returned to the calling routine only by an EOF or normal return.

Three buffers are allocated in the root segment of the Loader: B:INBLOCK, a single 512-word or double 180-word blocking buffer and general RAD I/O buffer; B:OL, a 60-word deblocking buffer; and B:CC, the Loader's control card buffer. No internal provision is provided for blocked I/O to the CC, DO or OC operational labels.

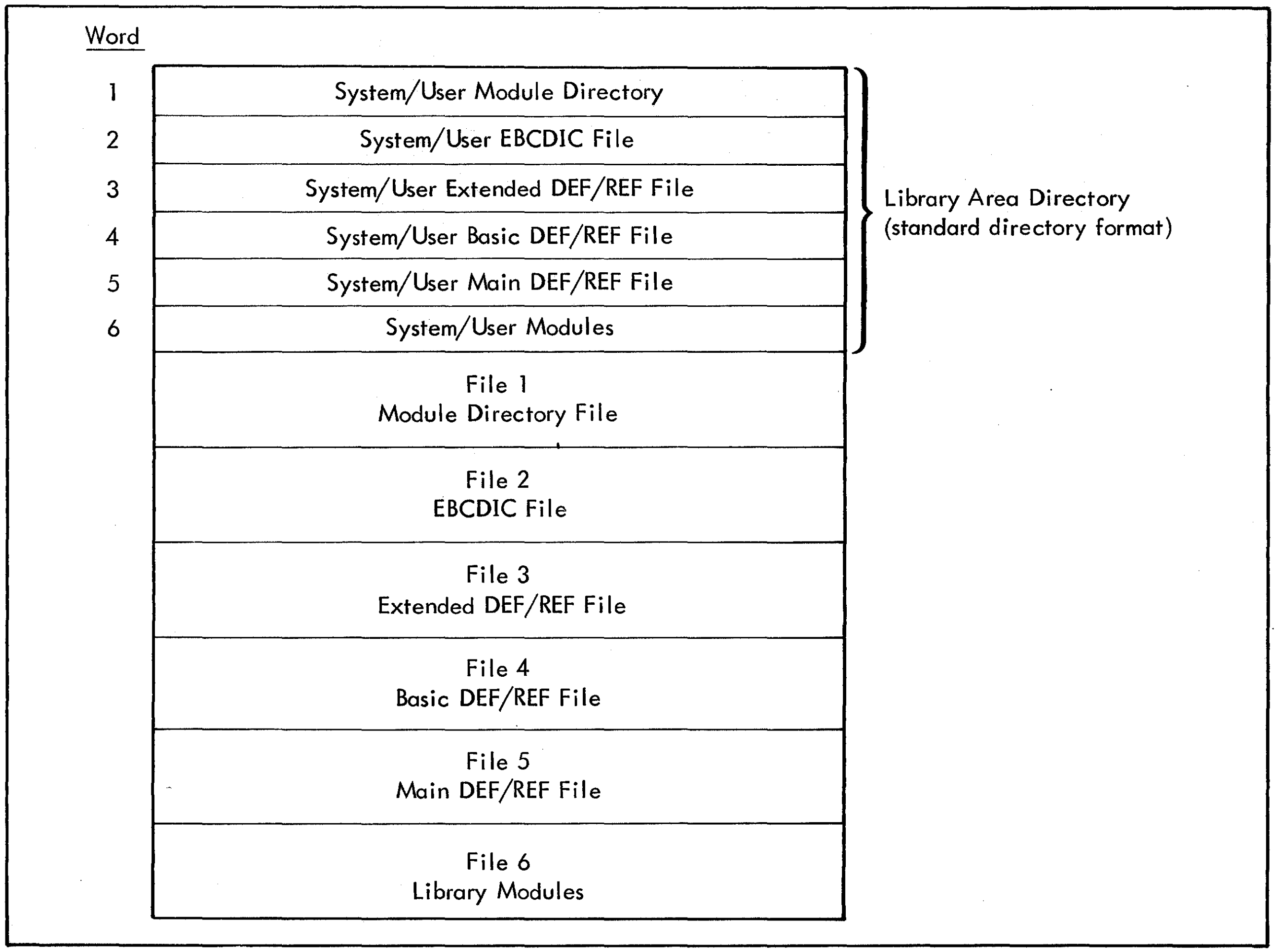
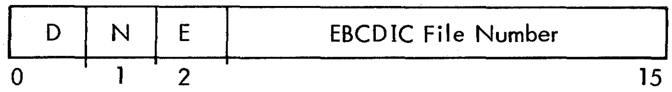


Figure 55. System/User Library Area Structure

Library Loading

If there are unsatisfied references within an input program, individual libraries will be searched for satisfying definitions. In the process, a Library Search Criterion Table (LSCT) will be built as shown below:



where

D = 1 indicates the EBCDIC symbol has been defined along the program path.

N = 1 indicates the EBCDIC symbol as a reference has been added to the table during the current library search.

E = 1 indicates the library module having the EBCDIC symbol as a DEF that should be excluded from loading.

EBCDIC File Number is the sequential count of the associated symbol in the EBCDIC file.

The LSCT will be comprised of those symbols defined within the library (System or User) and occurring in the program Symbol table. To do so, the EBCDIC file is matched against the program Symbol table to produce LSCT entries reflecting symbols already defined, those needing satisfaction, and those to be excluded from library loading. This procedure is shown in the flowchart of Figure 56.

Subsequently, the LSCT is matched against the individual library DEF/REF files to determine those modules to be loaded. A module to be loaded (i.e., one or more of its DEFs are required by the program) in turn needs its REFs satisfied. Consequently, the LSCT may be added to in this process. Should the LSCT exceed 300 entries, the process will abort with an "LS" diagnostic.

When an individual DEF/REF file is completely searched without adding new elements to the LSCT, library modules specified as being required will be loaded. Figure 57 shows the logical flow of this processing.

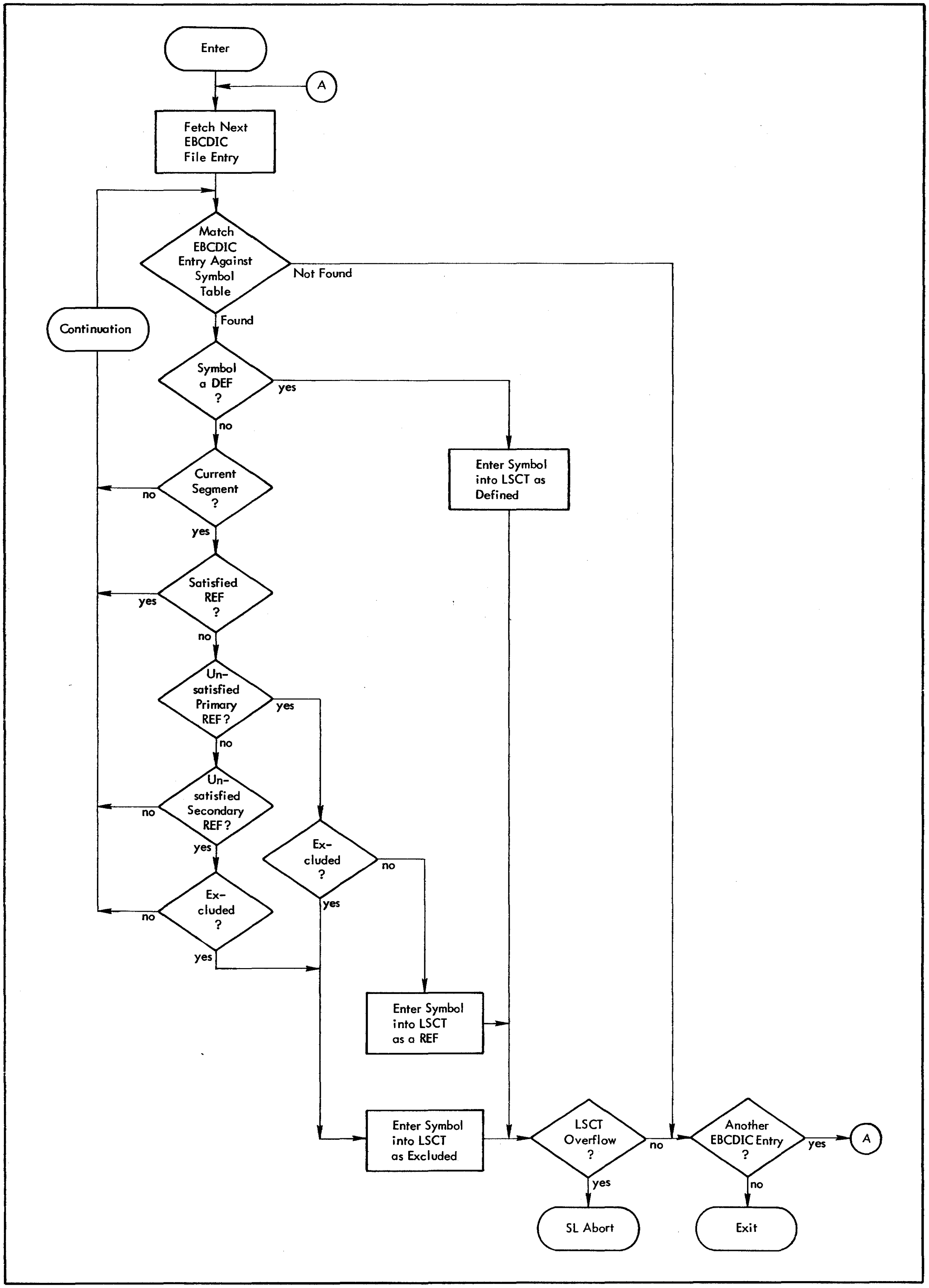

Enter
A
Fetch Next EBCDIC File Entry
Match EBCDIC Entry Against Symbol Table
Not Found
Found
Continuation
Symbol a DEF ?
yes
no
Enter Symbol into LSCT as Defined
Current Segment ?
no
yes
Satisfied REF ?
yes
no
Un-satisfied Primary REF ?
yes
no
Un-satisfied Secondary REF ?
no
yes
Ex-cluded ?
no
yes
Ex-cluded ?
no
yes
Enter Symbol into LSCT as a REF
Enter Symbol into LSCT as Excluded
LSCT Overflow ?
no
yes
SL Abort
Another EBCDIC Entry ?
yes
A
no
Exit

Figure 56. Building the Library Search Criterion Table

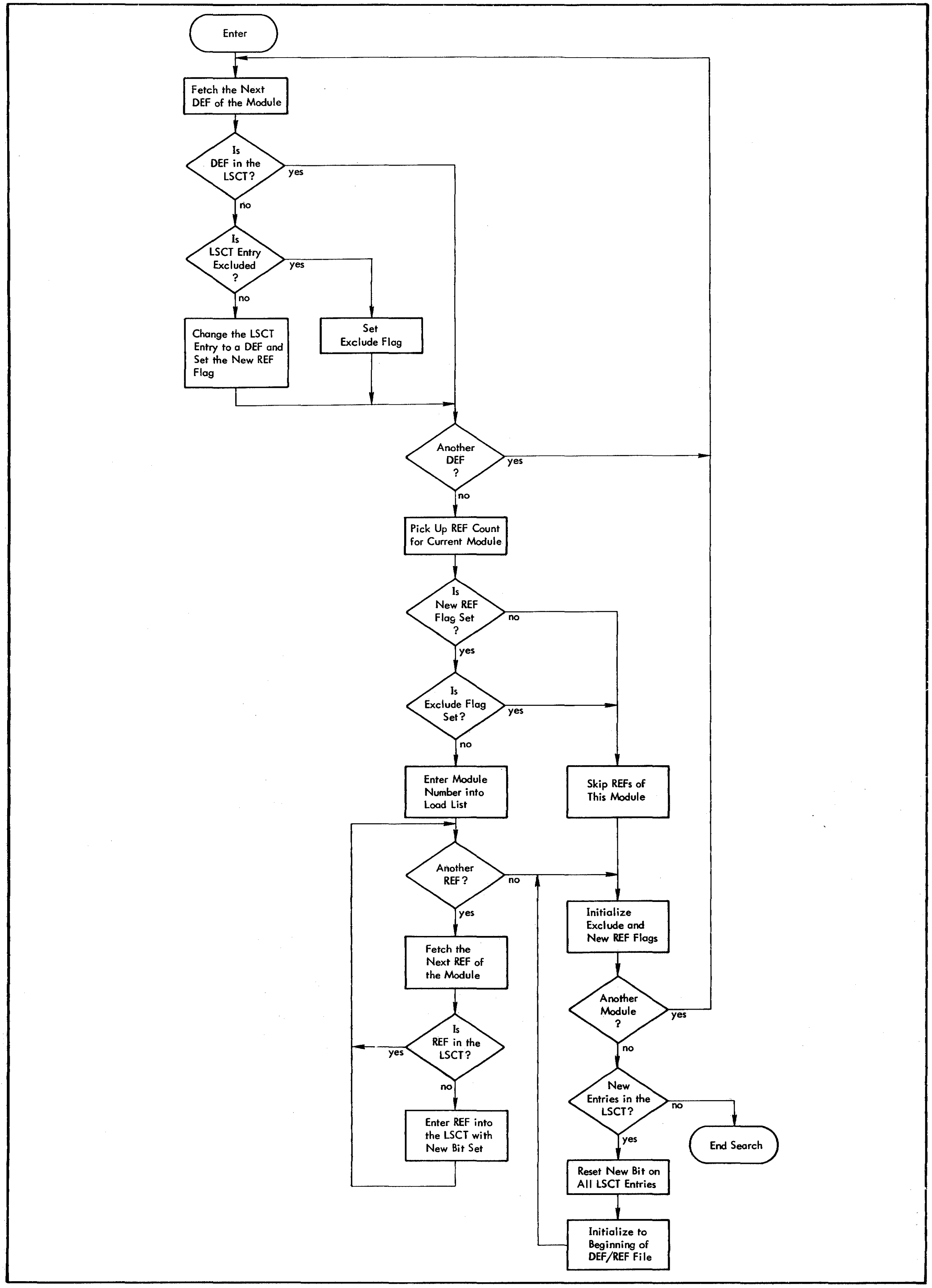

Enter
Fetch the Next DEF of the Module
Is DEF in the LSCT?
yes
no
Is LSCT Entry Excluded ?
yes
no
Change the LSCT Entry to a DEF and Set the New REF Flag
Set Exclude Flag
Another DEF ?
yes
no
Pick Up REF Count for Current Module
Is New REF Flag Set ?
no
yes
Is Exclude Flag Set?
yes
no
Enter Module Number into Load List
Skip REFs of This Module
Another REF?
no
yes
Initialize Exclude and New REF Flags
Fetch the Next REF of the Module
Another Module ?
yes
no
Is REF in the LSCT?
yes
no
New Entries in the LSCT?
no
yes
End Search
Enter REF into the LSCT with New Bit Set
Reset New Bit on All LSCT Entries
Initialize to Beginning of DEF/REF File

Figure 57. Building the Library Module Load List

BLOAD

The BLOAD loader facilitates the loading of large load modules by providing a paging mechanism for the core image of the object programs. BLOAD is syntax-compatible with OLOAD, but it does not have the capability of creating a Public Library. BLOAD has lower core requirements than OLOAD and is relocatable. BLOAD creates a load module one granule at a time, trading speed for the ability to load program segments larger than the available loading space.

The following discussion and flow charts detail the major differences between BLOAD and OLOAD. The tree structure of BLOAD is shown in Figure 58 and the flow charts are shown in Figures 59 through 66.

Segment numbering of BLOAD maintains the functional correlation of OLOAD as far as possible; this requires fewer modifications in the OLOAD sequence of calls to M:SEGLD. Also, BLOAD overlays 1-5 perform the same primary functions as OLOAD overlays 1-5. The significant differences are discussed below.

OLOAD overlays 1 and 2 contain duplicate code for reading, printing, and scanning control commands. This code is consolidated in BLOAD overlay 20.

OLOAD overlay 2 is preceded by routines to store load module data and insert symbol table entries required by overlays 3 and 4 (3 and 4 are biased high enough to avoid overlaying these routines). These routines are contained in BLOAD overlay 10.

BLOAD overlay 30 contains M:ASSIGN and M:READ FPTs and loader tables required by overlays 3 and 4 but not by overlay 2. These FPTs and tables in OLOAD are also loaded as part of overlay 2.

BLOAD overlay 6 contains the I/O diagnostic messages contained in OLOAD overlay 1.

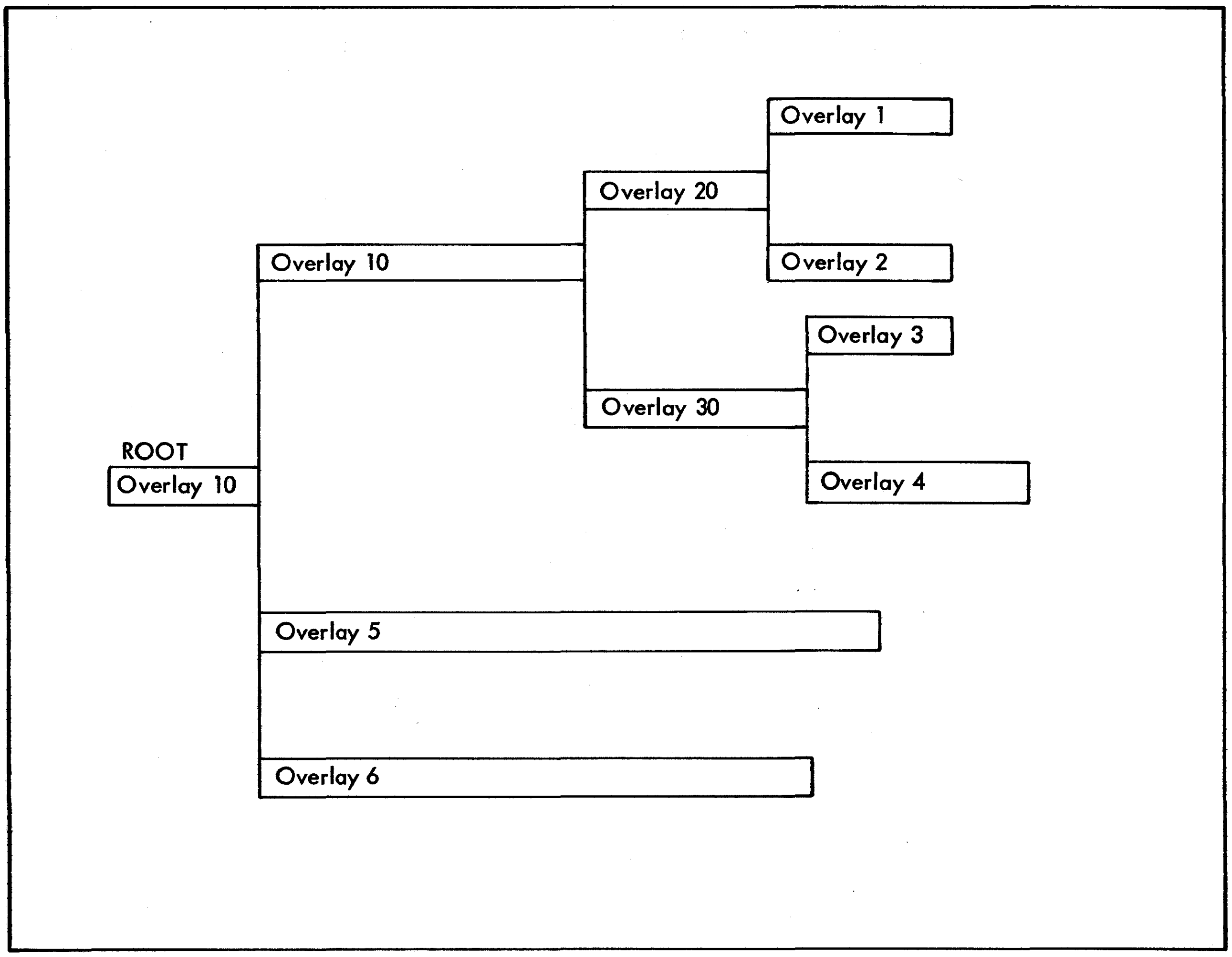


Figure 58. BLOAD Tree Structure

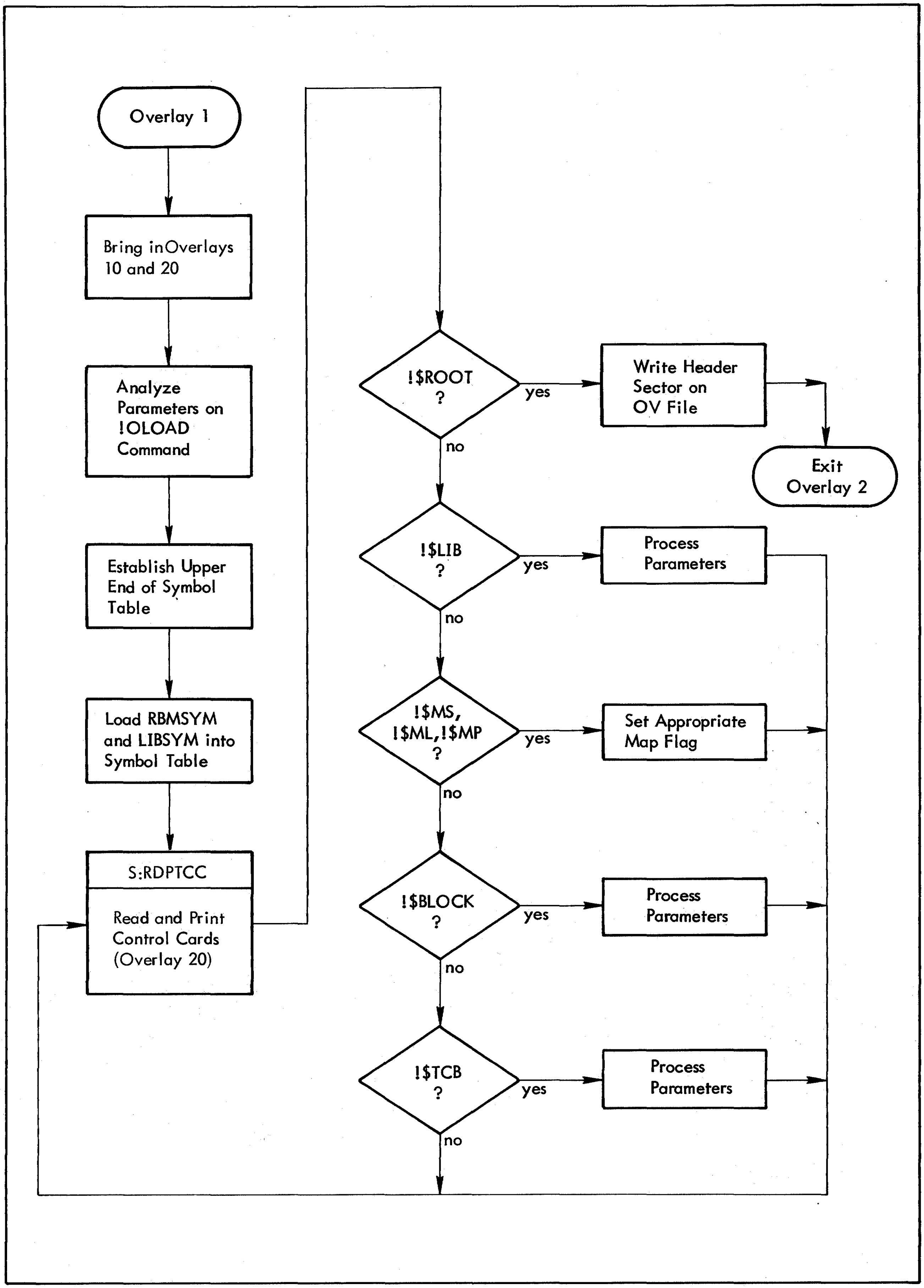

Overlay 1
Bring inOverlays 10 and 20
Analyze Parameters on !OLOAD Command
Establish Upper End of Symbol Table
Load RBMSYM and LIBSYM into Symbol Table
S:RDPTCC
Read and Print Control Cards (Overlay 20)
!$ROOT ?
yes
Write Header Sector on OV File
Exit Overlay 2
no
!$LIB ?
yes
Process Parameters
no
!$MS, !$ML,!$MP ?
yes
Set Appropriate Map Flag
no
!$BLOCK ?
yes
Process Parameters
no
!$TCB ?
yes
Process Parameters
no

Figure 59. BLOAD Overlay 1 Structure

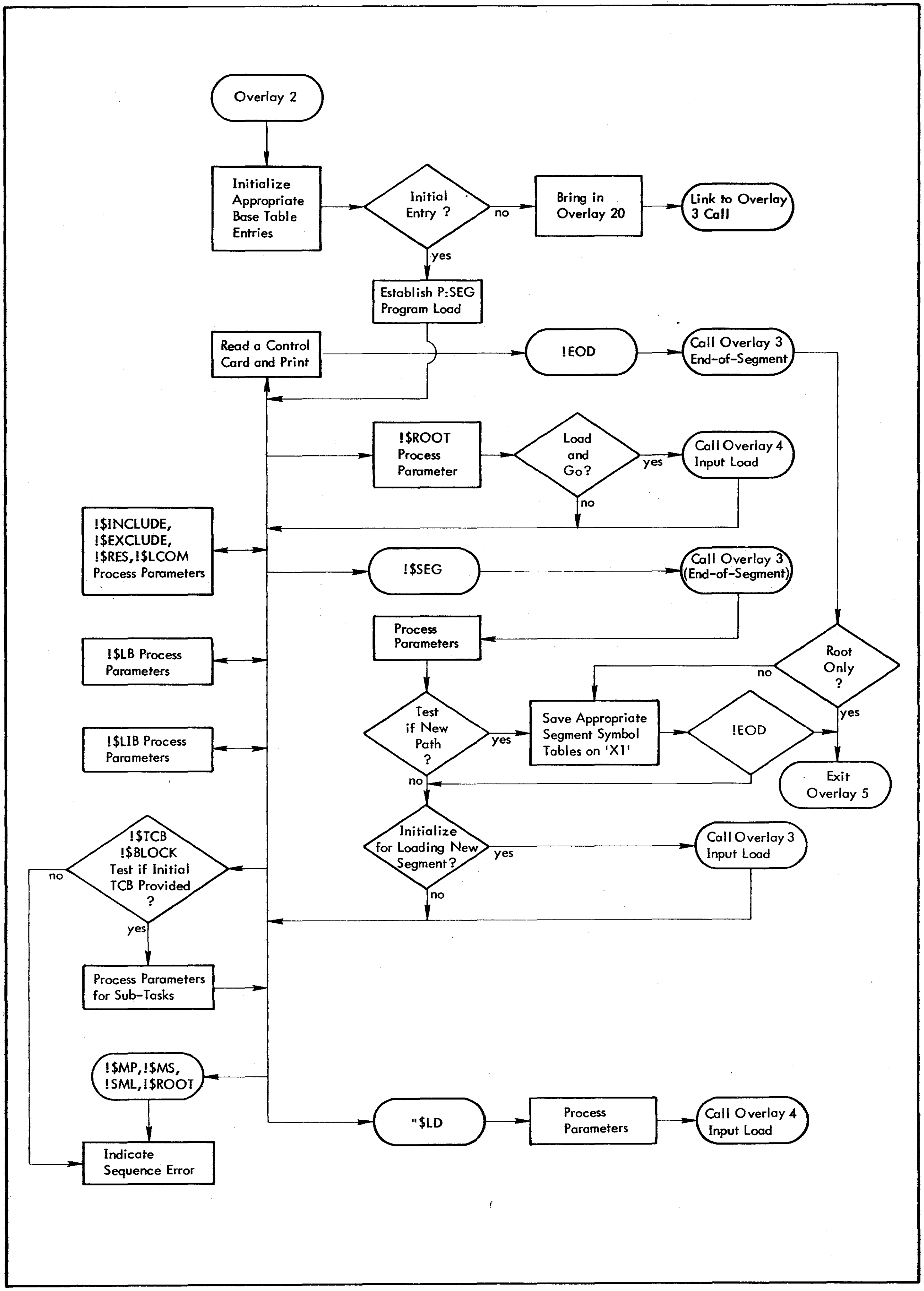

Overlay 2
Initialize Appropriate Base Table Entries
Initial Entry ?
no
Bring in Overlay 20
Link to Overlay 3 Call
yes
Establish P:SEG Program Load
Read a Control Card and Print
!EOD
Call Overlay 3 End-of-Segment
!$ROOT Process Parameter
Load and Go?
yes
Call Overlay 4 Input Load
no
!$INCLUDE, !$EXCLUDE, !$RES,!$LCOM Process Parameters
!$SEG
Call Overlay 3 (End-of-Segment)
Process Parameters
Root Only ?
no
yes
!$LB Process Parameters
Test if New Path ?
yes
Save Appropriate Segment Symbol Tables on 'X1'
!EOD
Exit Overlay 5
!$LIB Process Parameters
no
!$TCB !$BLOCK Test if Initial TCB Provided ?
no
yes
Initialize for Loading New Segment?
yes
Call Overlay 3 Input Load
no
Process Parameters for Sub-Tasks
!$MP,!$MS, !SML,!$ROOT
"$LD
Process Parameters
Call Overlay 4 Input Load
Indicate Sequence Error

Figure 60. BLOAD Overlay 2 Structure

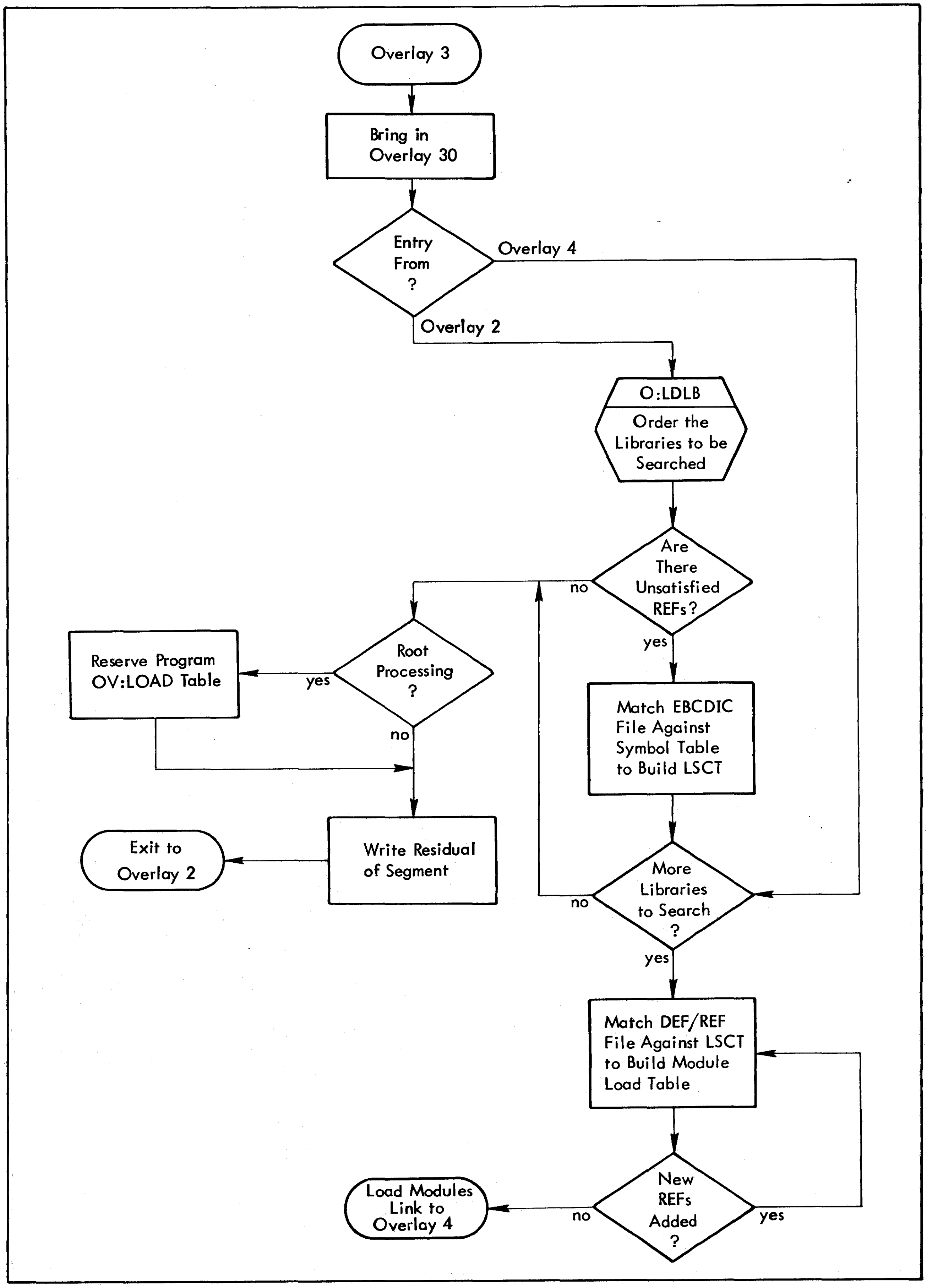

Overlay 3
Bring in Overlay 30
Entry From ?
Overlay 4
Overlay 2
O:LDLB
Order the Libraries to be Searched
Are There Unsatisfied REFs?
no
yes
Root Processing ?
yes
no
Reserve Program OV:LOAD Table
Match EBCDIC File Against Symbol Table to Build LSCT
Write Residual of Segment
Exit to Overlay 2
More Libraries to Search ?
no
yes
Match DEF/REF File Against LSCT to Build Module Load Table
New REFs Added ?
no
yes
Load Modules Link to Overlay 4

Figure 61. BLOAD Overlay 3 Structure

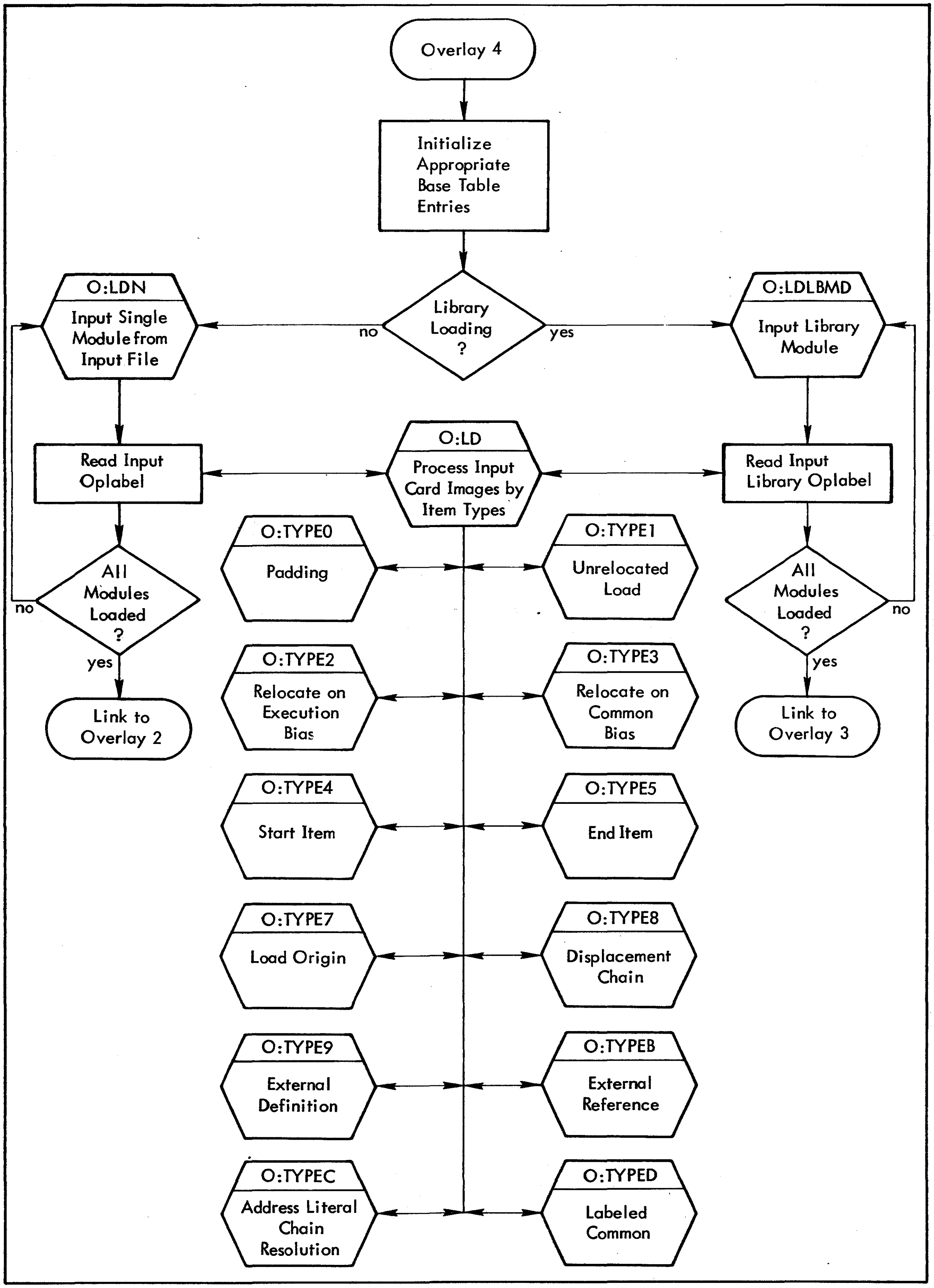


Figure 62. BLOAD Overlay 4 Structure

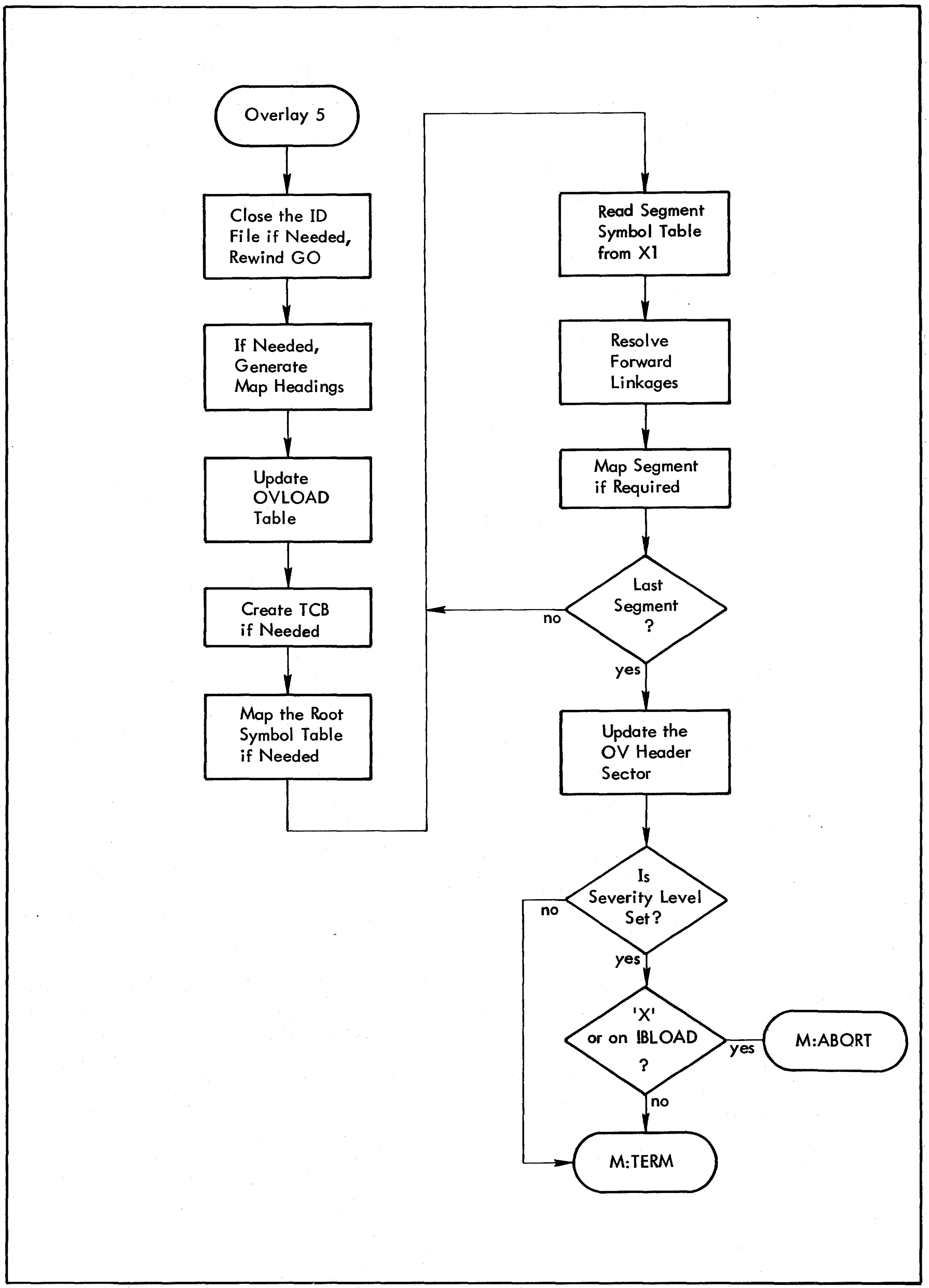


Figure 63. BLOAD Overlay 5 Structure

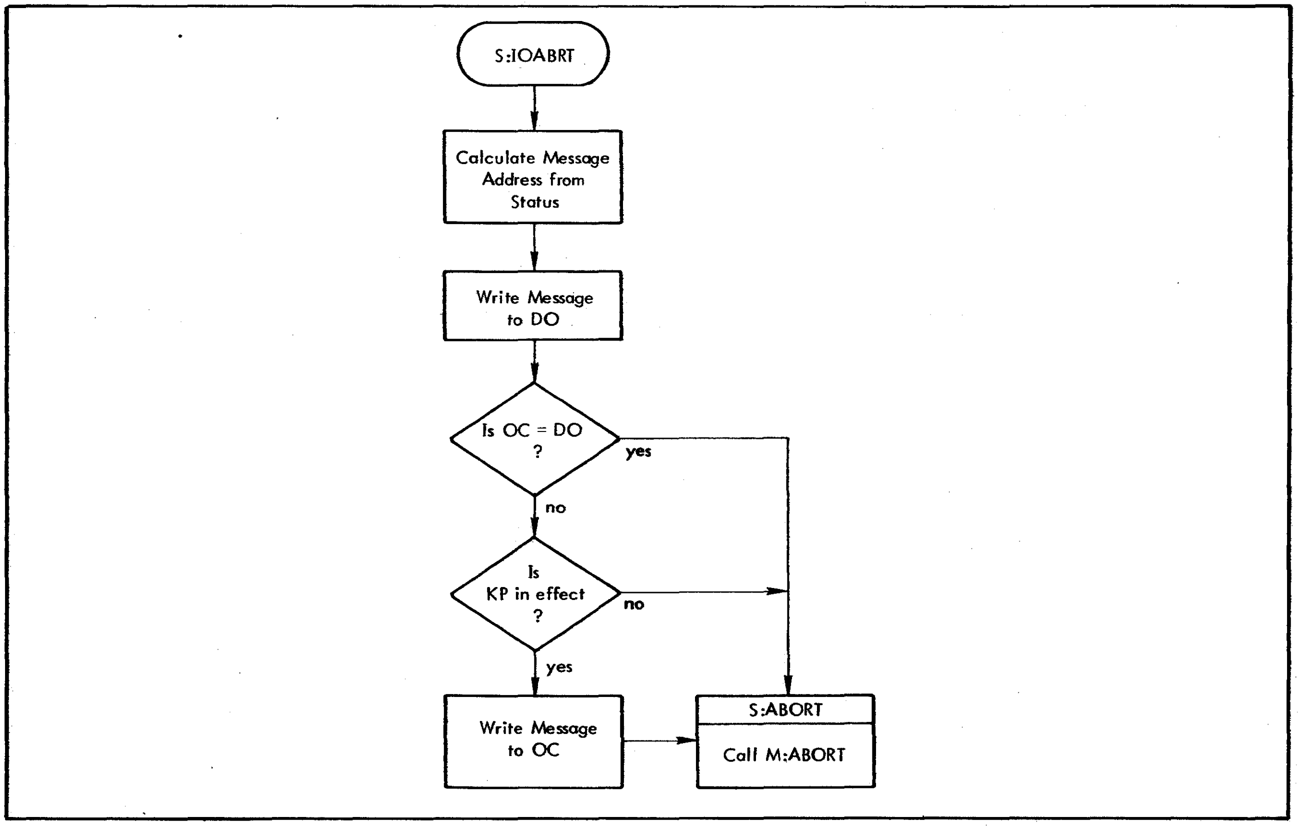


Figure 64. BLOAD Overlay 6 Structure

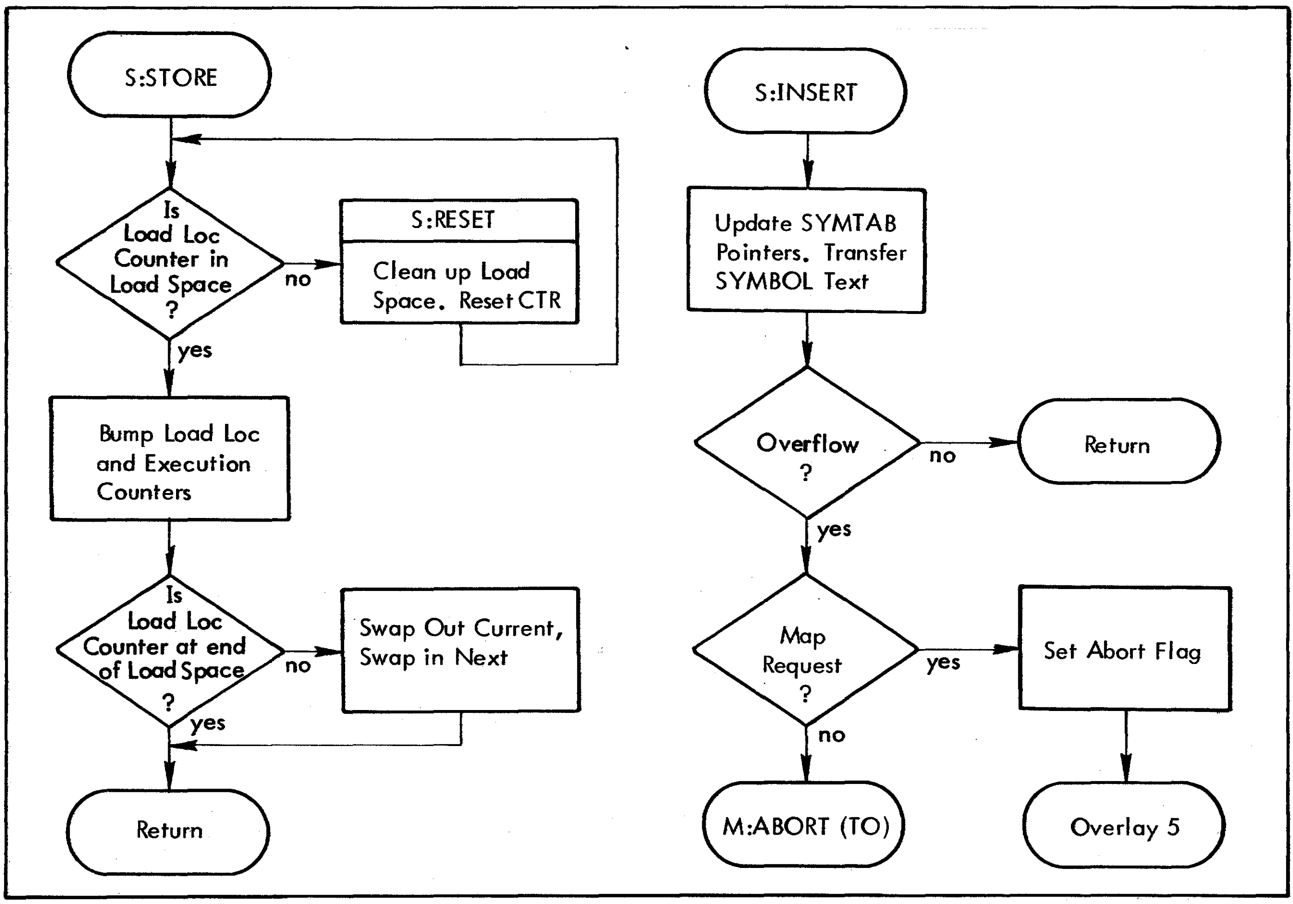

S:STORE
Is Load Loc Counter in Load Space ?
no
S:RESET
Clean up Load Space. Reset CTR
yes
Bump Load Loc and Execution Counters
Is Load Loc Counter at end of Load Space ?
no
Swap Out Current, Swap in Next
yes
Return
S:INSERT
Update SYMTAB Pointers. Transfer SYMBOL Text
Overflow ?
no
Return
yes
Map Request ?
yes
Set Abort Flag
no
M:ABORT (TO)
Overlay 5

Figure 65. Overlay 10 Structure

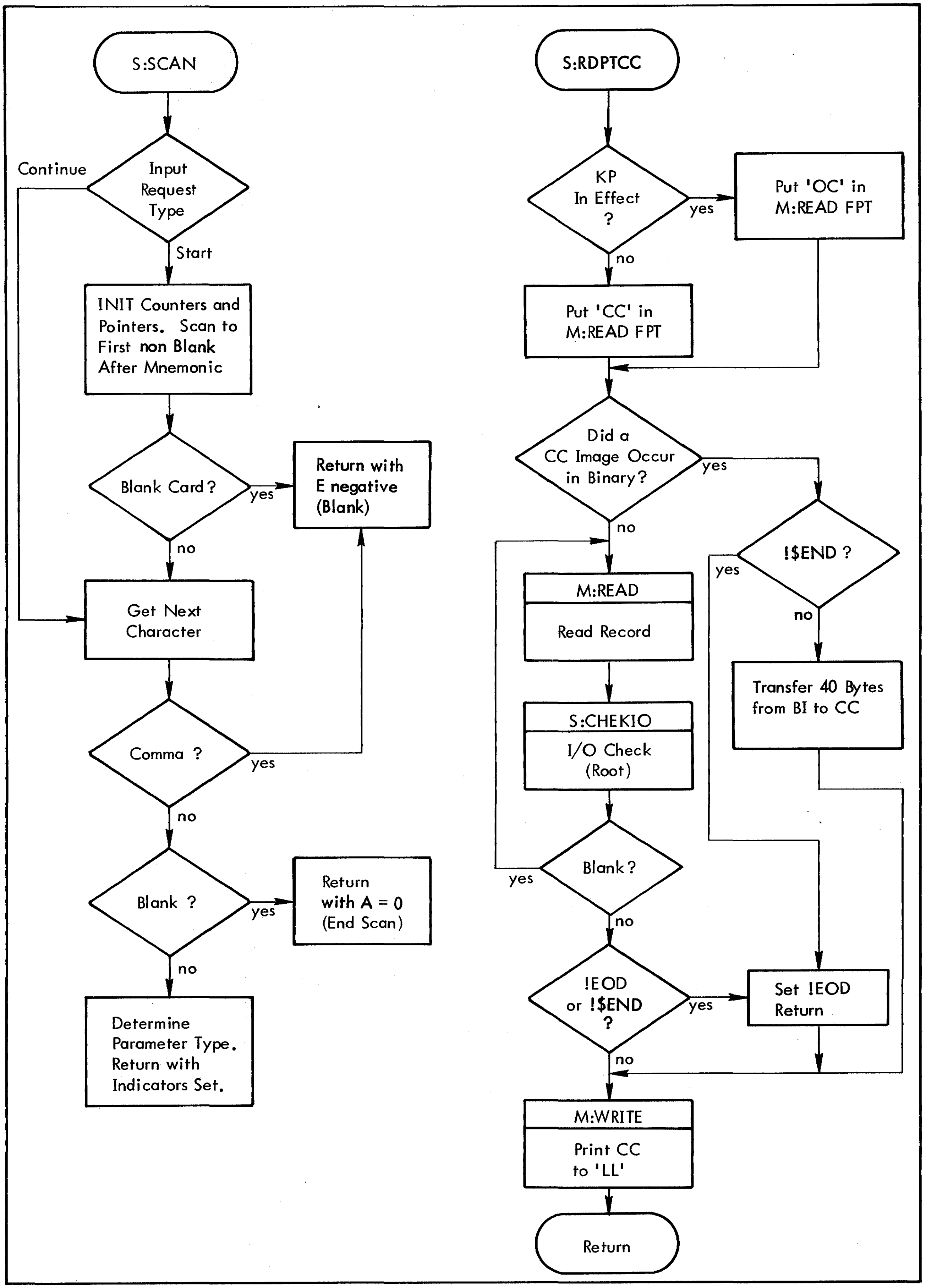


Figure 66. BLOAD Overlay 20 Structure

5. RADEDIT

Each current file area is defined by an entry in the RBM Master Directory. The relationship of the Master Directory, the file directory, and their corresponding files is shown in Figure 67.

The first file directory begins in sector 1, relative to the beginning of the disk or disk pack area. Word 2 of the Master Directory entry for an area contains the sector address of the first (label) sector of the area.

The first two words of every directory sector contain an identification entry, with the form:

<table>
<tr><td>0</td><td>C</td><td colspan="15">NEP or NFD</td></tr>
<tr><td>1</td><td colspan="16">NAS</td></tr>
<tr><td></td><td>0</td><td>1</td><td colspan="13"></td><td>15</td></tr>
</table>

where

C is the file directory sector for this area (0 = last file directory sector, 1 = not the last file directory sector).

NEP is the word offset (if C = 0) to the word following the last entry in this directory.

NFD is the sector offset (if C = 1) from the beginning of area to the next file directory sector for this area.

NAS is the next available sector (if C = 0) in the area (relative to the beginning of area) for the addition of a new file.

Each subsequent nine (K:FDSIZE) words define a file in the area as follows:

<table>
<tr><td>0</td><td colspan="8">n1</td><td colspan="8">n2</td></tr>
<tr><td>1</td><td colspan="8">n3</td><td colspan="8">n4</td></tr>
<tr><td>2</td><td colspan="8">n5</td><td colspan="8">n6</td></tr>
<tr><td>3</td><td colspan="8">n7</td><td colspan="8">n8</td></tr>
<tr><td>4</td><td colspan="2">F</td><td colspan="2">WP</td><td>C</td><td>T</td><td colspan="2"></td><td>RF</td><td colspan="6"></td><td>SR</td></tr>
<tr><td>5</td><td colspan="16">Record Size</td></tr>
<tr><td>6</td><td colspan="16">BOT (First Sector of File)</td></tr>
<tr><td>7</td><td colspan="16">EOF</td></tr>
<tr><td>8</td><td colspan="16">EOT (Last Sector of File + 1)</td></tr>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td colspan="5"></td><td>14</td><td>15</td></tr>
</table>

where

n1 – n8 is the name of the file (three to eight nonblank EBCDIC characters followed by blank EBCDIC characters to make a total of eight characters).

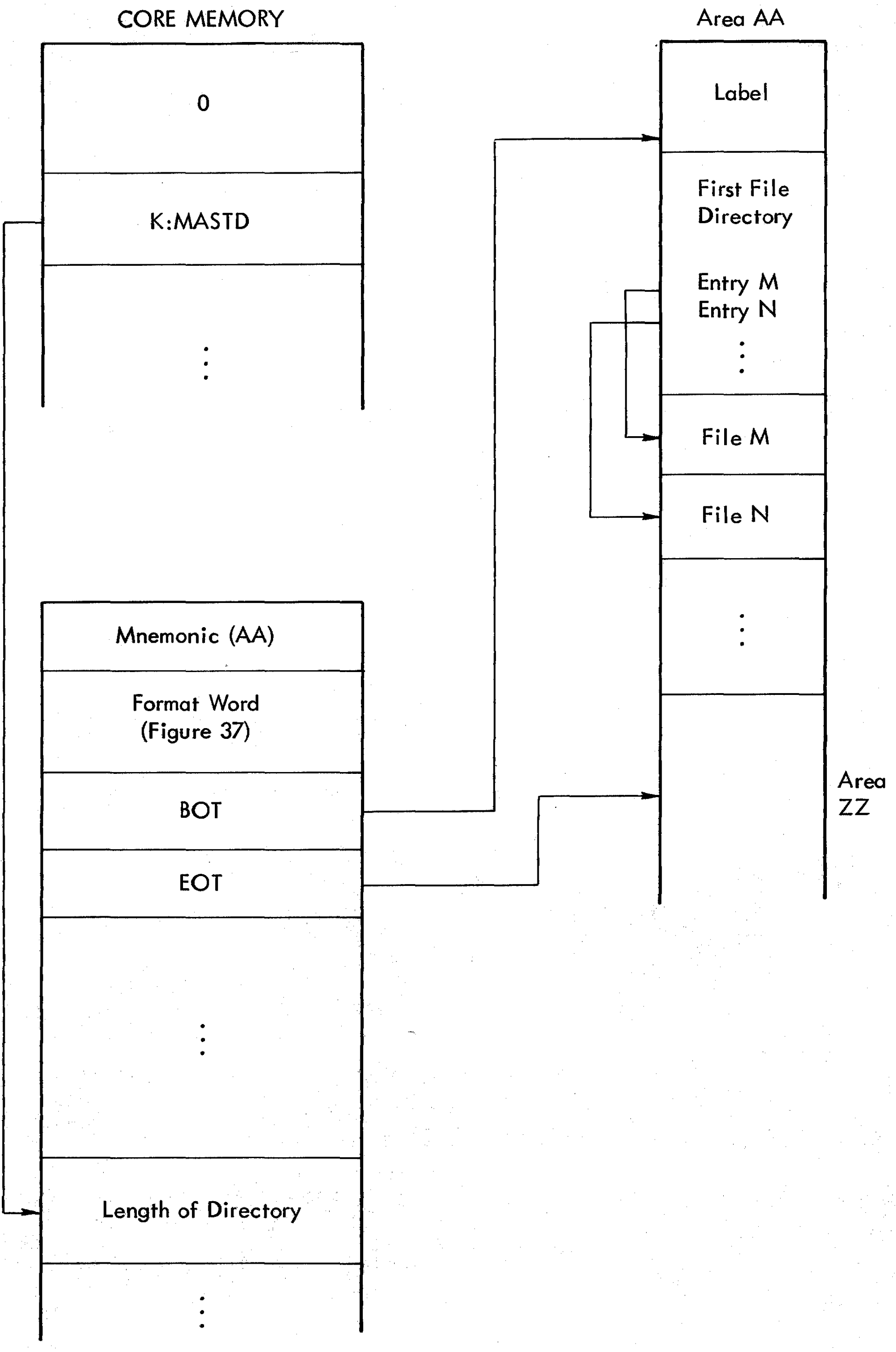


Figure 67. RBM File Structure

F is the format of the file

0 = unblocked.

1 = blocked.

2 = compressed.

3 = packed.

4 = random.

WP is the write-protection code for the file

0 = no protection.

1 = write permitted by background only.

2 = write permitted by foreground only (unless SY key-in is in effect).

3 = write permitted by RBM only (unless SY key-in is in effect).

C is a cylinder maintenance flag which indicates (if C = 1) that the BOT of the file is to be maintained on a cylinder boundary.

T is a track maintenance flag which indicates (if T = 1) that the BOT of the file is to be maintained on a track boundary.

RF is the resident foreground program flag for files in SP, UP, or FP areas. If set, the program is loaded by RBM at system boot time.

SR is the short record flag (see Disk File Directory).

Record Size is the number of words in a record.

BOT is the absolute sector address of the first sector defined for the file.

EOF is the pointer to the logical or pseudo file mark. If EOF = -1, no file mark has been written.

EOT is the absolute sector address of the last sector plus one defined for the file.

If a directory entry is deleted or empty, every word of the entry contains zeros. No entry extends over a file directory sector boundary.

Library File Formats

The System Library area and User Library area both have the same structure. Each contains six files: the Module Directory File (MODIR), the Library Module File (MODULE), the main DEF/REF File (MDFRF), the Extended DEF/REF File (EDFRF), the Basic DEF/REF File (BDFRF), and the EBCDIC File (EBCDIC). These files must be defined via !#ADD commands before attempting to generate them via !#LADD commands.

MODIR File

The MODIR file is random access, where each sector contains an integral number of six-word entries.

Entry 0 of the MODIR file contains the following identification entry:

0	Word Count
1	Next Available Module Sector
2 ⋮ 5	Unused
	0 15

where

Word Count is the number of active words in the MODIR file. The word count is 6n, where n is the number of entries in the file (including entry 0).

Next Available Module Sector is the relative sector within the MODULE file available for storing the next object module.

Each subsequent entry of the file contains MODIR entries numbered through n. A MODIR entry contains:

0	Lib	MODULE Record Number
1	Relocatable Length	
2 ⋮ 5	Identification	
	0 1	2 15

where

Lib indicates which DEF/REF (xDFRF) file contains the external definitions and references for the module (11 = MDFRF file, 10 = BDFRF file, 01 = EDFRF file, 00 = entry has been deleted).

MODULE Record Number is the relative sector within the MODULE file where the object module begins.

Relocatable Length is the relocatable length of the object module in the MODULE file.

Identification is the name from the start item of the object module beginning at MODULE sector number.

A deleted or empty MODIR entry contains all zeros.

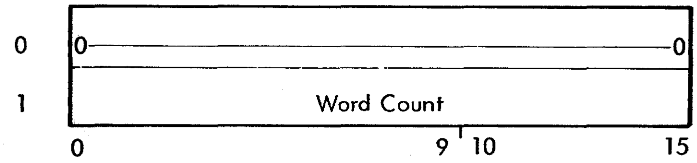
MODULE File

The MODULE file is a packed random access file containing object modules. The MODIR file acts as a directory to the object modules contained in the MODULE file. Each entry in the MODULE file is an object module. The 120-byte card images of the object module are blocked by RADEDIT.

MDFRF, BDFRF, and EDFRF Files

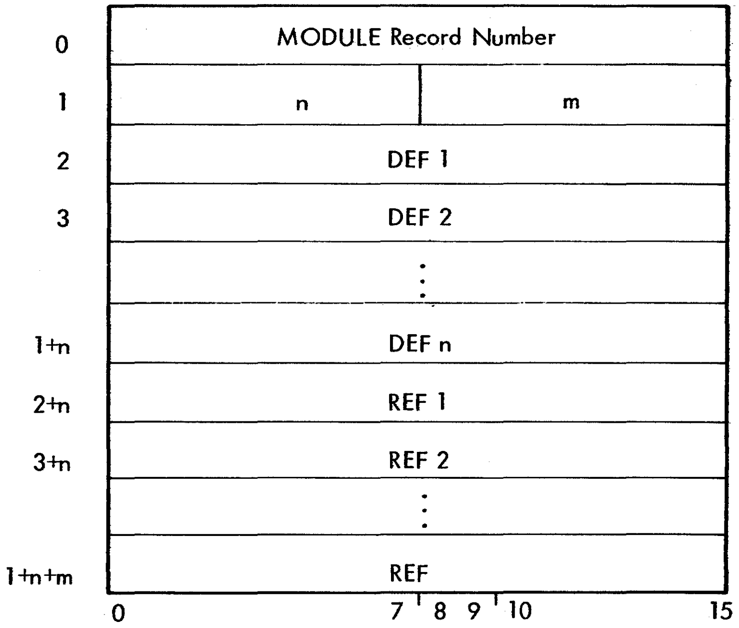
The MDFRF, BDFRF, and EDFRF files (xDFRF files) all have the same format. The files are random access with variable length entries.

Entry 0 in the file contains an identification entry as follows:



where Word Count is the number of active words in the file including entry 0. If no entries have been placed in the file, it is zero. This count can be used to compute the sector access and relative position within that sector where the next DEF/REF entry can be stored.

The remaining entries in the file are called DEF/REF entries. A DEF/REF entry never extends over a sector boundary. Empty entries are used to pad sectors. Each entry contains.



where

MODULE Record Number is the record number of the first record of the ROM in the MODULE file. If the MODULE entry number is -1, the entry is empty.

DEF_i is the entry number of an external definition symbol in the EBCDIC file.

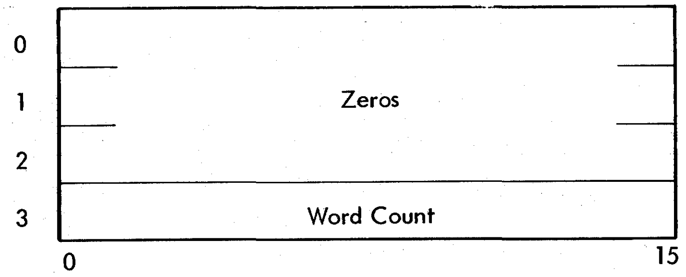
REF_i is the entry number of an external reference symbol in the EBCDIC file.

An empty or deleted DEF/REF entry contains a MODIR Entry No. of zero, and Entry Size is the length of the padding entry.

EBCDIC File

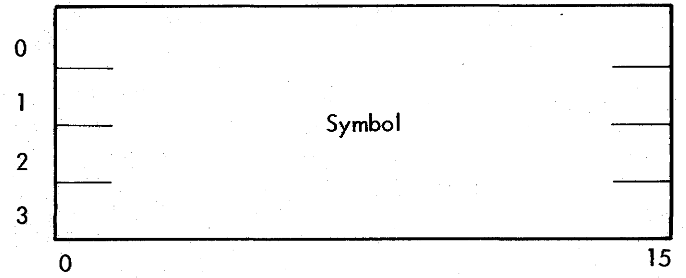
The EBCDIC file is a random access file where each entry contains four words. Every sector contains an integral number of entries.

Entry 0 is an identification entry that contains



where Word Count is the number of words in the EBCDIC file (including entry zero). The word count is 4n, where n is the number of entries in the file (including entry 0).

Each subsequent entry in the file is called an EBCDIC entry. Each contains:



where Symbol is an external definition or reference in EBCDIC, left-justified with trailing blanks.

Overlay Structure

The RAD Editor consists of a root segment plus 12 overlay segments. Each segment contains one object module. Before a command is executed, the appropriate segment(s) is loaded via M:SEGLD. Control is transferred to the command execution segment by the RAD Editor; not by M:SEGLD.

The overlay tree structure showing the function and segment number of each object module is illustrated in Figure 68.

Control Command Execution

RBM loads and transfers control to the RAD Editor upon reading a !RADEDIT command from CC. The Executive routine (R:EXEC) of the RAD Editor gains control. It initializes all flags and pointers and reads a control command from CC and scans it. The Execute Command routine (R:CMDEX) loads the overlay segments needed (if they are not

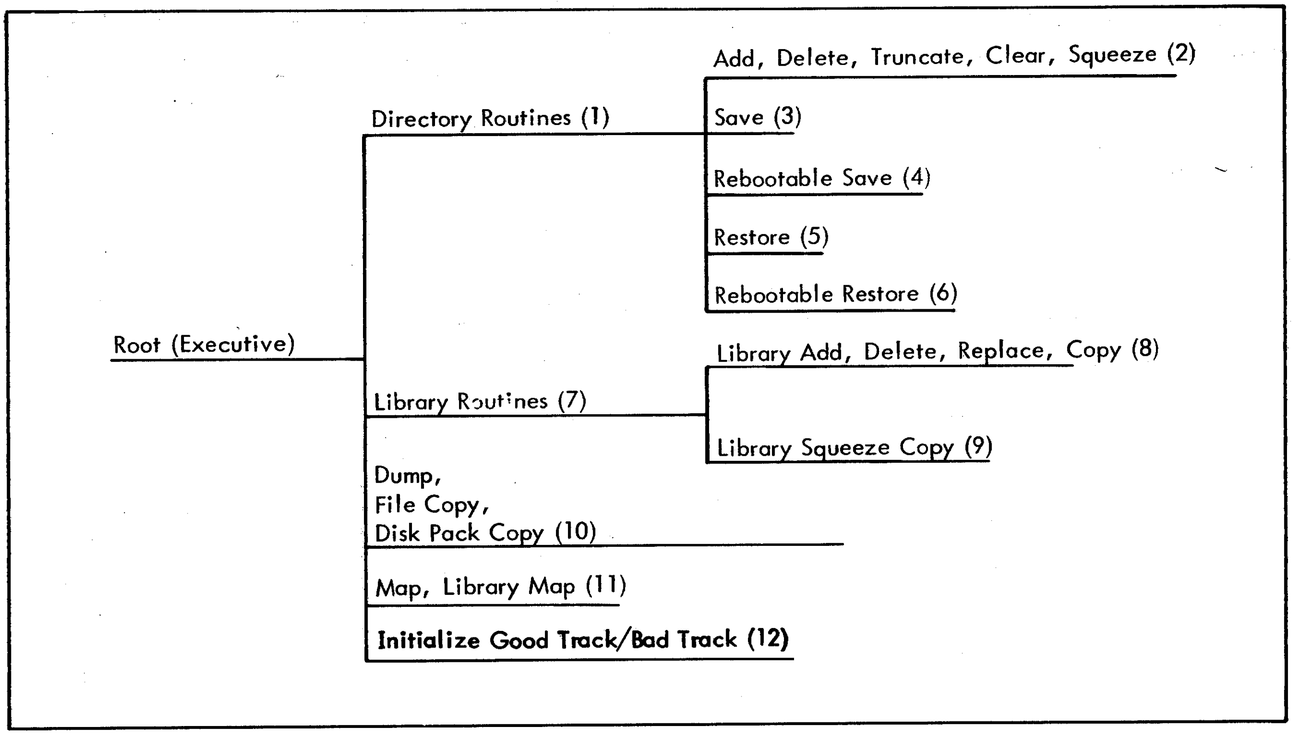


Figure 68. RADEDIT Tree Structure

already core and transfers control to the routine to process the command. On encountering a !#END command or an EOF status, (!EOD), the RAD Editor terminates by calling the Monitor service routine (M:TERM) which returns control to the Monitor. A functional flow diagram is shown in Figure 69.

For each command the root segment initializes the base table, reads and scans the control command, loads the segment(s) required for command segment for command execution (if necessary), and transfers control to the appropriate segment for command execution. Routines commonly referenced by other segments are also included in the root. The base table included in the root segment contains the addresses of the entry points to all routines in the root plus the addresses of error messages. Storage in the base table is also provided for flags, file entries, directory entries, and constants.

The entry points to routines included in the root segment and their functions are given in Table 1.

Area Maintenance Commands

The permanent file directories are maintained so that the directory entries in the permanent file directory appear in the same order as the actual files (i.e., the BOT in each directory entry is greater than the BOT in the previous entry). This ordering of entries and files facilitates maintenance, particularly execution of the !#SQUEEZE command.

To preserve this ordering of entries and files, the !#ADD, !#DELETE, !#CLEAR, !#TRUNCATE, and !#SQUEEZE commands must be executed as follows:

#ADD Command

The "area" parameter on the !#ADD command is used to determine the active area to be updated.

The "name", "fsize", "rsize", "format", "wp", "foreground", and "maintenance" parameters are used to form words 0, 1, 2, 3, 4, and 5 of a new directory entry. The EOF (word 7) is set to -1. The new entry is added to the

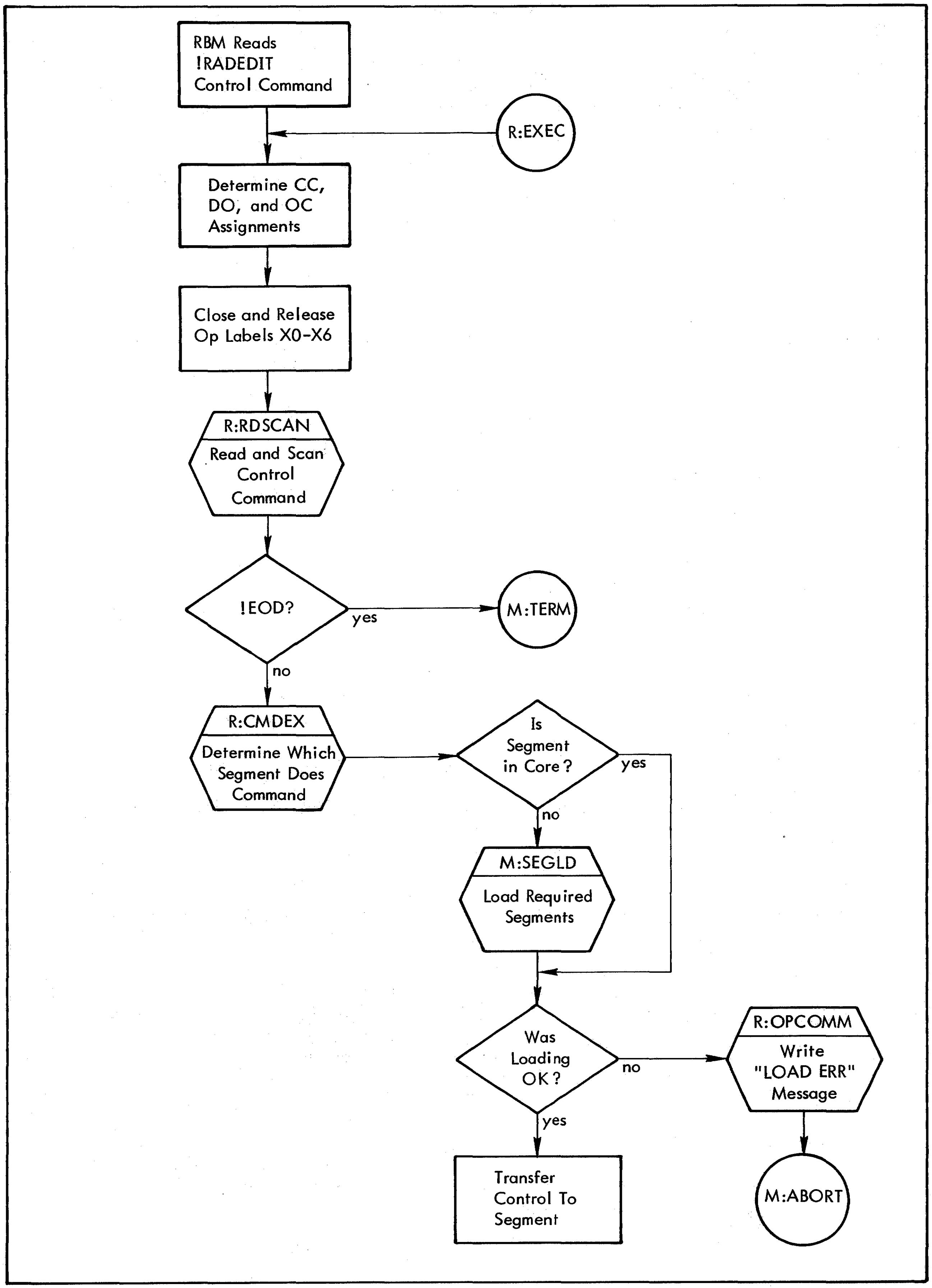

RBM Reads !RADEDIT Control Command
R:EXEC
Determine CC, DO, and OC Assignments
Close and Release Op Labels X0-X6
R:RDSCAN
Read and Scan Control Command
!EOD?
yes
M:TERM
no
R:CMDEX
Determine Which Segment Does Command
Is Segment in Core?
yes
no
M:SEGLD
Load Required Segments
Was Loading OK?
no
R:OPCOMM
Write "LOAD ERR" Message
yes
Transfer Control To Segment
M:ABORT

Figure 69. Control Command Execution Flow

Table 1. RADEDIT Root Segment Entry Points

Entry Point	Routine Name	Function
R:EXEC	RADEDIT Executive	Controls operation of the Editor and is entered to begin execution of the Editor.
R:CHKIO	Check I/O	Checks I/O status and, if necessary, writes appropriate error message. This routine is called after performing every I/O operation.
R:CLOSE	Close	Closes the indicated file in order to release the device file number assigned to it.
R:CMDEX	Command Execution	Determines which segment(s) are needed for command execution, loads the required segments, and calls the appropriate routine for actual command execution.
R:GDIR	Get Directory Entry	Gets the next entry from a permanent file directory.
R:GPAR	Get Next Parameter	Gets the address of the next entry in the parameter table created by R:RDSCAN.
R:OPCOMM	Operator Communication	Writes, solicits, and receives messages on the OC device.
R:RDSCAN	Read-Scan Command	Reads a control command from CC and creates a parameter table.
R:RADR	RAD Read	Reads a sector from a random access RAD file or file directory. This routine is called to read all RAD files.
R:RADW	RAD Write	Writes and check-writes a sector on a random access RAD file or file directory. This routine is called to write all RAD files.
R:RBI	Read BI	Reads a standard 120-byte binary card image from BI.

directory sector having an identification entry with C = 0 and is stored in the directory sector at NEP. The BOT (word 6) of the new entry is set equal to the next available sector (unless C or T is specified as a "maintenance" parameter). The EOT (word 8) of the new entry is computed from the "record" and "file" parameters. After the new entry has been added, the next available sector (word 1 of the last file directory sector) is set equal to the EOT of the new entry.

If the C or T is specified as a "maintenance" parameter, the BOT is set to the next available cylinder (C) or track (T) boundary.

To complete the !#ADD execution, the identification entry of the directory sector is updated. If there is room in the directory sector for another entry, the address in the identification entry is incremented by 9 (K:FDSIZE). If there is no room in the directory sector for another entry, the C-bit in the identification entry is set to 1, a new file directory sector is reserved, the volume sector number of the new file directory sector is stored as NFD, and the new file directory sector is initialized with the new volume of NAS.

#DELETE Command

The "area" parameter on the !#DELETE command is used to determine the active area to be updated. The "name" parameter is used to search the directory for the directory entry to be deleted. The file (defined by the BOT and EOT parameters) is deleted by zeroing out the directory entry. The space formerly occupied by the files becomes unused until a !#SQUEEZE command is executed. The deleted space is automatically recovered if the entry is the last one in the file directory.

#CLEAR Command

This command clears areas and/or files. Files are cleared by assigning the X0 oplabel to the area and then zeroing the file with a maximum byte count. Areas are cleared by assigning the X0 oplabel to the area and then zeroing out the area with a maximum byte count.

#TRUNCATE Command

This command is identical to !#DELETE, except that instead of the directory entry being zeroed out, the EOF is used to compute a minimum EOT value to replace the current EOT. The space between the new EOT and old EOT becomes unused until a !#SQUEEZE command is executed. The truncated space is automatically recovered if the entry is the last one in the file directory.

#SQUEEZE Command

The "area" parameters on the !#SQUEEZE command determine which permanent RAD areas to initialize. Executing a !#DELETE command causes the part of the permanent RAD area where directory entries are files were previously located to be lost from use (except when the file being deleted is the last file in the area). Executing !#ADD commands causes new entries and files to be added without attempting to regain any unused space. Squeezing eliminates these unused portions of the permanent RAD area. The directories are compacted and the files themselves are moved to regain these unused spaces. If C or T (cylinder or track) maintenance is specified for a file, it is moved to the next available cylinder or track boundary above the previously "squeezed" file. The BOT and EOT in the directory entries are updated as they are compacted to indicate the area occupied by the moved file. Figures 70 and 71 show a file area before and after squeezing.

Area Maintenance Routines

Segment 1 contains the common routines necessary for execution of area maintenance commands. The entry points, routine names, and functions are defined in Table 2.

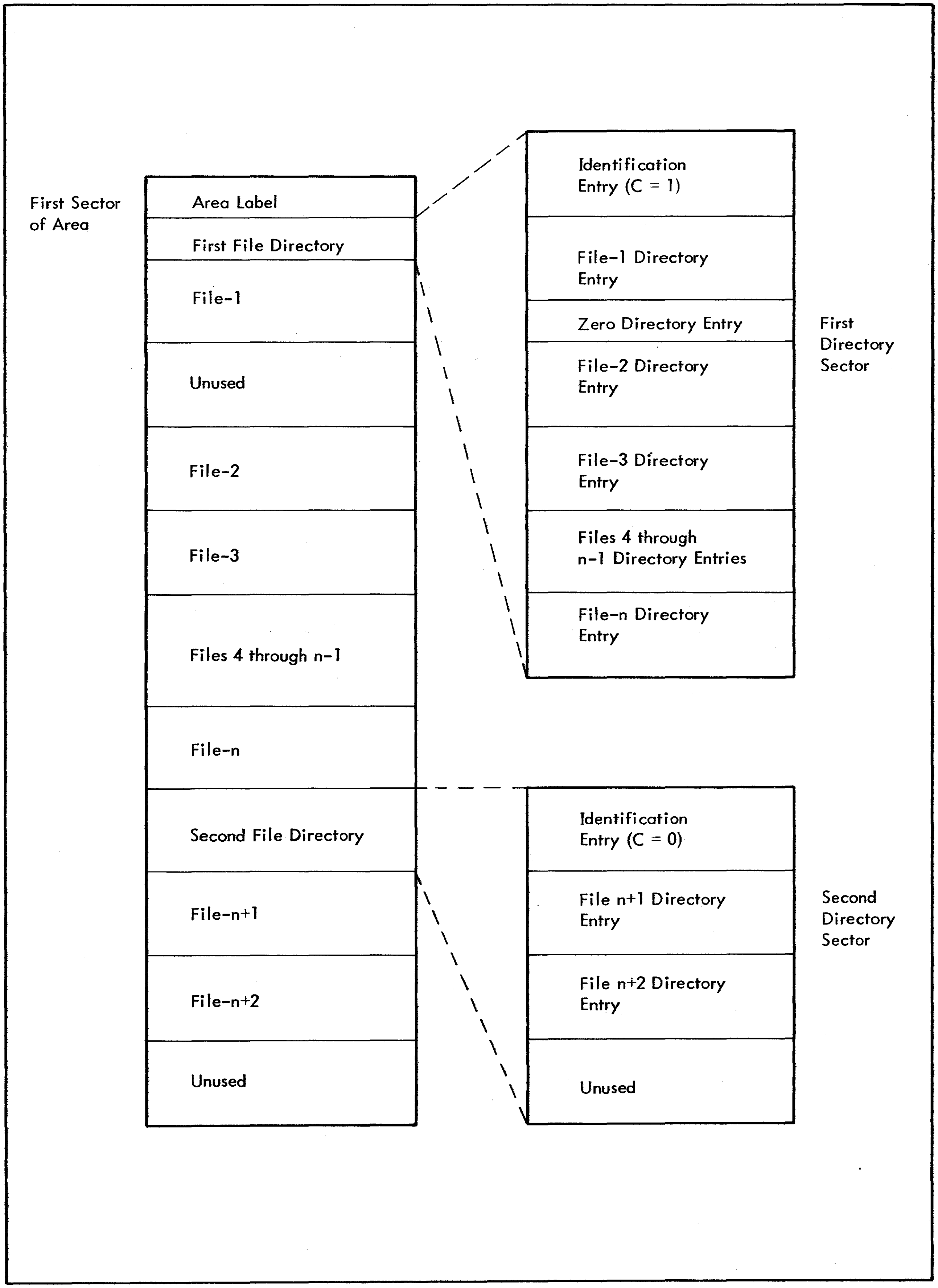


Figure 70. File Area Before Squeeze

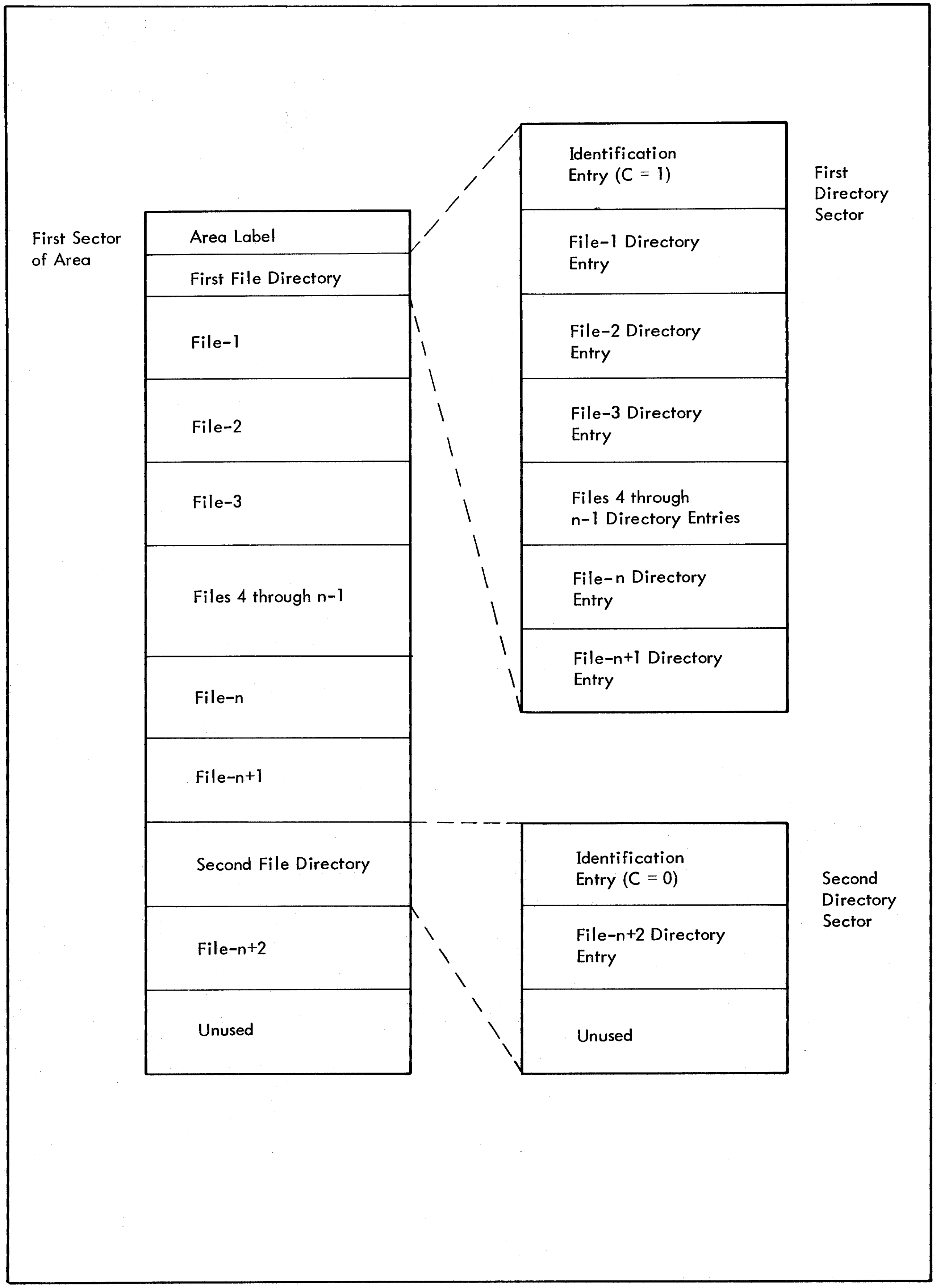


Figure 71. File Area After Squeeze

Table 2. Area Maintenance Routines

Entry Point	Routine Name	Function
R:DSRCH	Directory Search	Searches a permanent file directory for an entry having a designated name.
PDN	Process Directory and Name Parameters	Processes the "directory" and "name" parameters on an !#ADD, !#DELETE command by utilizing the parameter table created by R:RDSCAN.
R:SETY	Store Entry	Completes a directory entry by determining the BOT and EOT for the file it describes and stores the entry in the permanent file directory.
R:GNF	Get Next File	Gets the next "area/filename" from the control command.
PROTEST	Protection Test	Tests the current area protection code unless an SY key-in is in effect.

Library File Maintenance Commands

The library files are maintained through the execution of !#LADD, !#LREPLACE, and !#LDELETE commands. The entries in the MODIR file, MODULE file, and xDFRF files are all ordered the same way. The ith entry in the MODIR file identifies the ith object module in the MODULE file. The jth MODIR entry referencing a particular library corresponds to the jth entry in the xDFRF file for that library. The MODULE file is in a packed random format.

To preserve the ordering of these files, the !#LADD, !#LREPLACE, !#LDELETE, and !#LSQUEEZE commands must be executed as follows:

#LADD Command

The "area" parameter on the !#LADD command determines which active area contains the library files to be updated. For each object module added the following procedure is used:

- The "library" parameter determines the setting of the "lib" bits (word 0) in the new MODIR entry and which one of the xDFRF files to update.
- The MODULE record number (word 0) of the new MODIR entry is set equal to the "next available MODULE record" (MODIR identification entry). The remaining information stored in the library files is obtained from the object module read from BI.
- The object module is placed on the MODULE file beginning the "next available MODULE record".
- The identification and relocatable length are obtained from the module and stored to complete the MODIR entry.

The symbols for each external definition and reference in the object module are extracted and stored as entries in the EBCDIC file (if they are not already stored there). The entry number of the EBCDIC entry for each external definition and reference is saved to create the "DEF_i" and "REF_i" words of the DEF/REF entry written on the xDFRF file. The addition of the object module to the library is completed by updating the identification entries in the MODIR, MODFRF, BDFRF, EDFRF, and EBCDIC files.

#LDELETE Command

The "area" parameter on the !#LDELETE command determines which active area contains the library files to be updated. The MODIR entry containing the same "identification" indicated on the !#LDELETE command is zeroed out. The information in the MODIR entry is used to zero out the DEF/REF entry in the xDFRF file indicated by the "library" parameter. No changes are made to the identification entries in the MODIR and xDFRF files. No changes whatsoever are made to the EBCDIC file or MODULE file as the result of !#LDELETE command execution.

#LREPLACE Command

The "area" parameter on the !#LREPLACE command determines which active area contains the library files to be updated. The procedure followed is identical to executing a !#LDELETE command followed by a !#LADD command, where both commands have the same parameters.

#LSQUEEZE Command

The "area" parameter on the !#LSQUEEZE command determines which active area contains the library files to be squeezed. The command saves the MODIR file in core, clears the MODIR, EBCDIC, and xDFRF files, and then recreates them by performing a ! #LADD function using the old MODULE file as input.

Library File Maintenance Routines

Segment 7 contains the routines necessary for maintaining and referencing the library files via !#LADD, !#LREPLACE, !#LDELETE, and !#LCOPY commands. Routines in the root segment are referenced. The entry points to the routines in this segment are shown in Table 3.

Table 3. Library File Maintenance Routines

Entry Point	Routine Name	Function
R:ASSIGN	Assign Library File	Assign a designated library file to an operational label.
R:DRADD	DFRF Add	Place a new entry in either the MDFRF, BDFRF, or EDFRF file.
R:DRDELE	DFRF Delete	Deletes an entry from either the MDFRF, BDFRF, or EDFRF file by chaning the entry to a padding entry.
R:GBIN	Get Binary Card Image	Gets a binary card image from the MODULE file by deblocking sector images.
R:ESRCH	EBCDIC Search	Searches the EBCDIC file for a designated DEF or REF symbol. If the symbol is not found, it is added to the file. The entry number of the symbol is returned.
R:GMD	Get MODIR Entry	Obtains the next entry from the MODIR file.
R:MDADD	MODIR Add	Places a new entry in the MODIR file.

Table 3. Library File Maintenance Routines (cont.)

Entry Point	Routine Name	Function
R:MDDELE	MODIR Delete	Removes an entry from the MODIR file by zeroing it out.
R:MDSRCH	MODIR Search	Searches the MODIR file for an entry having a matching "IDNT" and library-mode (E, B, or M).
R:MOADD	MODULE Add	Reads an object module from BI and writes it on the MODULE file. From data in the module it forms or completes entries to add to the EBCDIC file, MODIR file, and either the MDFRF, BDFRF, or EDFRF file.
R:PLC	Process Library Command Parameters	Processes "area", "identification", and library parameters on the !#LADD, !#LDELETE, !#LREPLACE, and !#LCOPY commands by utilizing the parameter table created by R:RDSCAN.

Utility Commands

Utility functions are included in the RAD Editor to allow

- Dumping random access RAD files.
- Copying the contents from one random access RAD file into another.
- Copying a library routine from the System Library or User Library onto BO.
- Mapping the RAD areas.
- Saving contents of permanent RAD areas or files.
- Restoring the RAD using data saved.
- Initialization of new disk packs.
- Messages to the operator.
- Messages with operator response required.

#DUMP Command

The !#DUMP command outputs data on LO. The file must be random access and is read and dumped one sector at a time.

The format of the dump of each sector is

```
SECTOR    ssss
rrrr      dddd   dddd   ....   dddd
  .         .      .      .      .
  .         .      .      .      .
  .         .      .      .      .
rrrr      dddd   dddd   ....   dddd
```

where

ssss is the relative address (hexadecimal number) of the sector in the file defined by the "oplb" parameter.

rrrr is the relative address (hexadecimal number) of the first data word dumped.

dddd is the image of a data word (a hexadecimal number). The number of data words per line is eight if LO is assigned to a keyboard/printer. Otherwise, 16 data words per line are output.

#FCOPY Command

The !#FCOPY command copies the contents of the random access file assigned to the "input" operational label onto the random access file assigned to the "output" operational label. The data is read and written one sector at a time. When EOT is encountered on either file, the copy terminates.

#LCOPY Command

The !#LCOPY command outputs an object module on the BO device. The object module is found by searching the MODIR file for an entry having an identification matching the identification requested as a command parameter. From the MODULE sector number in the entry, the MODULE file is positioned and card images are deblocked and written on BO.

#MAP Command

The !#MAP command outputs data on LO. The "area" parameters determine which RAD area to map. Each area mapped may cause up to two items to be output. The items or parts of a map are:

1. Information from Master Directory consisting of the directory area identification, its beginning RAD address, and ending RAD address.

2. Information obtained from the permanent file directory about each file in the area, its name, format, write protection, foreground task indicator, logical record size, beginning RAD address, ending RAD address, and end-of-file pointer.

For CP or BT areas mapped, only part 1 is printed; for every area, parts 1 and 2 are printed.

Part 1 of the map has the format

```
AREA   aa   DEV   cc   BOT   bbbb   EOT   tttt
```

where

aa identifies the area.

bbbb is the volume sector address of the beginning of the area. It is in the form of a hexadecimal number.

cc is the device number.

tttt is the volume sector address of the last sector plus one of the area. It is in the form of a hexadecimal number.

Part 2 of the map has the format

NAME	FORMAT	WRITE	FORE	RECORD	CYL	TRACK	SECT	BOT	EOF	EOT
nnnnnnnn	f	w	i	r	cccc	tttt	ssss	bbbb	ffff	eeee
	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.

where

nnnnnnnn is the name of a file in the permanent RAD area.

f is the file format (U = unblocked; R = random; C = compressed, blocked sequential access; B = blocked sequential access; P = blocked random access).

w is the write protection for the file

SY is write permitted from RBM only.

FG is write permitted from foreground only.

BG is write permitted from background.

NO for no write protection.

i is the foreground task indicator. It is RF if the file contains a resident foreground task (only meaningful for files in the SP, UP, and FP areas). It is blank if the file does not contain a resident foreground task.

rrrr is the logical record length, in bytes, represented as a hexadecimal number.

cccc is the hardware cylinder address (if the file is contained on a disk pack) containing the BOT of the file.

tttt is the hardware track address containing the BOT of the file (included if the file is on a disk pack or 7232 RAD).

ssss is the hardware sector address containing the BOT of the file (included if the file is on a disk pack or 7232 RAD).

bbbb is the volume sector number of the first sector defined for the file. It is a hexadecimal number.

ffff is the end-of-file pointer. It is a hexadecimal number.

eeee is the volume sector number of the last sector plus one in the file, represented as a hexadecimal number.

#LMAP Command

The !#LMAP command maps the library files in the area specified by the command. The library map is printed for each object module in the format.

```
LIBRARY x    IDENT    iiiiiii    LENGTH    yyyy    SECTOR    ssss

DEFS         REFS

dddddddd     rrrrrrrr
   .            .
   .            .
   .            .
```

where

x indicates the library in which the library object module is located (M = Main, B = Basic, E = Extended).

iiiiiii is the identification (from start module item) of the object module.

yyyy is the relocatable length of the object module.

ssss is the relative record number of the module within the MODULE file.

dddddddd is the symbol for an external definition in the module.

rrrrrrrr is the symbol for an external reference in the module.

For each entry in the Master Directory, the following steps are repeated:

1. If the Master Directory entry is empty, no information is output; otherwise, Part 1 of the map is produced from the Master Directory entry.
2. If the Master Directory entry is for an active area, the permanent file directory for the area is read to produce Part 2 of the map. If the area is allocated but contains no permanent files, nothing is output.
3. If the active area is either the SL or UL, the libraries are mapped in the following order:
   a. Basic
   b. Extended
   c. Main
4. The contents of each library (Basic, Extended, or Main) are mapped by referencing the MDFRF, BDFRF, or EDFRF file for that library. Each nonpadding entry in the xDFRF file identifies an object module in the library. The information about the object module is obtained in the following manner:
   a. LIBRARY – this is the first character in the file name of the MDFRF, EDFRF, or BDFRF file being mapped.
   b. IDNT and LENGTH – each xDFRF entry contains a MODIR entry number. The corresponding entry in the MODIR file contains the identification and relocatable length of the object module.
   c. DEFs and REFs – each xDFRF entry contains n DEF_i words and m REF_i words describing the external definitions and references in the object module. Each DEF_i or REF_i is the entry number of a symbol in the EBCDIC file. The symbolic representation of each DEF and REF is obtained by referencing the indicated entry in the EBCDIC file.

#SAVE Command

The file save tape/deck created by the !#SAVE file command has the following format:

Control Records

The following records are output if the media is magnetic tape or paper tape:

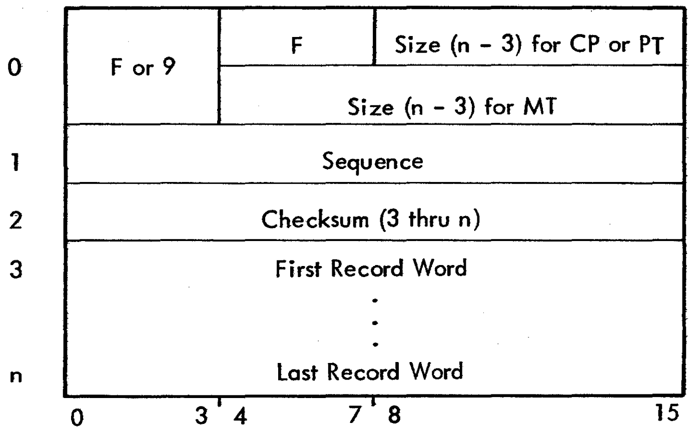
!BOR nn Where nn is a hexadecimal digit of value 1 for the first reel, 2 for the second reel, etc.

!EOR This record indicates the end of a tape reel and appears at the end of paper tape reels only.

!EOD This record indicates the end of a standard binary file on cards or paper tape.

Data Records

The following format is used for all saved files:



where

F or 9 if the media is cards or paper tape, word 0 begins with FF or 9F as the first byte. 9F means this is the last record of the file; FF means this is not the last record. If the media is magnetic tape, word 0 begins with F or 9 as the first digit (9 means this is the last record).

Size is the number of active words in the record. If the media is cards or paper tape, the maximum value of n is 59; if the media is magnetic tape, the maximum value of n is 4093.

Sequence is the sequence number for the current record.

Checksum is the arithmetic sum (with carry) of the n-3 active words after the record header.

File Definition Record

Record 0 of each file contains the directory information necessary to restore the file in words 3 through 12.

3	Area Mnemonic							
4	C_1					C_2		
5	C_3					C_4		
6	C_5					C_6		
7	C_7					C_8		
8	F	WP	C	T		RF		SR
9	Record Size							
10	1 if Standard Binary							
11	Relative EOF							
12	Relative EOT (fsize in sectors)							
13	First Record of File . . .							
n	Last Record of File							
	0 1 2	3 4	5	6	7	8	14	15

Standard Binary Files

If the output media is cards or paper tape, file being saved is blocked, the record size is 120, and the first three words of the first record conform to the format described in Appendix A of the Sigma 2/3 RBM Reference Manual (90 10 37) the file is simply copied to the BO device; otherwise, the file data is reformatted for output. Any record within a standard binary file found to have all zeros or all blanks is not saved.

Region of Save

If an EOF has been written on a file it will be saved up to and including the EOF. If no EOF has been written on the file and the first block or record of the file is all zeros, only the file definition is saved. Otherwise, the file is saved up to the EOT.

Listing of Saved Files

As each file is saved, it is listed on the LO device. The format of the output is

```
.   SAVED:   area, filename
.            .
.            .
             .
```

Rebootable Save Tape Format

The save tape created when the FILE parameter is absent has the following format:

Record	Content
1	Bootstrap to read record 2
2	Restore program
3	Area definition record
4	Data records
.	
.	
.	
n	End record
	Tape mark
	Tape mark

Area definition records, data records, and the end record have the following Header Format for words 0-2:

Word	Bit Position	
0	Type	Word Count
1	Sequence Number	
2	Checksum	
	0 1	2 15

where

Type identifies the record (1 = data record, 2 = area definition, 3 = end record).

Word Count is the number of active words in the record, excluding the three-word header.

Sequence Number is the sequential record number, beginning with record number 0, which contains the tape creation date and time.

Checksum is the arithmetic sum (with carry) of the active words in the record.

Area Definition Record

3	Sectors/Record	Device Number
4	Words Per Record	
5	Seek Address (Sector Number)	
6	Device Type	Channel Register
7	Area Mnemonic	
8 ⋮ 14	Date/Time (for record 0 only)	
	0 7	8 15

where

Sectors/Record is the number of sectors in the following data record.

Device Number is the hardware device number.

Words Per Record is the number of words in the following data record.

Seek Address is the hardware seek address for the following record expressed as a sector number.

Device Type signifies the model number (0 = 7202, 7203, or 7204; 1 = 3231; 2 = 3232/33; 3 = 7232; 4 = 7242 or 7246; 5 = 7251 or 7252; 6 = 3203/04; 7 = 7270).

Channel Register is the even-numbered channel register for the device.

Area Mnemonic indicates the area on the first definition record, and is zero on subsequent definition records.

Date/Time is the date/time the tape was created (as given by M:DATIME).

#RESTORE Command

This command restores files that have been saved with a previous !#SAVE command.

As each file is encountered on BI, tests are made to determine

1. If a restore of the file was requested.

2. If the file has an entry in the file directory (an entry is made if one does not already exist).

3. If the proper write authorization for the area is in effect (SY key-in).

Files to be restored are read/written using double buffering and I/O overlap wherever possible. As each file is restored, the message "RESTORED area,filename" is output on the LO device.

If the magnetic tape being from BI is a rebootable save tape, the areas saved on the tape will be restored in their entirety.

#INITIALIZE Command

This command initializes RAD/disk packs to conform to the requirements of the RBM file management system. The command is followed by a set of area definitions, which are used to build a label sector for each disk pack volume as follows:

For model 7242, sector zero of track 19 of cylinder 202 contains the disk pack label. The format of this label is

Bytes	Contents
0-3	Zero.
4-11	Volume serial number (left-justified and blank-filled).
12-19	Directory entry for first area on disk pack.
•	
•	
•	
132-139	Directory entry for 16th area on disk pack.

Each directory entry contains the same information as a Master Directory entry for an area, i.e.,

Word	Bits	Contents
0	0-15	Area name or zero if no corresponding area.
1	5-6	Area protection code.
2	0-15	Sector address of first sector of area.
3	0-15	Sector address of last sector of area +1.

For models 725x and 323x, the format of the label is identical, but is written in sector 1 of track 0 of cylinder 0.

#MESSAGE and #PAUSE Commands

#MESSAGE

This command outputs the control command image on the OC device. No RADEDIT overlay is used.

#PAUSE

This command is identical to !#MESSAGE, except that the RBM routine M:WAIT is executed before the next control command is read.

#GDTRACKS and #BDTRACKS Commands

The !#GDTRACKS and !#BDTRACKS Commands have the format

```
!#GDTRACKS dn,number [,number...].
```

The flow diagram for !#GDTRACKS and !#BDTRACKS is given in Figure 72.

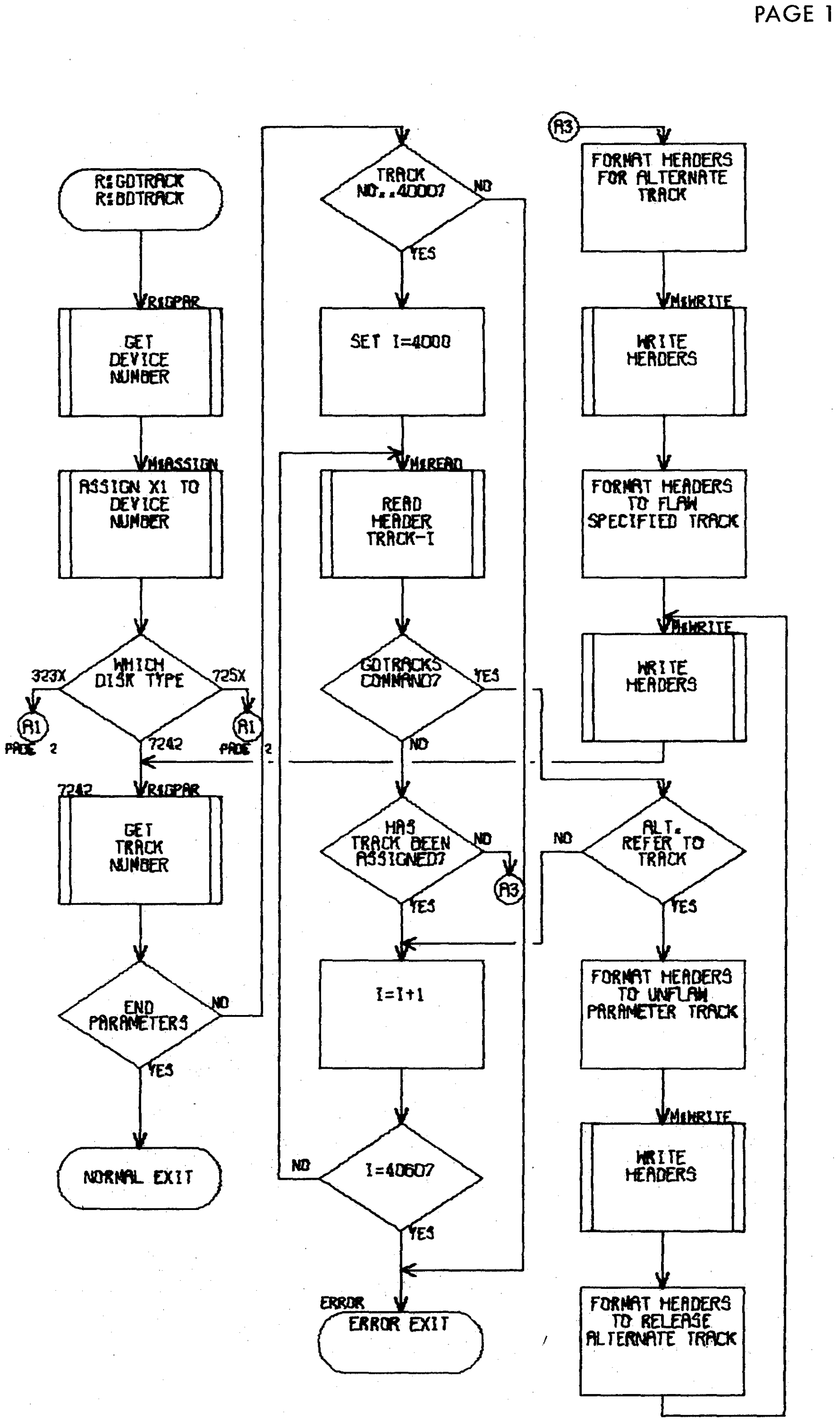


Figure 72. GDTRACKS, BDTRACKS Command Flow Diagram

PAGE 2

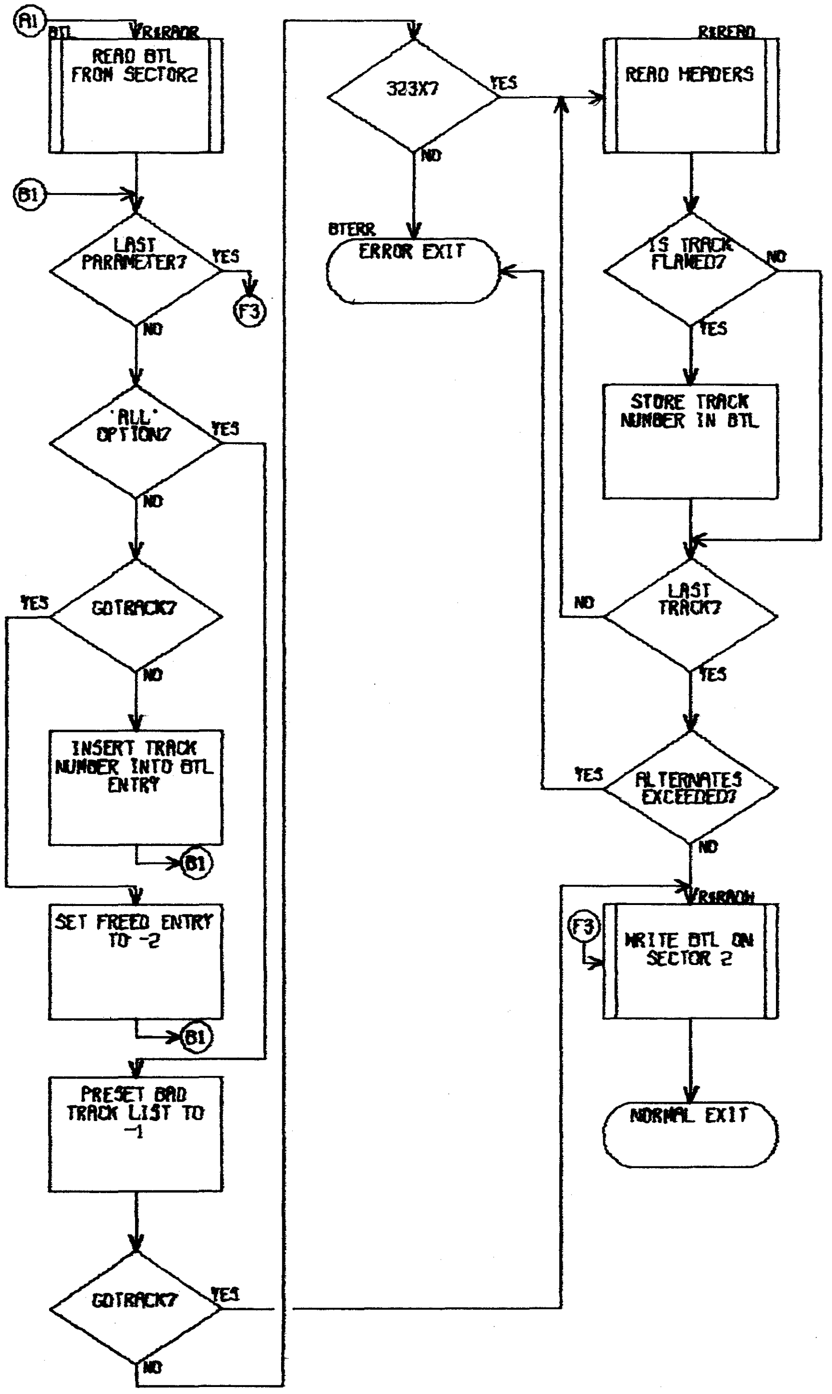


Figure 72. GDTRACKS, BDTRACKS Command Flow Diagram (cont.)

6. CHARACTER-ORIENTED COMMUNICATIONS HANDLER (R:COC)

Introduction

R:COC is a foreground program that serves as an interrupt handler for the Character-Oriented Communications (COC) Controller, Model 7611. R:COC should be assembled with parameters set to installation-dependent characteristics with regard to channel identity, interrupt address, and number of lines to be serviced. Lines are assumed to be a contiguous set (0 - n).

The program supports Teletypes (Models 33, 35, and 37) in addition to a Model 7555 keyboard display. Display-dependent code will be included as an assembly option. Unsupported devices will be serviced within certain limitations.

R:COC can be established in one of the processor areas and loaded into the foreground either automatically when the RBM system is loaded, or when called in by user command. When loaded, R:COC provides its own initialization to link the program to the appropriate interrupt address pair (input and output), establishes linkage for Monitor service requests (M:COC), and provides continuous input to a circular buffer within R:COC.

R:COC Input Buffer

The circular buffer for input data coming from the COC is normally 64 words in length but may be enlarged by assembly option. Input flows to the buffer from the COC as initiated by up to 64 communication lines. The COC transfers a full word to the CPU for each character initiated by a communication line. This word contains the originating line number (0-63) in the first byte and the generated character (ANSCII) in the second byte. An input interrupt is provided with each word transferred.

Once activated by an input interrupt, R:COC processes individual characters and adds to the user message buffer as appropriate. R:COC maintains an index pointer to identify the next character to be processed and, after processing a character, replaces it with a mask of all 1's. An unprocessed character is identified in the circular buffer by bit position one (1) being a zero. All input characters that may have been stacked in the circular buffer are processed individually before exiting the input interrupt level (see Figure 73).

Character Output

The first character of an output message (or prompt character to elicit input) is initiated outside the handler in M:COC. Upon completing the transmission of a character, the COC controller will provide an output interrupt. The interrupt will be processed by R:COC, which will "read" the line number associated with the interrupt and then continue the output message by transmission of either the next character from the user buffer or from the appropriate EOM sequence. After initiating character transmission, the interrupt level is exited to allow subsequent interrupts. If no further transmission is required on the line, a "stop transmit" is performed before exiting.

Processing flow of the output interrupt handler is diagrammed in Figure 74.

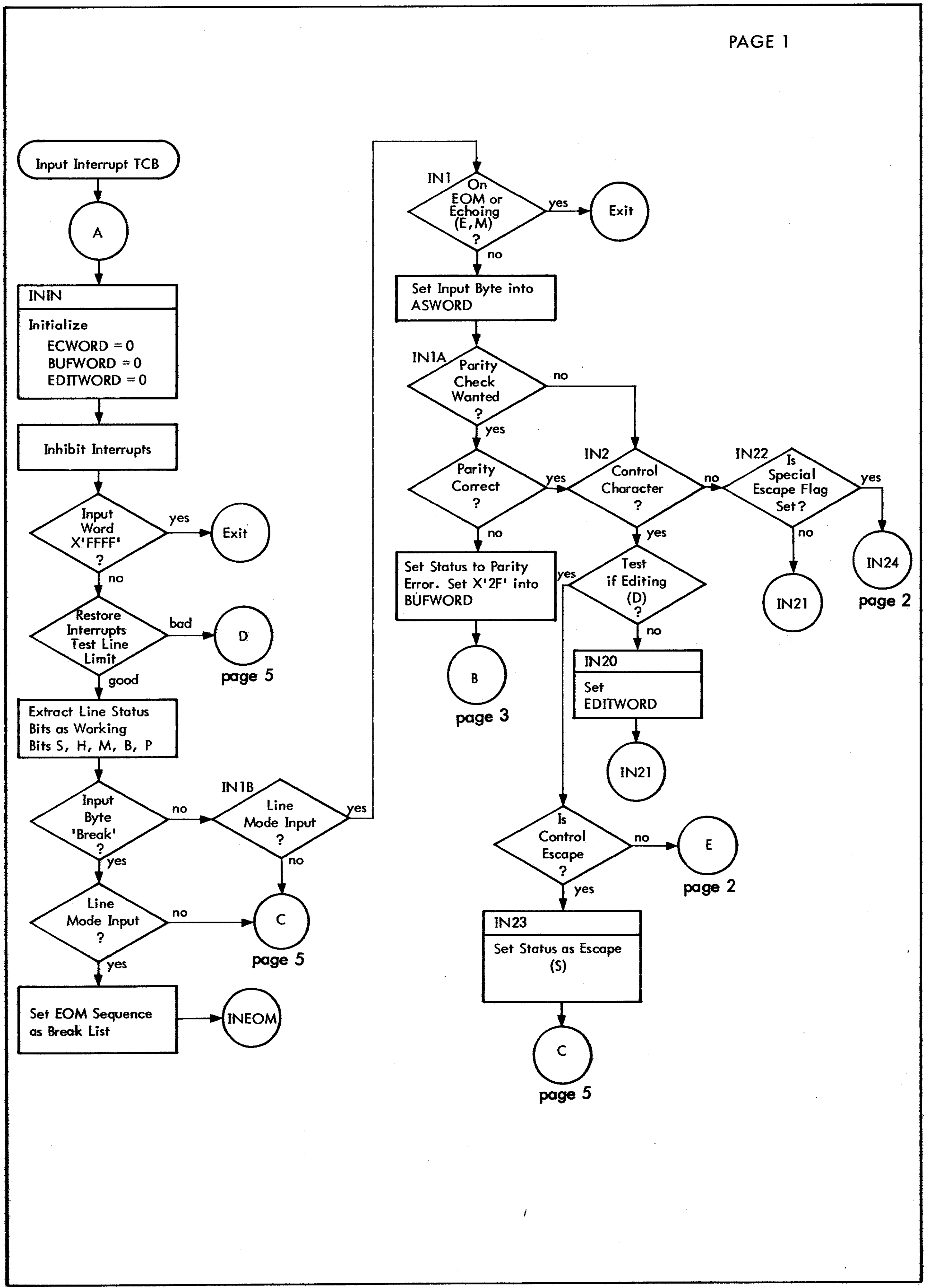


Figure 73. R:COC Input Interrupt Handler

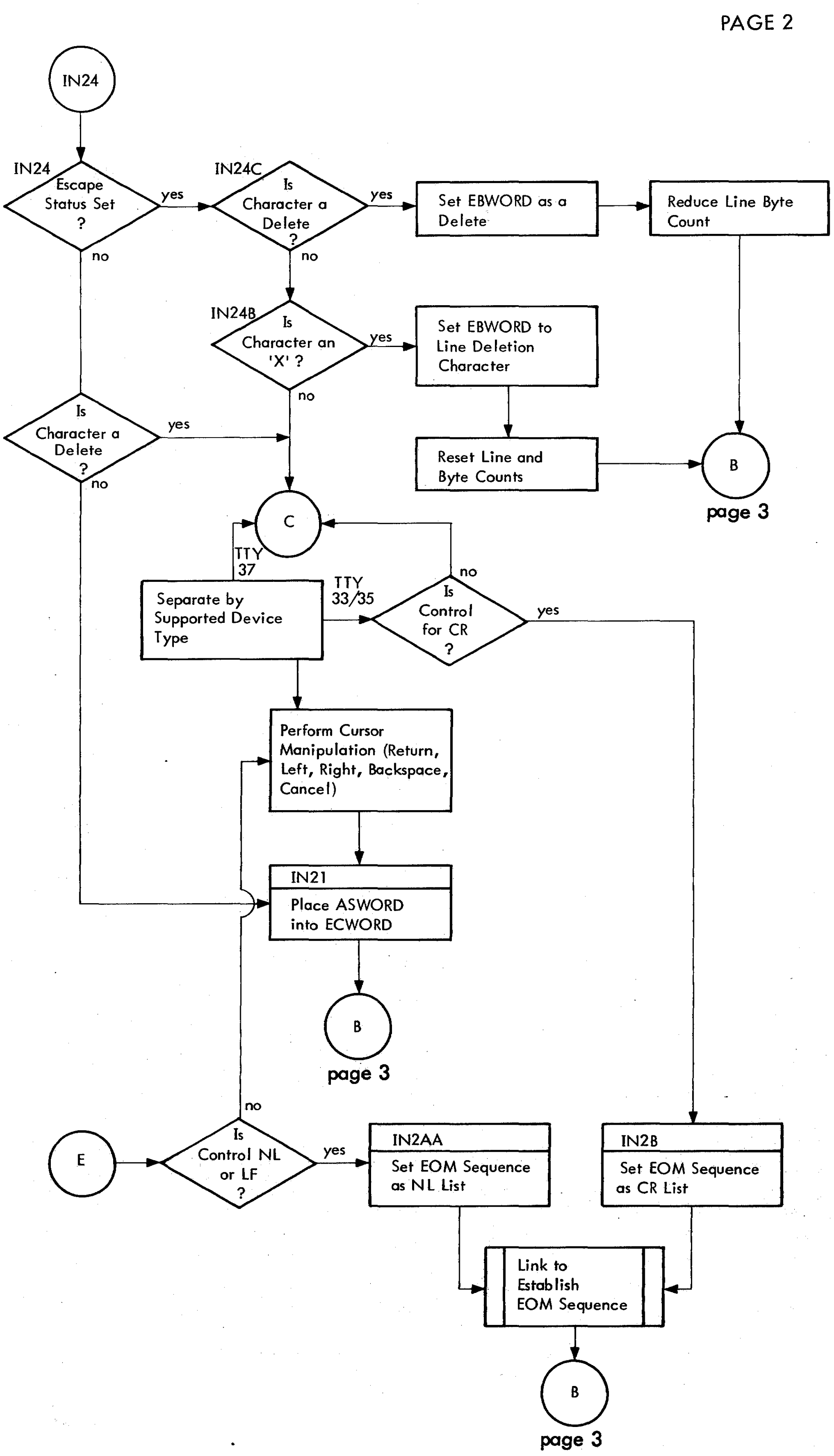


Figure 73. R:COC Input Interrupt Handler (cont.)

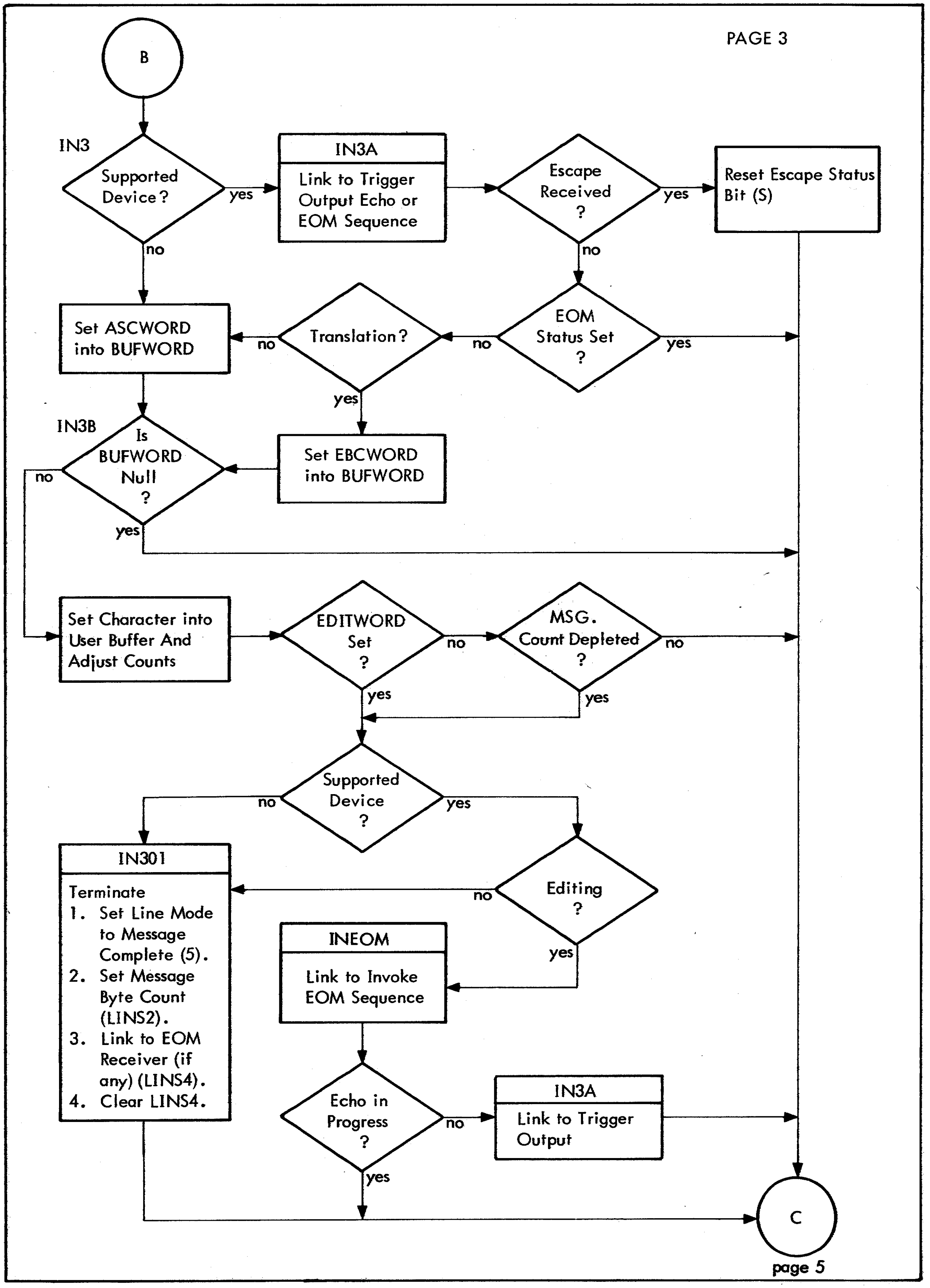


Figure 73. R:COC Input Interrupt Handler (cont.)

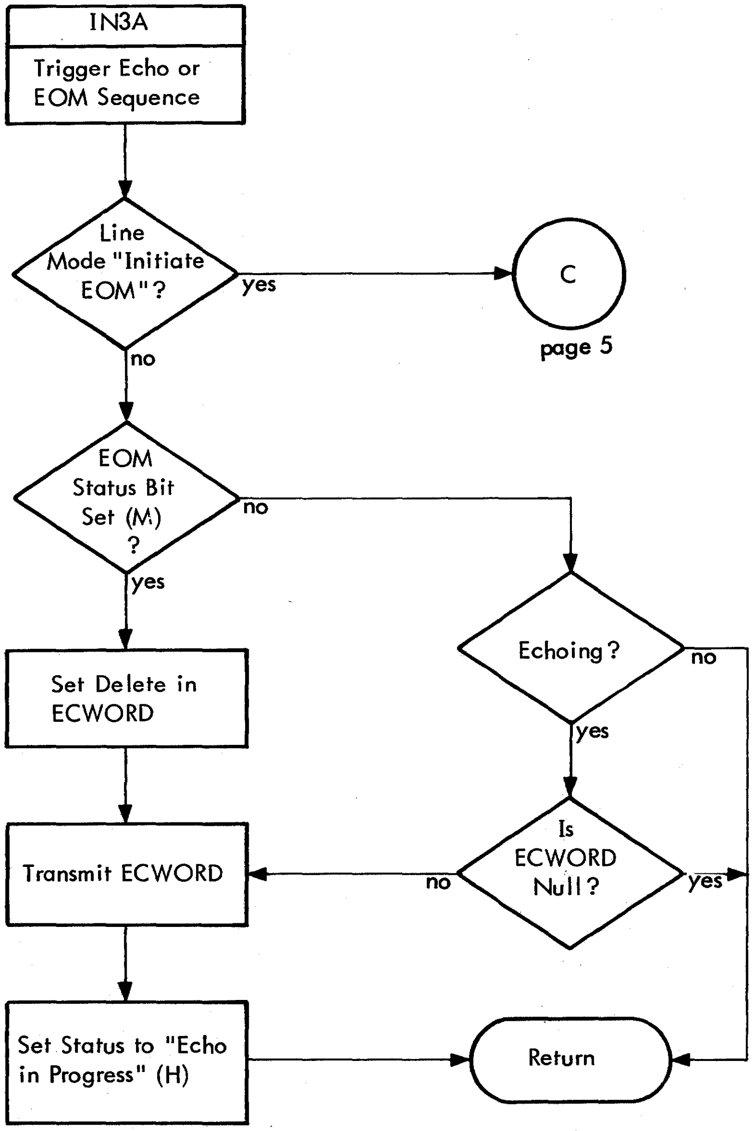


Figure 73. R:COC Input Interrupt Handler (cont.)

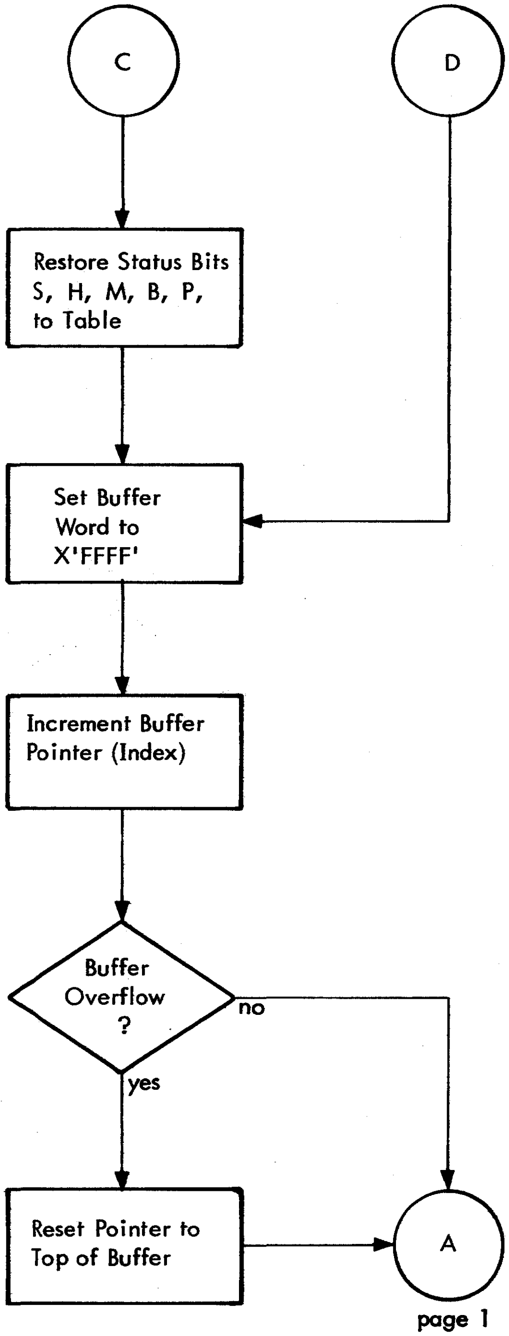

C
D
Restore Status Bits S, H, M, B, P, to Table
Set Buffer Word to X'FFFF'
Increment Buffer Pointer (Index)
Buffer Overflow ?
no
yes
Reset Pointer to Top of Buffer
A
page 1

Figure 73. R:COC Input Interrupt Handler (cont.)

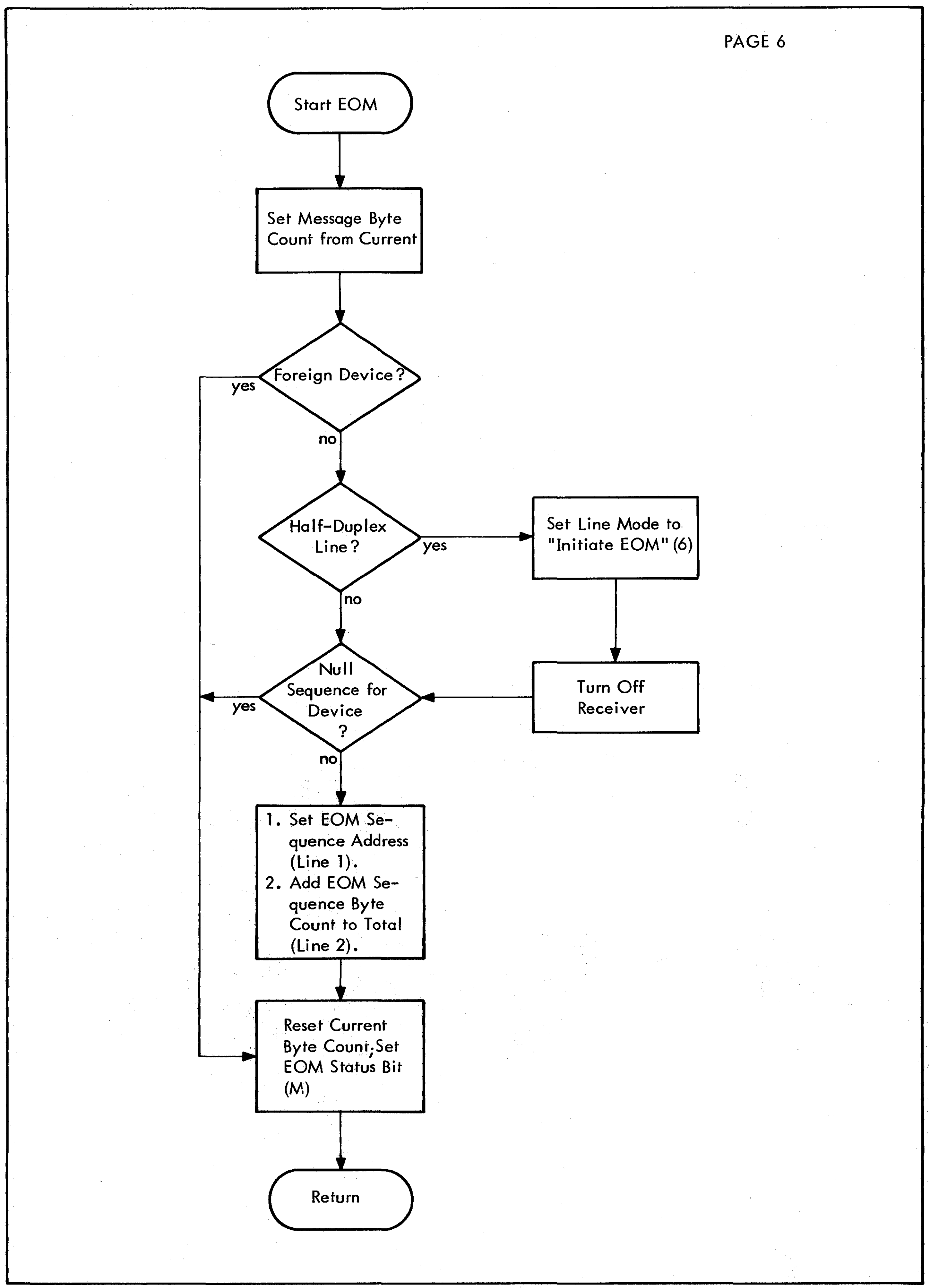

PAGE 6
Start EOM
Set Message Byte Count from Current
Foreign Device?
yes
no
Half-Duplex Line?
yes
Set Line Mode to "Initiate EOM" (6)
no
Turn Off Receiver
Null Sequence for Device ?
yes
no
1. Set EOM Sequence Address (Line 1).
2. Add EOM Sequence Byte Count to Total (Line 2).
Reset Current Byte Count, Set EOM Status Bit (M)
Return

Figure 73. R:COC Input Interrupt Handler (cont.)

PAGE 1

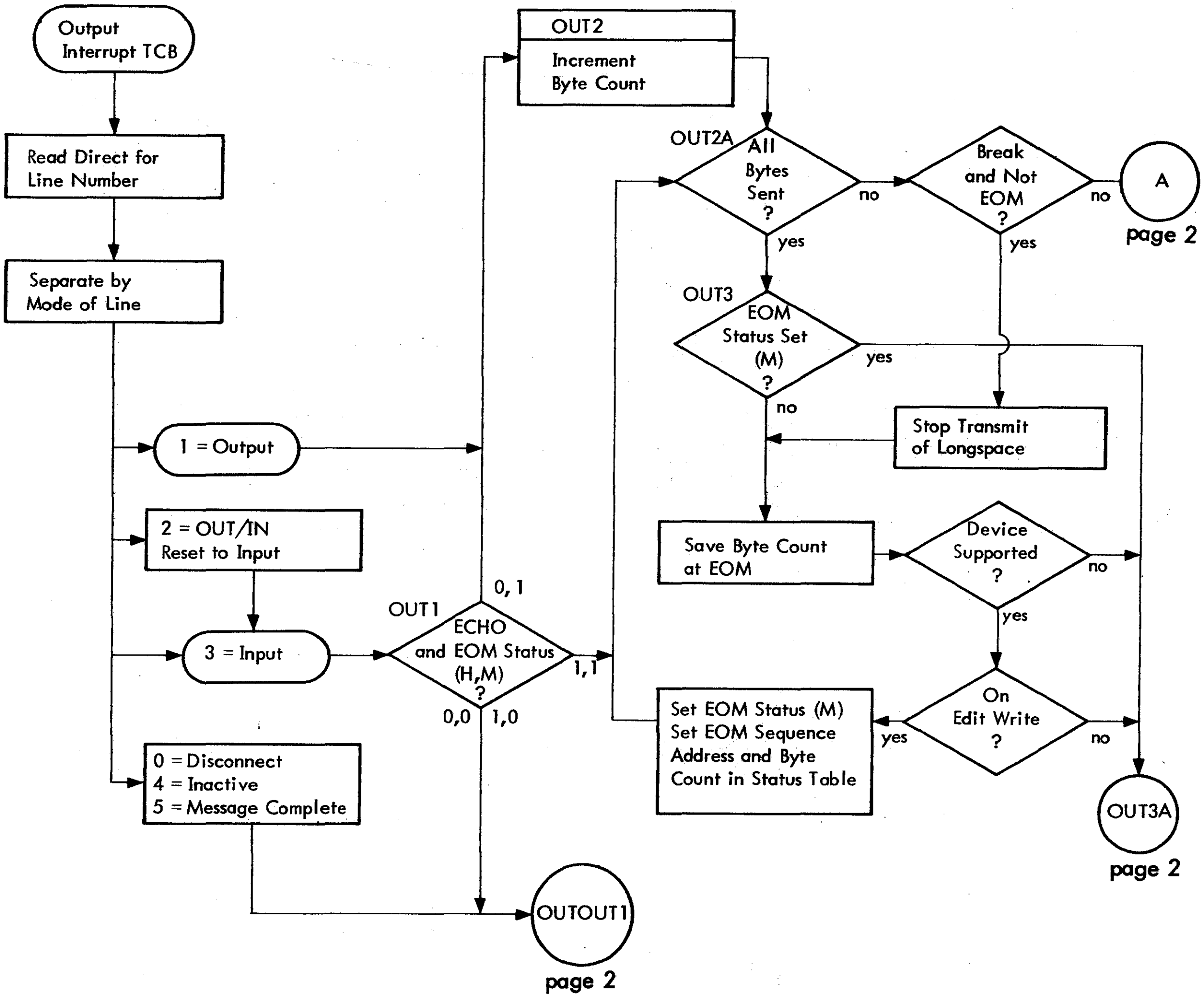


Figure 74. R:COC Output Interrupt Handler

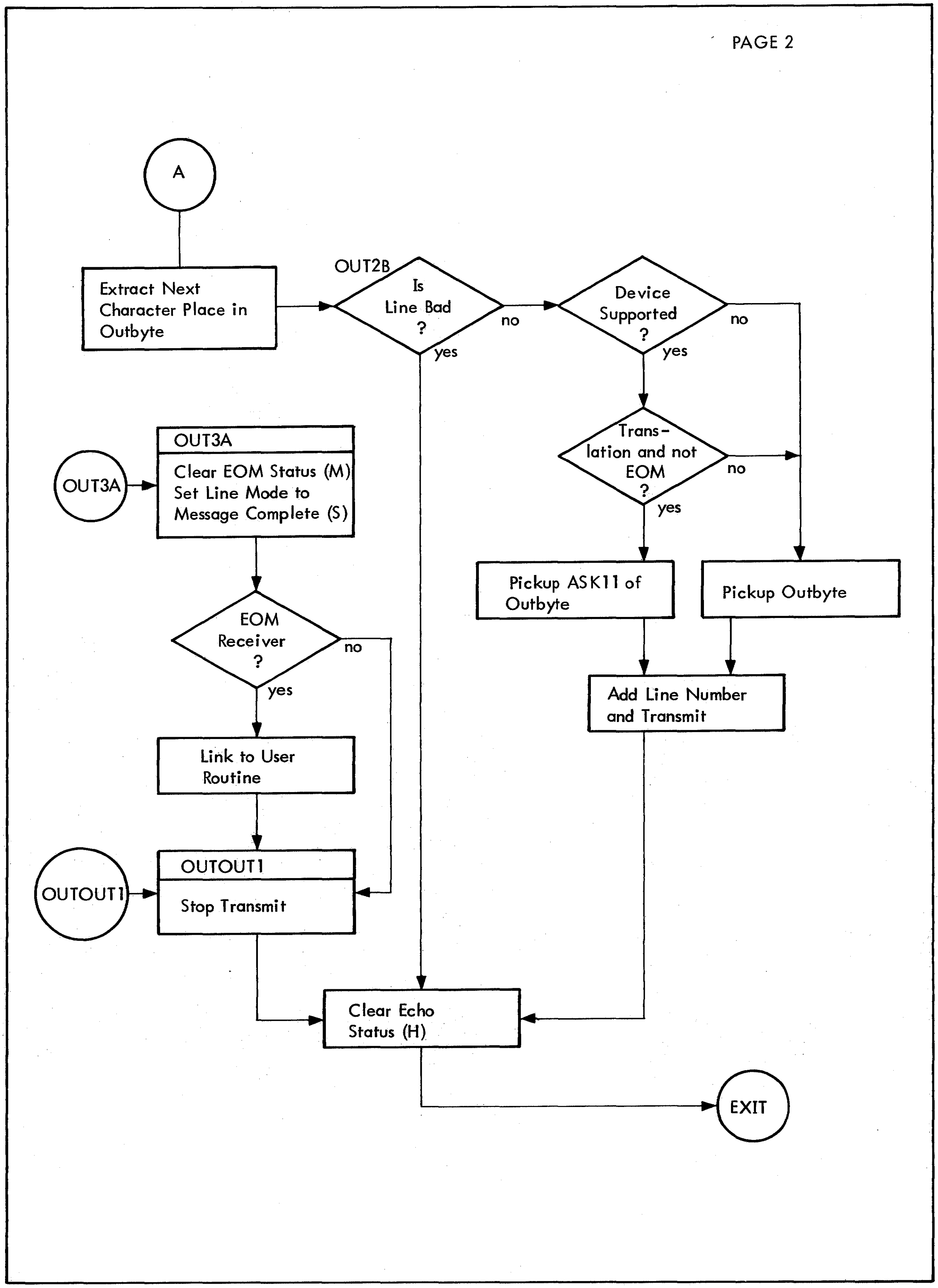


Figure 74. R:COC Output Interrupt Handler (cont.)

Monitor Service Request (M:COC)

The user interacts with the communications handler (R:COC) through Monitor service calls (M:COC), which can be used to connect or disconnect communication lines, initiate message reads and writes, break off a line function, and check on line or operation status. Coordination between R:COC and M:COC is maintained through status tables in the handler code (see Figure 75 below). Upon receiving any request, M:COC as a Monitor overlay is activated and will first shift a specific block (assembly dependent) of COC write directs and status table pointers from the foreground handler to the Temp Stack of the caller. If connection is required, the appropriate send and/or receive modules of the COC are activated, the line status checked, and the status tables for the indicated line are then set to the characteristics prescribed by the caller. Once a line is connected, read/write requests will be serviced.

While processing a read or write message, the line mode is set as "busy" until message complete" status is registered in the status tables by the handler. If need be, a message may be terminated whilein progress through a "break" call to M:COC, in which case the line mode is forced to output status and a "long-space" is generated. When this "long-space" is recognized by the output handler, the message will be terminated in a normal fashion with the appropriate activation of the EOM receiver and EOM sequence. In all cases, the EOM receiver is called from the R:COC interrupt level (input or output) and should be used in the same manner as prescribed for an AIO receiver (see Chapter 5 of the Sigma 2/3 RBM/RT,BP Reference Manual, 90 10 37).

Half-duplex lines, which require switching receiver modules off in order to transmit, are given special attention when prompting, initiating EOM sequences, or activating an intervening "break" (long-space). A variable time delay (equipment dependent) is required to complete the turning off of receiver modules and this delay is always incurred at the user program level (in M:COC) and not at the handler interrupt level.

The processing flow for requests to M:COC is illustrated in Figure 76.

Translation Tables

Conversion between ANSCII and EBCDIC characters is provided by two tables containing 256 words each. Both tables are resident in R:COC. EBCDIC to ANSCII translation is performed on a byte-per-character basis, and the ANSCII to EBCDIC table is built on a word-per-character basis. In the latter case, the first byte of each word is used to flag the control characteristic of the ANSCII character. Format identification is illustrated in Figure 77.

	0	1	2	3 4	5	6	7	8	9	10 11	12	13 15
LINS 0	E	T	CP	TYPE	S	H	M	B	PE	LINE	D	MODE
LINS 1	FWA of User Buffer											
LINS 2	Byte Count at EOM							Total Byte Count				
LINS 3	Cursor Position							Current Byte Count				
LINS 4	EOM Receive Address											

where

E = 1 indicates input is to be echoed.

T = 1 indicates byte translate required.

CP = 1 indicates parity check required.

type = 0 identifies device as TTY model 33 or 35.

= 1 identifies device as TTY model 37.

= 2 identifies device as keyboard display (7555).

= 3 identifies device as unknown (unsupported).

S = 1 indicates previous character was escape (ESC).

H = 1 indicates echo in progress.

M = 1 indicates EOM sequence being echoed.

B = 1 indicates "break" was received.

PE = 1 indicates a parity error has been detected.

Line = 0 indicates a full duplex line.

= 1 indicates a simplex-send line.

= 2 indicates a simplex-receive line.

= 3 indicates a half-duplex line.

D = 1 indicates current request requires editing.

mode = 0 indicates line is logically disconnected.

= 1 indicates current request is output.

= 2 indicates first character is prompt output, then to become input.

= 3 indicates current request is input.

= 4 indicates line connected but inactive.

= 5 indicates message has been completed.

= 6 indicates EOM sequence must be initiated by M:COC for a half duplex line.

Figure 75. Line Status Table Format

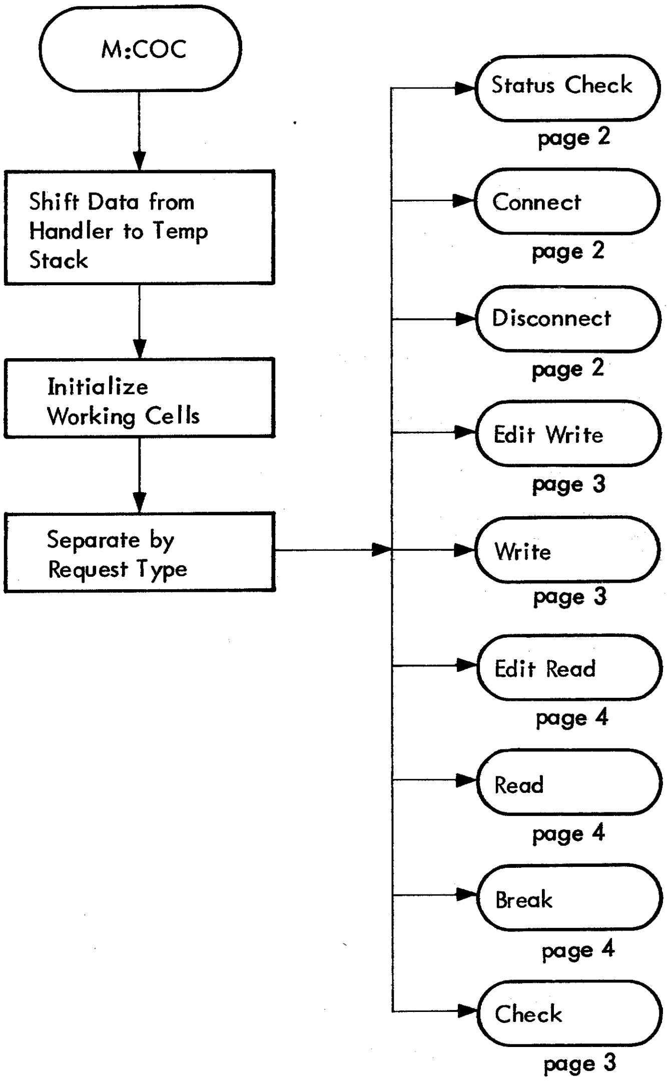


Figure 76. M:COC Request Processing

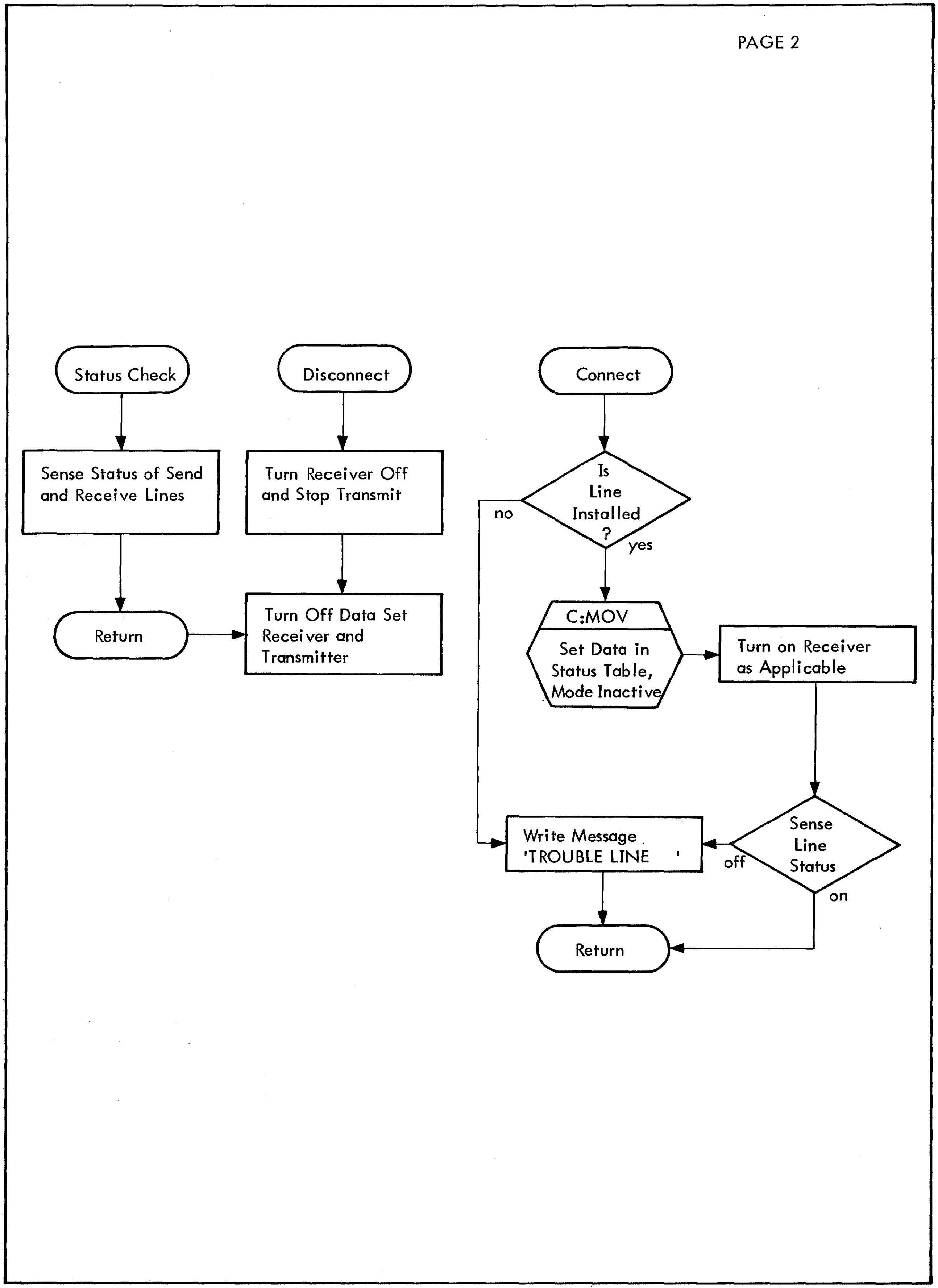

PAGE 2
Status Check
Sense Status of Send and Receive Lines
Return
Disconnect
Turn Receiver Off and Stop Transmit
Turn Off Data Set Receiver and Transmitter
Connect
Is Line Installed ?
no
yes
C:MOV
Set Data in Status Table, Mode Inactive
Turn on Receiver as Applicable
Sense Line Status
off
on
Write Message 'TROUBLE LINE '
Return

Figure 76. M:COC Request Processing (cont.)

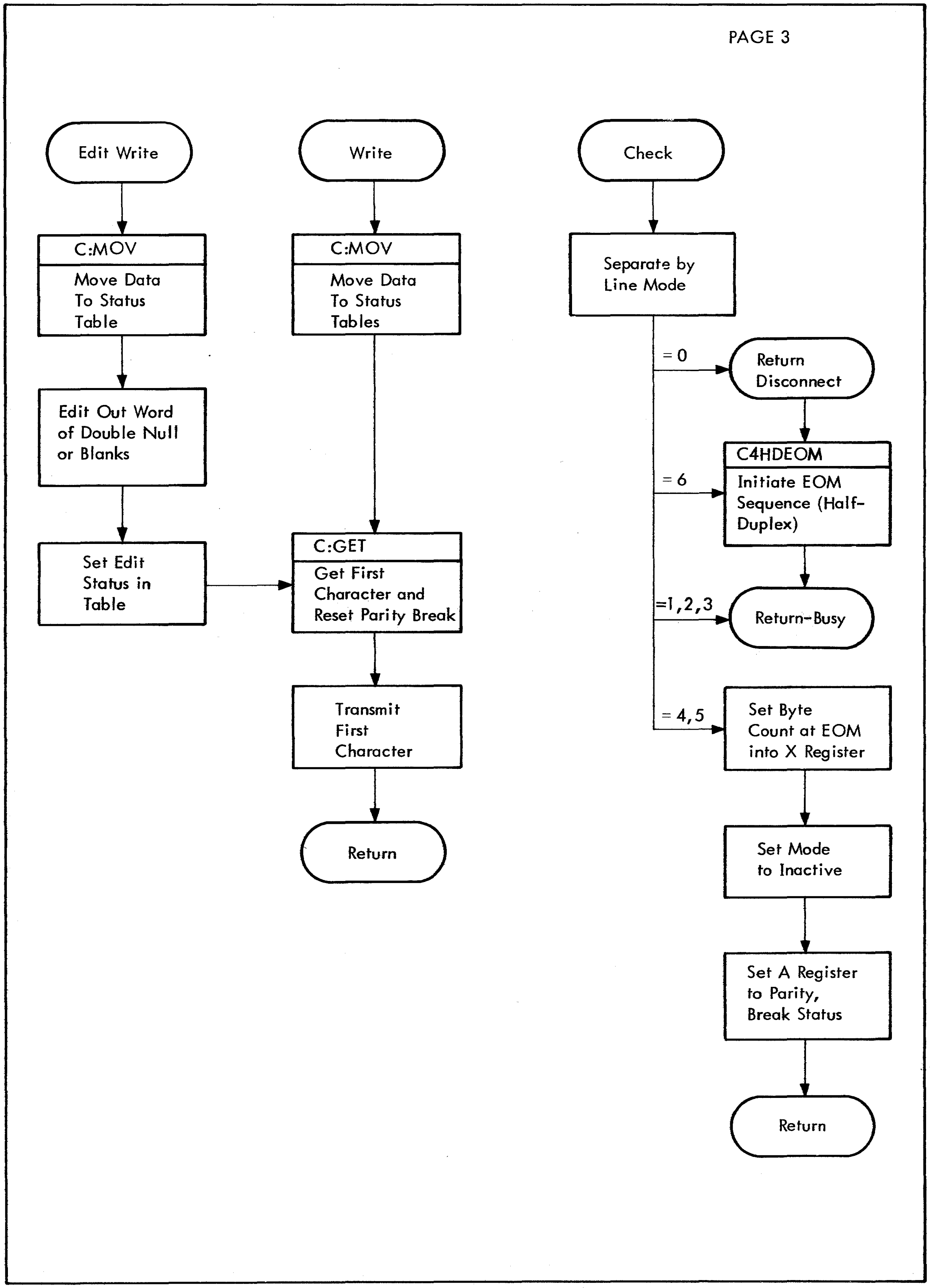

PAGE 3
Edit Write
C:MOV
Move Data To Status Table
Edit Out Word of Double Null or Blanks
Set Edit Status in Table
Write
C:MOV
Move Data To Status Tables
C:GET
Get First Character and Reset Parity Break
Transmit First Character
Return
Check
Separate by Line Mode
= 0
Return Disconnect
= 6
C4HDEOM
Initiate EOM Sequence (Half-Duplex)
=1,2,3
Return-Busy
= 4,5
Set Byte Count at EOM into X Register
Set Mode to Inactive
Set A Register to Parity, Break Status
Return

Figure 76. M:COC Request Processing (cont.)

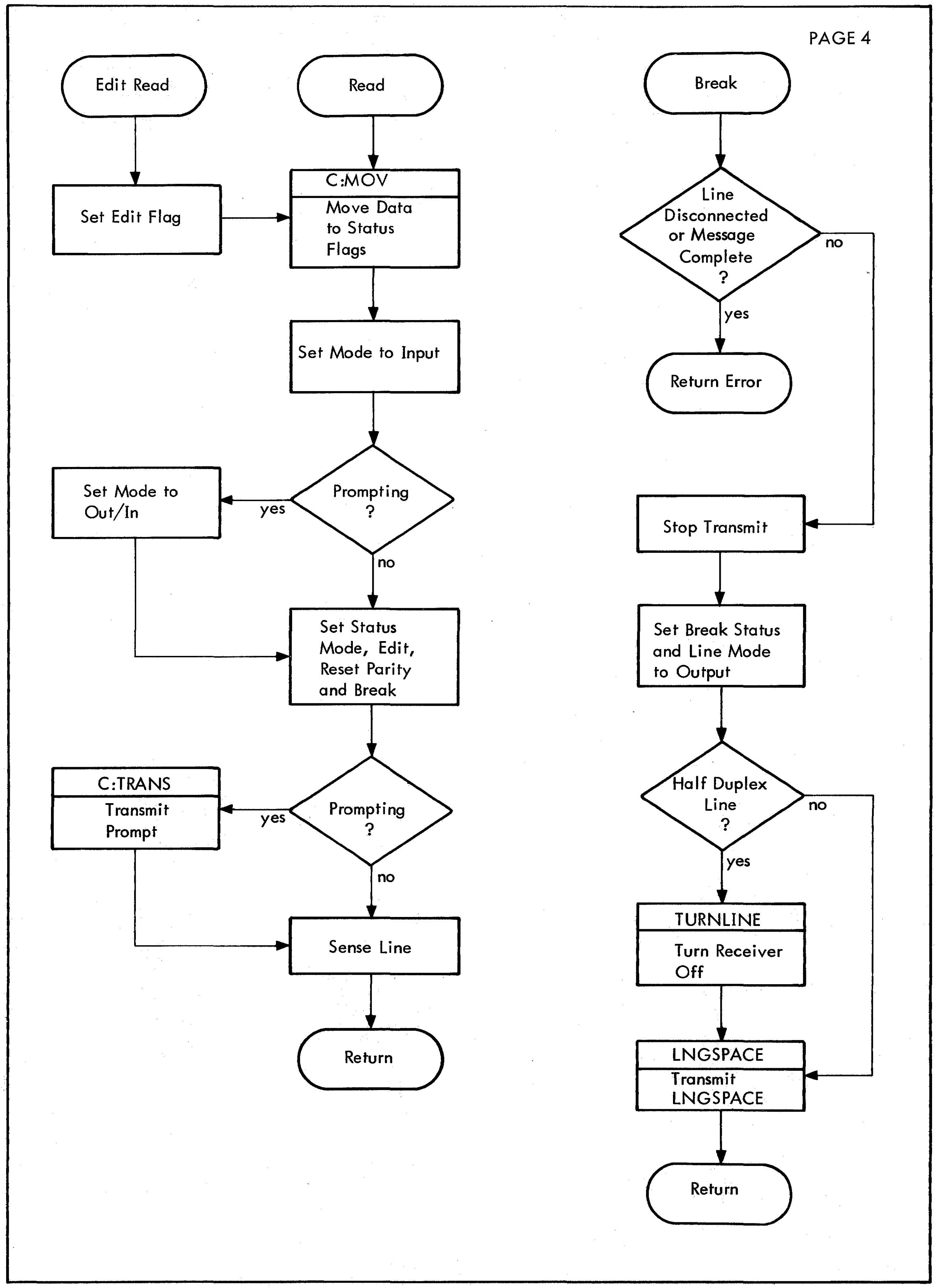


Figure 76. M:COC Request Processing (cont.)

CC	E	SP	NL	CR	Code	EBCDIC Character
0	1	2	3	4	5 7	8 15

where

CC = 1 flags a control character.

E = 1 flags the escape key-in.

SP = 1 flags a character appropriate after the escape key-in.

NL = 1 flags the NEW LINE character.

CR = 1 flags the carriage (error) return character.

Code is a 3-bit identifier for special keyboard display functions as indicated:

= 0 ignore.

= 1 cancel (CAN).

= 2 backspace (BS).

= 3 cursor left (EM).

= 4 cursor right (HT).

= 5 cursor return (CR).

= 6 cursor up (BEL).

= 7 cursor down (SUB).

Figure 77. ANSCII to EBCDIC Table Entry Format

7. SYSERR ANALYSIS

Resident SYSERR Routine

The resident routine M:SYSERR is responsible for shutting the system down in an orderly fashion in the event of a catastrophic system failure. In addition, if the SYSGEN option "ANALYSIS" is selected, the routine must preserve all hardware context information available for later analysis. M:SYSERR is entirely resident and is completely standalone; i.e., it does not use any zero-table constants or pointers, monitor tables, or monitor service routines.

The calling sequence for the SYSERR routine is as follows:

```
RCPYI   P,A

B       *V:SYSERR

DATA    SYSERR Code
```

M:SYSERR will perform the following functions if SYSERR analysis is not selected.

1. Inhibit interrupts.
2. Issue RIO if Sigma 3 or Xerox 530.
3. Halt with SYSERR code in accumulator.

If analysis is selected, M:SYSERR will perform the following:

1. Inhibit interrupts and save PSD status.
2. Save register contents.
3. Save SYSERR code.
4. Save data switch settings.
5. Save interrupt-system status if Sigma 3 or Xerox 530.
6. TDV, HIO all devices; save device status. (Note: TDV first.)
7. Save all channel register contents.
8. Save Fault Register contents if Xerox 530.
9. Save contents of location zero; store pointer to data area in location zero.
10. Issue RIO if Sigma 3 or Xerox 530; if Sigma 2, clear all interrupt levels.
11. Call user's error receiver (if any) with error severity = 3.
12. Output "!!SYSERR xx" message to the console (where xx is the SYSERR code).
13. Copy memory to CP area on RAD or to tape in 512-word blocks.
14. Call user's error receiver (if any) with error severity = 4.
15. Halt with SYSERR code in accumulator.

If SYSERR analysis is selected, location zero of the memory dump will be modified to point to the beginning of the data saved by M:SYSERR. This data will be organized into six blocks, each of which has a key-word preceding it.

The key-word contains a key in the most significant byte position and the number of elements in the block in the least significant byte position. The organization of the data is as follows (asterisked items are preset by SYSGEN):

Key = 1, Count = 7

*Word 1 – Software version (2-character EBCDIC)
* 2 – Computer type (same as K:CPU)
* 3 – Core size (same as K:UNAVBG)
4 – SYSERR code
5 – Original contents of location zero
6 – Current data switch settings
* 7 – Number of 512-word blocks in SYSERR dump.

Key = 2, Count = 7

Word 1 – L-register contents
2 – T-register contents
3 – X-register contents
4 – B-register contents
5 – E-register contents
6 – A-register contents
7 – PSD status indicators

Key = 3, Count = 0, if Sigma 2

Key = 3, Count = 4, if Sigma 3

Word 1 – Group '0' interrupts enabled
2 – Group '0' interrupts armed or waiting
3 – Group '0' interrupts waiting or active
* 4 – Group '0' interrupts not implemented

Key = 3, Count = 12, if Xerox 530

Word 1 – Group '0' interrupts enabled
2 – Group '0' interrupts armed or waiting
3 – Group '0' interrupts waiting or active
* 4 – Group '0' interrupts not implemented
5 – Group '5' interrupts enabled
6 – Group '5' interrupts armed or waiting
7 – Group '5' interrupts waiting or active

* 8 – Group '5' interrupts not implemented

9 – Group '6' interrupts enabled

10 – Group '6' interrupts armed or waiting

11 – Group '6' interrupts waiting or active

* 12 – Group '6' interrupts not implemented

Key = 4, Count = 2n (where n = number of physical devices on the system)

Word (2n-2) + 1 – HIO status byte (0-7), device address *(8-15)

(2n-2) + 2 – TDV status byte (0-7), HIO O,C (12-13), TDV O,C (14-15)

Key = 5, Count = 3m (where m = the number of I/O channels on the system)

*Word (3m-3) + 1 – Even channel register address

(3m-3) + 2 – Contents of even channel register

(3m-3) + 3 – Contents of odd channel register

Key = 6, Count = 0, if Sigma 2 or Sigma 3

Key = 6, Count = 2 if Xerox 530

Word 1 – Fault Register, 1st read direct

2 – Fault Register, 2nd read direct

Key = X'FF', Count = 0 Last Key

If memory is saved on disk at SYSERR time, it is written in 512-word blocks regardless of the sector size. Writing starts in the first sector of the CP area and continues until the CP area is full. Since the first word of the dump is non-zero (it contains a pointer to the SYSERR-time context save area), it serves as a flag to indicate that a SYSERR dump resides in the checkpoint area. The CHECKPOINT, RESTORE and SYSLOAD routines store a zero into the first word of the checkpoint area. In the case of CHECKPOINT, the first word written is the first word of the background TCB. This location is cleared prior to the first write operation (this word is unused for background operations). If a disk error is detected, an error message will be output, and the computer will halt with the seek address in the E- and A-registers. Clearing the halt will result in restarting the save process.

If memory is saved on tape at SYSERR time, the operator will be instructed to mount a save tape and insert the device address of the tape unit in the data switches, and the computer will come to a halt. When the halt is cleared, writing will begin on the selected tape unit. Tape errors will result in an error message to the console, the tape unit being rewound, and the save process will be repeated. If the tape unit is not ready or the tape is write-protected when the halt is cleared, the operator will again be instructed to mount the save tape.

In order to reduce the number of address literals required for calls to M:SYSERR, zero-table location X'6B' contains an ADRL pointer to the SYSERR routine. This location is labeled V:SYSERR.

If a user's error receiver is provided, its address must be stored in SYSERRXR by the user. (The contents of SYSERRXR will default to the address of an RCPY L,P instruction.) When the receiver is called, the registers will be set as follows:

L = Return link

X = Address of context block organized as follows:

Word 0 -- Error severity

Word 1 -- EBCDIC SYSERR code

The receiver will be called twice; just before saving the memory image, and just before halting. In the first call, the error severity will be 3; in the second, the severity will be 4. The second call is to permit a user-written automatic restart routine.

SYSGEN Considerations

The following operations are performed during SYSGEN in order to support the optional SYSERR analysis feature:

1. Determine if SYSERR analysis is desired (ANALYSIS input parameter).

2. If analysis is not desired, select the abbreviated version of the SYSERR routine.

3. If analysis has been selected, determine whether the SYSERR dump is to go th the CP area on disk or to tape, and select the appropriate extended version of the SYSERR routine. This routine will be tailored to the hardware/software configuration as follows:

   a. A skeleton of the data area where the SYSERR-time context information is to be stored will be constructed.

   b. If the dump is to go to disk and the CP area size was not specified, allocate CP area with enough room to contain all of memory.

   c. If the dump is to go to disk, the following information will be stored into the SYSERR disk handler:

      - Disk device address.
      - Seek address of the first sector of the CP area.
      - The number of sectors per 512-word block of memory.
      - The number of sectors per track.
      - The number of tracks per cylinder.
      - The number of bits in the sector field of the seek address.
      - The number of bits in the track field of the seek address.
      - The number of 512-word blocks to be written. (Function of memory size and CP area size.)

4. If the dump is to go to disk and a CP area size was specified which is not large enough to contain all of memory, output an alarm indicating the size problem.

5. Store a pointer to M:SYSERR in zero-table location V:SYSERR.

Operator-Forced SYSERR

If ANALYSIS is selected at SYSGEN time, location zero will contain the address of an operator-SYSERR routine which allows the operator to force a SYSERR condition via the PCP. This routine is, in fact, an alternate entry point to M:SYSERR which sets a flag to indicate that an operator-forced SYSERR has occurred. In this way, all register contents can be preserved with a minimum amount of temp space required. The SYSERR routine will test this flag prior to fetching the SYSERR code and, if found set, will use the default SYSERR code 'OP'.

Background SYSERR-Analysis Program

The background SYSERR-analysis program, ANALYZE, is comprised of a root segment, a number of level-2 overlay segments, and an assembly-procedure set contained in the S24RBM file. This structure was chosen to minimize the amount of background space required to perform a SYSERR analysis.

Root Segment

The root segment includes a base table, I/O buffers, common subroutines, an initialization routine, and control routines. These components are described in the paragraphs below.

Base Table

The base table contains variables, pointers, and frequently used constants which are used by the several overlay segments and the common subroutines in the root. In addition, there are temporary storage cells for exclusive use by the overlay segments and storage cells for use in passing parameters to the overlays.

The base table is created by use of the procedure A:BASE (described later). The B register will be initialized to point at the stack when ANALYZE is started and is not to be modified. Implicit base-table references are used.

I/O Buffers

All necessary I/O is performed by common subroutines (described later) to or from one of seven I/O buffers located in the root. All but one of these buffers has an associated base-table pointer. (To facilitate the description of buffer usage, these buffers will be referred to by the label of the base-table location containing the buffer pointer, rather than by the buffer label itself.)

The significance of six of the seven buffers is as follows:

A:BLKO – A 512-word buffer used to contain the first data block of the SYSERR file if oplabel SI is not assigned to zero.

INBUF – A 512-word buffer used to contain the current data block being accessed (other than the first).

LINEBUF – A 65-word image buffer used for the construction of formatted output.

TYPEBUF – A 65-word image buffer used for the construction and output of error messages.

HEADER1 – A 65-word image buffer used for the construction and storage of an optional header line.

HEADER2 – A 65-word image buffer used for the construction and storage of an optional second header line.

A seventh buffer is used as a "side" buffer for print operations to permit compute-I/O overlapping.

Direct usage of these buffers by overlay segments is discouraged except when absolutely necessary. Data may be moved into these buffers by use of common subroutines and procedures described later in this chapter.

Common Subroutines

The functions of the common subroutines fall into one of four general categories: (1) data acquisition; (2) display line construction; (3) display line and error-message output; and (4) control. Subroutine linkage is made through the base table; all registers but the base register are considered volatile.

Data Acquisition. To acquire the contents of a particular location of the memory image being analyzed, function overlays need only to specify an address. Actual location of the data on disk, tape, or in memory will be handled automatically by common subroutines in the root.

If operational label SI is assigned to zero, requests for the contents of memory locations will result in the actual access of the corresponding real memory locations. If SI is assigned to a disk file or to tape, the data will be fetched from the appropriate device as follows:

- The first 512-word block of the file will be read into A:BLKO at initialization time. All subsequent references to locations 0 through 511 will result in an access from this area.

- Accesses to dump locations higher than 511 result in the determination of the block number containing the desired address (block number = address/512). If the block is currently contained in INBUF, the data will be fetched from that buffer. If the block is not currently resident, it will be read from the appropriate storage device.

The decision to keep block zero of the SYSERR file resident is based on the frequency of accesses to the SYSERR-dump zero table. Failure to keep this block resident could result in large amounts of time required to perform an analysis.

Three subroutines are available for accessing data for analysis. They are described below.

GTCTXTWD Get a specified word from the SYSERR context-save area.

Call: LINK A:GTCTXT

DATA block key

DATA displacement from block keyword.

Exit: If no error, carry will be set and value will be in the A-register.

If an error is detected (invalid key, error in context area, no context area pointer, or displacement larger than the block size), and cell A:CXTERX is zero, the subroutine will return with carry reset. If A:CXTERX is nonzero, the subroutine will exit to the address contained in that location. (A:CXTERX may be set with the SETERRX procedure.)

LDWD Fetch the contents of a specified location from the SYSERR file.

Call: LINK A:LDWD

Entry: A-register contains the address.

Exit: A-register contains the contents of the specified address. A:CURADD contains the specified address.

If there is no error encountered while attempting to fetch the requested data, the subroutine will return to (L) + 1 with carry set. If there is an error, location A:LDERRX will be tested and, if found to contain zero, return will be made to (L) + 1 with carry reset. If A:LDERRX contains a nonzero value, the subroutine will exit to that point.

LDNXT Fetch the contents of the next location from the SYSERR file.

Call: LINK A:LDNXT

Entry: A:CURADD contains the last address accessed via LDWD or LDNXT.

Exit: A-register contains the contents of the next consecutive location.

This routine is, in fact, a special entry to the LDWD subroutine. Instead of passing an address to the subroutine, the effective address is implicitly (A:CURADD) + 1. Use of this routine can significantly reduce access time for consecutive locations in the SYSERR file.

The error returns are the same as for the LDWD subroutine.

See Figure 78 for the flow of the LDWD and LDNXT routines.

Display Line Construction. To facilitate the problem of constructing formatted-display lines, four image buffers and a set of conversion and text-handling common subroutines are provided. The construction of all messages and display lines is done in the image buffers; conversion and text storage subroutines will deal exclusively with them.

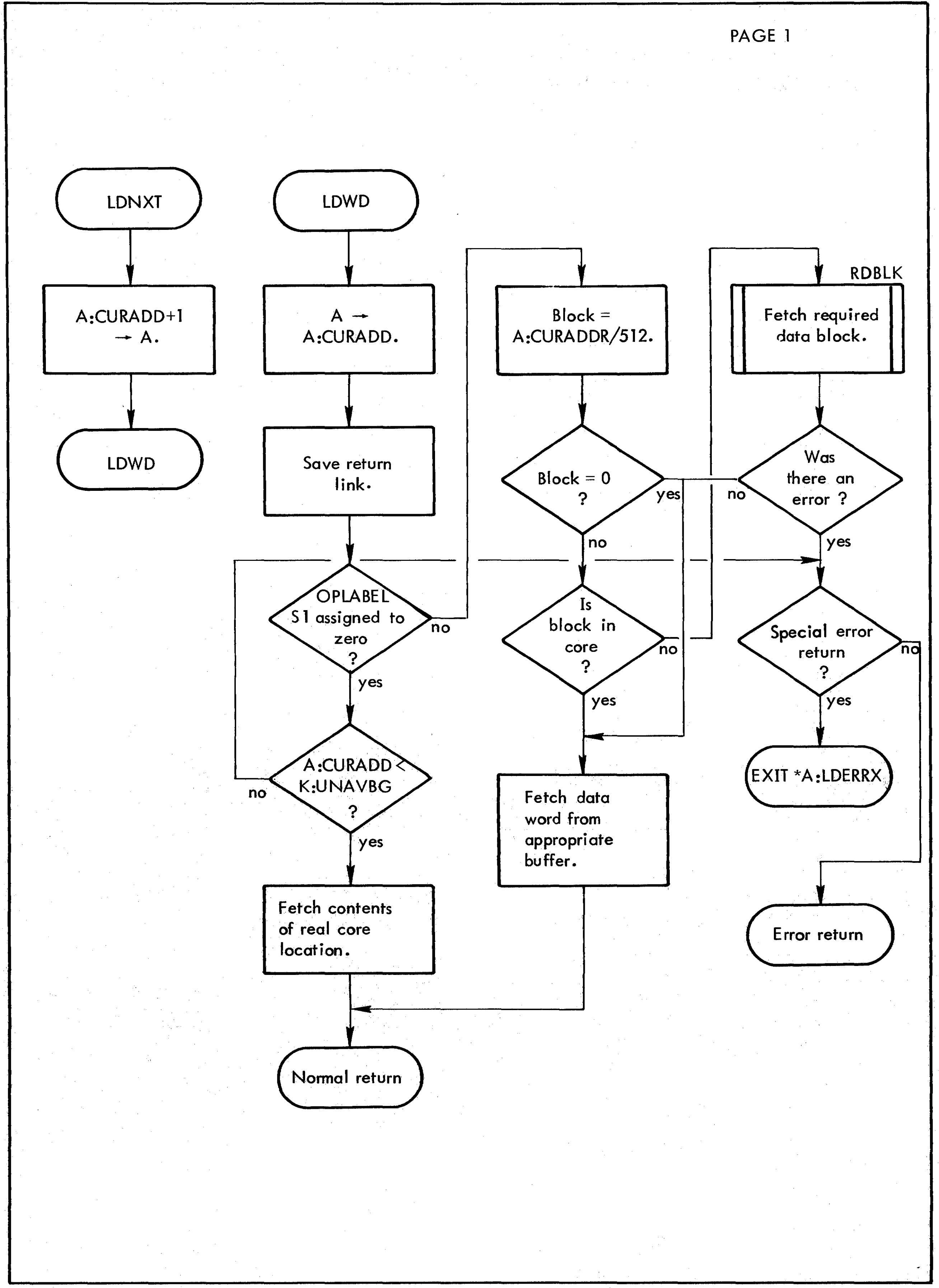


Figure 78. Data Acquisition Subroutines LDWD and LDNXT

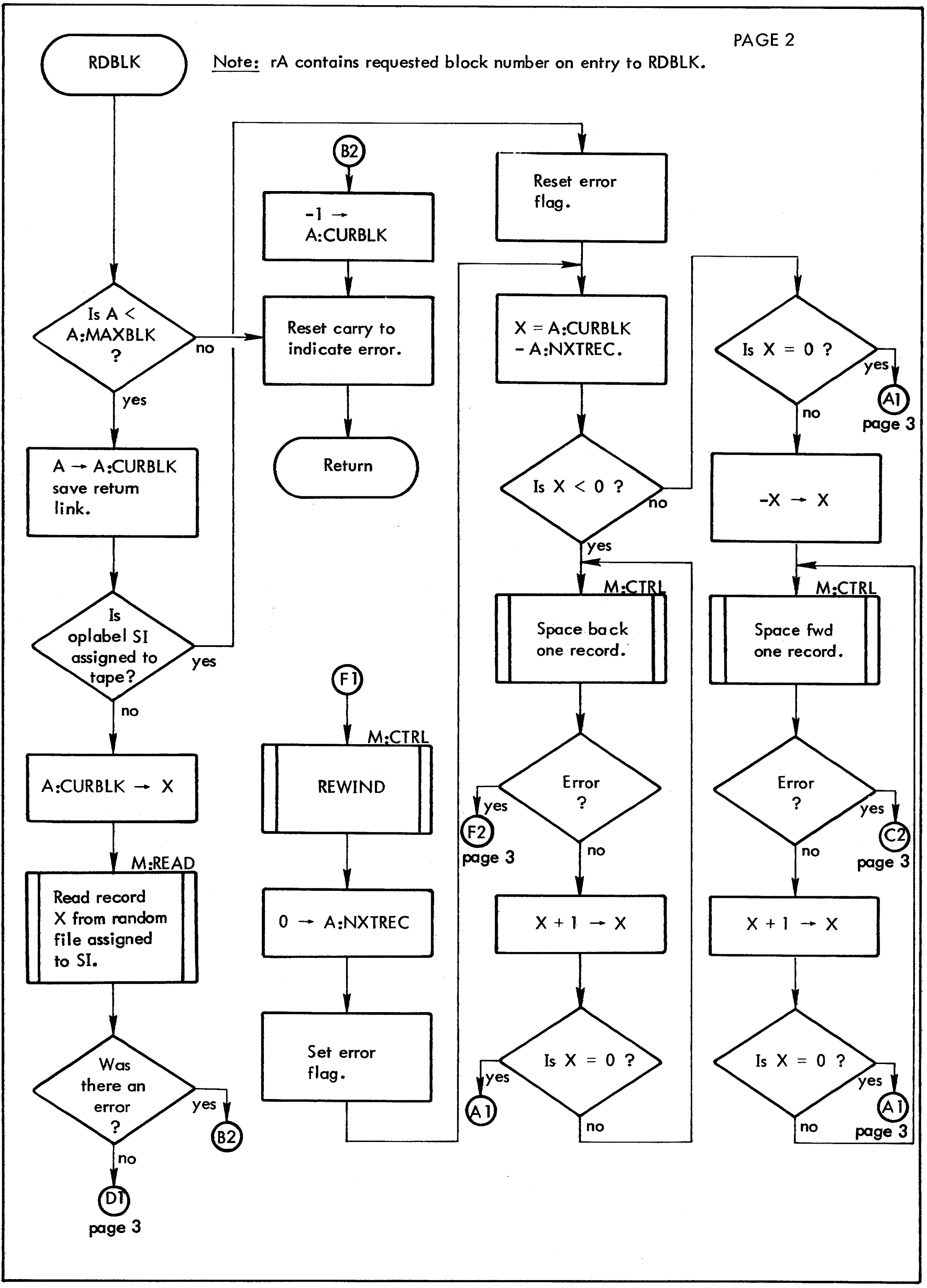


Figure 78. Data Acquisition Subroutines LDWD and LDNXT (cont.)

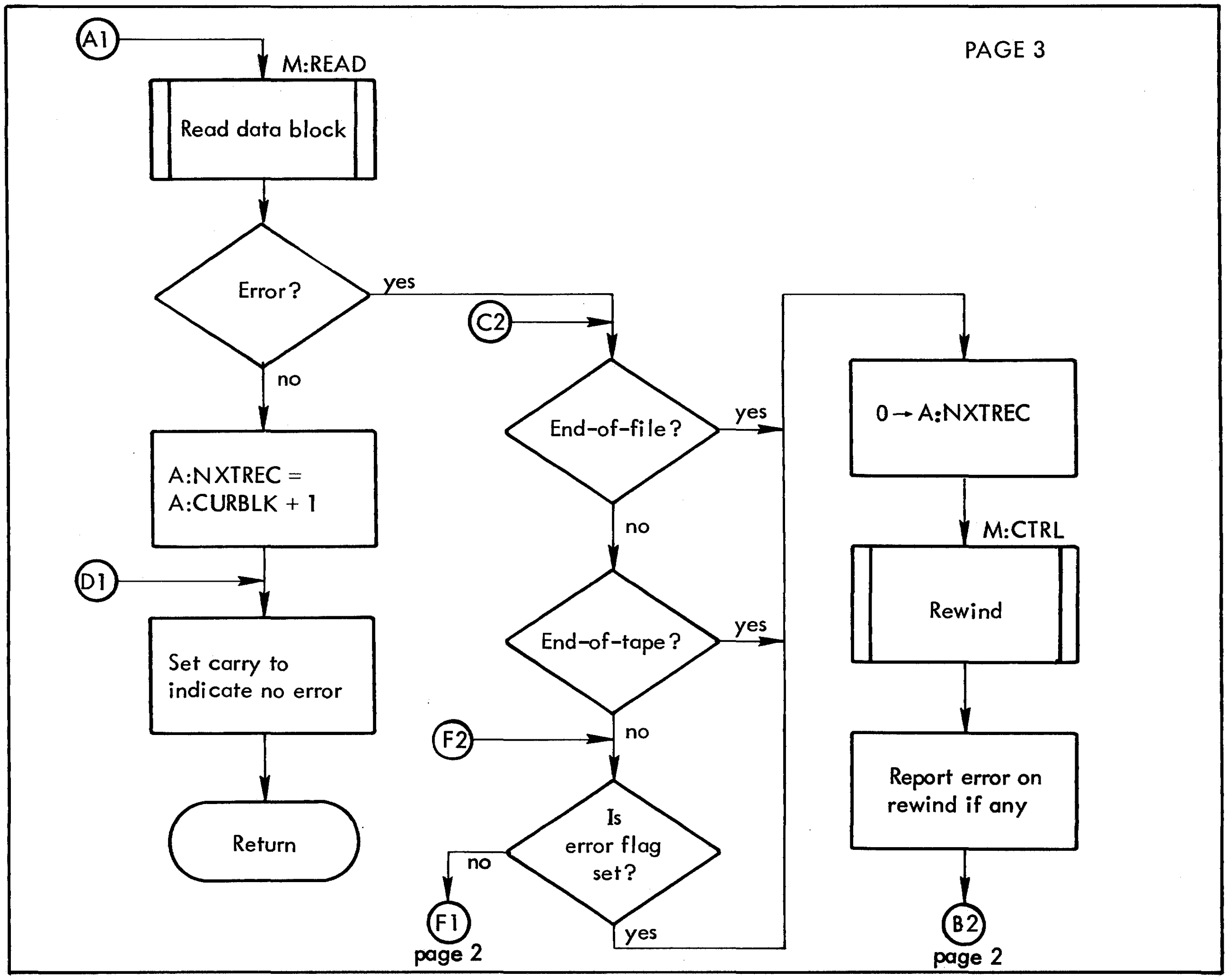


Figure 78. Data Acquisition Subroutines LDWD and LDNXT (cont.)

The four buffers are identified as LINEBUF, TYPEBUF, HEADER1, and HEADER2. Function overlays, using common subroutines, may construct display lines in any or all of the image buffers. In general, however, LINEBUF is used for the construction of the current display line and HEADER1 and HEADER2 are optionally used for construction and storage of top-of-page and/or bottom-of-page header lines. Use of TYPEBUF should be restricted to error message construction and output. Unless otherwise indicated, these buffers will not be output automatically — they must be explicitly output via the subroutines described under "Display-Line and Error-Message Output".

The five subroutines which perform data manipulation and conversion are described below.

BURST Expand a 16-bit word comprised of n nonzero-length fields into n consecutive locations in the temp stack, starting at A:FLD1 and extending through A:FLDn (where $1 \leq n \leq 16$).

Call: LINK A:BURST

Entry: A-register contains the data word to be expanded.

X-register contains the first-word address of an associated pattern table.

Exit: Contents of the n fields stored into A:FLD1 through A:FLDn (right-justified, no sign extension).

The pattern table must contain n nonzero words (where n is the number of fields in the word), each of which contains the corresponding field length (in bits). The sum of the field sizes must equal sixteen (16).

As an example, the contents of FCT1 contains eight fields whose lengths are (from left to right) 1, 1, 1, 5, 1, 1, 1, and 5 bits. To burst the contents of FCT1 entry, the following pattern table must be used:

```
DATA    1,1,1,5,1,1,1,5
```

CVDEC Converts a binary number to packed decimal.

Call: LINK A:CVDEC

Entry: A-register contains the binary number to converted. The number must be in the range $0 \leq n \leq 9999_{10}$.

Exit: A-register contains the number in packed decimal format.

CVSTORE Convert a binary number to EBCDIC (decimal or hexadecimal) and store it into a specified image buffer.

Call:

```
LINK    A:CVSTOR

DATA    X'8000'*a + X'4000'*b + X'2000'*c

DATA    d

DATA    e

DATA    f
```

where

a = 1 if number is hexadecimal.
= 0 if number is decimal.

b = 1 if T contains negative character count.
= 0 if character count is in e.

c = 1 if X contains starting column.
= 0 if starting column is in f.

d = buffer indicator, as follows:

0 = LINEBUF
1 = TYPEBUF
2 = HEADER1
3 = HEADER2

e contains positive character count and is present only if b (above) = 0.

f contains starting column number and is present only if c (above) = 0.

Entry: A-register contains binary number to be stored.

Exit: Number will be stored in appropriate position within specified buffer.

Remarks: Decimal numbers will have leading zeros suppressed.

DELZRO Delete leading zeros from an EBCDIC number contained in the E- and A-registers.

Call: LINK A:DELZRO

Entry: E- and A-registers contain a four-character, EBCDIC number.

Exit: E- and A-registers contain the same number with the first three leading zero characters (X'F0') converted to blanks (X'40').

NDECCH Calculate the number of characters needed to represent a given binary number in decimal.

Call: LINK A:NDECCH

Entry: A-register contains the binary number.

Exit: A-register contains the number of decimal EBCDIC characters required.

The above subroutines are used in conjunction with the monitor service routine M:INHEX (binary integer to hexadecimal representation in EBCDIC) to aid in data separation and conversion to text.

The following four subroutines are used to move text strings and characters into image buffers.

BLANK Blank-fill an image buffer.

Call: LINK A:BLANK

Entry: X-register contains the first-word address of an image buffer (obtained from the temp stack).

Exit: The indicated buffer will be blank-filled.

MOVE Move a text string into an image buffer.

Call: LINK A:MOVE

DATA n

where

n indicates the buffer, as follows:

0 = LINEBUF
1 = TYPEBUF
2 = HEADER1
3 = HEADER2

Entry: A-register contains the first-word address of the message.

X-register contains the column number where the first character is to be stored.

Exit: Message will have been moved into the appropriate position within the designated buffer.

The message must be in TEXTC format.

The first column printed is in column 1. If the message contains a format control character as the first byte, it must start in column 0.

STBYTE Store a byte into an image buffer.

Call: LINK A:STBYTE

Entry: A-register contains the column number where the character is to be printed (0 for format byte).

E-register contains the total number of bytes in the message (129 for the line printer and 85 for the teletype).

X-register contains the first-word address of the image buffer (obtained from temp stack).

T-register contains the data byte right-justified.

Exit: Data byte will be stored in the appropriate position within the indicated buffer.

STCHAR Store EBCDIC characters contained in the E- and A-registers into designated positions within an image buffer.

Call: LINK A:STCHAR

DATA n

where

n indicates image buffer, as follows:

0 = LINEBUF
1 = TYPEBUF
2 = HEADER1
3 = HEADER2

Entry: E- and A-registers contain the character string right-justified.

T-register contains the negative of the number of characters to be stored (-1 through -4).

X-register contains the column number where the first character is to be stored (0 for format byte).

Exit: The characters will be stored into the appropriate positions within the indicated image buffer.

It should be noted that the above subroutines require the column numbers where characters are to be printed and not the byte position within the buffer. This is because even/odd byte count adjustments are made within the subroutines.

Two special-purpose subroutines are available which format and output decimal number sequences. They are described below.

SEQADD Adds a number to the decimal number sequence being constructed in LINEBUF.

Call: LINK A:SEQADD

Entry: A-register contains the binary number to be added. If the number is negative, the sequence will be terminated.

Exit: The number will be added to the sequence.

The subroutine will construct the sequence in LINEBUF in decimal EBCDIC representation. When the image buffer is full or when a negative number is input to terminate the string, the current contents of LINEBUF will be printed automatically. Printing of partially completed sequences is transparent to the calling program.

The subroutine will determine whether or not a number input is part of a range. If not, the number will be converted to decimal EBCDIC and appended to the current contents of LINEBUF individually. If a number is part of a range (i.e., a series of consecutive numbers), the range upper limit will be updated. The entire range will be added to the sequence (in the form XX-YY) when the first number is input which is not equal to the previous number plus one.

Note: Numbers may be added in any order, but a number range will be constructed only if the numbers are input in incrementally increasing order.

If the first number input to the subroutine is negative, the message "NONE" will be output in place of a sequence.

The variable A:CURCOL will be used to indicate the column (within LINEBUF) where the next number is to be stored. LINEBUF may not be used while a sequence is being constructed.

SEQST Initialize the decimal number sequence for the SEQADD subroutine.

Call: LINK A:SEQST

Return: LINEBUF will be blank-filled; A:CURCOL will be preset to 4.

This subroutine must be called prior to calling SEQADD to add the first element to a sequence.

Display-Line and Error-Message Output. To provide for the maximum rate of output, a PRINT subroutine is provided that handles all display-line output in a manner permitting compute-print overlap. This routine is described below along with a message-printing subroutine (for LO).

PRINT Output the contents of an image buffer to the LO device.

Call: LINK A:PRINT

Entry: A-register contains the address of the image buffer (obtained from the temp stack).

Exit: Output operation will have been initiated.

A:LINENO will show the line number of the next line to be printed.

If the printer is busy when the subroutine is entered, it will wait until the printer becomes free. The contents of the designated buffer will then be moved to a side buffer for output and a write will be initiated with error record recovery and without wait. The image buffer will, therefore, be free for use upon return.

If the attempt to initiate the I/O is unsuccessful, an error message will be output indicating the error-return and ANALYZE will be aborted with code 'IO'.

MESSAGE Output a message to the LO device.

Call: LINK A:MSG

DATA message address

DATA Substitute format byte or 0.

The message must be in TEXTC format with a format control character as the first character.

If the second argument is zero, formatting will be as specified within the message. If the second argument is nonzero, it is assumed to contain a valid format byte which is to be substituted for the one in the message. Valid format bytes are as follows:

EBCDIC	Hex	Function
0	F0	Double-space
1	F1	Top-of-form
space	40	Single-space
-	60	Inhibit auto upspacing

The subroutine uses LINEBUF.

Two additional subroutines are available for outputting error-diagnostic information. They are described below.

IOERR Format and output an error message to the OC device for I/O errors.

Call: LINK A:IOERR

Entry: A-, E-, and X-registers contain the status returned from M:READ, M:WRITE, or M:CTRL.

Exit: An error message will have been output to the OC device.

The image buffer TYPEBUF will be used for construction and output of the error message. It will be free for use upon return; original contents will not be preserved. I/O is performed with wait.

TYPE Output the current contents of TYPEBUF to the OC device.

Call: LINK A:TYPE

Entry: Message in TYPEBUF.

Exit: Message output to OC with wait.

I/O will be performed with wait and error recovery. If an unrecoverable I/O error occurs, ANALYZE will be aborted with code 'IO'.

Control Subroutines. Several of the common subroutines are used primarily for control purposes; i.e., selection of display functions. They are as follows.

CPSTRING Compare a text string with a parameter table entry.

Call: LINK A:CSTRNG

DATA comparison text-string pointer

Entry: X-register points to word one of a parameter table entry.

Exit: Carry indicator will be set if the strings are the same and will be reset if they are different. The subroutine call may, therefore, be followed by a BE or BNE instruction.

X-register contents will be preserved.

The parameter entry must be a valid, non-null, EBCDIC entry. The comparison text string must be in TEXTC format; the string lengths must be equal.

GETPAR Get the address of the next entry in the parameter table created by the SCAN subroutine (described following).

Call: LINK A:GETPAR

(return if no more parameters)

(normal return)

Entry: A:PARE contains the address of the entry to be processed.

Exit: A:PARE unpated to show the address of the following entry. X-register contains the address of word 1 of the current entry.

E-register contains the number of words in the parameter excluding the key word (word 0).

A-register contains a code indicating the entry type as follows:

A = -1	null
A = 0	EBCDIC
A = 1	single integer
A = 2	integer range
A = 3	illegal (syntax error)

A:PAR incremented by one. This variable shows the parameter number currently being processed. (Used for error messages.)

The parameter entries are organized as a key word (word 0) followed by n words containing the parameter ($n \geq 0$). Decimal and hexadecimal values will be converted to binary, EBCDIC strings will be stored in consecutive locations

(2 bytes/word) in TEXTC format. EBCDIC strings containing an even number of characters will have a trailing blank appended. The following examples show the significance and structure of the various entry types:

Null Entry:

	0 1	2 15
Word 0	0	0

EBCDIC Entry:

<table>
<tr><th></th><th>0 1</th><th>2 7</th><th>8 15</th></tr>
<tr><td>Word 0</td><td>0</td><td colspan="2">$\frac{\text{Char. Count} + 1}{2}$</td></tr>
<tr><td>Word 1</td><td colspan="2">Char. Count</td><td>Char. 1</td></tr>
<tr><td>Word 2</td><td colspan="2">Char. 2</td><td>Char. 3</td></tr>
<tr><td>⋮</td><td colspan="3">⋮</td></tr>
<tr><td>Word (n/2) + 1</td><td colspan="2">Char. n-1</td><td>Char. n</td></tr>
</table>

Single Integer Entry:

<table>
<tr><th></th><th>0 1</th><th>2 15</th></tr>
<tr><td>Word 0</td><td>1</td><td>1</td></tr>
<tr><td>Word 1</td><td colspan="2">Number</td></tr>
</table>

Integer Range Entry:

<table>
<tr><th></th><th>0 1</th><th>2 15</th></tr>
<tr><td>Word 0</td><td>2</td><td>2</td></tr>
<tr><td>Word 1</td><td colspan="2">First Number</td></tr>
<tr><td>Word 2</td><td colspan="2">Second Number</td></tr>
</table>

Illegal Entry:

	0 1	2 15
Word 0	3	0

Last Entry:

	0 1	2 15
Word 0	3	X'3FFF'

Return will be made to the call plus one if there are no more parameters; otherwise, return will be made to the call plus two.

SCAN Scan the !ANALYZE control card and create a parameter table composed of entries whose formats were described above.

Call: LINK A:SCAN

(return if no parameters)

(normal return)

Exit: A:PARE contains the address of the first entry.

A:PARTBL points to a parameter-entry table of n + 1 entries (where n is the number of !ANALYZE parameters). Each entry is of the form described above for GETPAR.

A:PAR initialized to zero. (This variable contains the number of the parameter currently being processed).

Return will be made to the call plus one if there were no parameters; otherwise, return will be made to the call plus two. The last entry in the table will always contain a key of -1.

SEGLD Load and transfer to a display-function overlay.

Call: LINK A:SEGLD

DATA segment-ID

(error return)

(normal return)

Prior to calling the function overlay, the subroutine will set the LDWD error-transfer address (A:LDERRX) and GTCTXTWD error-transfer address (A:CXTERX) to exit back to the root.

Initialization Routine

The initialization routine is entered when ANALYZE is first entered. It will perform the following functions:

- Initialize base register.
- Verify that the LO oplabel is assigned and set variables dependent upon the output-device type.
- Initialize run-time variables in the base table.
- Read first data block of SYSERR file into A:BLK0 if oplabel SI is not assigned to zero.

The initialization routine is flowcharted in Figure 79.

Control Routines

The central control routine first determines whether there are any parameters on the !ANALYZE control card. If not, all display functions are individually called in a fixed order. If there are parameters, the appropriate group-control routine is called to set up the parameters and call the individual displays within the group. When the group is completed, ANALYZE will exit to JCP through M:TERM.

The control routines are flowcharted in Figure 80.

Overlay Segments

The overlay segments contain the code which actually produces the various displays — one display per segment. Regardless of the function of a segment, several rules must be followed in its construction. They are:

- A pseudo base table must be assembled into the beginning of each segment. This may be accomplished with the procedure A:BASE, having an argument greater than zero.
- Overlays should not do their own I/O — the subroutines provided in the root should be used.
- Upon completion, overlays must exit *A:OVEXIT. This will effect a return to the segment-load subroutine (SEGLD) which will, in turn, return to the appropriate control routine.

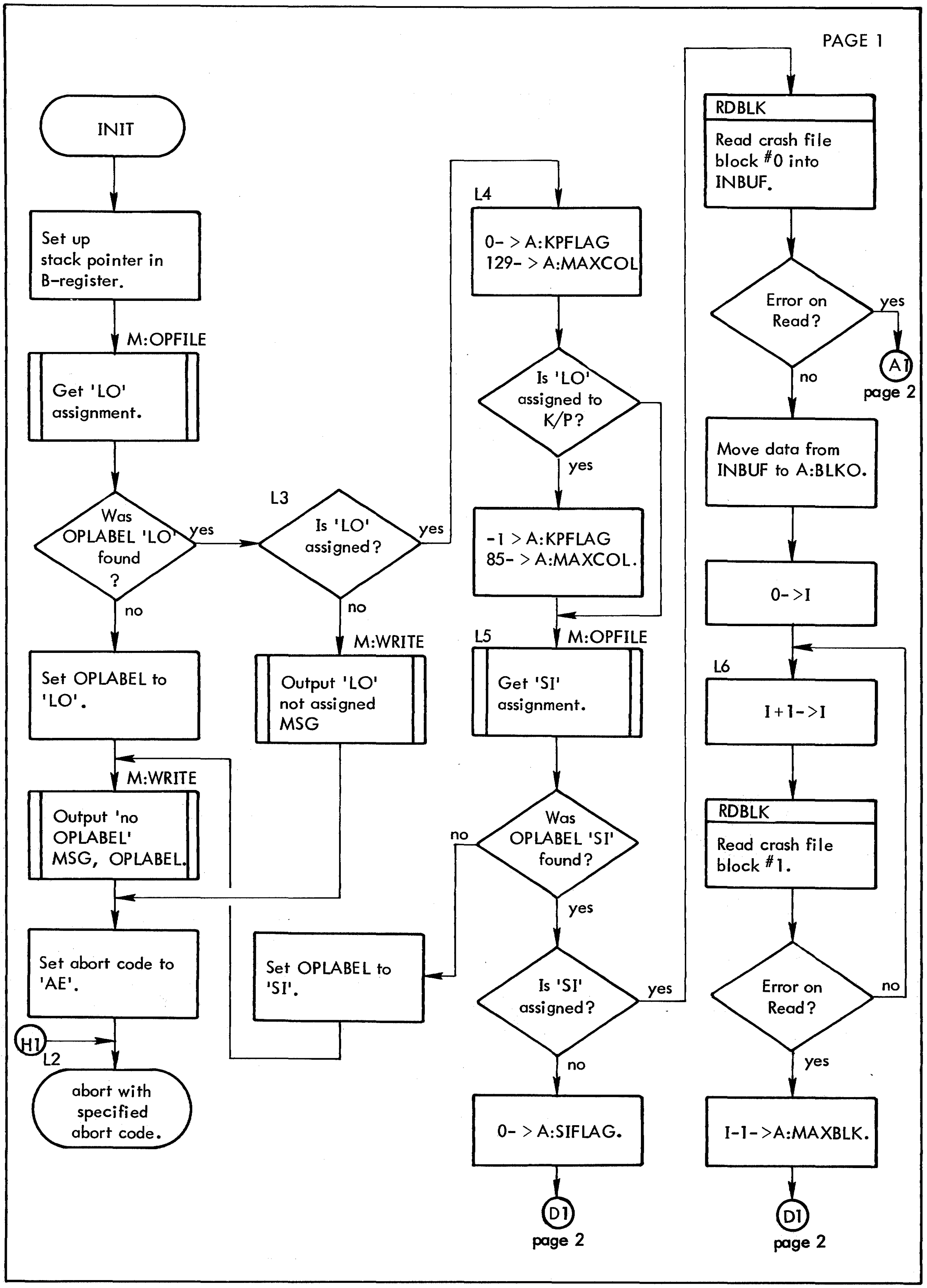


Figure 79. Initialization Routine

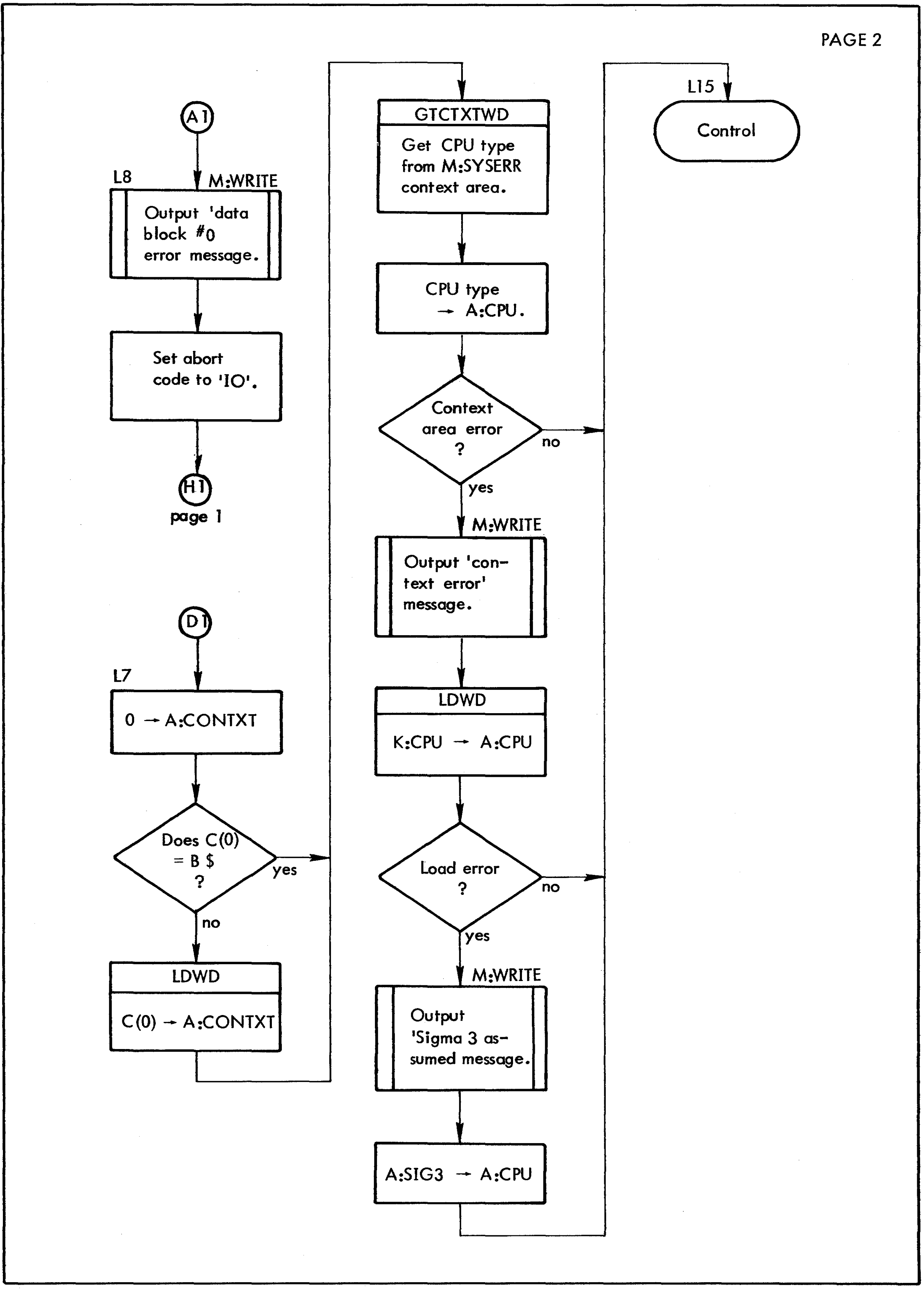


Figure 79. Initialization Routine (cont.)

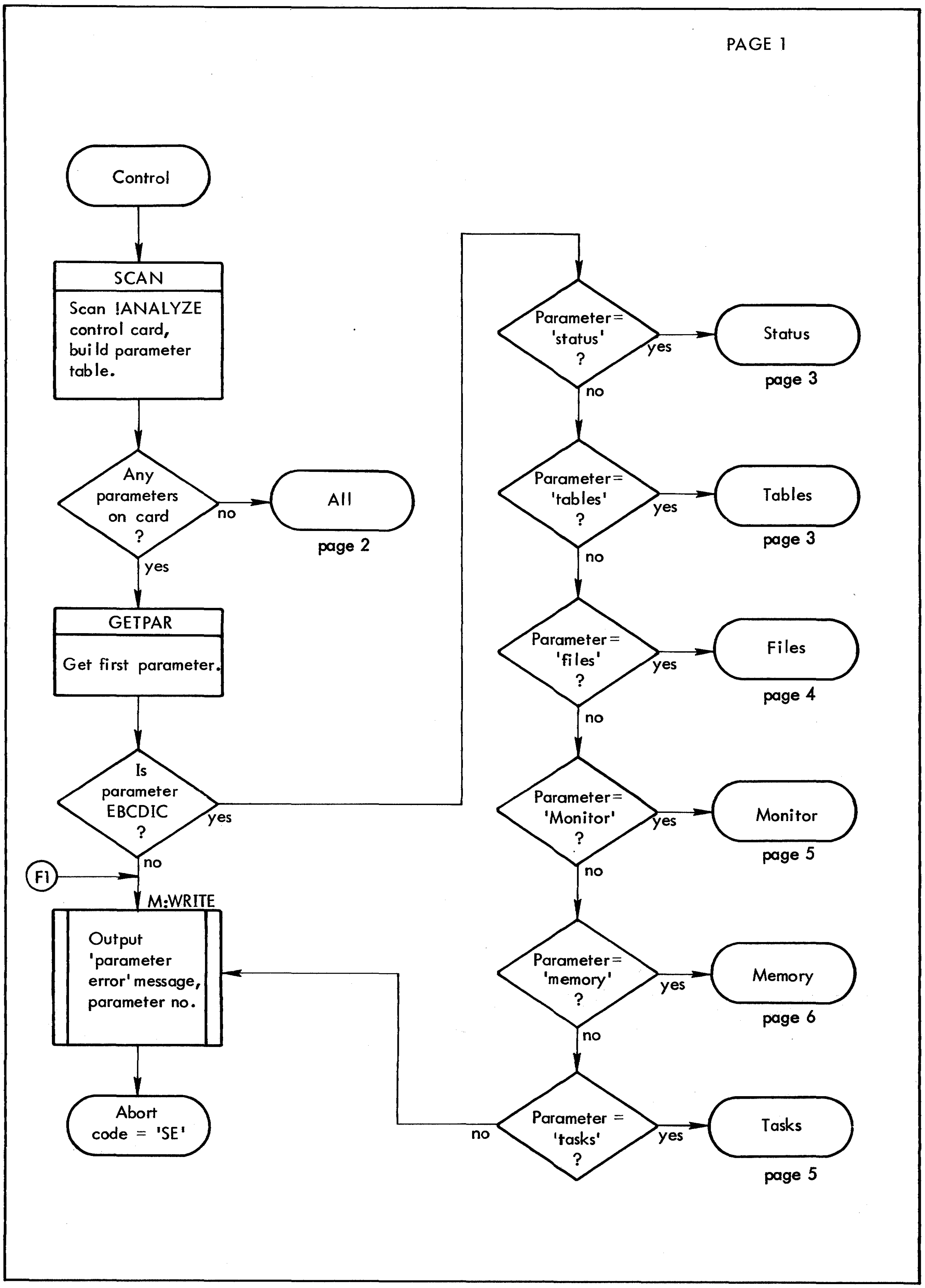
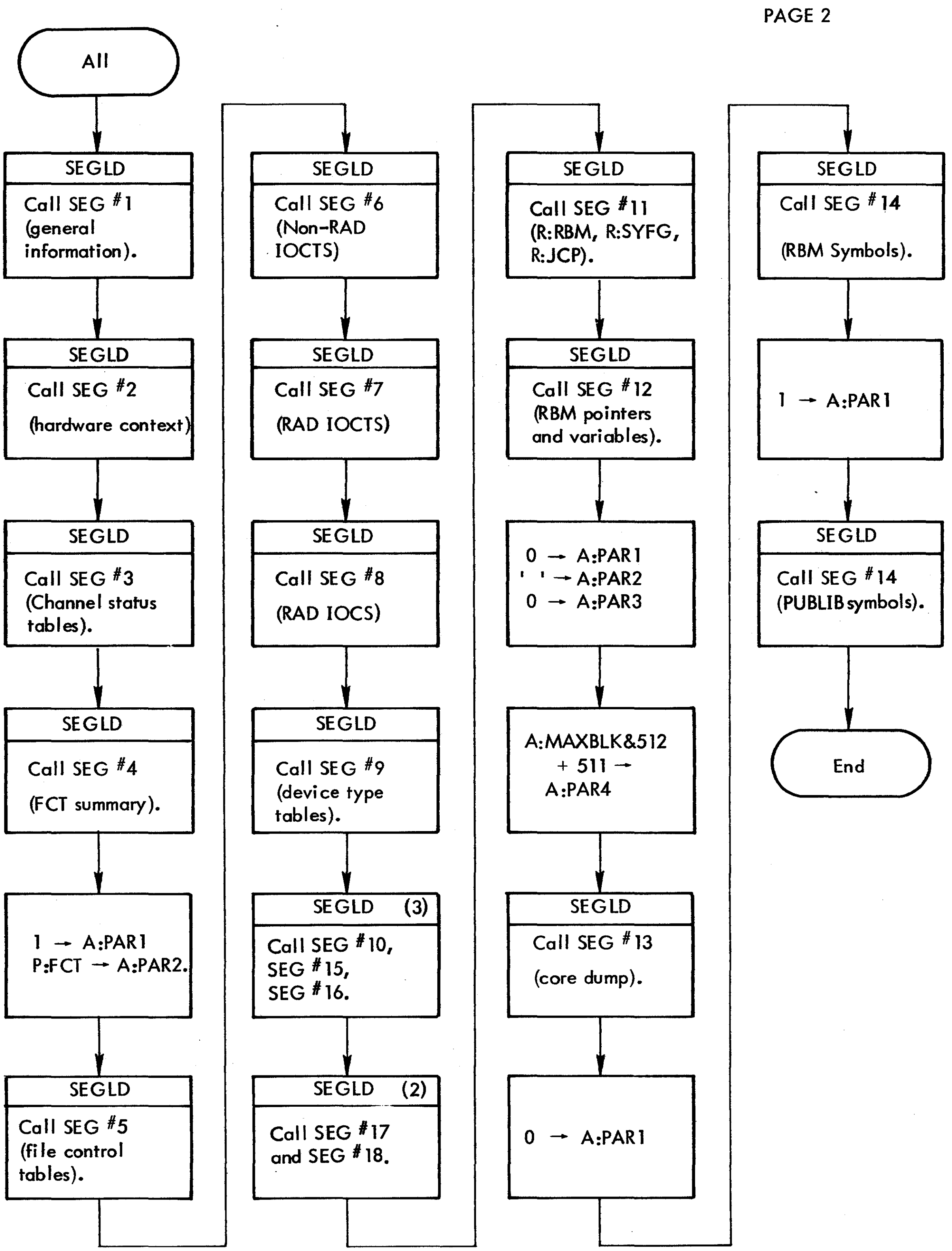
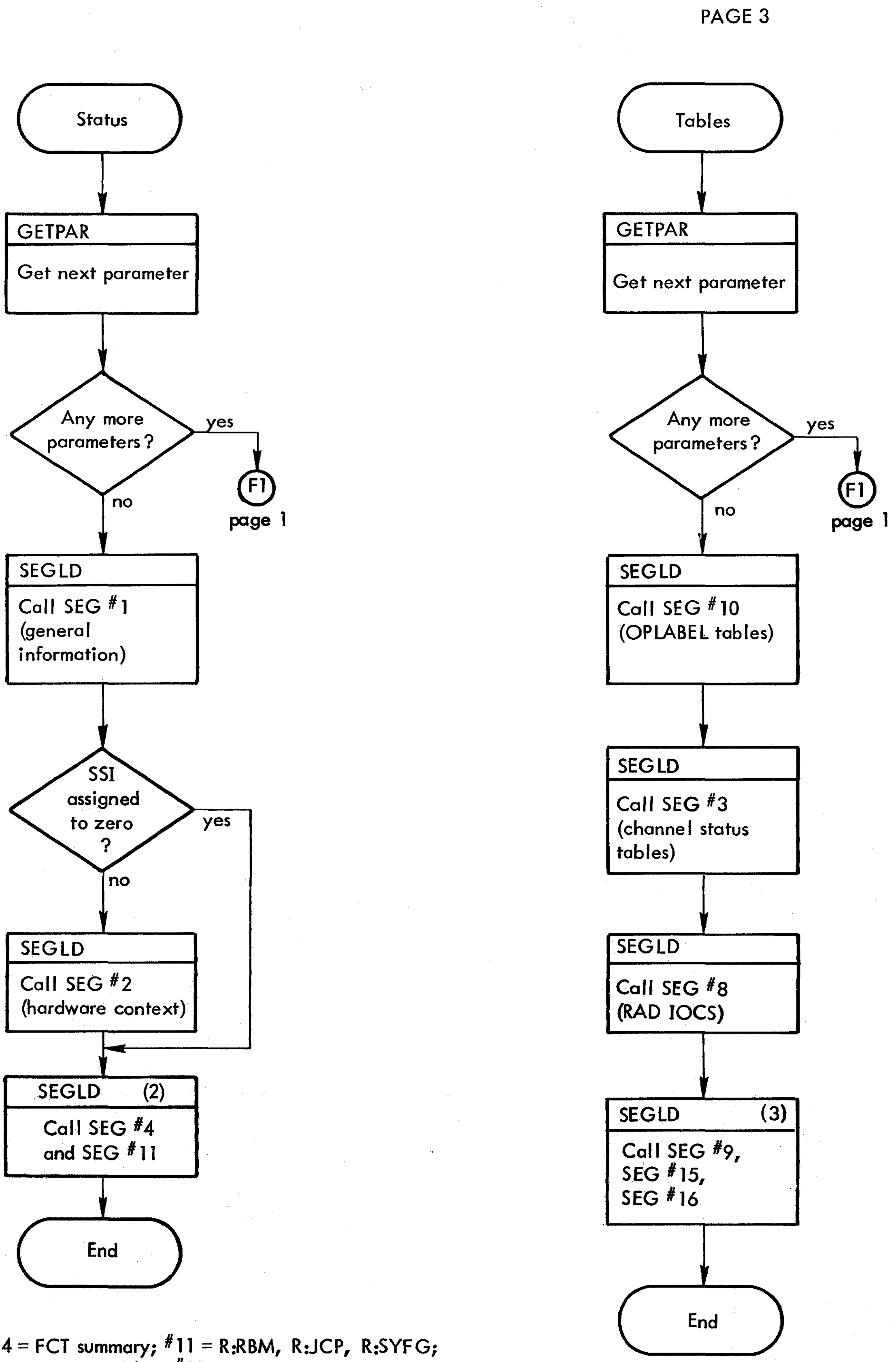


Figure 80. Control Routines


PAGE 2
All
SEGLD
Call SEG #1 (general information).
SEGLD
Call SEG #2 (hardware context)
SEGLD
Call SEG #3 (Channel status tables).
SEGLD
Call SEG #4 (FCT summary).
1 → A:PAR1
P:FCT → A:PAR2.
SEGLD
Call SEG #5 (file control tables).
SEGLD
Call SEG #6 (Non-RAD IOCTS)
SEGLD
Call SEG #7 (RAD IOCTS)
SEGLD
Call SEG #8 (RAD IOCS)
SEGLD
Call SEG #9 (device type tables).
SEGLD (3)
Call SEG #10, SEG #15, SEG #16.
SEGLD (2)
Call SEG #17 and SEG #18.
SEGLD
Call SEG #11 (R:RBM, R:SYFG, R:JCP).
SEGLD
Call SEG #12 (RBM pointers and variables).
0 → A:PAR1
' ' → A:PAR2
0 → A:PAR3
A:MAXBLK&512 + 511 → A:PAR4
SEGLD
Call SEG #13 (core dump).
0 → A:PAR1
SEGLD
Call SEG #14 (RBM Symbols).
1 → A:PAR1
SEGLD
Call SEG #14 (PUBLIB symbols).
End

SEG #10 = Oplabel tables; #15 = Master dictionary; #16 = RBM overlay table; #17 = Interrupt PSDs; #18 = Active task TCBs.

Figure 80. Control Routines (cont.)



SEG #4 = FCT summary; #11 = R:RBM, R:JCP, R:SYFG;
#9 = Device type tables, #15 = Master dictionary;
#16 = RBM overlay table.

Figure 80. Control Routines (cont.)

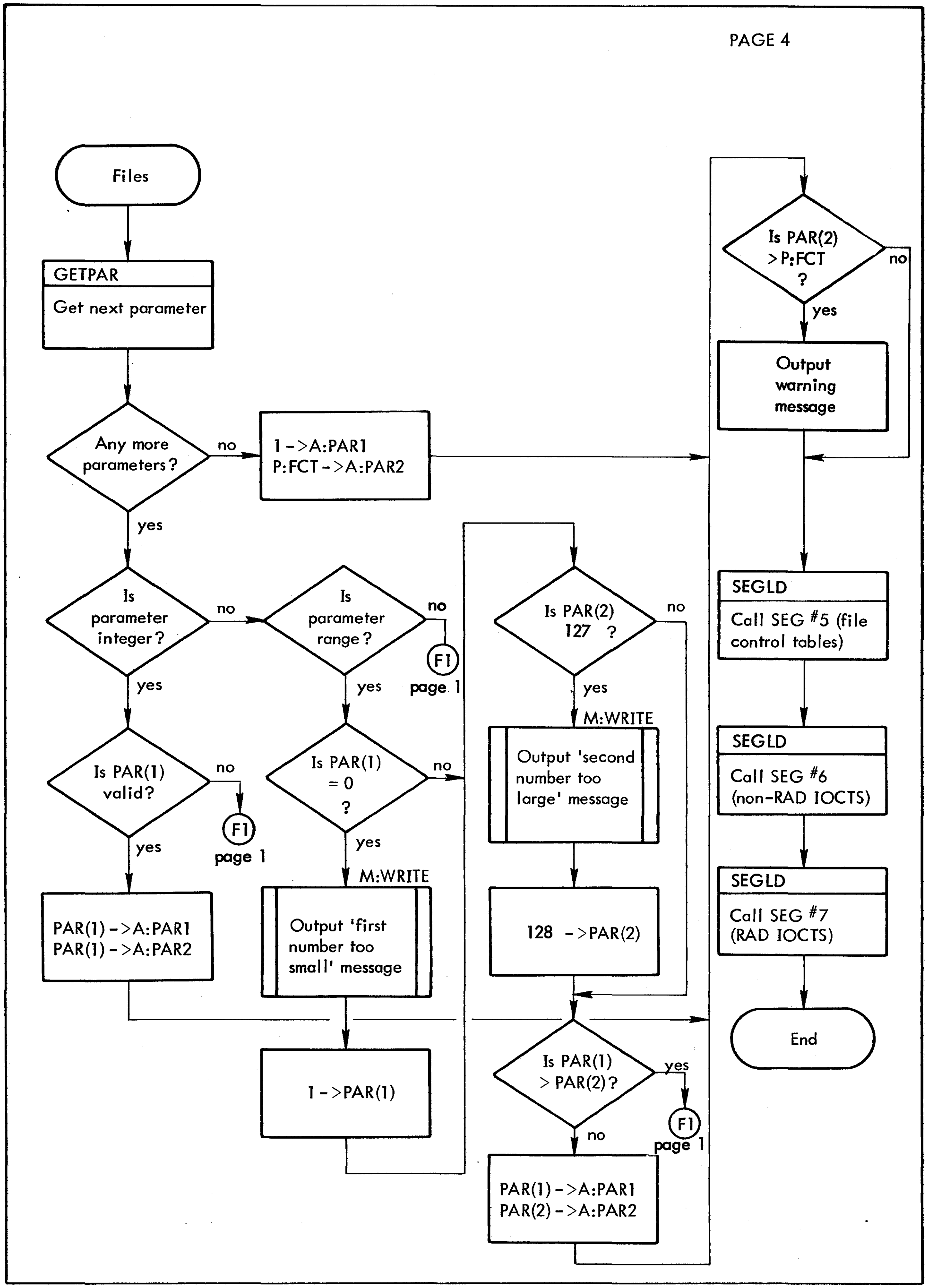


Figure 80. Control Routines (cont.)

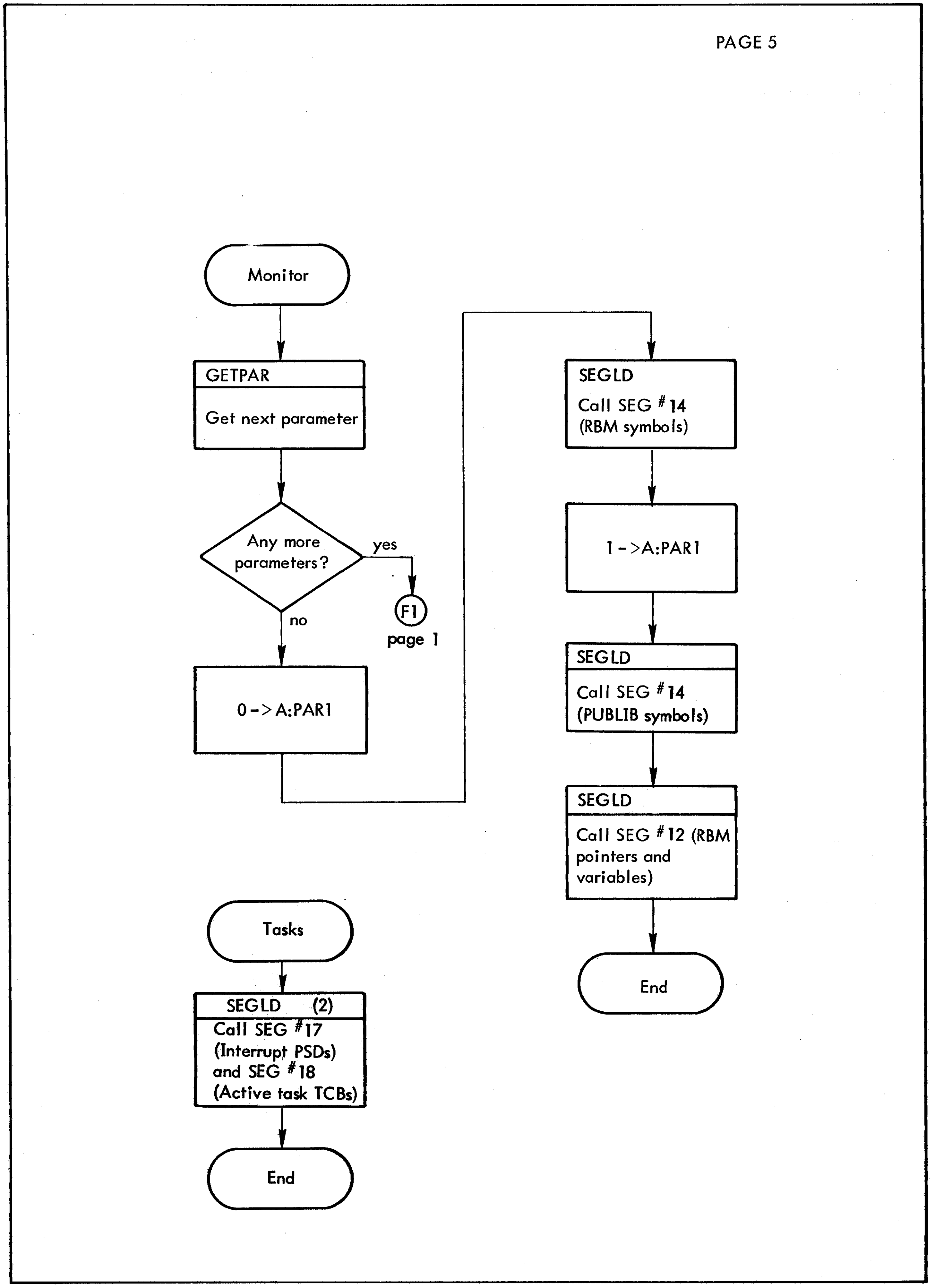

PAGE 5
Monitor
GETPAR
Get next parameter
Any more parameters?
yes
F1
page 1
no
0->A:PAR1
SEGLD
Call SEG #14 (RBM symbols)
1->A:PAR1
SEGLD
Call SEG #14 (PUBLIB symbols)
SEGLD
Call SEG #12 (RBM pointers and variables)
End
Tasks
SEGLD (2)
Call SEG #17 (Interrupt PSDs) and SEG #18 (Active task TCBs)
End

Figure 80. Control Routines (cont.)

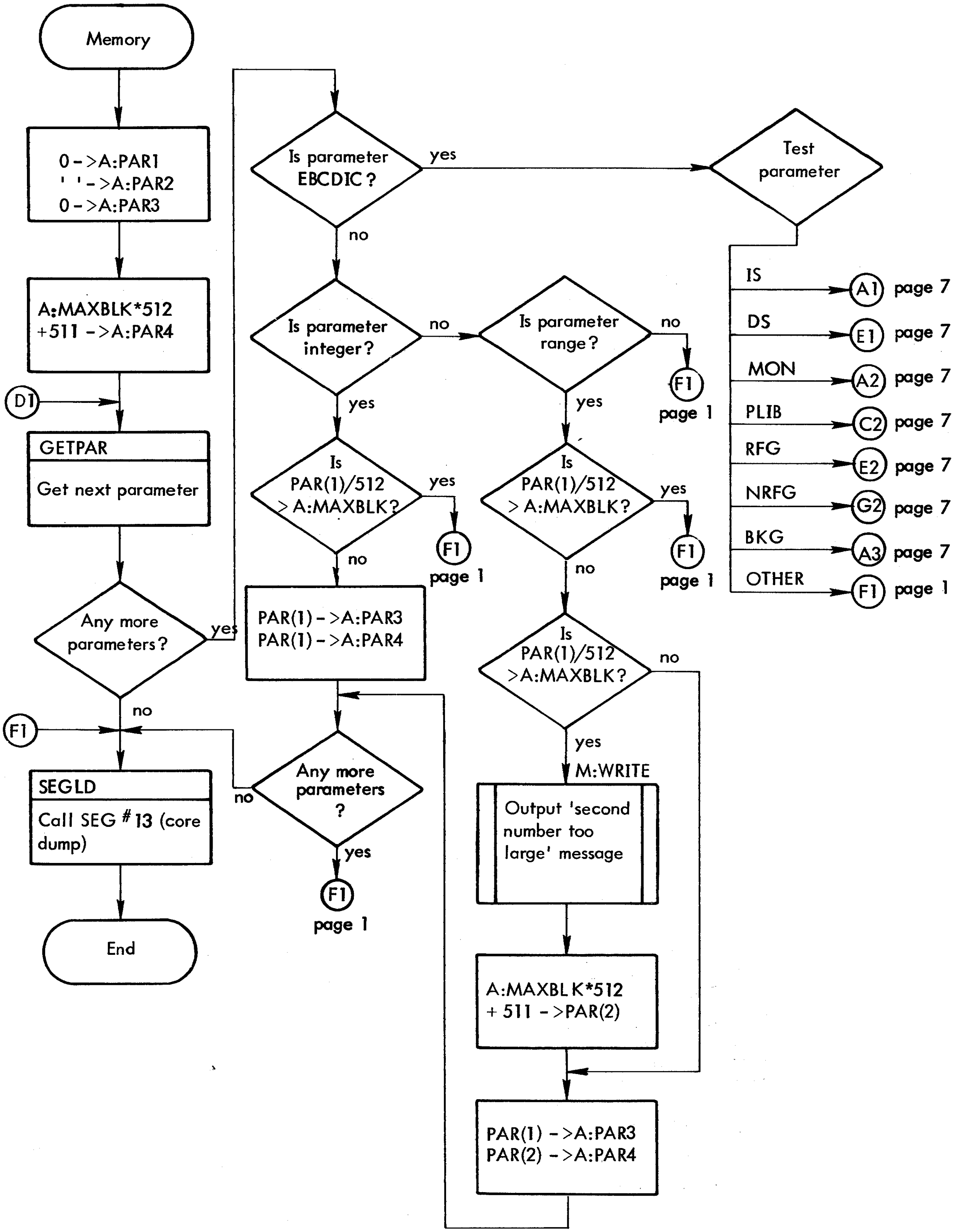


Figure 80. Control Routines (cont.)

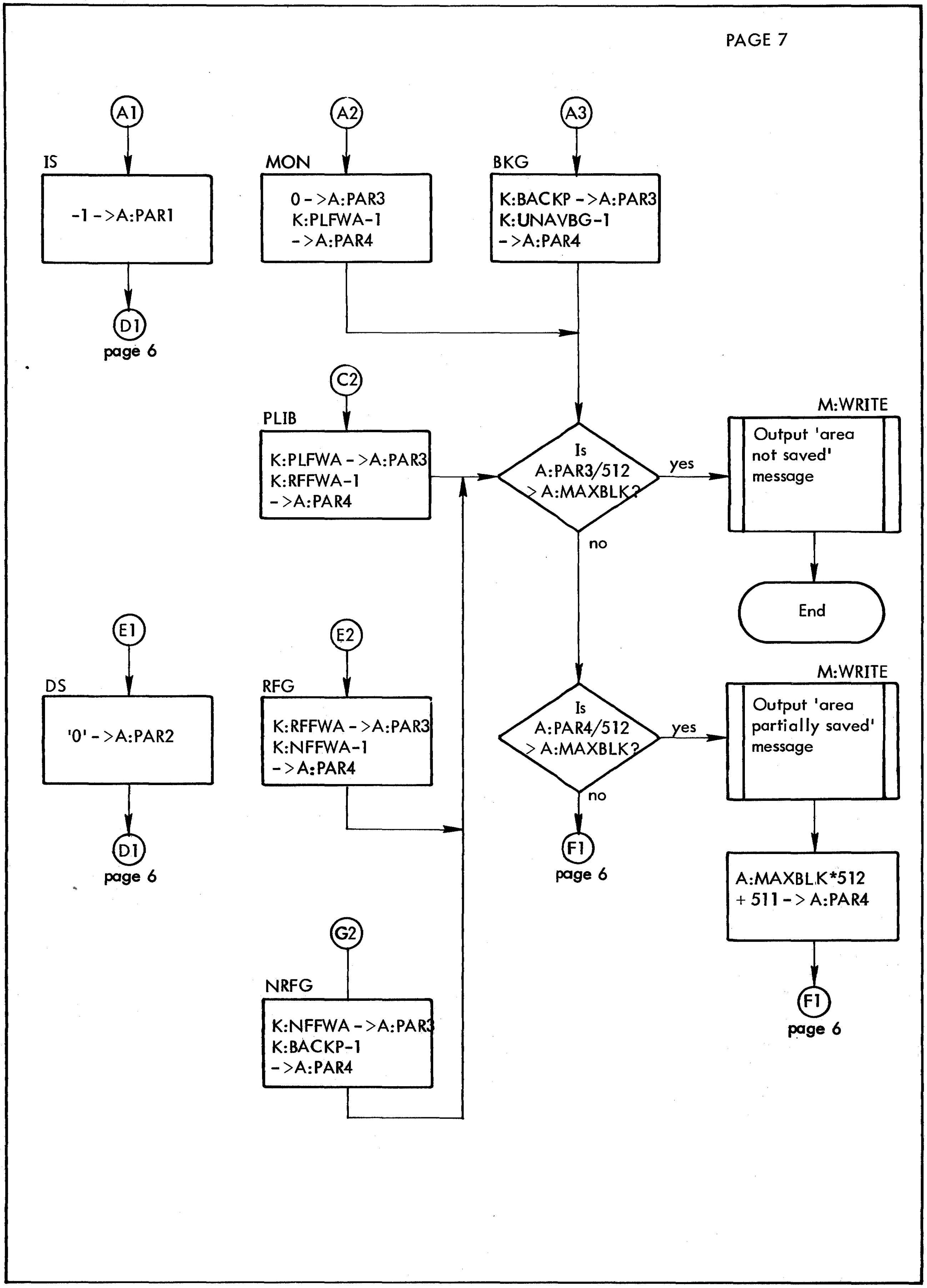


Figure 80. Control Routines (cont.)

- Overlays may not call other overlays.
- Overlays may use the temp locations A:TEMP1 through A:TEMP20 for their temporary storage. Base table locations A:PAR1 through A:PAR10 are used to pass control information and boundary conditions to overlays. These locations may not be changed by the overlays.
- Upon entry to the overlay segments, the base register will point to the base table.
- Upon entry to the overlay segments, the LDWD, LDNXT, and GTCTXTWD error exits contain a pointer back to the root.
- A transfer address must exist on the END statement for the overlay.

Procedures

The ANALYZE procedure set will be assembled into the S24RBM file if the label #ANALYZE is equated with YES in the source file. Some of these procedures utilize one or more common subroutines to accomplish their function, others do not. It should be assumed that all procedures that generate executable code will utilize all registers with the exception of the base register (B). The exceptions are ENTRY (A-register altered), LITERALS (no registers altered), and SETERRX (A-register altered). The procedures and their descriptions are as follows.

BLANK Blank-fill an image buffer.

Call: [label] BLANK buffer

where buffer = LINEBUF, TYPEBUF, HEADER1, or HEADER2

BURST Expand the data word in the accumulator into n words in the temp stack starting at A:FLD1.

Call: [label] BURST pattern

where pattern is the first-word address of the burst pattern (see the description of the BURST subroutine under "Common Subroutines", above).

CPSTRING Compares a text string (in TEXTC format) with the parameter entry pointed to by the X-register.

Call: [label] CPSTRING address

where address is the address of a comparison text string if no argument field asterisk (AFA). If an AFA is present, address contains a pointer to the text string.

The X-register must point to word 1 of a valid non-null, EBCDIC entry and will not be altered. The procedure call may be followed by a BE or BNE instruction.

CVSTORE Convert a binary word to printable hexadecimal or decimal and store it into an image buffer.

Call: [label] CVSTORE, cf_2 [, cf_3] af_1, af_2, af_3

where

cf_2 indicates the output format: DEC or HEX.

cf_3 optionally specifies the address of binary data to be processed. If not present, the current A-register contents will be used.

af_1 equals the number of characters if no argument field asterisk (AFA). If an AFA is present, af_1 is the address of a location containing the number of characters.

af_2 indicates the target buffer (LINEBUF, TYPEBUF, HEADER1, or HEADER2).

af_3 specifies the column number where the first character is to be stored if there is no AFA. If there is an AFA, af_3 specifies the address of a location containing the starting column number.

Example: CVSTORE,DEC,LOC 3,LINEBUF,*ADDR

Convert the binary number contained in LOC to decimal EBCDIC, and store the three least-significant decimal characters into LINEBUF, starting at the column number contained in ADDR.

DISPLAY Control PROC expansion listing.

Call: DISPLAY value

If value is nonzero, the listing of the PROC expansion will be controlled by the LIST directive currently in effect. If value is zero, the expansion will not be listed and the location counter will be displayed.

ENTRY Mark the entry to a subroutine and optionally save one or more registers.

Call: [label] ENTRY [loc, r_1[,$r_2 \ldots r_n$]]

where

loc is the address where the contents of the first register are to be stored.

r_1 is the register number of the first register to be saved.

r_i are the second and subsequent registers to be saved.

Registers will be stored in consecutive locations (beginning at loc) in the order indicated in the call. If the A-register is to be saved, it must be the first register specified, since storage of other registers requires use of the accumulator.

GET#CHAR Find the number of characters necessary to represent a binary number in decimal.

Call: [label] GET#CHAR[,cf_2] [af_1]

where

cf_2 is the optional address of the location containing the binary number. If not specified, the current accumulator contents will be used.

af_1 is the optional address of the location where the character count is to be stored. If not supplied, the count will be returned in the accumulator.

GETPAR Get the pointer to the next parameter in the parameter list.

Call: [label] GETPAR

Return will be to the call plus one if there are no more parameters; otherwise, return will be to the call plus two. Upon return, registers will be set as described for the GETPAR subroutine (described under "Common Subroutines").

GTCTXTWD Get the contents of a specified word within the M:SYSERR context area.

Call: [label] GTCTXTWD af_1,af_2

where

af_1 is the key of the desired context block if no AFA. If an AFA is specified, af_1 is the address of the location containing the key.

af_2 is the displacement of the desired word from the top of the entry if no AFA. If AFA is specified, af_2 is the address of the location containing the displacement.

IOERR Output I/O error status to the OC device (with WAIT).

Call: [label] IOERROR

LITERALS Generate a literal pool with a branch around it.

Call: [label] LITERALS

LOAD Load the contents of a specified location (from the SYSERR file) into the accumulator.

Call: [label] LOAD [[*]address[, index]]

Computation of the effective address is the same as for a LDA instruction with one exception: "index" is the location containing the index value.

If no address is specified, the current contents of the accumulator will be used as the effective address.

At return, A:CURADD will contain the effective address of the LOAD.

Examples: LOAD K:SEGIN

Fetch the contents of K:SEGIN from the SYSERR context.

```
LDA      =5
STA      INDEX
LOAD     *P:FCT5,INDEX
```

Fetch the contents of the fifth entry in the table whose pointer is contained in P:FCT5.

LOADNEXT Load the contents of the next consecutive location from the SYSERR dump into the accumulator.

Call: [label] LOADNEXT

The effective address will be contained in A:CURADD upon return.

LOADSEG Load and transfer to a display segment.

Call: [label] LOADSEG id

where id is the number of the desired segment.

MESSAGE Output a message to the LO device using LINEBUF.

Call: [label] MESSAGE[, cf_2] address

where

cf_2 is the optional address of a format control byte to be substituted (TOF, SS, DS, NS are the standard format bytes for Top-of-Form, Single-Space, Double-Space, and No-Space).

address is the address of a text string in TEXTC format. The first byte of the message is assumed to be the format byte.

MOVE Move a TEXTC message into a designated image buffer.

Call: [label] MOVE [af$_1$], af$_2$ [, af$_3$]

where

af$_1$ is a TEXTC message address if there is no argument field asterisk (AFA). If an AFA is present, af$_1$ is the address of a location containing the message address.

af$_2$ is the image buffer (LINEBUF, TYPEBUF, HEADER1, or HEADER2).

af$_3$ is the starting column number if there is no AFA. If an AFA is present, af$_3$ specifies the location containing the starting column number.

If af$_1$ is omitted, the accumulator must contain the message address and af$_3$ must be present.

If af$_3$ is omitted, the buffer will be first blank-filled and the message will start at column zero (format byte column). If af$_3$ is present, the buffer will not be initially blank-filled.

The message must be in TEXTC format.

PRINT Output the contents of an image buffer through oplabel LO.

Call: [label] PRINT [buffer]

where buffer is the image buffer to be printed (LINEBUF, HEADER1, or HEADER2). If not specified, LINEBUF will be printed.

SEQADD Add a number to the decimal number sequence being constructed in, and output from, LINEBUF.

Call: [label] SEQADD [address]

where address (if specified) is the location containing the number to be added. If address is not specified, the current A-register will be used.

SEQEND Terminate the decimal number sequence constructed by SEQADD.

Call: [label] SEQEND

SEQSTART Initialize a decimal number sequence.

Call: [label] SEQSTART

This PROC must be issued prior to building a decimal sequence with SEQADD. The sequence must be terminated with SEQEND.

SETERRX Define an error transfer address from the LDWD subroutine.

Call: [label] SETERRX $\left\{\begin{array}{l}\text{LOAD}\\ \text{LOADNEXT}\\ \text{GTCTXTWD}\end{array}\right\}$ [, address]

where address is the address to which the indicated subroutine is to transfer if it encounters an error. If an AFA is present, address specifies the location containing the transfer address.

If "address" is omitted, the transfer address will be cleared. A subsequent error will cause a return to the subroutine call plus one with the carry indicator reset.

SKIPLINE Output a blank line to the LO device.

Call: [label] SKIPLINE

This PROC uses LINEBUF.

STBYTE Store the byte right-justified in the accumulator into the designated buffer.

Call: [label] STBYTE af_1,af_2

where

af_1 is the destination buffer (LINEBUF, TYPEBUF, HEADER1, or HEADER2).

af_2 is the column number if no AFA. If an AFA is present, af_2 specifies the location containing the column number.

TYPE Output the current contents of TYPEBUF through oplabel OC (with WAIT).

Call: [label] TYPE

8. ERROR LOGGING AND DEVICE ISOLATION

Error Logging

When error logging is specified at SYSGEN time, the default M:DOW linkage code – simply a call to the diagnostic-output-writer overlay (A6) – is replaced by an alternate module (DEF = M:DOWE), and an additional overlay (06) is included in the monitor. Figure 81 shows the flow of the resident M:DOW code when error logging is specified. Figure 82 shows the flow of the associated non-resident code (overlay 06), which essentially performs the I/O between the resident ("circular") buffer and the error-log file when required.

Also at SYSGEN time, the error-log file is automatically allocated in the SD area with the following attributes:

Name:	ERRFILE
Logical record size:	30 if K:BLOCK = 180. 32 if K:BLOCK = 2^n (where n is any positive integer).
File size:	16 blocks if K:BLOCK = 180. 6 blocks if K:BLOCK = 512. (100 records assuming default resident buffer size.)

Error Log Formats

Figures 83 through 94 show the detailed format of Xerox 530 and Sigma 2/3 RBM error-log entries. The following generalities relating to the formats should be noted:

- Relative time is expressed as milliseconds since midnight and is included only if CLOCK 1 is dedicated to RBM at SYSGEN time. Otherwise, relative time will always be that last entered by the operator.
- All system error-log entries are fixed length. The byte length may be either 30 or 32 but only the first 30 bytes are meaningful. Unused words always contain zeros.
- The current release of RBM error logging defines the following entry codes:

Code	Entry Type
00	Null entry (byte 0 = 0).
18	System startup ("boot").
20	Power on.
23	Date and time ("time stamp").
27	Operator message.
91	SIO rejection.
92	Device timeout (only if CLOCK 1 dedicated).
93	Spurious I/O interrupt.
95	Device error: any condition where the UE flag is set but some flag other than IL is also set. Software and hardware write-protect violations will not be logged, however.
9E	Lost entries.
A1	Configuration.
A2	System identification.
B1	Machine fault interrupt.

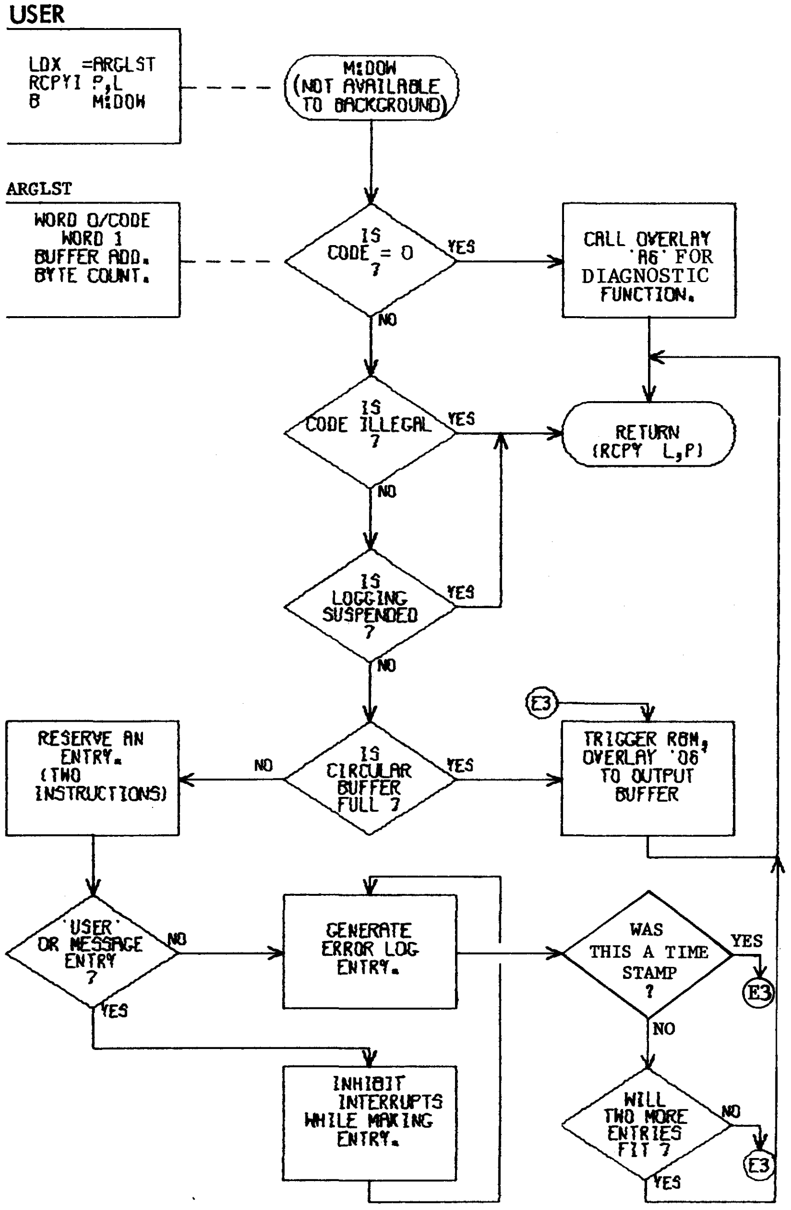

USER
LDX =ARGLST
RCPYI P,L
B M:DOW
M:DOW
(NOT AVAILABLE TO BACKGROUND)
ARGLST
WORD 0/CODE
WORD 1
BUFFER ADD.
BYTE COUNT.
IS CODE = 0 ?
YES
NO
CALL OVERLAY 'A6' FOR DIAGNOSTIC FUNCTION.
IS CODE ILLEGAL ?
YES
NO
RETURN
(RCPY L,P)
IS LOGGING SUSPENDED ?
YES
NO
E3
RESERVE AN ENTRY.
(TWO INSTRUCTIONS)
NO
IS CIRCULAR BUFFER FULL ?
YES
TRIGGER RBM, OVERLAY '06' TO OUTPUT BUFFER
'USER' OR MESSAGE ENTRY ?
NO
YES
GENERATE ERROR LOG ENTRY.
WAS THIS A TIME STAMP ?
YES
E3
NO
INHIBIT INTERRUPTS WHILE MAKING ENTRY.
WILL TWO MORE ENTRIES FIT ?
NO
E3
YES

Figure 81. Resident Error Log Code

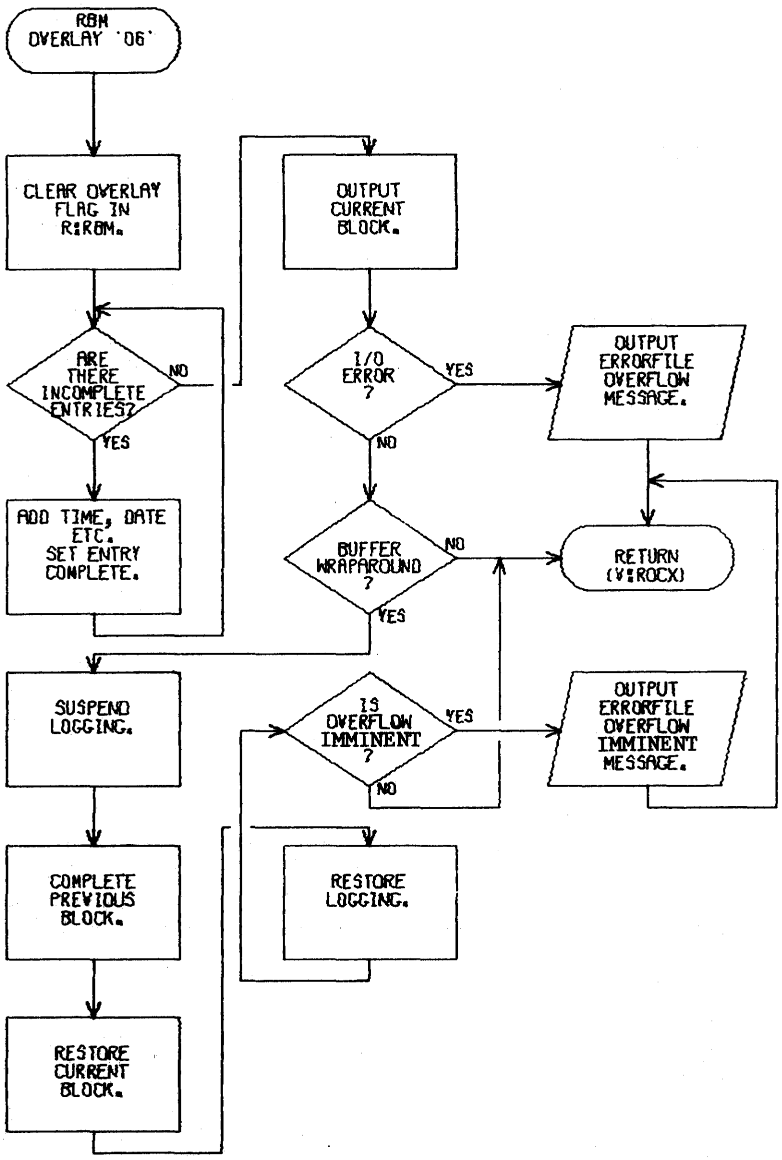


Figure 82. Nonresident Error Log Code (Overlay '06')

System Startup, Code = 18 (Recorded at boot time.)

1	X'18'	Count = 6		
2	0 ——— 0		i	
3	Relative time			
4	Relative time		ii	
5	Year — 1900			Binary
6	Julian day		iii	Binary
7				
8			iv	
9				
10			v	
11				
12			vi	
13				
14			vii	
15				
	0 7	8 15		

Figure 83. System-Startup Entry

Power On, Code = 20 (Recorded at power-on time.)

1	X'20'	4	
2	K:TCB @ Power failure		i
3	Relative time		Time of Power Off
4	Relative time		ii
5			
6			iii
7			
8			iv
9			
10			v
11			
12			vi
13			
14			vii
15			
	0 7	8 15	

Figure 84. Power-On Entry

Time and Date (Time Stamp), Code = 23 (Recorded at startup, at half-day, and whenever a D or T key-in is entered.)

1	X'23'	Count = 6		
2	▨	▨	i	
3	Relative time			
4	Relative time		ii	
5	Year — 1900			Binary
6	Julian day		iii	Binary
7				
8			iv	
9				
10			v	
11				
12			vi	
13				
14			vii	
15				
	0 7	8 15		

Figure 85. Time-and-Date Entry

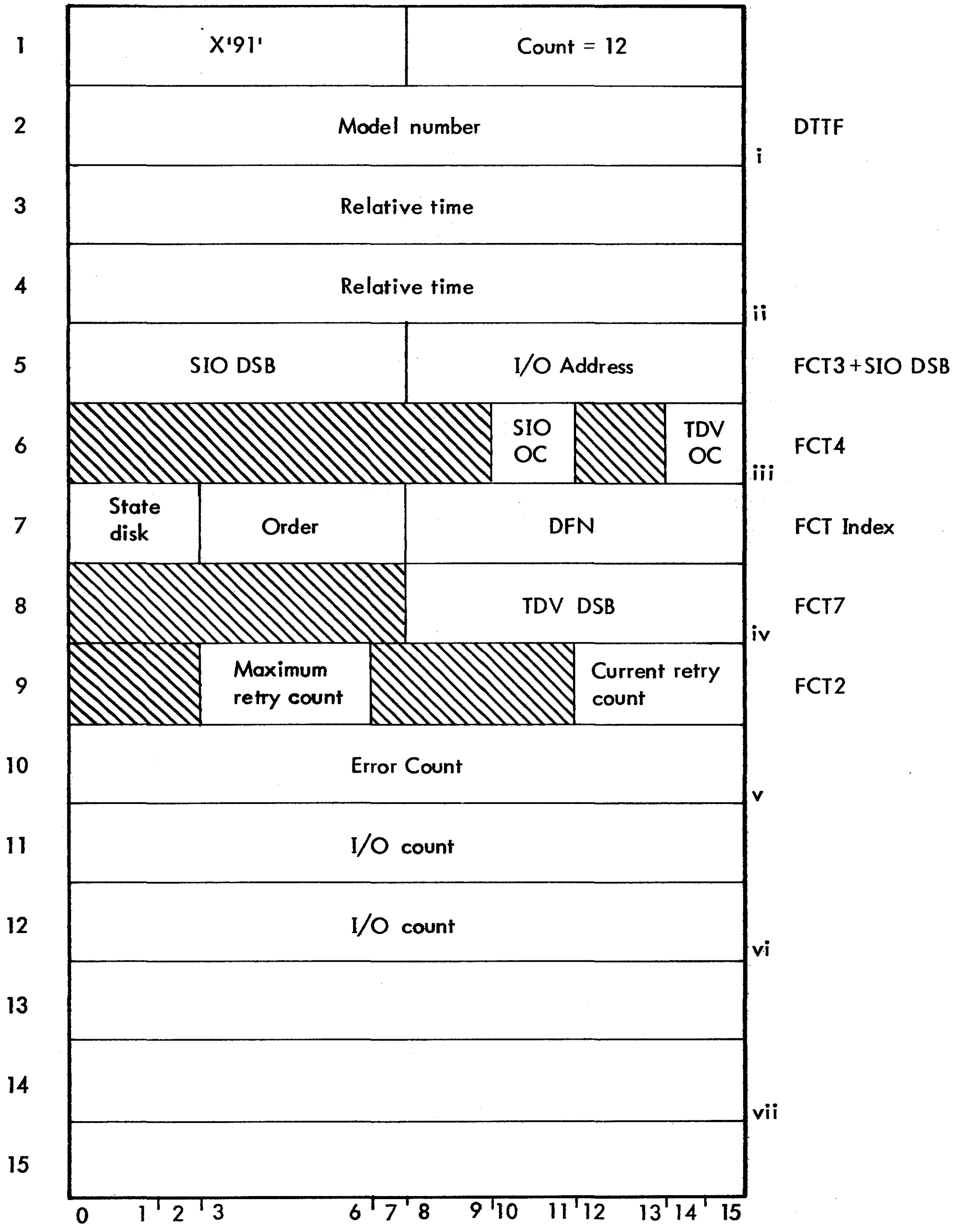
Operator Message, Code = 27

1	X'27'	Count = 15		
2	K:TCB		i	
3	Relative time			
4	Relative time		ii	
5				
6			iii	
7				
8			iv	EBCDIC message, 20 byte maximum — unused words will contain blanks.
9				
10			v	
11				
12			vi	
13				
14			vii	
15				
	0	7 8	15	

Note: An L key-in will allow the operator to enter an 18-byte message into the Error Log. Excess bytes will be truncated. Operator messages may also be recorded by user programs via M:DOW.

Figure 86. Operator-Message Entry

SIO Rejection, Code = 91



1	X'91'			Count = 12		
2	Model number				i	DTTF
3	Relative time					
4	Relative time				ii	
5	SIO DSB			I/O Address		FCT3+SIO DSB
6	▨	SIO OC	▨	TDV OC	iii	FCT4
7	State disk	Order		DFN		FCT Index
8	▨			TDV DSB	iv	FCT7
9	▨	Maximum retry count	▨	Current retry count		FCT2
10	Error Count				v	
11	I/O count					
12	I/O count				vi	
13						
14					vii	
15						

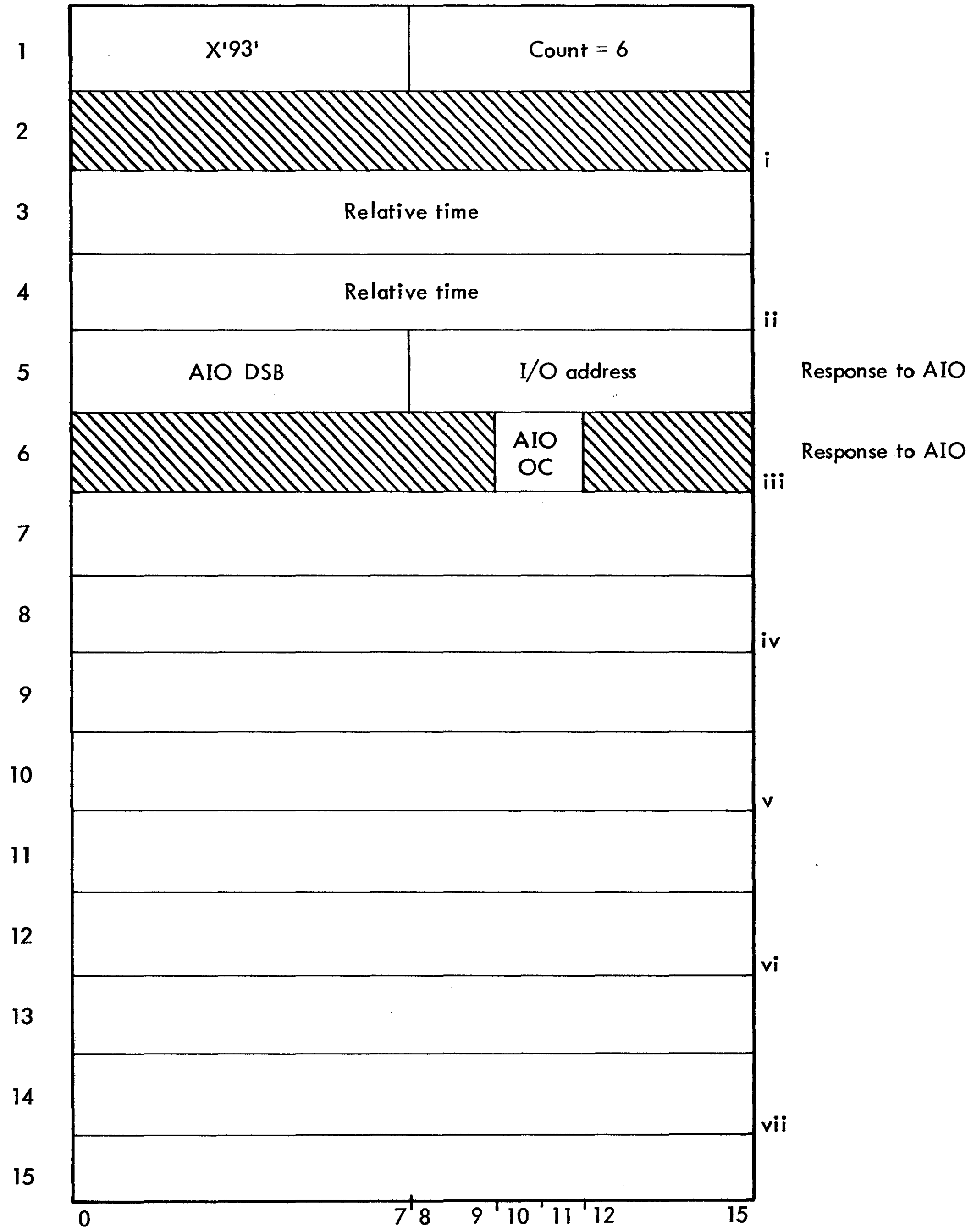
Figure 87. SIO-Rejection Entry

Device Timeout, Code = 92

<table>
<tr><td>1</td><td colspan="8">X'92'</td><td colspan="8">Count = 14</td><td></td><td></td></tr>
<tr><td>2</td><td colspan="16">Model number</td><td>i</td><td>DTTF</td></tr>
<tr><td>3</td><td colspan="16">Relative time</td><td></td><td></td></tr>
<tr><td>4</td><td colspan="16">Relative time</td><td>ii</td><td></td></tr>
<tr><td>5</td><td colspan="8">HIO DSB</td><td colspan="8">I/O address</td><td></td><td>FCT3</td></tr>
<tr><td>6</td><td colspan="8">OSB</td><td>C</td><td>▨</td><td colspan="2">HIO OC</td><td colspan="2">TIO OC</td><td colspan="2">TDV OC</td><td>iii</td><td>FCT4</td></tr>
<tr><td>7</td><td colspan="3">State (disk)</td><td colspan="5">Order</td><td colspan="8">DFN</td><td></td><td>FCT Index</td></tr>
<tr><td>8</td><td colspan="8">TIO DSB</td><td colspan="8">TDV DSB</td><td>iv</td><td>FCT7</td></tr>
<tr><td>9</td><td colspan="3">▨</td><td colspan="4">Maximum retry count</td><td colspan="5">▨</td><td colspan="4">Current retry count</td><td></td><td>FCT2</td></tr>
<tr><td>10</td><td colspan="16">Error count</td><td>v</td><td></td></tr>
<tr><td>11</td><td colspan="16">I/O count</td><td></td><td></td></tr>
<tr><td>12</td><td colspan="16">I/O count</td><td>vi</td><td></td></tr>
<tr><td>13</td><td colspan="16">PRAD</td><td></td><td>(FCT6) +5</td></tr>
<tr><td>14</td><td>E</td><td>D
C</td><td>I</td><td colspan="13">Byte count residue</td><td>vii</td><td>FCT5</td></tr>
<tr><td>15</td><td colspan="16"></td><td></td><td></td></tr>
<tr><td></td><td>0</td><td>1</td><td>2</td><td colspan="3">3</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td></td><td></td></tr>
</table>

Figure 88. Device-Timeout Entry

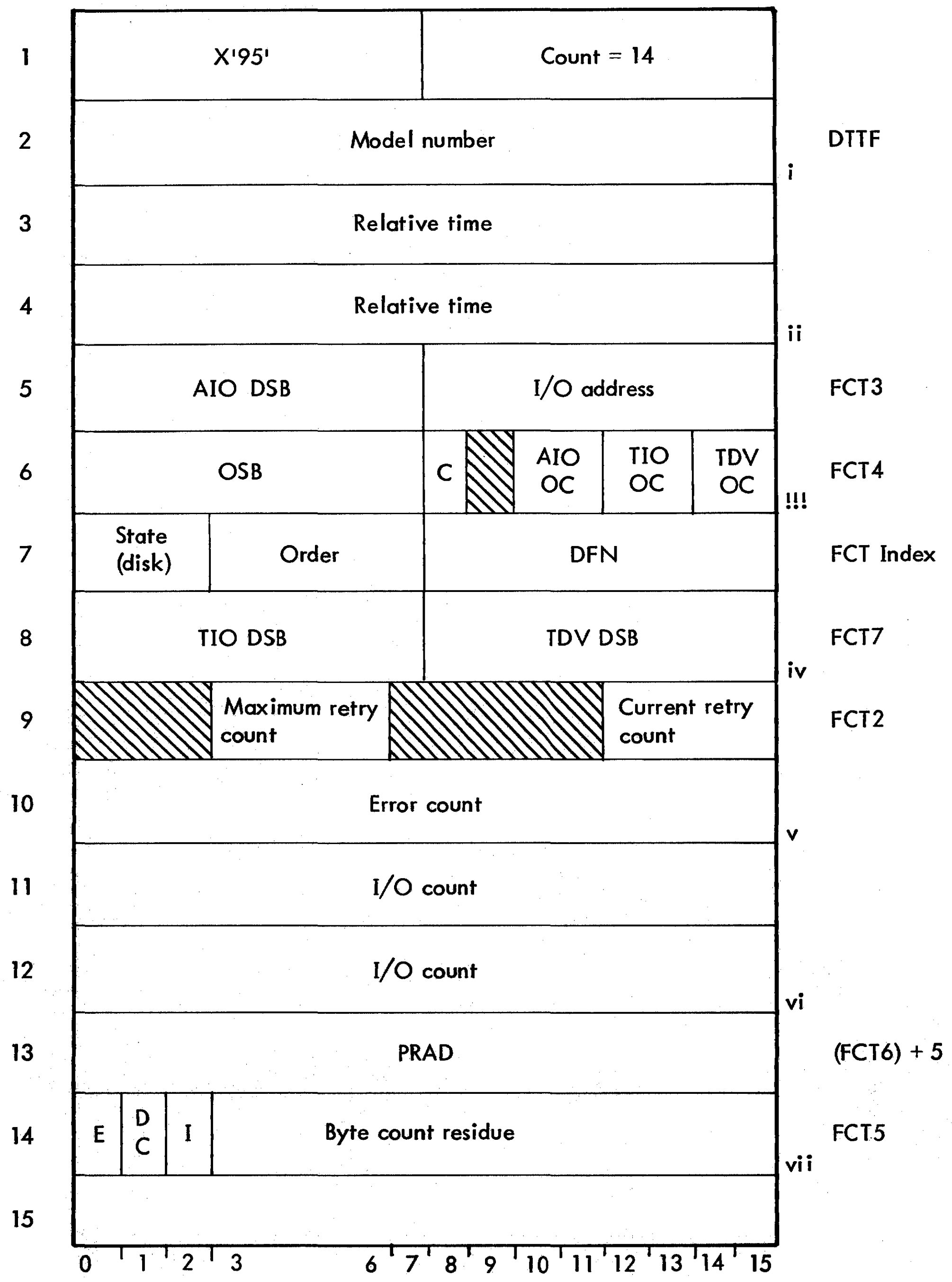
Spurious I/O Interrupt, Code = 93



Recorded by the I/O interrupt task whenever an AIO interrupt is received from a device which RBM believes to be inactive or nonexistent.

Figure 89. Spurious-I/O-Interrupt Entry

I/O Error, Code = 95



1	X'95'	Count = 14		
2	Model number		i	DTTF
3	Relative time			
4	Relative time		ii	
5	AIO DSB	I/O address		FCT3
6	OSB	C ▨ AIO OC TIO OC TDV OC	!!!	FCT4
7	State (disk) Order	DFN		FCT Index
8	TIO DSB	TDV DSB	iv	FCT7
9	▨ Maximum retry count	▨ Current retry count		FCT2
10	Error count		v	
11	I/O count			
12	I/O count		vi	
13	PRAD			(FCT6) + 5
14	E D C I Byte count residue		vii	FCT5
15				

Figure 90. I/O-Error Entry

Lost Entries, Code = 9E

1	X'9E'	Count = 8	
2	Count of lost entries		i
3	Relative time of last lost entry		
4			ii
5	Relative time of first lost entry		
6			iii
7	Last entry type lost		
8	First entry type lost		iv
9			
10			v
11			
12			vi
13			
14			vii
15			
	0 7	8 15	

Recorded when buffering constraints make error-log recording temporarily impossible.

Figure 91. Lost-Entries Entry

Configuration, Code = A1

	0 7	8 15		
1	X'A1'	Count = 7		
2	▨	▨	i	
3	Relative time			
4	Relative time		ii	
5	▨	I/O address		
6	▨	DTT_x	iii	FCT1
7	Model number			
8			iv	
9				
10			v	
11				
12			vi	
13				
14			vii	
15				

Note: If CLOCK1 is dedicated for accounting at SYSGEN time, one configuration record with model number = 8111 will be added. (DTT_x and I/O address = 0 for 8111).

Recorded as part of the ERRFILE initialization sequence by the !PURGE, EL, R command.

DTTx and Model Number = 0 for devices declared for M:IOEX usage only.

Figure 92. Configuration Entry

System Identification, Code = A2

1	X'A2'	Count = 6	
2	Number of 8K-word blocks	Relative time resolution = 1 (2 ms)	i
3	Relative time		
4	Relative time		ii
5	K:VRSION		
6	K:CPU		iii
7			
8			iv
9			
10			v
11			
12			vi
13			
14			vii
15			
	0 7	8 15	

Recorded as part of the ERRFILE initialization sequence by the !PURGE, EL, R command.

Figure 93. System-Identification Entry

Machine Fault, Code = B1

	0–7	8–15	
1	X'B1'	Count = 7	
2	K:TCB	K:TCB	i
3	Relative time	Relative time	
4	Relative time	Relative time	ii
5	PSD1	PSD1	
6	PSD2	PSD2	iii
7	Fault register	Fault register	
8			iv
9			
10			v
11			
12			vi
13			
14			vii
15			

For Sigma 2/3, fault register will be 8102 for memory parity error.

For Sigma 3, fault register will be 2020 for any IOP timeout.

For Sigma 3, fault register will be 2010 for watchdog timeout (incorrect DIO address).

For Sigma 3, fault register will be 2030 for watchdog and IOP timeout.

Figure 94. Machine-Fault Entry

Glossary for Error-Log Formats

A glossary of terms pertinent to the error-log entry descriptions follows.

AIO DSB an 8-bit value representing the device status byte as returned by the hardware in response to an AIO instruction. Device specific, see the appropriate device reference manual.

AIO O and C a 2-bit value representing the overflow and carry (in that order) as returned by the hardware in response to an AIO instruction. Device specific, see the appropriate device reference manual.

byte count residue a 14-bit value representing the number of bytes not transferred in the I/O operation. This value is established by the I/O processor and is available in the odd channel.

C a 1-bit value which indicates whether bits 5-15 of the even I/O channel register were all zeros at time of AIO (C = 0 if yes; C = 1 if bits 8-15 were non-zero).

code an 8-bit value in the first byte of the error log message indicating message type. A value of zero (0) indicates a null entry.

count an 8-bit value representing the number of useful 16-bit words contained in the error-log message. Includes the first word in the count.

current retry count a 4-bit value representing the retry attempt at which either the operation was successful or a value equal to maximum retry count when all allowable retries have been exhausted. Thus, the range of current retry count is 1 through maximum retry count. When current retry count exceeds the maximum retry count, an unrecoverable device error has occurred.

DC a 1-bit value indicating whether data chaining (1 = yes) was specified for the I/O operation. This is obtained by the I/O interrupt task from bit 1 of the odd channel register.

DFN an 8-bit value representing the device-file number which is used as a file control table index. The value may be utilized in many cases to determine the task involved in a device-error condition. Ambiguity results when there is multi-task usage of the same DFN.

DTT_x an 8-bit value representing the device type table index. There is one entry in the table for each unique physical device type in the configuration.

E a 1-bit value indicating a memory fault or data parity error during an I/O operation. This is obtained by the I/O interrupt task from bit 0 of the odd channel register.

HIO DSB an 8-bit value representing the device status bytes as returned by the hardware in response to an HIO instruction. Device specific, see the appropriate device reference manual.

HIO O and C a 2-bit value representing the overflow and carry (in that order) as returned by the hardware in response to an HIO instruction. Device specific, see the appropriate device reference manual.

I a 1-bit value which indicates whether the IOP was requested to interrupt (1=yes) at the completion of the I/O. This is obtained by the I/O interrupt task from bit 2 of the odd channel register.

I/O address an 8-bit value representing the physical I/O address. (E.g., X'92' represents multiunit device 2 on device-controller 1 on I/O channel 1; X'C' if EIOP or Xerox 530 IOP-2.)

I/O count a 32-bit value which records channel activity. (For F00, the 32-bit value obtained from the channel-activity counts.)

Julian day a 16-bit binary value representing the Julian day of year. (E.g., March 1, 1976 would be represented as X'3D'.)

K:TCB a 16-bit value indicating the address of the Task Control Block associated with the task possibly affected by the fault condition.

K:CPU a 16-bit value which indicates CPU hardware options, as follows:

Bit 0 set indicates normalized-shift present.

Bit 1 set indicates extended-arithmetic present.

Bit 2 set indicates MUL/DIV hardware present.

Bit 3 set indicates floating-point hardware present.

Bit 4 set indicates field-addressing hardware present.

Bits 5-10 are unused.

Bits 11-15 = 00010 (2) if Sigma 2.
= 00011 (3) if Sigma 3.
= 11110 (30) if 530.

K:VRSION a 2-byte EBCDIC value, assigned at SYSGEN time, that identifies the system version.

maximum retry count a 4-bit value representing the maximum retry count after which a device error is returned to the requester. When current retry count exceeded the maximum retry count, an unrecoverable device error has occurred.

model number a 16-bit number which uniquely identifies peripheral devices.

order a 5-bit value representing the actual device order which resulted in the device error. Device specific, see the appropriate device reference manual.

PRAD a 16-bit value representing the absolute sector number at which the latest disk transfer began. The range is 0 through n-1 where n represents the number of physical sectors on the device. PRAD is meaningless for other than RAD or disk-pack operations.

relative time a 32-bit value representing milliseconds since midnight. Resolution is 2ms.

SIO DSB an 8-bit value representing the device status byte as returned by the hardware in response to an SIO instruction. Device specific, see the appropriate device reference manual.

SIO O and C a 2-bit value representing the overflow and carry (in that order) as returned by the hardware in response to an SIO instruction. Device specific, see the appropriate device reference manual.

STATE a 3-bit value for current disk state, as follows:

0 = seek to read flawed header.

2 = read flawed header.

4 = seek for requested operation.

5 = restore.

6 = perform requested operation; 'order' specifies read or write.

7 = header read following restore.

TDV DSB an 8-bit value representing the device status byte as returned by the hardware in response to a TDV instruction.

TDV O and C a 2-bit value representing the overflow and carry (in that order) as returned by the hardware in response to a TDV instruction. Device specific, see the appropriate device reference manual.

TIO DSB an 8-bit value representing the device status byte as returned by the hardware in response to a TIO instruction. Device specific, see the appropriate device reference manual.

TIO O and C a 2-bit value representing the overflow and carry (in that order) as returned by the hardware in response to a TIO instruction. Device specific, see the appropriate device reference manual.

year a 16-bit binary value representing current year minus 1900, e.g., 1973 expressed as X'49'.

Device Isolation

Device Key-in Implementation

Three key-ins are provided for device isolation. They are: DU (Device Unavailable), DA (Device Available), and DS (Device Substitution).

The DU key-in is used to make nonrotating memory devices unavailable for all but special M:IOEX, M:CTRL, M:READ, and M:WRITE operations. When this key-in is input, KEYIN will obtain the 2-digit hexadecimal device address included as a parameter of the key-in. First, a check will be made to determine if the input address matches the address of the operator's console (contained in FCT3(1)). If there is a comparison, KEYIN will output the message !!KEY ERR and KEYIN will be reentered. If there is no address comparison, all non-disk File Control Tables will be tested for device address comparison with the input address. If a match is found, bit 7 of FCT2(DFN) will be set to a one to indicate that the DFN is unavailable and the search will continue. If a DFN is marked down, a flag will be set. When the last DFN has been checked, this flag will be tested and, if found reset, a !!KEY ERR message will be output and KEYIN will be reentered. If the flag was found to be set, KEYIN will exit.

The DA key-in is used to make a previously-unavailable device available for normal usage again. The same actions will be taken as for the DU key-in described above except that the tests for operator console will be bypassed and the device-unavailiaility bits will be reset. A !!KEY ERR message will only be output if there is no address comparison.

Key errors will be generated for both DA and DU key-ins if no address is specified, if the argument contains other than two characters, or if a nonhexadecimal value is input.

The DS key-in is used to substitute one device address for another in one or more DFNs. KEYIN first checks for the presence of valid hexadecimal numbers in argument fields one and two of the key-in (old and new device addresses respectively), converts the fields to binary, and stores them into the stack. If a third argument is present, it also will be converted to binary, checked for validity, and stored in the stack; if not present, a zero will be stored. If a syntax error is detected, or the third argument contains a value higher than the number of DFNs or references a disk DFN, a key error will be generated. If the third argument is present, that DFN will be checked to determine if the address contained in FCT3 matches the address contained in argument field one of the key-in. If there is no match, a key error will be generated; if the addresses do compare, the address contained in the second argument field will be stored in FCT3(DFN) and KEYIN will exit. If there is no third argument in the key-in, all non-disk DFNs will be checked for an address comparison with argument field one. If found, the address contained in the second argument field will be stored into FCT3, the message "CHANGED, DFN xx" will be output to the operator's console, and the search will continue. If no address comparison is found, a key error will be generated. If the file is found active, however, the message "UNCHANGED, DFN xx" will be output and no action will be taken. This is necessary to prevent deadlock conditions and the redefinition of device addresses during intermediate I/O operations. Prior to changing the addresses in the FCT3 entries, a check will be made to determine if the address is known to the system. If it is a known disk address, a key error will be generated. If the address is known, a check for device-type comparison will be made. If this test fails, a key error will be indicated. The unavailability bit will be set according to that of another file referencing the same device, if the address is found, or will be unconditionally reset if no DFNs reference the new device address.

The ID of the nonoptional overlay that processes the DA, DS, and DU key-ins is '76'. KEYIN, part 1 (overlay ID07) recognizes these three key-ins and calls overlay ID76 to handle the processing.

Tests for "Down" Devices

Device-Unavailable status is maintained in the File Control Table for non-disk devices. Bit 7 of FCT2 is used to indicate the availability of a non-disk device associated with a DFN. If this bit is set, the device will be unavailable for normal M:READ, M:WRITE, M:CTRL and M:IOEX operations. If bit 7 is reset, access will be permitted. These service routines will test this bit prior to attempting I/O operations in conjunction with bit 7 of word 0 of the user's argument list. If argument list (0) bit 7 is set, device access will be permitted only if the device is unavailable. If the device is unavailable and argument list (0) bit 7 is reset, or if bit 7 is set and the device is available, device-unavailable status will be returned.

Special Receiver Group

Several special purpose receivers allow user access to additional RBM services. These are provided for the use of Xerox application programs and are not intended for general use. The documentation is included here for completeness and should RBM users wish to take advantage of these facilities, they must be aware that these services are subject to change as future requirements dictate.

The following receivers have been defined:

	Name	Absolute Location
Global AIO Receiver	GAIORXR	x'1B8'
Dismissal Receiver	DRXR	x'1B9'
M:TERM Receiver	TERMRXR	x'1BA'
Q:ROC Receiver	QRXR	x'1BB'
Keyin Receiver	KEYRXR	x'1BD'
M:ABORT Receiver	ABORTRXR	x'1AF'
JOB/FIN Receiver	JOBRXR	x'1B0'

All receivers connect by first saving the current contents of the receiver location at their entry address -1 and then storing their entry address into the receiver location.

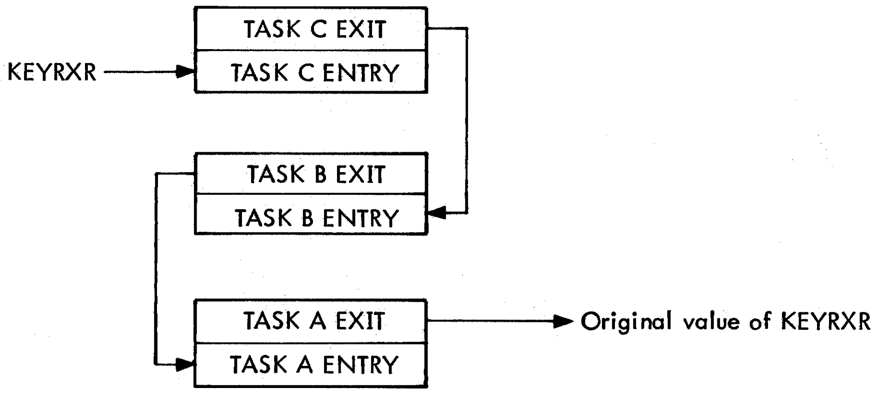
The delinking process requires a search of the receiver chain for the position within the chain of the delinking task and a substitution of the delinking's task exit address for that position within the chain.

It should be noted that interrupts should be inhibited whenever the chain is manipulated. The following code might be utilized to connect and to delink from the chain.

To connect:

```
INHIBIT     R:PSW1

LDA         xxxRXR

STA         MYENTRY-1

LDA         =MYENTRY

STA         xxxRXR

RESTORE     R:PSW1
```

Assuming tasks A, B, and C had connected in that order to the keyin receiver, the keyin receiver chain would be as follows:



To disconnect:

```
            INHIBIT     R:PSW1
            LDX         =KEYRXR
SEARCH      LDA         0,1
            CP          =MYENTRY
            BNC         $+2
            B           ITSME
            RCPY        A,X
            RADD        *Z,X
            B           SEARCH
ITSME       LDA         MYENTRY-1
            STA         0,1
            RESTORE     R:PSW1
```

Global AIO Receiver

Location GAIORXP (ref: S24RBM) is a pointer to the global AIO receiver. Just prior to transferring control to conventional user AIO receivers, RBM will route control through the global AIO receiver chain with the A register containing AIO status as received from the device and X containing the RBM channel status table index.

The global AIO receiver must always restore the contents of A and X and return by a B *ENTRY-1.

Dismissal Receiver

The dismissal receiver is entered at a dismissal opportunity for either primary or secondary foreground tasks. The receiver then dictates whether dismissal may occur or, in the case of a software scheduler, may defer service to another secondary task.

Upon entry: Interrupts are inhibited, with status in R:PSW1.

B is a pointer to the M:READ/M:WRITE temp stack.

L is a pointer to the No-Dismiss return.

Upon exit: Register B and register L must be preserved.

If the dismissal receiver opts for normal dismissal, it may branch directly to M:EXIT. This, however, may only be done for primary tasks; secondary software scheduled tasks cannot undergo normal dismissal. For these tasks, return must be eventually be made to the 'L' register after the interrupt status has been restored from R:PSW1.

M:TERM Receiver

The M:TERM receiver is entered upon termination of any background, primary, foreground, or secondary foreground task. All registers are volatile. The M:TERM receiver must be reentrant.

Q:ROC Receiver

The Q:ROC receiver will give notification when an RBM overlay request has been made and when the RBM overlay area is again free.

Upon entry: 'E' – Pre/Post Flag

$E \geq 0$ means overlay requested.
$E < 0$ means overlay area free.

'B' – Q:ROC temp pointers.

Upon exit: The E register and B register must be preserved.

The Q:ROC receiver must be reentrant.

Note: There will not necessarily be a one to one correspondence between 'E' negative and 'E' non-negative entries to this receiver. The receiver will not be entered for overlays declared as resident at SYSGEN time, but the receiver may be entered ($E \geq 0$) even if no I/O is required for requested overlays already residing in the overlay area.

Keyin Receiver

A keyin receiver pointer is contained at location KEYRXR (ref: S24RBM). After RBM examines a keyin and determines that it is not an RBM keyin, control will be routed through user-connected keyin receivers. In addition, a foreground task may initiate a command to be processed by the keyin receiver chain. The calling sequence is:

L register = return path.

(L) = command not recognized.
(L) + 1 = command recognized.

X register is a pointer to ARGLST as follows:

ARGLST Word 0 = Word address of buffer containing command in TEXT format.

Word 1 = Byte Count (always K:KEYBUF*2 for RBM keyin subtask).

Word 2 = Deferred Status Reply Address (0 if no reply is desired; always 0 for keyin subtask).

If return is to (L) + 1, the 'A' register indicates return status as follows:

A = 0 = Command acknowledged (however, processing may be deferred).

A < 0 = Command recognized but cannot be accepted now.

A > 0 = Command recognized but byte count illegal, obvious syntax violation, or immediate processing has detected an error. Error messages if any, must be output by the processing task.

Details:

Location 'KEYRXR' is initialized by SYSGEN to a pointer to a RCPY L, P instruction. A foreground task connects to the keyin receiver chain by first saving the current contents of location KEYRXR at its entry address -1 and then storing its entry address at KEYRXR. This will serve as its exit address and provides a procedure for delinking. A task may pass a command through the keyin receiver chain (i.e., RBM keyin subtask) by first pointing X to the ARGLST. An RCPYI P, L followed by a branch to the contents of location KEYRXR will cause the request to be examined by the reentrant keyin receiver chain.

If a receiver acknowledges the request, it will typically move the command to its own buffer and save the other ARGLST information. The task to accomplish the actual processing is then triggered; the A register is set to zero and return is made to (L) + 1 to inform the initiator that the command has been accepted for processing. Immediate or deferred processing will report a reply if requested in the ARGLST.

If a receiver recognizes the command but cannot accept it now because of processing constraints, A is set to negative and a return is made to (L) + 1.

If a receiver recognizes the command but initial (or complete) processing causes the command to be rejected, A is set to a positive value and a return is made to (L) + 1.

If a receiver does not recognize the command, control is transferred to the next receiver (B *ENTRY-1) with the X register and L register unchanged.

In particular, the RBM keyin subtask will react in the following manner:

1. Return to (L + 1)

A ≥ 0	Command accepted, exit keyin subtask.
A < 0	Output !!BUSY message on dfn 1.

2. Return to (L); output !!KEYERR message.

M:ABORT Receiver

The M:ABORT receiver is entered at an abort of a background, primary foreground, or secondary foreground task.

Upon entry: 'L' – Abort Location.
'X' – Abort Code.

Upon exit: L and X contain abort location and code (may be modified).

The M:ABORT receiver must be reentrant.

JOB/FIN Receiver

The JOB/FIN receiver is entered at two points, either before !JOB command processing takes place ('E' register greater than -1) or after !FIN command processing ('E' register less than zero) but before WAIT. For the second case (i.e., !FIN), a return of (L)+1 will cause Z:JSAVCC to be set in R:JCP. This will retain the current assignment of 'CC' for one !JOB or !FIN command. A return of (L)+1 for the !JOB command case will have no effect.

File Directory Receiver

DBUF

With the intent to provide support for in-core directory to minimize disk accesses, a Directory Buffer Receiver was added to F01. M:ASSIGN and M:CLOSE will link to the address in DBUF (location X'1BC') with the following register significance:

L = return address

X = -1	Request has been made for assign to a file.
X = -2	An assign-to-file request has been satisfied.
X = -3	M:CLOSE is about to update the associated file directory to record the new EOF pointer, please update the in-core directory.

The DBUF receiver will have the M:ASSIGN (overlay 'B1') and M:CLOSE temp stacks at its disposal. In all cases M:ASSIGN and M:CLOSE will proceed as normal if return is made.

9. BASIC SPOOL SYSTEM (BSS)

The Basic Spool System (BSS) is an independent foreground program that copies data from one foreground oplabel to another. The data is diverted to a disk file so that the input process can proceed independently of the output. A typical application for the BSS would be the transfer of data from a fast device to a slow device (i.e., magnetic tape to line printer). If the input oplabel is a "logical device" (available in G00) the BSS can, with certain modifications, serve as a line printer symbiont (see description of the #LPSPOOL assembly option).

Loading of the BSS is accomplished by assigning the CC oplabel to the release media containing the JCL and binary data required for loading BSS. The operator will be queried during the loading process as to the form of the !$TCB and !$ROOT cards, so that the user may specify the priority at which the BSS is to run and the memory location.

Various assembly options control the BSS, as described below. These options must be modified at the source level. This is accomplished by acquiring a source copy of the BSS (available in compressed form on the release tape) and then modifying the appropriate source lines, as given below.

Option	Function
+20, 20 #LPSPOOL EQU YES	This option, when assembled as a YES, will define the following options as shown:

```
#KEYIN      EQU   NO

#OUTOPLB    EQU   'LP'

#INOPLB     EQU   'LD'

#COMPRESS   EQU   YES

#SUPPRESS   EQU   YES
```

With #LPSPOOL on (#LPSPOOL EQU YES), the BSS will become a line printer symbiont. The BSS reads from the 'LD' oplabel and writes to the 'LP' oplabel. All background output otherwise destined for the same line printer as that referenced by the foreground 'LP' oplabel, will be directed to the background equivalent of the foreground 'LD' oplabel.

Other secondary changes will also occur. Primarily, a larger portion of the spooling file will be maintained so that backspacing may allow recovery from paper jams.

Option	Function
NO	This option, when assembled as a NO, will define the following options as shown:

```
#KEYIN      EQU   NO

#OUTOPLB    EQU   '  '

#INOPLB     EQU   '  '

#COMPRESS   EQU   YES

#SUPPRESS   EQU   YES
```

Modification of these options is possible as described below.

Option | Function

+31, 31
#KEYIN EQU YES
(Default is NO)

This option, when assembled as a YES will cause the BSS to determine its input and output operational labels and (optionally) the spooling file name from a key-in of the following format

i kkk αα to ββ[VIA filename[,area]]

where

i is ignored if a one character field (this facilitates use of the Q key-in).

kkk is ignored (again to facilitate use of the Q key-in).

αα is the input oplabel.

ββ is the output oplabel.

filename[,area] specifies the spooling file name. 'area' defauts to 'SD' if not specified. If the filename is not specified, this option will be satisfied by the assembly option #FILNAME, as described below.

The location of the key-in buffer will default to the RBM key-in buffer area. In order to facilitate activation by other foreground tasks, the contents of the DEFed item 'KEYBUF' (which resides in the BSS initialization routine) may be used as a pointer to a foreground mailbox location which in turn points to a foreground buffer containing a BSS specification record of the format just described. Thus the 'KEYBUF' location may be modified by reassembly on a !$MODIFY command to allow foreground tasks to initiate a BSS copy function completely without operator intervention.

NO

NO is the default for #KEYIN. If NO is specified, the default values for the options #INOPLB and #OUTOPLB may be specified (see the #INOPLB and #OUTOPLB assembly options described below).

+32, 32
#OUTOPLB EQU 'ββ'
(Default is zero)

This assembly option specifies the oplabel which the BSS is to use for output. ββ will default to 0 which will cause the BSS to query the operator. If upon accessing ββ, the BSS determines the oplabel to be invalid or assigned to zero, the message "#STOPPED ββ" will be output to the operator console. The operator may then properly assign ββ through use of an FL key-in or !ASSIGN command and enter a "#GO ββ" key-in to restart the BSS.

+33, 33
#INOPLB EQU 'αα'

This assembly option specifies the oplabel which the BSS is to use for input. αα will default to 0 which will cause the BSS to query the operator. If upon accessing αα, BSS determines the oplabel to be invalid or assigned to zero, the message "#STOPPED αα" will be output to the operator console. The operator may then properly assign ββ through use of an FL key-in or !ASSIGN command and enter a #GO ββ" key-in to restart the BSS.

Option		Function
+34,34 #COMPRES EQU (Default is YES)	YES	When #COMPRES is YES, the record will be compressed before it is moved to the spool file. Compression is achieved by replacing two or more consecutive words which are identical with the value of the word and a count of the number of words. If the value of the iterative word is X'4040', the entire field is replaced by only a count of the number of words.
	NO	When #COMPRES is NO, no compression is performed.
+35,35 #SUPRESS EQU (Default is NO)	YES	When #SUPRESS is YES, combinations of blanks and/or zeros are removed from the end of the record, with a subsequent reduction in the record size. Blank/zero suppression occurs before compression if #COMPRES is also specified.
	NO	#SUPRESS should be NO if the record length must be fixed, as for binary cards.
+40,40 #FILNAME TEXT	'YYYYYYYY'	YYYYYYYY specifies the name of the spooling file. If YYYYYYYY contains leading blanks or zeros (as in the default case) a file name of "ββSPOOL" is assumed (ββbeing the output oplabel). If the spooling file YYYYYYYY does not spool, the BSS will attempt to use "ββ SPOOL" in the 'UD' area. If this file does not exist, the BSS will abort with code #F.
+41,41 #AREA EQU (Default is SD)	'ss'	ss specifies the area name, which contains the spool file. If ss is zero or blanks (as in the default case) the area will be assumed to be 'UD'.
+42,42 #EVEN EQU (Default is NO)	YES	If #EVEN is YES, records with an odd byte size will be padded with one byte of zeros.
	NO	If #EVEN is NO, no padding will occur.
+43,43 #BYTES EQU (Default is 134)	X	X indicates the maximum byte size passed to M:READ. The maximum value for X is 2^{10}.
+44,44 #GRACE EQU (Default is 150 but not greater than 20%)	X	X indicates the number of records which are guaranteed for a #BACK key-in.
+45,45 #EOF DATA (Default is 2)	-X	X indicates the number of consecutive EOFs which will terminate the BSS stream.

As soon as the BSS resolves the parameters specified above, the copy will proceed.

Line Printer Symbiont

A copy of the BSS is available on the SYSTEM Release Tape which is suitable for a single device Line Printer Symbiont. In the process of loading this version of BSS, the user will be queried for

1. Interrupt level.

2. Load location.

3. Permanent file name for that copy of BSS ('OV' file).

4. Spool file size. The name of the spool file will default to 'LPSPOOL' on the SD area.

This version of BSS contains special code which will redirect background output through the spooler. However, for this process to be effective, the following conditions must be met.

1. All of the appropriate background oplabels (i.e., 'LO', 'LL', 'DO', etc.) should point at a background DFN which references a Line Printer (same as existing SYSGEN).

2. The foreground 'LP' oplabel must point at the same background Line Printer via a foreground DFN.

3. The foreground 'LD' oplabel must point at a foreground DFN which references a logical device, which, in turn specifies a device address unique from the line printer.

4. The background must also be supplied with a background DFN which references the same device address as the foreground 'LD' oplabel. This DFN need not have an oplabel assigned to it.

This can be accomplished by adding the following SYSGEN considerations:

1. Assuming that a background DFN for the Line Printer already exists in the SYSGEN deck and defines a Line Printer of device address 'dn', model 7445, add under "DEVICE FILE INFO"

7445/dn, F	DFN = x	Foreground Printer
LP/$\alpha\alpha$, B	DFN = y	Background Logical Device
LD/$\alpha\alpha$, F	DFN = z	Foreground Logical Device

   where $\alpha\alpha$ is an otherwise unused device address.

   Note: The 'LP' mnemonic on the background logical device definition is required by FORTRAN and COBOL.

2. Assuming that the appropriate background Line Printer oplabels already exist and point at the background Line Printer DFN (as in the existing SYSGEN deck), add, under "FGD. OP. LBL."

   LD = y

   LP = x

If these requirements are met, the Line Printer Symbiont will take control automatically of the background Line Printer whenever the system is rebooted.

Blocking/Compression Scheme

Each record in the spool file will be accompanied with at least two words of control, as follows:

Word 0	Meaning
X'0000'	End of block.
X'DEOF' X'0000'	End of spool file. The output routine will terminate when it encounters this value.
X'0EOF'	End of file; this value represents a logical end of file and establishes a backup point for use in the #BACK key-in.
X'0EOD'	This value indicates the end of data for the previous record.

where

Bit 0 when on, indicates that this record should be written with a write EBCDIC order byte.

Bits 6-15 indicate the total record size, in bytes. If these bits are all zero, the record size is 2^{10} bytes, which is the maximum record size.

Bits 1-5 must be zero or the BSS will consider itself lost and search the spool file for the next valid word 0.

Word 1

Bit 0 when on, indicates that this record is not compressed or does not cross a block boundary. Bits 1-5 must be zero, or the BSS will consider itself lost and search for another valid word 0. No X'0EOD' value will follow a noncompressed record.

Bit 1 when set, indicates that a noncompressed series follows. The length of the series is given in bits 6-15. Bits 2-5 must be zero or the BSS will consider itself lost, and search for a valid word 0.

When reset, indicates that an iteration follows. The value for this iteration is given in word 2, unless the iteration value is X'4040'. An iteration value of X'4040' is indicated by bit 2 being set. Bits 3-5 must be zero or the BSS will consider itself lost and search for a valid word 0.

APPENDIX A. XEROX 16-BIT STANDARD OBJECT LANGUAGE

Introduction

The Xerox 16-bit standard object language provides a means of expressing the output of a processor in a standard format. All programs and subprograms in this object format can be loaded by the Overlay Loader. The complete standard object language contains 13 load item types.

An object module consists of the ordered set of binary records generated by an assembly or compilation for later loading. The Overlay Loader has the facility to load and link several object modules together to form an executable program.

The Absolute Loader can load a single module (absolute subset) to form an executable program. The following load item types from the standard object language comprise the absolute subset:

1. Record Header
2. Record Padding (type 0, subtype 0)
3. Repeat Load (type 0, subtype 1)
4. Unrelocated Load (type 1)
5. Start Module (type 4)
6. End Module (type 5)
7. Absolute Load Origin (type 7, subtype 1)

All load item types are acceptable input to the Overlay Loader except Absolute Load Origin (type 7, subtype 1).

Description of Object Modules

General Description

An object module consists of a set of binary object records, each containing an integral number of load items after a standard three-word record header (see Figure A-1). Each binary record in the module is a 120-byte record.

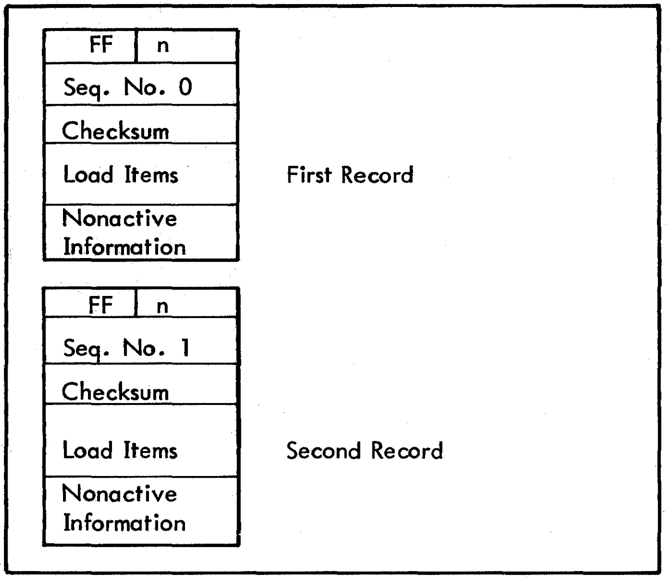


Figure A-1. Typical Object Module of M Records

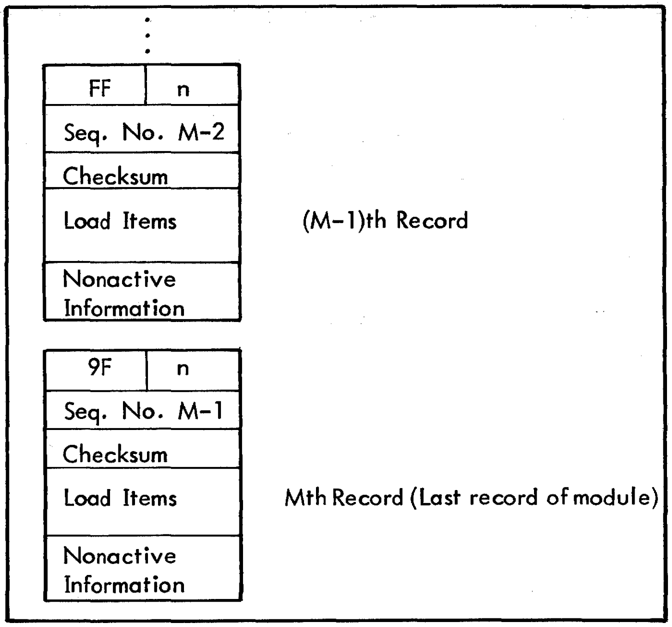
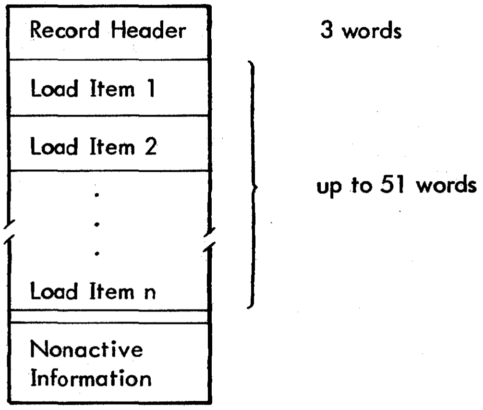


Figure A-1. Typical Object Module of M Records (cont.)

Each load item consists of a header word followed by a variable number of data words. The first load item in an object module is a start-module item and the last item (other than record padding) is an end-module item. There are 15 types of load items, described below.

Binary Object Record Format

Each 120-byte binary record in an object module consists of these parts: Record Header, Load Items, and Nonactive Information in the following arrangement. The Record Header and Load Items are considered the "active" portion of the record.



The "active" portion of the record is that information concerning type, sequence number, checksum and binary data usually processed by loaders. The "nonactive" portion may contain sequence or identification information, or it may be empty. It is not processed by the loaders.

Format of Record Header

The first byte of the record header may be either X'F' or X'9'. X'F' denotes that this is a standard record of the object module: X'9' denotes that this is the last record of the object module.

word 0

Control word		
F or 9	F	0 0 n n n n n n
0 3	4 7	8 9 10 11 12 13 14 15

word 1

S	C	Record sequence no.
0	1	2 15

word 2

Checksum
0 15

nnnnnn in the first word is the number of active words in the record, excluding the record header. "Active" denotes data to be processed by a loader. There may be some padding words or sequence information at the end of the record that is not included in the "active" count. The maximum value of n is 51. Note that although the physical record size is fixed at 120 bytes (80 columns of binary data) the number of active words may vary from 3 to 54. This effectively standardizes the reading of binary object records but allows versatility in the generation of active data. The record sequence number starts at 0 and takes on consecutive integer values for all the records in one file. The S bit is a sequence override. If this is a 1, the loader ignores sequence checking for the record. The checksum is an arithmetic sum, with carry, of the n-3 active words after the record header. If the C bit is a 1, the checksum is ignored.

Load Item Format

Each load item consists of a one-word header and an optional variable-length body of data.

Load Item Header	Load Item
Load Item Data	

Format of Load Item Control (Header) Word

Every header word has the same general format:

bits 0-3 Type

bits 4-7 Subtype or control.

bits 8-15 Number of data words in the load item (excluding item header).

This number plus 1 is equal to the size of the load item. All words of a load item must be contained in the same physical record.

Summary of Load Item Formats

RECORD PADDING (Type 0, Subtype 0)

word 0

Control word			
0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
0 3	4 7	8 11	12 15

There is no body of data. Padding words are ignored by the loader. The object language allows padding as a convenience for processors.

REPEAT LOAD (Type 0, Subtype 1)

word 0

Control word			
0 0 0 0	0 0 0 1	0 0 0 0	0 0 0 1
0 3	4 7	8 11	12 15

word 1

Repeat count
0 15

This item repeats the next load item a specified number of times. The load item (type 1, 2, or 3 only) immediately following the repeat load is repeated (i.e., loaded) in its entirety the number of times indicated by the data word.

UNRELOCATED LOAD (Type 1)

word 0

Control word			
0 0 0 1	0 0 0 0	0 0 n n	n n n n
0 3	4 7	8 11	12 15

word 1

First data word
0 15

⋮

word n

Last data word
0 15

This item loads n words without relocation.

RELOCATED LOAD-MODULE BASE (Type 2)

word 0

Control word			
0 0 1 0	0 0 0 0	0 0 n n	n n n n
0 3	4 7	8 11	12 15

word 1

First data word

0 15

⋮

word n

Last data word

0 15

This item loads n words with module relocation. The relocation bias of the current object module is added to each data word in the item.

RELOCATED LOAD-COMMON BASE (Type 3)

word 0

Control word															
0	0	1	1	0	0	0	0	0	0	n	n	n	n	n	n
0			3	4			7	8			11	12			15

word 1

First data word

0 15

⋮

word n

Last data word

0 15

This item loads n words with a common base relocation.

START MODULE (Type 4)

word 0

Control word								
0	1	0	0	0	0	0	0	n + 1
0			3	4			7	8 15

word 1

Common size allocation

0 15

word 2

First character	Second character
0 7	8 15

⋮

word n + 1

(2n-1)th character	Last character (or blank)
0 7	8 15

This item identifies the start of the object module. The characters in words 2 through n + 1 are the program name (identification) for the module.

END MODULE (Type 5)

word 0

Control word															
0	1	0	1	0	0	0	r	0	0	0	0	0	0	1	1
0			3	4			7	8			11	12			15

word 1

Starting address

0 15

word 2

Severity level

0 15

word 3

Relocatable size (or zero)

0 15

This item identifies the end of the object module. In the control word (word 0), the starting address is defined in bit 7

where

r = 1 indicates absolute starting address.
r = 0 indicates relocatable starting address.

The severity level in word 2 is defined as the highest level reached during processing.

The loader uses the relocatable module size to determine the starting location for the next relocatable section.

A starting address of absolute 0 indicates there is no starting address for this module.

LOAD ORIGIN (Type 7)

word 0

Control word															
0	1	1	1	0	0	0	r	0	0	0	0	0	0	0	1
0			3	4			7	8			11	12			15

word 1

Origin address

0 15

This item sets the origin within the object module. In the control word (word 0), the origin is defined in bit 7

where

r = 0 indicates relocatable origin.
r = 1 indicates absolute origin.

RELATIVE LOCATION POINTER (Type 8)

word 0

Control word			
1 0 0 0	0 0 0 r	0 0 0 0	0 0 0 1
0 3	4 7	8 11	12 15

word 1

Chain base address
0 15

This item establishes the chain base for later chain resolution. In the control word (word 0), the chain base address is defined in bit 7

where

r = 0 indicates a relocatable address.
r = 1 indicates an absolute address.

NAME DEFINITION (Type 9)

word 0

Control word		
1 0 0 1	0 0 1 0	n + 1
0 3	4 7	8 15

word 1

First data word
0 15

word 2

First character	Second character
0 7	8 15

⋮

word n + 1

(2n-1)th character	Last character (or blank)
0	15

This item identifies a name as a definition within the object module.

All name definitions immediately follow the start-module item and must precede all other load items. For each name definition, an address definition should appear later in the object module.

ADDRESS DEFINITION (Type 9)

word 0

Control word		
1 0 0 1	0 0 0 r	n + 1
0 3	4 7	8 15

word 1

First data word definition - address
0 15

word 2

First character	Second character
0 7	8 15

⋮

word n + 1

(2n-1)th character	Last character or blanks
0 7	8 15

This item associates a location in the module with a definition name (characters in words 2 through n + 1) for other modules to reference. In the control word (word 0), the definition address is defined in bit 7

where

r = 0 indicates relocatable definition address.
r = 1 indicates absolute definition address.

EXTERNAL REFERENCE (Type A)

word 0

Control word		
1 0 1 0	0 0 0 r	n + 1
0 3	4 7	8 15

word 1

Chain address (or zero)
0 15

word 2

First character	Second character
0 7	8 15

⋮

word n + 1

(2n-1)th character	Last character (or blank)
0 7	8 15

This item states a name (characters in words 2 through n + 1), defined in another module, whose definition address must be inserted in a chain of locations within the module. In the control word (word 0), the chain address is defined in bit 7

where

r = 0 indicates a relocatable chain address.
r = 1 indicates an absolute chain address.

Note: If there is no chain address, the reference address is zero and is used for library searching purposes only.

SECONDARY REFERENCE (Type B)

word 0

Control word		
1 0 1 1	0 0 0 r	n + 1
0 3	4 7	8 15

word 1

First data word chain address
0 15

word 2

First character	Second character
0 7	8 15

⋮

word n + 1

(2n-1)th character	Last character (or blank)
0 7	8 15

This item states a name (characters in words 2 through n + 1), defined in another module, whose address may be inserted in a chain of locations within the module. This item is identical to type A, above, except that it does not force loading of the routine from the library. In the control word, the chain address is defined in bit 7

where

r = 0 indicates a relocatable chain address.
r = 1 indicates an absolute chain address.

ADDRESS LITERAL CHAIN RESOLUTION (Type C, subtypes 0, 1, 2, and 3)

word 0

Control word			
1 1 0 0	0 0 q r	0 0 0 0	0 0 1 0
0 3	4 7	8	15

word 1

Resolution address
0 15

word 2

Chain address
0 15

This item defines a location within the module (called the resolution address) whose address must be inserted in a chain of displacement fields within the module. In the control word, the chain address is defined in bit 6

where

q = 0 indicates a relocatable chain address.
q = 1 indicates an absolute chain address.

The resolution address is defined in bit 7

where

r = 0 indicates a relocatable resolution address.
r = 1 indicates an absolute resolution address.

An address literal chain is a threaded list of forward references to a single location in a program. The definition value (called the resolution address) can be output as an address literal chain resolution (Type C, subtypes 0, 1, 2, and 3). The chain address points to the beginning of the threaded list which is terminated by an absolute zero value. The resolution address and the chain address may be absolute or relocatable.

Note: Because the terminator of the chain is zero, no program may have an address literal chain whose last link is at absolute zero (i.e., the item would reference zero and would thus appear to terminate the chain).

Note that external reference (REF) (type A) and secondary reference (SREF) (type B) chains are structured in the same manner, but resolved by the loader using an external definition value (type 9).

DISPLACEMENT CHAIN RESOLUTION (Type C, subtypes 6, 7, A, and B)

word 0

Control word			
1 1 0 0	p p q r	0 0 0 0	0 0 1 0
0 3	4 7	8 9 11	12 15

word 1

Resolution address
0 15

word 2

Chain address
0 15

This item defines a location (called the resolution address) within the module whose relative displacement must be inserted in a chain of displacement fields within the module. In the control word, the displacement chain is defined in bits 4-5

where

pp = 01 indicates that an indirect bit is not set in each instruction in the displacement chain.

pp = 10 indicates that an indirect bit is set in each instruction in the displacement chain.

q = 1 always indicates absolute displacement of the last item in the chain (relative to the chain base declared in item type 8).

The resolution address is defined in bit 7

where

r = 0 indicates a relocatable resolution address.
r = 1 indicates an absolute resolution address.

When forward references occur during one-pass processing, and the possibility of resolving the reference by a definition or literal may occur within 255 locations, the 8-bit displacement field of the instruction may be used to form a displacement chain. The item types 8 (relative location pointer — establish chain-base) and C (displacement-chain resolution) must be used together to resolve the chain by substituting actual displacements determined at load time.

In the creation of a displacement chain, the pointer in the type 8 item defines the relative location in the program to be established as the chain base. Each new type 8 item can define a new chain base. The values in the displacement field of the instructions included in any given displacement chain refer to the absolute displacement of that instruction relative to the currently established chain base; e.g., if the chain base is established to be X'100' and an instruction is located at X'125', the displacement of that instruction for purposes of the displacement chain is X'125'-X'100' or X'25'. This point is emphasized since the loader will use this displacement only to determine the final displacement of the instruction relative to the location of literal or target locations.

When the displacement chain connects instructions that reference a literal or a specific target location within range of the chain base (e.g., LDA=3 LDA=LAB, B XR), no indirect bit is set in each instruction (pp = 01 in Header — Type C).

When the chain connects references to an external symbol or forward reference whose value will be given in some literal within range of the chain base, pp is set to 2 in the type C header, to set the indirect bit in each instruction in the chain (e.g., LDA X, which will be resolved as LDA *$+n, where n is the displacement of ADRL X relative to the instruction).

The chain base address (in the type 8 item) may be declared as an absolute or relocatable value. The resolution address (first data-word of a Type C item) is the address of the target location or literal expressed as a location, and not as a displacement on the chain base. Note that although the resolution address is defined at this point, the value of the literal at that resolution may not be defined until later. In fact, it may be an element of an address-literal chain (type C) or external reference chain (type A). The address-literal or external chain resolution is independent of the displacement chain resolution.

The chain address given in the second data word is the obsolute displacement of the last item in the chain, relative to the chain base declared in type 8 (e.g., if the effective chain base were X'1000' and the value of the chain address were X'20', the last item of the displacement chain would be located at X'1020').

A separate displacement chain will be created for each unique variable in a given displacement region. Thus, many displacement chains may be built using the same chain base. As a matter of fact, the chain base may not be changed until a displacement chain resolution item has been output for each displacement chain. An unresolved displacement chain is a serious error condition in the output, and is unacceptable for execution.

The format of the displacement chain is described in the example in Figure A-2.

Example: Let a chain base be declared at 109(R). (Numbers given are decimal.) It is assumed that the ADRL for XLB will be ultimately loaded at 140(R). Note that the displacement field of each instruction before resolution is a pointer to the location of the next item in the threaded list relative to the chain base.

Relative Location Counter	Symbolic	Displacement From Chain Base	Displacement Field of Instruction Before Loading	Displacement Field of Instruction After Resolution
110	LDA XLB	1	00 (end of chain)	30 (140-110)
125	STA XLB	16	01	15 (140-125)
134	CP XLB	25	16	06 (140-134)
136	STA XLB	27	25	04 (140-136)
140				
	Item Type C, Displacement Chain Resolution			
	Resolution Address 140(R)			
	Chain Address 27(A)			

Figure A-2. Displacement Chain Format

LABELED COMMON (Type D, Subtypes 0, 1, and 2)

word 0

Control word		
1 1 0 1	0 0 k k	n+1
0 3	4 7	8 15

word 1

Labeled COMMON index	
0	15

word 2

Labeled COMMON size, zero, or displacement	
0	15

word 3

Content (first word)	
0	15

⋮

word n+1

Content (last word)	
0	15

Subtype 0 -(k=0)- Labeled COMMON Definition. This subtype conveys the block size in words and an index value for the block being defined. The contents of the load item designate the alphanumeric name for the Labeled COMMON block. The index value is relative only to the module being loaded and is sequenced from the integer one. It is used only to economize on space in the reference and data subtypes.

This subtype will follow the start module and name definition items. It must precede the reference and data subtypes for Labeled COMMON.

Subtype 1 -(k=1)- Labeled COMMON References. This subtype carries as content a set of words that continue the load program and to which a Labeled COMMON base will be added. The particular base address to be added is indicated by the index value in the load item. The word to which the base is added may contain positive or negative content. Should the index value be zero on this subtype, then the blank COMMON will be the added base value.

The third word (word 2) of this item is non-functional and is carried as zero.

Subtype 2 -(k=2)- Labeled COMMON Data. This subtype will load Labeled COMMON with a set of contiguous data. Again the COMMON block is identified by an index value. The starting displacement from its base is identified in the third word (word 2) of the load item.

APPENDIX B. CRITICAL RBM TIMES

Routine	Time (microseconds)
M:SAVE	
Registers Only, No Temp, No Accounting	59
With Accounting, No Temp	99
Without Accounting, No Temp	90
With Accounting and Temp	138
M:EXIT	
Registers Only, No Temp	69
With Temp	116
Maximum Interrupt Inhibit by RBM	100

Multiply Simulation (average)
Minimum = 81
Maximum = 280
	250

Divide Simulation (average)
Minimum = 86
Maximum = 340
	310

Control Panel Interrupt	29
I/O Interrupt	
No Command Chaining	315
Command Chaining Without Receiver	457
Command Chaining With Receiver	
Keyboard/Printer (per character)	415
Card Punch (per row)	315
Disk Pack (no device waiting)	485
Disk Pack (device waiting)	586
Interrupt on Channel Active (Seek Overlap; Set Device Waiting)	min 217 max 557

Note: Figures are given for Sigma 2. For Sigma 3, subtract 15 percent from each figure.

APPENDIX C. MAGNETIC TAPE HANDLING

It is assumed that the reader has a general knowledge of the structure of M:READ/M:WRITE, which is flowcharted in Figure 24 of this manual.

When an RBM user makes a request for magnetic tape I/O through M:READ, M:WRITE, or M:CTRL, several different routines unique to magnetic tape handling may come into play. Which routines are called is a function of the service routine used (M:CTRL or M:READ/M:WRITE); the desired function (write binary, read BCD (7T), etc.); the model of magnetic tape unit being used (9-track or 7-track); and the device status, both before and after the I/O operation. Some of these routines are resident and others are overlays. With the exception of M:CTRL, all are SYSGEN optional and are included only when the system has a requirement for the routines.

Magnetic Tape Command Chaining Receiver (Resident)

The command chaining receiver for magnetic tapes has two purposes; to allow mode control for 33xx magnetic tapes, and to acquire SENSE information from 9-track tape controllers to provide the capability for correctable read error recovery. The sense must be performed at I/O interrupt time to prevent the potential loss of track-in-error information caused by subsequent intermediate controller operations by tasks of higher priority than the one that initiated the current operation. To simplify the code within the routine, a SENSE operation is issued to 9-track tapes only if a transmission error is detected, the E-flag bit is set, if the byte count residue is nonzero, if a timeout occurred or if any of the I/O left the overflow or carry indicators set in FCT4. The mode order is command chained to precede any reads or writes moving the tape from load point (33xx magnetic tapes only).

Resident Magnetic Tape Pre-I/O Edit

The resident tape pre-I/O edit routine is called by M:READ/M:WRITE prior to the issuance of the SIO for binary 7-track and all 9-track tape operations. Its purpose is to check for I/O attempts past end-of-tape, device manual or unrecognized, and build the command chaining necessary in the IOCT. Write EOF and read or write with error recovery suppressed are the operation items permitted beyond end-of-tape. In this way, end-of-volume sentinels can be written or read past the end-of-tape marker. If the I/O operation cannot be performed because of the position of the tape, EOT status will be returned to the user and the SIO will not be attempted.

7-Track BCD Tape Pre-I/O Edit and BCD Conversion Overlay

This overlay is called from M:READ/M:WRITE or the magnetic tape error recovery overlay for the following functions:

1. Pre-I/O edit for BCD 7-track tape operations.

2. Post-I/O edit for BCD 7-track tape operations.

3. Post-read edit for BCD card operations.

4. Error recovery for BCD card reader operations.

Only the code pertaining to BCD 7-track tape operations is discussed below.

For all post-I/O operations, the overlay converts any special BCD characters in the user's buffer to EBCDIC and then exists. If the overlay was called from M:READ/M:WRITE, the return status will be "successful I/O completion". If the BCD conversion overlay was called from the magnetic tape error recovery overlay, the return status will be "incorrect length", which is the only condition for which BCD-EBCDIC translation is performed following an error.

For pre-I/O edit operations, code similar to that of the resident tape pre-I/O edit routine is first executed. The actions taken are the same with two major exceptions:

1. There is no read-backwards order for 7-track tapes, so checks for that condition are not performed.

2. If the operation is to be permitted, special EBCDIC characters in the user's buffer will be converted to BCD. These special characters will be translated back to EBCDIC when the overlay is called to perform post-I/O editing.

For a further discussion of the EBCDIC-BCD translation feature in RBM, see Appendix D.

Magnetic Tape Error Recovery Overlay

The resident magnetic tape error recovery module screens abnormal conditions for simple incorrect length. If any other conditions exist, the appropriate error recovery overlay is called, depending on the device model number.

One of the magnetic tape error recovery overlays will be called from M:READ/M:WRITE if any of its magnetic tape operations result in the detection of an abnormal condition. However, an abnormal condition for tapes may or may not be a "real" error. In addition to real errors, such conditions as end of file, beginning of tape, write protect violations and incorrect record length may be detected. These conditions are reported to the calling program but are not treated as real errors. The error recovery overlay will be called to process abnormal device status even if the calling program does not specify standard error recovery, due to the degree of analysis required to ensure correct status reporting.

If a genuine error occurs, it is either recoverable or irrecoverable. The conditions under which "irrecoverable-I/O" status is returned to the calling program are as follows:

1. Error recovery is not specified on the user call.
2. Indeterminate tape position (i.e., the tape position is lost).
3. Ten recovery attempts were performed without success.
4. An error occurred while repositioning tape prior to a retry attempt.
5. The nature of the error makes recovery impossible.
6. Device and/or channel status are in conflict and it is impossible to determine the exact nature of the problem.

If an error is recoverable, a retry sequence will be initiated. In general, one or more intermediate positioning operations will be attempted (the overlay will not exit while they are in progress). If they are successful, the overlay will exit back to M:READ/M:WRITE with status which indicates that the original operation is to be retried.

For write operations, two recovery sequences are used, based on the current retry count. If the retry count is less than three (i.e., 0, 1, or 2), the sequence is

SET CORRECTION – BACKSPACE – BACKSPACE – READ – SET ERASE – RETRY.

The purpose of this sequence is to ensure that the write attempt did not result in the generation of multiple records due to a bad spot on tape (i.e., generating one record with gaps in its middle). If there is such an error, the second backspace operation will not position the tape at the beginning of the previous record, but instead will stop in the middle of a record. The following read operation will then result in the detection of a transmission error. In this case, a "bad tape" message will be output to the operator's console and the error recovery overlay will exit with "irrecoverable–I/O" status. If the read operation in the above sequence does not result in the detection of a transmission error but the retry attempts continue to fail (due to an inability to erase past the bad spot on tape), a second recovery sequence will be attempted. If the retry count is three or greater, the following sequence will be used:

SET CORRECTION – BACKSPACE – SET ERASE – RETRY.

This sequence will allow the tape to erase approximately 25 inches of tape before the retry count is exhausted. If the operation cannot be performed successfully before the maximum number of retries is reached, the operator will be notified of a tape fault and "irrecoverable–I/O" status will be returned to the calling program.

For read errors, two recovery sequences are also used, depending upon the type of magnetic tape unit and the nature of the error. If the read error is correctable, the following recovery sequence will be used:

BACKSPACE – SET CORRECTION – RETRY.

The SENSE data used for the SET CORRECTION operation is that which was gathered at I/O Interrupt time by the command chaining receiver. If the error is noncorrectable, the following retry sequence will be used:

BACKSPACE – RETRY.

Noise Record Correction

A maximum of 10 recovery attempts will be made before declaring the error irrecoverable. Under certain circumstances, an irrecoverable read error will be ignored. If the retry count becomes exhausted, a transmission error is reported, and there is an incorrect length with the number of bytes actually transmitted numbering seven or fewer, the error will be designated a noise record. In this event, the operator will be notified of a noise record and the next record on tape will be read. (If the user has specified "no error recovery" this sequence is not used.) This does not mean that the user cannot write and read records of fewer than eight bytes, but does mean that if there are irrecoverable errors in short records, the records may be ignored.

If the E-flag (bit 0 of the odd I/O channel register) is set, a memory parity error is indicated. In this case, the error recovery routine will scan the user's buffer and/or IOCT via LDA instructions. If there is a real memory error, the Machine Fault interrupt will be triggered and the task (or job) will be aborted. If the MFI is not triggered, a further analysis will be made to determine if standard recovery techniques may be employed.

M:CTRL Overlay

Status at I/O interrupt time is analyzed to determine which status to return to the calling program. Table C-1 shows the various possible combinations and the status returned.

Table C-1. M:CTRL Magnetic Tape Operations Status Returns

Device Status			
EOF	BOT	UE	Status Returned to Program
N		N	Successful - I/O
N		Y	Irrecoverable - I/O
Y	N		End-of-File
Y	Y		Beginning-of-Tape

There is no attempt at error recovery for M:CTRL operations because of the possibility of incorrect tape position.

Recommended Practices

Several general practices are recommended for programs that support magnetic tape I/O under RBM.

1. Specify standard error recovery on all M:READ and M:WRITE service calls. This permits complete and automatic recovery from errors whenever possible. This technique also prohibits the calling program from writing or reading off the end of the reel.

2. Maintain a pair of indicators that always contain the current file and record numbers. If "irrecoverable-I/O status is returned for a tape operation, there is no guarantee that the tape is positioned exactly where the program assumes it to be. If this status is returned, the recommended procedure is to rewind the tape, position to the end of the last known bad record on tape, and continue from that point.

3. Although the Xerox magnetic tape drives handle a much wider range of record lengths, it is recommended that values in the range of 16-4096 bytes be used, with record lengths of 1K to 2K considered optimal. This permits a moderately high packing density with a relatively low probability of errors.

APPENDIX D. BCD/EBCDIC CODE CONVERSION

Introduction

A feature of the Xerox card equipment and 7-track magnetic tape is hardware conversion of user's BCD inputs to EBCDIC codes for Sigma computer internal use. The outputs are also hardware converted from EBCDIC to BCD. A problem arises with the definition of BCD. The tape drives and card equipment are designed with the commercial (COBOL) character set as the basis for conversion. Most of the Sigma installations operate using the scientific (FORTRAN) set. Therefore, the RBM I/O routines provide pre-I/O and post-I/O software conversion for those characters that present conflicts in the two BCD sets when selected by appropriate users options. Note that BCD cards are produced on an 026 keypunch or equivalent, and EBCDIC cards are produced on an 029 keypunch or equivalent.

SYSGEN Options

RBM performs character conversion when I/O is requested on the following device types:

Device Type Name	Characteristics
B7	7-track magnetic tape with BCD option.
BR4	400 and 1500 cpm card reader.
BP1	100 cpm card punch.
BP3	300 cpm card punch.

Table D-1 contains those character codes that are modified by the RBM I/O editing routines.

Table D-1. Special Character BCD/EBCDIC Conversions

Internal Code (Hex)			
BCD Character	I/O[†]	Program[†]	EBCDIC Character
% or (	6C	4D	(
⌑ or)	4C	5D	)
# or =	7B	7E	=
& or +	50	4E	+
@ or '	7C	7D	'
<	4E	4C	<
>	7E	6E	>
:	7D	7A	:
?	4A	6F	?

[†] The I/O value is the hexadecimal value in memory just after input or just before output. The program value is the actual value used by the user program.

The characters in Table D-1 are modified as follows:

1. If any of the BCD codes are encountered when reading from device type B7 or BR4, they are converted to the corresponding EBCDIC codes by a post-I/O editing routine in RBM (i.e., after the data transfer).

2. If any of the EBCDIC codes are encountered in an output buffer for devices B7, BP1, or BP3 they are converted to the corresponding BCD codes by a pre-I/O editing routine in RBM (i.e., before the data transfer). If the output device type is BP3, the output buffer in RBM is converted and the output buffer in the program is not altered. However, for device types B7 and BP1, the user's buffer is temporarily altered by the pre-I/O edit routine. After output is complete, the characters are reconverted to their original values. If I/O is performed with wait (for completion) the code conversion is not ordinarily apparent to the user.

Programming Considerations

There are two conditions that will cause the user's buffer to temporarily contain erroneous data.

1. If output is to device type B7 or BP1 and the argument list specifies "no wait", the user must not initiate another output operation from the same buffer until a "check" operation is performed after the first operation is complete. When using UTILITY COPY, the user must not specify more than one device of type B7 or BP1 in a list of operational labels for output. Device type B7 or BP1 may be included in an operational label list with UTILITY COPY provided that it is the last label in the list. For example, if operational label BO is assigned to device type BP1 and operational label RD is assigned to a RAD file, the following UTILITY control command must be used:

```
!*OPLBS    RD,BO
```

However, the following command will cause incorrect data to be written to the RAD file:

```
!*OPLBS    BO,RD
```

If BO is assigned to device type BP1 and MT is assigned to device type B7, it is improper to copy to both devices at once, and the following control command must not be used:

```
!*OPLBS    BO,MT
```

The reason for these restrictions is that UTILITY COPY performs I/O without wait to several devices concurrently (if several devices are specified).

2. If output is to device type B7, BP1, or BP3 and the data is to be later input using device type B7 or BR4, the output buffer must not contain any EBCDIC character codes that do not have corresponding scientific BCD character codes. For example, if an output buffer contains the EBCDIC character code "&" (X'50'), this character will be output to tape as an "&" in octal code. When input, the "&" is converted by the hardware to an X'50' but the BCD post-I/O editing routine will convert this code to X'4E' (+). Therefore, the programmer must be extremely careful when outputting any of the following EBCDIC characters to device types B7, BP1, or BP3 (in BCD mode):

Initial EBCDIC Character (Memory)	Initial EBCDIC Code (Hex)	Initial BCD Character	Converted Value After Reading	
			Code (Hex)	EBCDIC Character
%	6C	(or %	4D	(
#	7B	#	7E	=
&	50	& or +	4E	+
@	7C	@ or '	7D	'
¢	4A	¢	6F	?

Other Considerations

All use of standard RBM operations to 7-track tape requires the packed binary option. This is also true of the LOAD procedure initiated by the processor control panel. The BCD option can be used only for user data in the proper BCD subset.

The unpacked binary feature is only available using M:IOEX.

APPENDIX E. ERROR SUMMARY ACCOUNTING

Optional assembly code is provided to keep track of the total number of M:READ and M:WRITE operations on each I/O channel and the number of errors (including retry attempts). These counters provide the operator or Field Engineer with a means of measuring the reliability of the peripheral device (s) on each channel.

To avoid penalizing installations that do not desire this feature, the code is assembled out of the system. To include it, the #ERRSUM EQU NO source cards must be changed to #ERRSUM EQU YES in both the RBM Monitor and the S24RBM procedure file. Files that must be reassembled with these switches set are

- RBM Monitor.
- Overlay ID #07 (Unsolicited Key-In Subtask, Part 1).
- Overlay ID #35 (Buffered Line Printer Error Recovery).
- Overlay ID #36 (BCD Card and 7-Track Tape Handler).
- Overlay ID #37 (BCD Low Cost Card Punch Handler).
- Overlay ID #38 (BCD High Speed Card Punch Handler).

These changes will result in an increase in residency of 24_{10} words plus four times the number of I/O channels defined at SYSGEN time. One additional overlay will be included in the SP area on the RAD.

To display and reset these operation and error counters, two unsolicited operator key-ins are provided if error summary accounting is assembled in the system; DC (Display Counters) and RC (Reset Counters). The key-ins are invalid if the assembly switch is off.

The format of the key-ins is as follows:

$$\begin{Bmatrix} \text{DC} \\ \text{RC} \end{Bmatrix} \begin{bmatrix} \text{CHAN,chan} \\ \text{DEV,dev} \\ \text{DFN,dfn} \\ \text{OPLB,}\begin{Bmatrix}\text{fdun}\\ \text{oplb}\end{Bmatrix}\left[\text{,}\begin{Bmatrix}\text{F}\\ \text{B}\end{Bmatrix}\right] \end{bmatrix}$$

where

chan is a one- or two-digit hexadecimal number that represents the channel number. The limits on chan are $0 \le \text{chan} \le 27_{10}$.

dev is the two-digit hexadecimal address of the device in question.

dfn is a one- or two-digit hexadecimal number that indicates a Device File Number.

fdun is a FORTRAN device unit number. If the second parameter begins with "F:" or a numeral, an fdun is assumed.

oplb is a two-character operational label. It may not start with a numeral.

F or B if present, indicates that the specified operational label or FORTRAN device unit number is for the foreground or background respectively. If not specified, the oplabel is assumed to be for the background.

If no parameters are specified, all channel error and access counters will be displayed or reset, as appropriate.

If an error is detected while processing a DC or RC key-in, the message "!!KEY ERROR" will be output to the operator's console and the Key-In Subtask will be reentered. The following errors will cause a !!KEY ERROR message:

1. Syntax errors in key-in statement.
2. Reference to an I/O channel number not defined at SYSGEN time.

3. Reference to a device address not defined at SYSGEN time.

4. Reference to an invalid Device File Number.

5. Reference to an undefined operational label or FORTRAN device unit number.

6. Reference to an oplb or fdun currently assigned to zero.

All error and access counts will be reset to zero if RBM is rebooted.

The format of the message (directed to the operator's console) that is output in response to a DC key-in is as follows:

```
CHAN   cc   ERRORS   eeee   ACCESSES   aaaaaaaa
```

All numbers will be displayed in hexadecimal.

After processing a valid DC or RC command, the Key-In Subtask will be called again. At this time, the operator may elect to key in another DC or RC command or else input an "S" key-in to return to the background.

APPENDIX F. LINE PRINTER VFC's (WRITE BINARY)

Pseudo VFC	Print with Format Definition	Real VFC	Print Order	Data Chained to Text (Yes/No)	Printer Model
X'60'	Print, suppress upspace	X'60'	PF	Yes	A, B, C
X'80'	Print, suppress upspace	X'60'	PF	Yes	A, B, C
X'81'	Print, then space 1 line	X'C0'	PF	Yes	A, B, C
X'82' - X'8F'	Print, then space n lines (2-15)	1) X'60' 2) X'C0'+n	PF F	Yes No	A, B, C
X'90' - X'9F'	Print, then skip to channel n	1) X'60' 2) X'F0'+n	PF F	Yes No	A, B, C
X'A0' - X'AF'	Space n lines, print and inhibit upspace	1) X'C0'+n 2) X'60'	F PF	No Yes	A
		X'E0'+n	PF	Yes	B, C
X'B0' - X'BF'	Skip to channel n, print and inhibit upspace	1) X'F0'+n 2) X'60'	F PF	No Yes	A
		X'D0'+n	PF	Yes	B, C
X'C0' - X'CF'	Space n lines, print and upspace	X'C0'+n	PF	Yes	A, B, C
X'D0' - X'DF'	Skip to channel n, print and inhibit upspace	1) X'F0'+n 2) X'60'	F PF	No Yes	A
		X'D0'+n	PF	Yes	B, C
X'E0' - X'EF'	Space n lines, print and inhibit upspace	1) X'C0'+n 2) X'60'	F PF	No Yes	A
		X'E0'+n	PF	Yes	B, C
X'F0' - X'FF'	Skip to channel n, print and upspace	X'F0'+n	PF	Yes	A, B, C

Legend

PF – Print with format.

F – Format.

A – Printer models 3451, 7440, 7445.

B – Printer models 7441, 7442, 7446, 3461, 3463, 3464, 3465, 3466.

C – Printer model 7450.

n – Number of lines to skip or channel number. N is limited by line printer capabilities (e.g., a skip to channel > 1 for the 7450 line printer will result in a skip to channel 1).

Invalid VFC's result in a single space (X'C0') operation.

APPENDIX G. LOGICAL DEVICES

General

It is assumed that the reader has a general knowledge of the strcuture of M:READ/M:WRITE, which is flowcharted in Figure 24 of this manual.

An RBM user makes an Input or Output request to a Logical Device (LD) through calls to M:READ and M:WRITE, respectively.

Overview

The concept of a Logical Device arises from the need to be able to pass information and data between tasks. Logical Devices are defined at SYSGEN via a two-character mnemonic[†] (for model number), and an accompanying pseudo-device number (which indicates a channel number, preferably unique). The user performs Reads and Writes on DFNs (or assigned oplabels) associated with the LDs via calls on M:READ and M:WRITE.

Oplabels to be used by tasks for intertask communication may be specified at SYSGEN via the DFNs assigned to the same pseudo-device number of an LD. Communication between foreground and background tasks is accomplished by use of the foreground (F)/background (B) SYSGEN option at definition of the LD. One example of possible use would be where a task receives data from a hardware device via a standard oplabel or DFN. This data may be manipulated (if desired) by the task and passed on to another task via a pair of DFNs associated with the same LD. The receiving task may, if desired, pass the data to another DFN of the same LD, a different LD, or to a real physical device.

There are no restrictions as to direction of flow of information. Any DFN associated with an LD may be used to read or write to any other DFN associated with the same LD. At least two DFNs must be associated with one pseudo-device number to define an LD. Only two DFNs associated with an LD can be involved in any given LD data transfer.

SYSGEN Considerations

It is strongly recommended that the system be SYSGENed with the DISMISS option. This is necessary since the I/O interrupt task is not triggered for LDs until a READ/WRITE pair of operations is satisfied. Dismissal prevents a task from locking up the system waiting for an I/O operation to complete which cannot be completed until that task relinquishes control.

Similarly, it is recommended that the pseudo-device number (channel) used to specify the LD be unique. RBM can allow only one data transfer per channel at a time. Since an LD I/O operation requires both the read and write requests to be completed, the LD handler sets the channel busy when it processes the first request (Read or Write). The channel will not be available until the corresponding request is handled. The problem with having a physical device on the same channel as the LD is clear: no physical I/O transfers can be processed until the "handshake" LD request is processed. Since there is no timeout logic for LD operation, this could present a significant problem.

A discussion of the action taken by SYSGEN will aid in understanding the LD concept.

Real device definition at SYSGEN is implemented by requiring the user to specify the relationship between the model numbers of his hardware units and the hardware device number for the unit as follows:

$$\text{model/dn,}\begin{Bmatrix}\text{B}\\ \text{F}\\ \text{DI}\\ \text{DO}\end{Bmatrix}\begin{bmatrix}\text{I}\\ \text{,E}\end{bmatrix}[\text{,BCD}]$$

[†] The mnemonic "LD" or any other two-character mnemonic other than RD or XX can be used. This mnemonic may indicate the "device type" the Logical device is to represent; e.g., LP for line printer as required by the printer symbiont.

Refer to the RBM System Management Reference Manual, 90 30 36, for parameter definition. To each such definition, RBM assigns sequentially a DFN (Device File Number).

Logical Device definition at SYSGEN is implemented by requiring the user to specify the distinct logical groupings representing a Logical Device as follows:

$$\text{model/yy, }\left\{\begin{matrix}\text{B}\\\text{FF}\end{matrix}\right\}$$

where

model can be 'LD' or any other two-character mnemonic other than RD or XX. This mnemonic may indicate the 'device type' the Logical Device is to represent; e.g., LP for line printer as required by the printer symbiont. The mnemonic is placed in File Control Table 7 (FCT7) for that DFN and the mnemonic "LD" is stored in Device Type Table 1 (DTT1) for all Logical Device definitions.

yy is a pseudo-device number. This pseudo-device number will indicate a channel number that is preferably unused by any real device, X'00' ≤ yy ≤ X'FF'.

For example, the definition of two Logical Devices might appear as follows:

$$\left.\begin{matrix}\text{LD/08, F}\\\text{LD/08, F}\end{matrix}\right\}\ \text{LD \#1}$$

$$\left.\begin{matrix}\text{B1/09, B}\\\text{F1/09, F}\\\text{F2/09, F}\end{matrix}\right\}\ \text{LD \#2}$$

To each of the above specifying lines, SYSGEN will assign a unique DFN.

All Logical Device definitions should be grouped in the SYSGEN deck and placed immediately after the real device definitions (this is for the sake of clarity). The user makes pseudo-device number assignments within the range yy = hexadecimal 00 to FF. These device numbers represent channels that are preferably not used by real devices.

For example, a real device assigned a device number from X'91' to X'9F' would be serviced by the same channel register pair as a device assigned to X'01'. Thus, if a real device were assigned to any of the above device numbers, no physical device transfer could be made while the channel was busy with a Logical Device transfer.

Since there is no timeout value for Logical Devices, this could create significant delay problems. Therefore, it is suggested that the user avoid conflicts between hardware device numbers associated with hardware model numbers and pseudo-device numbers associated with the LD model specifications.

The number of LDs that can be defined for a given system configuration is suggested to be the maximum number of available channels (28) less the number of channels occupied by real devices.

In the previous example, SYSGEN would have made DFN assignments for the LD definitions as follows:

	Device Definition	DFN
(real devices)	model/dn$_1$	DFN$_1$
	⋮	
	model/dn$_n$	DFN$_n$
(Logical Devices)	LD/08, F	DFN$_{n+1}$
	LD/08, F	DFN$_{n+2}$
	B1/09, B	DFN$_{n+3}$
	F1/09, F	DFN$_{n+4}$
	F2/09, F	DFN$_{n+5}$

If desired, the user could specify oplabel association for the LDs as follows:

SYSGEN BCKG. OP. LBL. $OP_1 = DFN_{n+3}$

SYSGEN FGD. OP. LBL. $OP_2 = DFN_{n+1}$

$OP_3 = DFN_{n+2}$

$OP_4 = DFN_{n+4}$

$OP_5 = DFN_{n+5}$

where $OP_1/OP_4/OP_5$ represent one LD and OP_2/OP_3 represent another LD.

Implementation

SYSGEN processes LDs in the same way as other devices. DTT and IOCT entries are established. M:RSVP checks the DTT for LD and treats such requests as valid but performs no operation for an LD. M:READ/M:WRITE bypass the call to Q:LOADC for LDs.

M:CTRL requests an LD's receive 'operational not meaningful' status. M:CKREST will ignore active DFNs included in a Logical Device I/O operation when it is allowing I/O to run down prior to a checkpoint of background.

An Overlay contains a pre-I/O edit routine, a post I/O edit routine, and an error recovery routine for LDs (see Figure G-1). This overlay is reentrant and optionally resident.

Pre-I/O Edit Routine

Since one channel per Logical Device is assigned at SYSGEN, all read/write operations to a member DFN of the LD share the same channel status table entry. (Note that each DFN has its own FCT and IOCT entry.)

One restriction is imposed when background is involved in a Logical Device operation. All LD transfers involving background will take place at the background level. Therefore, between background and foreground, the foreground request will always claim the channel. Then the background request will be honored and the transfer completed. If background is the first to issue an LD request, the user will receive an artificial busy return on a NO-WAIT request or will be held at the system level waiting for the corresponding foreground request if background issued a WAIT request.

To describe the "sharing" of a CST entry, consider that the first operational request on an associated inactive channel will cause the DFN of the requestor to be placed in the CST thus specifying the requestor as the "owner" of the CST.

All LDs are specified by the same DTT entry. The pre-I/O edit, post-I/O edit, and error recovery routines specified in the DTT perform those operations necessary to satisfy the LD requests as indicated in M:READ/M:WRITE calls.

Subsequent operational requests utilizing the same channel (i.e., requests on member DFNs of the same LD), which is now "owned" by the first requestor, are satisfied according to the following rules:

1. If the "owner" of the CST entry is a write request, a subsequent read satisfies the operation. The CST entry is "frozen" and the data is moved. Completion is posted for both the read and write. Other read requests occurring after the first read request will receive a busy indication. All further write requests will receive a busy indication.

2. If the "owner" of the CST entry is a read request, a subsequent write request will satisfy the operation. At that time, the CST entry is "frozen", the data is moved, and completion is posted for both the read and write.

   Subsequent read and write requests will receive a busy indication.

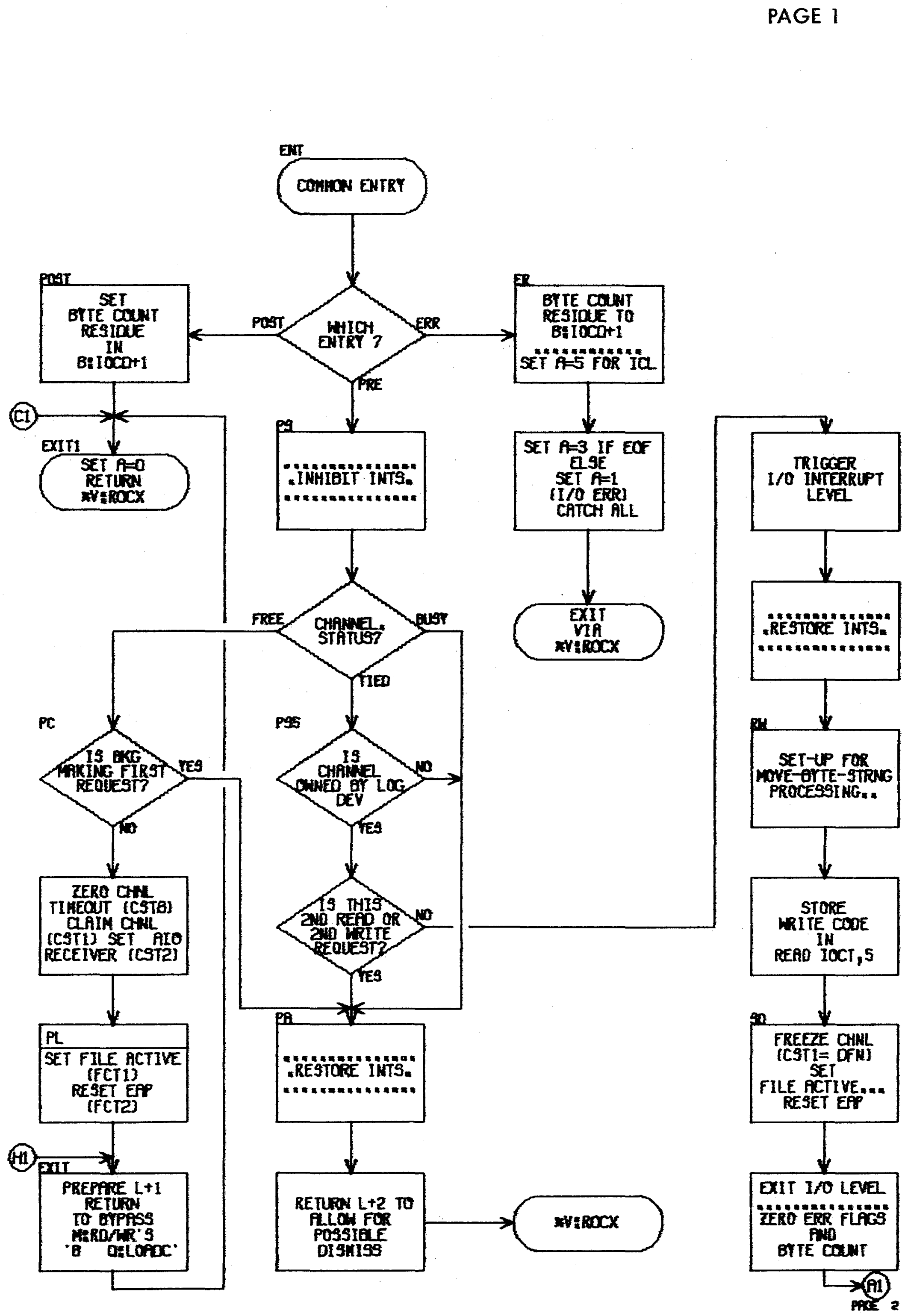

PAGE 1
ENT
COMMON ENTRY
WHICH ENTRY 7
POST
ERR
PRE
POST
SET BYTE COUNT RESIDUE IN B:IOCD+1
ER
BYTE COUNT RESIDUE TO B:IOCD+1
SET A=5 FOR ICL
C1
EXIT1
SET A=0 RETURN *V:ROCX
P9
.INHIBIT INTS.
SET A=3 IF EOF ELSE SET A=1 (I/O ERR) CATCH ALL
EXIT VIA *V:ROCX
TRIGGER I/O INTERRUPT LEVEL
CHANNEL. STATUS7
FREE
BUSY
TIED
.RESTORE INTS.
PC
IS BKG MAKING FIRST REQUEST7
YES
NO
P95
IS CHANNEL OWNED BY LOG DEV
NO
YES
RW
SET-UP FOR MOVE-BYTE-STRNG PROCESSING..
ZERO CHNL TIMEOUT (CST8) CLAIM CHNL (CST1) SET AIO RECEIVER (CST2)
IS THIS 2ND READ OR 2ND WRITE REQUEST7
NO
YES
STORE WRITE CODE IN READ IOCT,S
PL
SET FILE ACTIVE (FCT1) RESET EAP (FCT2)
PA
.RESTORE INTS.
SO
FREEZE CHNL (CST1= DFN) SET FILE ACTIVE... RESET EAP
H1
EXIT
PREPARE L+1 RETURN TO BYPASS M:RD/WR'S 'B O:LOADC'
RETURN L+2 TO ALLOW FOR POSSIBLE DISMISS
*V:ROCX
EXIT I/O LEVEL
ZERO ERR FLAGS AND BYTE COUNT
A1
PAGE 2

Figure G-1. Logical Device Handler

PAGE 2

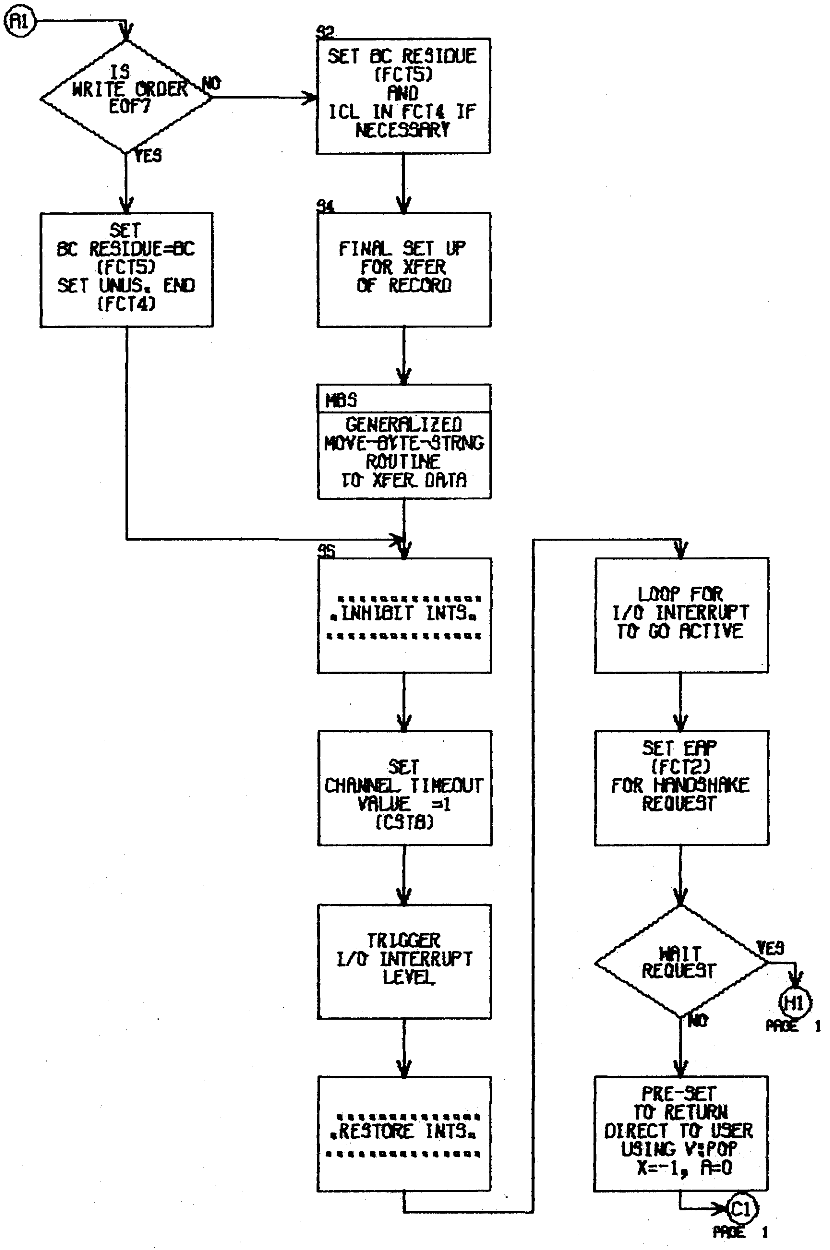

A1
IS WRITE ORDER EOF7
NO
YES
S2
SET BC RESIDUE (FCT5) AND ICL IN FCT4 IF NECESSARY
SET BC RESIDUE=BC (FCT5) SET UNUS. END (FCT4)
S4
FINAL SET UP FOR XFER OF RECORD
MBS
GENERALIZED MOVE-BYTE-STRNG ROUTINE TO XFER DATA
S5
.INHIBIT INTS.
SET CHANNEL TIMEOUT VALUE =1 (CST8)
TRIGGER I/O INTERRUPT LEVEL
.RESTORE INTS.
LOOP FOR I/O INTERRUPT TO GO ACTIVE
SET EAP (FCT2) FOR HANDSHAKE REQUEST
WAIT REQUEST
YES
H1
PAGE 1
NO
PRE-SET TO RETURN DIRECT TO USER USING V:POP X=-1, A=0
C1
PAGE 1

Figure G-1. Logical Device Handler (cont.)

3. Because there is no real device to operate an I/O interrupt, a channel timeout for LDs must be simulated when the pre-I/O edit routine determines that an I/O operation is satisfied. The mechanization of this is documented in the code.

Error Recovery Routine

This routine merely checks the indicators set in the FCT by the pre-I/O edit routine and posts the appropriate completion status.

Post-I/O Edit Routine

The post-I/O edit routine currently stores the byte count residue and returns to the user.

Use of M:READ/M:WRITE

M:READ and M:WRITE are used exactly as for an operation on a real device with the following exceptions:

1. Channel timeout does not apply and will be ignored if specified.

2. Read backward is not meaningful (order X'0C').

3. Read binary and read automatic are not differentiated. Only one record, as specified by buffer address and byte count, is transferred per request (orders X'02' and X'06').

4. Check write is not meaningful (order X'07').

5. Write binary and write EBCDIC are not differentiated (orders X'01' and X'05').

Coding of M:READ/M:WRITE calls should check for a status return indicating the AIO Receiver, if specified, will not be entered. If requested, only the AIO Receiver of the channel "owner" will be entered. All other requests to the channel (as long as the channel is "owned") will return with the X-register set = -1 indicating the AIO Receiver will not be entered.

All status returns and completion codes retain the same meaning, where applicable, as for real devices.

All no-wait operations should be followed by a CHECK operation as standard. Wait/no-wait I/O and the dismiss function are handled as for real device I/O.

Recommended Practices

1. SYSGEN with the DISMISS option.

2. SYSGEN the LDs with unique pseudo-device numbers; i.e., allow no real devices to co-exist on the same channel with a Logical Device.

3. Check status on return from M:READ/M:WRITE to determine whether an AIO Receiver, if specified, will be entered.

4. A CHECK operation should be performed for all no-wait M:READ/M:WRITE calls.

5. The overlay containing the pre-I/O edit, post-I/O edit, and error routines for LDs should be made resident to facilitate rapid response to M:READ/M:WRITE calls. This overlay is reentrant.

6. Channel timeout is not presently implemented for LDs. If timeout is required for some application, use of a clock routine combined with checking status could be used.

Xerox Corporation
701 South Aviation Boulevard
El Segundo, California 90245

XEROX

Reader Comment Form

We would appreciate your comments and suggestions for improving this publication.

Publication No.	Rev. Letter	Title	Current Date

How did you use this publication?

□ Learning □ Installing □ Sales
□ Reference □ Maintaining □ Operating

Is the material presented effectively?

□ Fully Covered □ Well Illustrated □ Well Organized □ Clear

What is your overall rating of this publication?

□ Very Good □ Fair □ Very Poor
□ Good □ Poor

What is your occupation?

Your other comments may be entered here. Please be specific and give page, column, and line number references where applicable. To report errors, Please use the Xerox Software Improvement or Difficulty Report (1188) instead of this form.

Your Name & Return Address

Thank You For Your Interest. (fold & fasten as shown on back, no postage needed if mailed in U.S.A.)

Staple Staple

Fold

First Class
Permit No. 229
El Segundo,
California

BUSINESS REPLY MAIL

No postage stamp necessary if mailed in the United States

Postage will be paid by

Xerox Corporation
701 South Aviation Boulevard
El Segundo, California 90245

Attn: Programming Publications

Fold

XEROX

701 South Aviation Boulevard
El Segundo, California 90245
213 679-4511