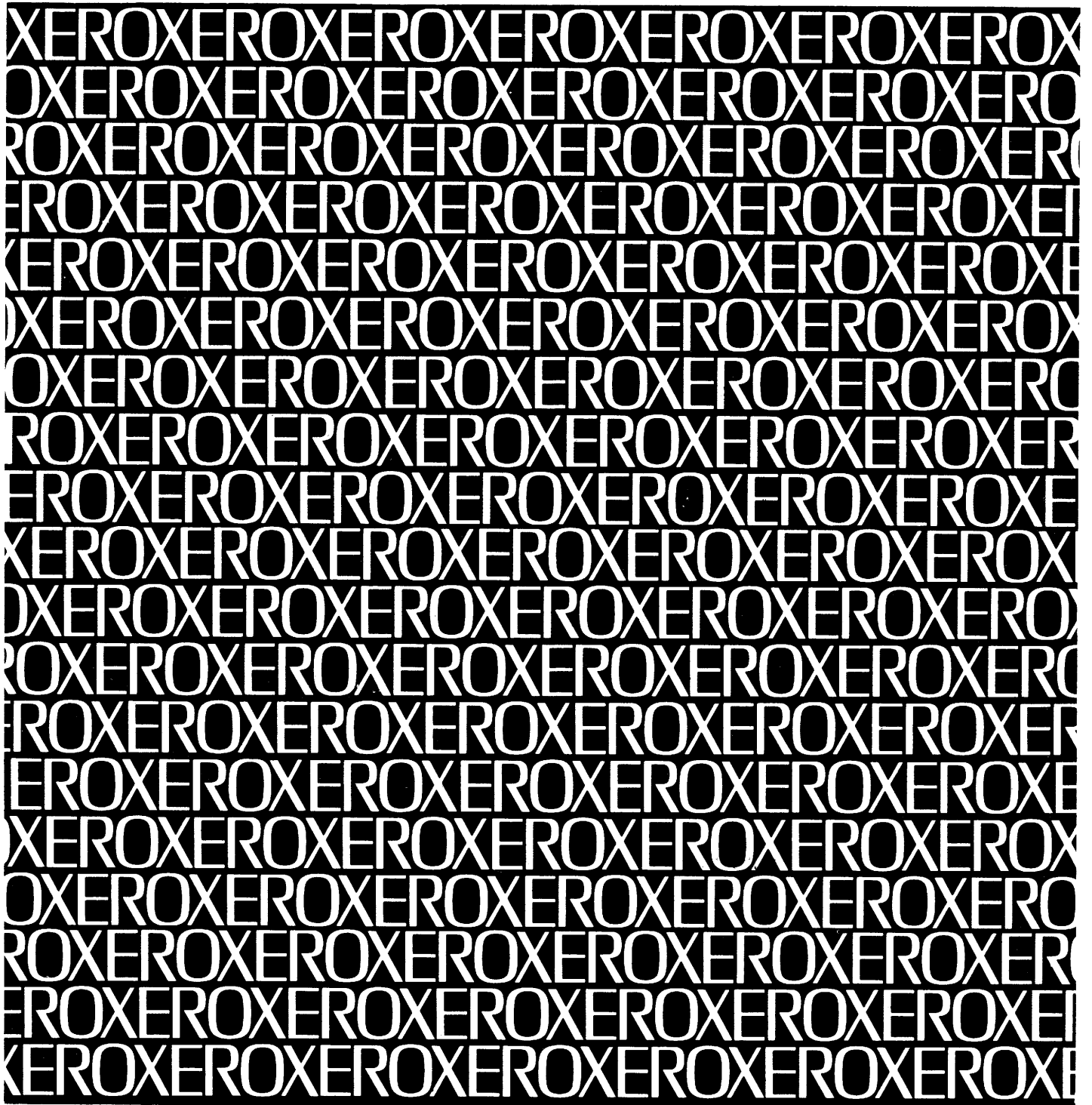


Xerox Real-Time Batch Monitor (RBM)

Sigma 5-9 Computers

System
Technical Manual



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

XEROX

Xerox Real-Time Batch Monitor (RBM)

Sigma 5-9 Computers

System

Technical Manual

90 17 00C

March 1971

Price: \$6.25

REVISION

This publication is a revision of the Xerox Real-Time Batch Monitor (RBM)/System Technical Manual for Sigma 5-9 Computers, Publication Number 90 17 00B (dated March 1971). The manual incorporates changes from the 90 17 00B-1(10/71) revision package, which reflects version C01 of the RBM operating system. No other changes beyond those in the 90 17 00B-1(10/71) revision package are included. A change in the text from that of the previous manual is indicated by a vertical line in the margin of the page.

RELATED PUBLICATIONS

| <u>Title</u> | <u>Publication No.</u> |
|--|------------------------|
| Xerox Sigma 5 Computer/Reference Manual | 90 09 59 |
| Xerox Sigma 6 Computer/Reference Manual | 90 17 13 |
| Xerox Sigma 7 Computer/Reference Manual | 90 09 50 |
| Xerox Sigma 8 Computer/Reference Manual | 90 17 49 |
| Xerox Sigma 9 Computer/Reference Manual | 90 17 33 |
| Xerox Real-Time Batch Monitor (RBM)/RT,BP Reference Manual | 90 15 81 |
| Xerox Real-Time Batch Monitor (RBM)/OPS Reference Manual | 90 16 47 |
| Xerox Real-Time Batch Monitor (RBM)/RT,BP User's Guide | 90 16 53 |

Manual Type Codes: BP - batch processing, LN - language, OPS - operations, RBP - remote batch 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

| | | | |
|---|----|---|----|
| PREFACE | vi | JCP Loader | 65 |
| 1. RBM INITIALIZATION ROUTINE | 1 | Job Accounting | 68 |
| 2. RBM CONTROL TASK | 3 | Background TEMP Area Allocation | 68 |
| Structure | 3 | 5. FOREGROUND SERVICES | 71 |
| Function and Implementation | 3 | Implementation | 71 |
| Resident Control Task | 3 | RUN | 71 |
| Key-In Processor | 6 | RLS | 71 |
| Foreground Release (FGL1) | 22 | MASTER/SLAVE | 71 |
| Foreground Loader (FGL2) | 22 | STOPIO/STARTIO | 71 |
| Background Loader (BKL1) | 24 | IOEX | 71 |
| Background Loader (BKL2) | 24 | TRIGGER, DISABLE, ENABLE, ARM, DISARM, CONNECT | 71 |
| Checkpoint Restart (CKPT) | 25 | Task Control Block (TCB) | 72 |
| Abort/Exit | 25 | 6. MISCELLANEOUS SERVICES | 75 |
| Postmortem Dump (PMD) | 26 | SEGLOAD | 75 |
| 3. I/O HANDLING METHODS | 27 | Trap Handling | 75 |
| Channel Concept | 27 | Trap CAL | 75 |
| Handling Devices | 27 | Trap Processing | 75 |
| Single Interrupt Mode | 27 | TRTN (Trap Return) | 77 |
| Interrupt-to-Interrupt Mode | 27 | 7. RBM SIZES | 78 |
| System Tables | 27 | 8. RBM TABLE FORMATS | 79 |
| IOQ (Request Information) | 27 | RAD File Table (RFT) | 79 |
| DCT (Device Control) | 28 | Device Control Table (DCT) | 80 |
| CIT (Channel Information) | 28 | DCT Format | 80 |
| Handler Tables | 28 | SYSGEN DCT Consideration | 82 |
| Separation of Priorities and Control Task | 29 | Channel Information Table (CIT) | 83 |
| INTSIM Routine | 30 | I/O Queue Table (IOQ) | 84 |
| CTTEST Routine | 30 | Blocking Buffers | 86 |
| Initiating I/O Requests | 30 | Foreground Program Table (FGT) | 86 |
| I/O Interrupt Processing | 36 | Master Dictionary | 87 |
| I/O Cleanup | 36 | Operational Label Table (OPLBS) | 88 |
| Miscellaneous Basic I/O Subroutines | 41 | Interrupt Label Table (INTLB) | 88 |
| REQCOM (Request Complete) | 41 | OVLOAD Table (for RBM Overlays Only) | 89 |
| CUPCORE, CUPDCB Cleanup, End-Action Routines | 44 | Write Lock Table (WLOCK) | 89 |
| MSGOUT (Message Out) | 45 | 9. OVERLAY LOADER | 90 |
| QUEUE | 45 | Overlay Structure | 90 |
| User I/O Services | 46 | Overlay Loader Execution | 90 |
| OPEN | 46 | Dynamic Table Area | 91 |
| CLOSE | 46 | Dynamic Table Order | 92 |
| READ/WRITE | 48 | T:SYMBOL and T:VALUE | 92 |
| PRINT | 49 | T:VALUE Entry Formats | 93 |
| TYPE | 49 | T:SYMBOL Entry Formats | 94 |
| DFM | 49 | T:PUBVAL and T:PUBSYM | 94 |
| DVF | 49 | T:PUBVAL Entry Formats | 94 |
| REWIND | 49 | T:PUBSYM Entry Formats | 95 |
| WEOF | 49 | T:VALX | 95 |
| PREC | 49 | T:DCB | 96 |
| PFILE | 50 | T:SEG | 97 |
| 4. JOB CONTROL PROCESSOR | 51 | B:MT | 98 |
| Overview | 51 | | |
| ASSIGN Command Processing | 51 | | |

| | |
|---|-----|
| T:DECL | 98 |
| T:CSECT | 99 |
| T:FWD | 99 |
| T:FWDX | 99 |
| T:MODULE | 100 |
| T:ROMI | 100 |
| T:DCBV | 101 |
| T:MODIFY | 101 |
| Use of the Dynamic Table Area During LIB | 102 |
| T:LDEF | 103 |
| T:LROM | 103 |
| MODULE File | 104 |
| EBCDIC File | 104 |
| MODIR File | 104 |
| DEFREF File | 104 |
| Use of Dynamic Table Area During PASSTWO | 105 |
| T:GRAN | 105 |
| T:ASSN | 106 |
| MAP Use of Dynamic Table Area | 106 |
| DIAG Use of Dynamic Table Area | 107 |
| Root Tables | 107 |
| T:PL | 108 |
| T:DCBF | 108 |
| Scratch Files | 109 |
| Program File Format | 110 |
| Foreground/Background Program Header | 111 |
| Public Library Header | 111 |
| Logical Flow of the Overlay Loader | 112 |
| Logical Flow of CCI | 112 |
| Logical Flow of PASSONE | 112 |
| Logical Flow of LIB | 113 |
| Logical Flow of PASSTWO | 113 |
| Logical Flow of MAP | 113 |
| Logical Flow of DIAG | 114 |
| Loader-Generated Table Formats | 114 |
| PCB | 114 |
| DCBTAB | 115 |
| INTTAB | 115 |
| OVLOAD | 115 |
| Loading Overlay Loader | 116 |
| | |
| 10. RAD EDITOR | 126 |
| Functional Flow | 126 |
| Permanent RAD Area Maintenance | 126 |
| Permanent File Directory | 126 |
| Control Commands | 128 |
| :ALLOT | 128 |
| :DELETE | 129 |
| :TRUNCATE | 129 |
| :SQUEEZE | 129 |
| Library File Maintenance | 129 |
| Algorithms for Computing Library File Lengths | 129 |
| Library File Formats | 131 |
| MODIR File | 132 |
| MODULE File | 132 |
| EBCDIC File | 132 |
| DEFREF File | 133 |
| Command Execution | 133 |
| :ALLOT | 134 |
| :COPY | 134 |
| :DELETE | 134 |
| :SQUEEZE | 134 |

| | |
|---------------------------|-----|
| Bad Track Handling | 134 |
| Command Execution | 134 |
| :BDTRACK | 134 |
| :GDTRACK | 135 |
| Use of IOEX for Disk Pack | 135 |
| Utility Functions | 135 |
| :MAP | 135 |
| :CLEAR | 137 |
| :COPY | 137 |
| :DUMP | 137 |
| :SAVE | 138 |
| :RESTORE | 139 |

| | |
|---|-----|
| 11. SYSTEM GENERATION | 151 |
| Overview | 151 |
| SYSGEN/SYSLOAD Flow | 152 |
| Loading Simulation Routines, RBM, and RBM Overlays | 152 |
| SYSGEN I/O | 159 |
| Rebootable Deck Format | 159 |
| Stand-Alone SYSGEN Loader | 160 |

APPENDIXES

| | |
|----------------------------------|-----|
| A. RBM SYSTEM FLAGS AND POINTERS | 161 |
| B. PAPER TAPE STANDARD FORMAT | 166 |

FIGURES

| | |
|--|----|
| 1. Initialize Routine Core Layout | 1 |
| 2. RBM Initialize Routine Overall Flow | 2 |
| 3. Resident Control Task Flow | 4 |
| 4. Key-In Processor Flow | 6 |
| 5. Operator Key-In Flow "C" | 7 |
| 6. Operator Key-In Flow, "W" | 7 |
| 7. Operator Key-In Flow, "X" | 8 |
| 8. Operator Key-In Flow, "TY" and "TC" | 8 |
| 9. Operator Key-In Flow, "CC" | 10 |
| 10. Operator Key-In Flow, "DT" and "DE" | 10 |
| 11. Operator Key-In Flow, "SY" and "SYC" | 11 |
| 12. Operator Key-In Flow, "FG", "FGC", "FSC", and "SFC" | 11 |
| 13. Operator Key-In Flow, "RUN" | 12 |
| 14. Operator Key-In Flow, "RLS" | 13 |
| 15. Operator Key-In Flow, "INTLB" | 14 |
| 16. Operator Key-In Flow; "yyndd" | 15 |
| 17. Operator Key-In Flow; "STDLB" | 16 |
| 18. Operator Key-In Flow, "FMEM" | 19 |
| 19. Operator Key-In Flow, "CINT" | 20 |
| 20. Operator Key-In Flow, "DM", "BB", and "DF" | 21 |
| 21. Operator Key-In Flow, "DED" and "UND" | 22 |
| 22. SERDEV Routine Flow | 31 |
| 23. INTSIM Routine Flow | 34 |
| 24. CTTEST Routine Flow | 35 |
| 25. STARTIO Routine Flow | 37 |
| 26. IOINT Routine Flow | 39 |
| 27. CLEANUP Routine Flow | 42 |
| 28. QUEUE Subroutine Flow | 47 |
| 29. JCP General Flow | 52 |
| 30. JOB Command Flow | 54 |

| | |
|---|-----|
| 31. FIN Command Flow | 55 |
| 32. ASSIGN Command Flow | 55 |
| 33. DAL Command Flow | 56 |
| 34. ATTEND Command Flow | 56 |
| 35. MESSAGE Command Flow | 56 |
| 36. PAUSE Command Flow | 57 |
| 37. CC Command Flow | 57 |
| 38. LIMIT Command Flow | 57 |
| 39. STDLB Command Flow | 58 |
| 40. NAME Command Flow | 59 |
| 41. RUN Command Flow | 61 |
| 42. ROV Command Flow | 61 |
| 43. POOL Command Flow | 61 |
| 44. ALLOBT Command Flow | 62 |
| 45. LOAD Command Flow | 63 |
| 46. PMD Command Flow | 64 |
| 47. PFIL, PREC, SFIL, REWIND, and UNLOAD Command Flows | 64 |
| 48. WEOF Command Flow | 64 |
| 49. Core Layout During JCP Execution | 65 |
| 50. Pre-PASS1 Core Layout | 66 |
| 51. ARM, DISARM, and CONNECT Function Flow | 73 |
| 52. Overlay Structure of the Overlay Loader | 90 |
| 53. Overlay Loader Core Layout | 91 |
| 54. LIB Reorganization of Dynamic Table Area | 102 |
| 55. PASSTWO Reorganization of Dynamic Table Area | 105 |
| 56. MAP Table Reference | 107 |
| 57. Program File Format | 110 |
| 58. Overlay Loader Flow, !OLOAD | 116 |
| 59. Overlay Loader Flow, CCI | 117 |
| 60. Overlay Loader Flow, PASSONE | 118 |
| 61. Overlay Loader Flow, PASSTWO | 121 |
| 62. Overlay Loader Flow, MAP | 123 |
| 63. Overlay Loader Flow, RDIAG | 124 |
| 64. Overlay Loader Flow, RDIAGX | 124 |

| | |
|---|-----|
| 65. Overlay Loader Flow, DIAG | 125 |
| 66. RAD Editor Functional Flow | 127 |
| 67. Permanent RAD Area | 130 |
| 68. RAD Editor Flow, ALLOT | 140 |
| 69. RAD Editor Flow, COPY | 141 |
| 70. RAD Editor Flow, SQUEEZE | 146 |
| 71. RAD Editor Flow, SAVE | 148 |
| 72. RAD Editor Flow, RESTORE | 150 |
| 73. SYSGEN and SYSLOAD Layout Before Execution | 151 |
| 74. SYSGEN and SYSLOAD Layout After Execution | 152 |
| 75. SYSGEN/SYSLOAD Flow | 153 |

TABLES

| | |
|--------------------------------------|-----|
| 1. ASSIGN Table | 51 |
| 2. RAD File Table Allocation | 80 |
| 3. DCT Subtable Formats | 81 |
| 4. IOQ Allocation and Initialization | 84 |
| 5. Foreground Program Subtables | 87 |
| 6. Overlay Loader Segment Functions | 90 |
| 7. T:DCBF Entries | 108 |
| 8. Background Scratch Files | 109 |
| 9. Standard SYSLOAD DEFs | 158 |
| A-1. RBM System Flags and Pointers | 161 |

PREFACE

The primary purpose of this manual is to provide a guide for better comprehension of the program listings supplied with the Xerox Real-Time Batch Monitor (RBM) operating system. The programs and processors included are the System Generation program, the Monitor and its associated tasks and subprocessors such as the Job Control Processor, Overlay Loader, and RAD Editor.

The manual is intended for Sigma RBM users who require an in-depth knowledge of the structure and internal functions of the RBM operating system for system maintenance purposes. Since the RBM Technical Manual and program listings are complementary, it is recommended that the listings be readily available when referencing this manual.

The Delta Debug package referenced in this manual is an internal Xerox development and maintenance tool and is not suitable or available for customer use.

1. RBM INITIALIZATION ROUTINE

The RBM Initialize routine is entered from the RAD Bootstrap every time the system is booted from the RAD, and it sets up core prior to the execution of RBM. It also modifies the resident RBM system (including all system tables), the RBM overlays, and the Job Control Processor. Modifications may be made from the C, OC, or SI device that is selected by a corresponding sense switch setting (1, 2, or 3). If sense switch 4 is reset, the Initialize routine loads all programs on the FP area of the RAD designated as resident foreground into the foreground area. The Initialize routine extends into the background and can be overwritten by background programs, since it executes only once. In Figure 1 below, the background first word address is the first page boundary after RBMEND (the end of resident RBM). The Initialize routine terminates by setting the idle subtask bit and triggering the RBM Control Task.

The general flow of the Initialize routine from entry from RAD Bootstrap to triggering the Control Task interrupt is illustrated in Figure 2.

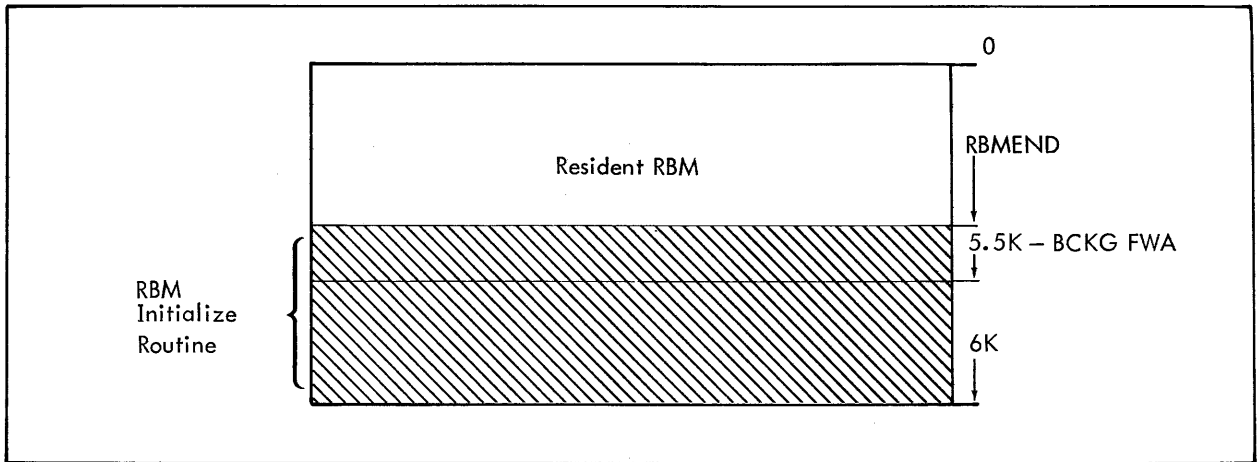


Figure 1. Initialize Routine Core Layout

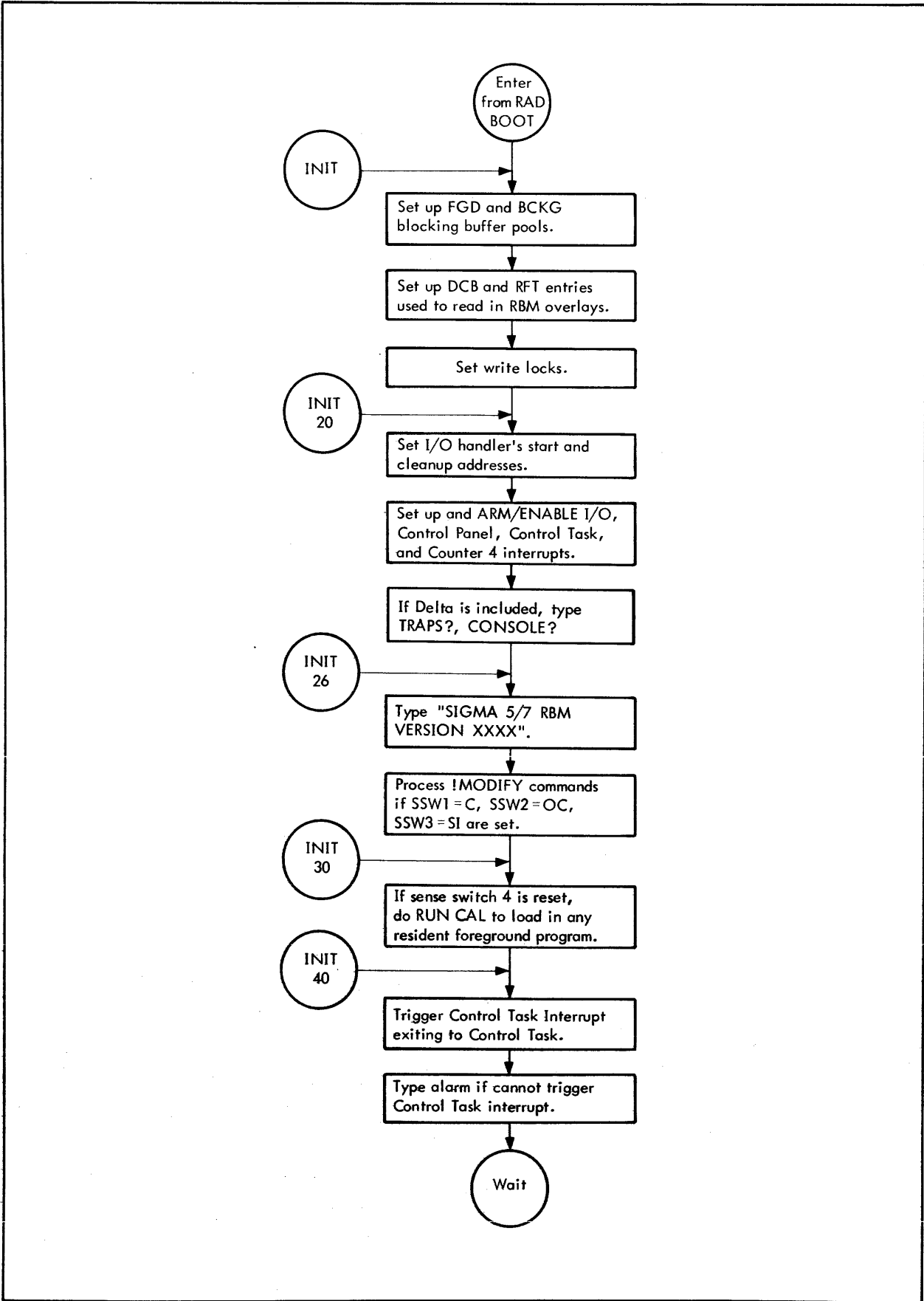


Figure 2. RBM Initialize Routine Overall Flow

2. RBM CONTROL TASK

The RBM Control Task is connected to the lowest priority system interrupt. Among the functions performed by the Control Task are

- Key-in processing
- Foreground program "RUN"
- Foreground program "RELEASE"
- Background program Load
- Background Checkpoint
- Background Restart
- Background Exit
- Background Abort
- Background Wait
- Postmortem Dump
- Deferred I/O processing
- Periodic service of all devices.

In facilities where there are no system interrupts, the Control Task is connected to the Control Panel interrupt (see "Key-In Processor" later in this chapter).

Structure

The Control Task consists of a resident portion and a number of nonresident portions that overlay each other in a single area of core. The overlays are

- Foreground program "RELEASE" (FGL1)
- Foreground program "RUN" (FGL2)
- Background program Loader part 1 (BKL1)
- Background program Loader part 2 (BKL2)
- Checkpoint/Restart (CKPT)
- Abort/Exit (ABEX)
- Postmortem Dump (PMD)
- Key-in Processor part 1 (KEY1)
- Key-in Processor part 2 (KEY2)

These overlays are reloadable from the RAD after being partially executed. Memory locations X'22' to X'3F' are used for the necessary temporary storage.

Function and Implementation

Resident Control Task

The resident portion of the Control Task functions as a scheduler for the various subtasks. The priority of the subtasks is determined by the order that the resident Control Task tests the signal bits, with Checkpoint (bit 0) being the highest priority of the tasks represented. The logic is depicted in Figure 3.

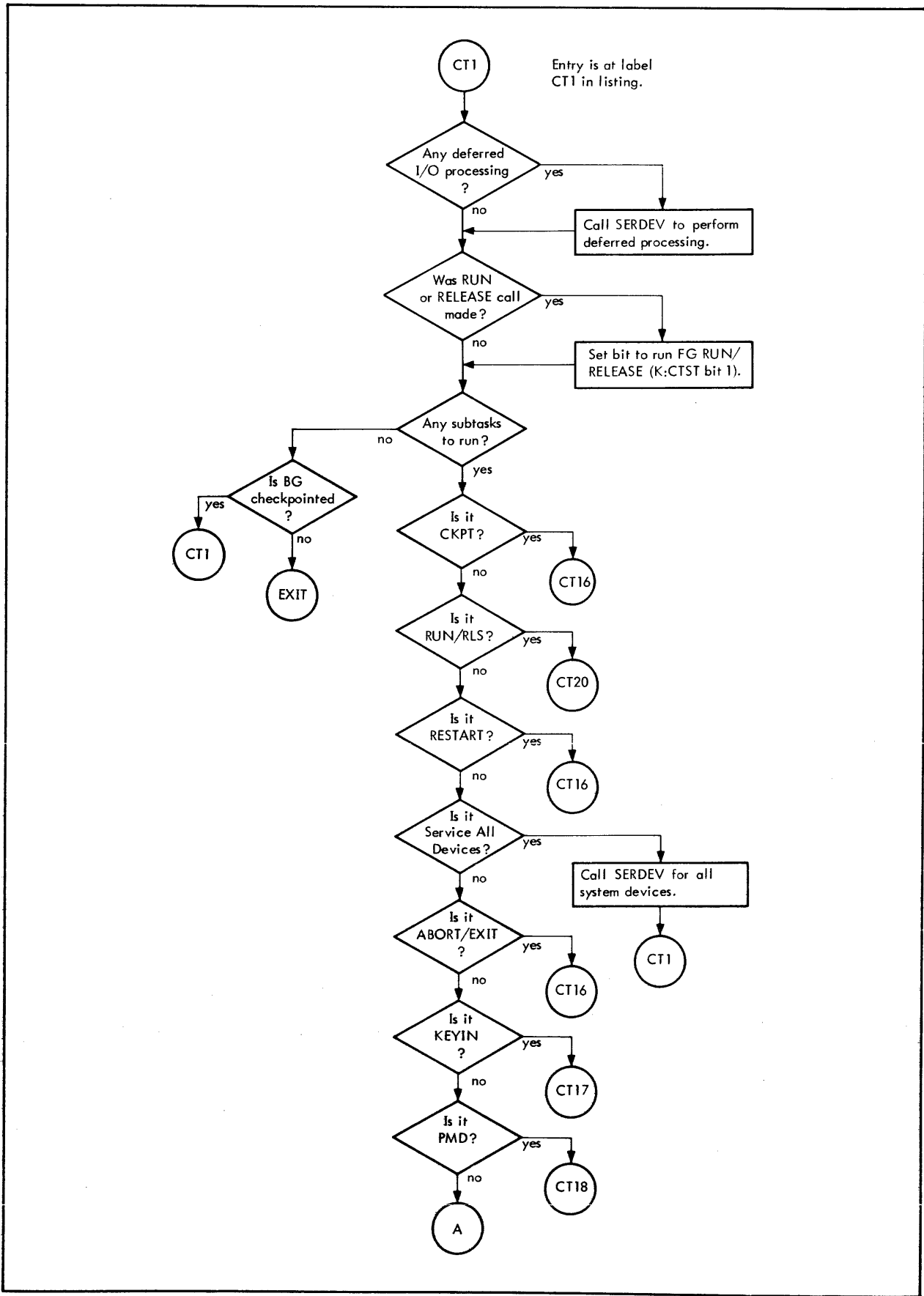


Figure 3. Resident Control Task Flow

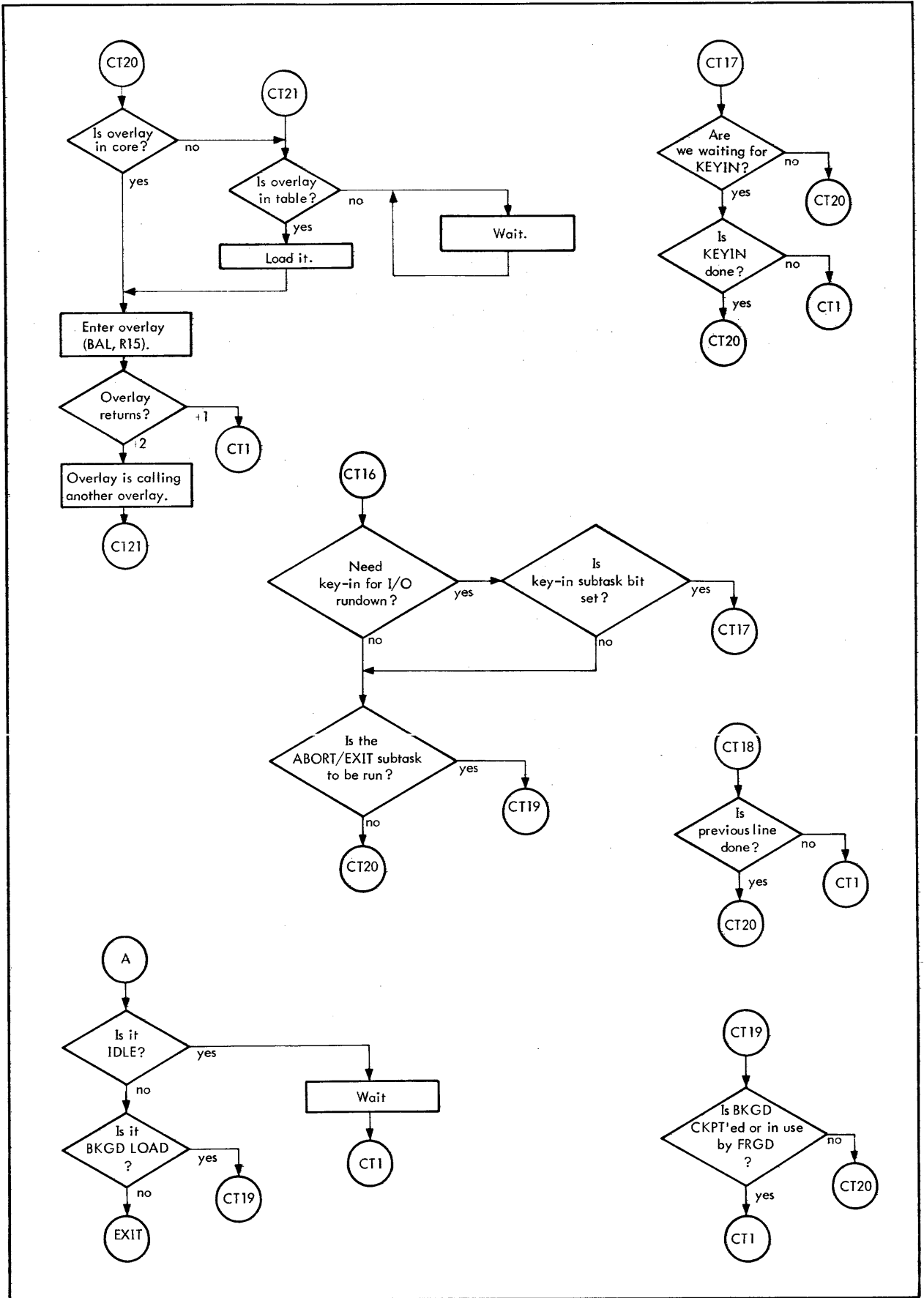


Figure 3. Resident Control Task Flow (cont.)

Key-In Processor

The key-in processor portion of the Control Task processes all operator key-ins that are initiated when the operator depresses the console interrupt. When the console interrupt becomes active, the key-in bit (bit 5) is set in K:CTST and the Control Task interrupt is triggered if the Control Task is connected to a system interrupt. Figure 4 illustrates the Key-In Processor flow, where CPINT is the label used in the code.

The key-in request is recognized in the Control Task, and if the system is not waiting for key-in or if a key-in input has been completed, the key-in overlay is entered. In the key-in overlay, the message

!!KEY-IN

is typed on the OC device, the OC device is enabled for input, and control is returned to the resident Control Task. The resident Control Task periodically tests for completion of the key-in input, and when this occurs, the key-in overlay is again entered. At this entry to the key-in overlay, illegal inputs are diagnosed and if any occurred,

!!KEY ERR

is typed on the OC device.

Legal key-ins are processed as illustrated in Figures 5 through 21.

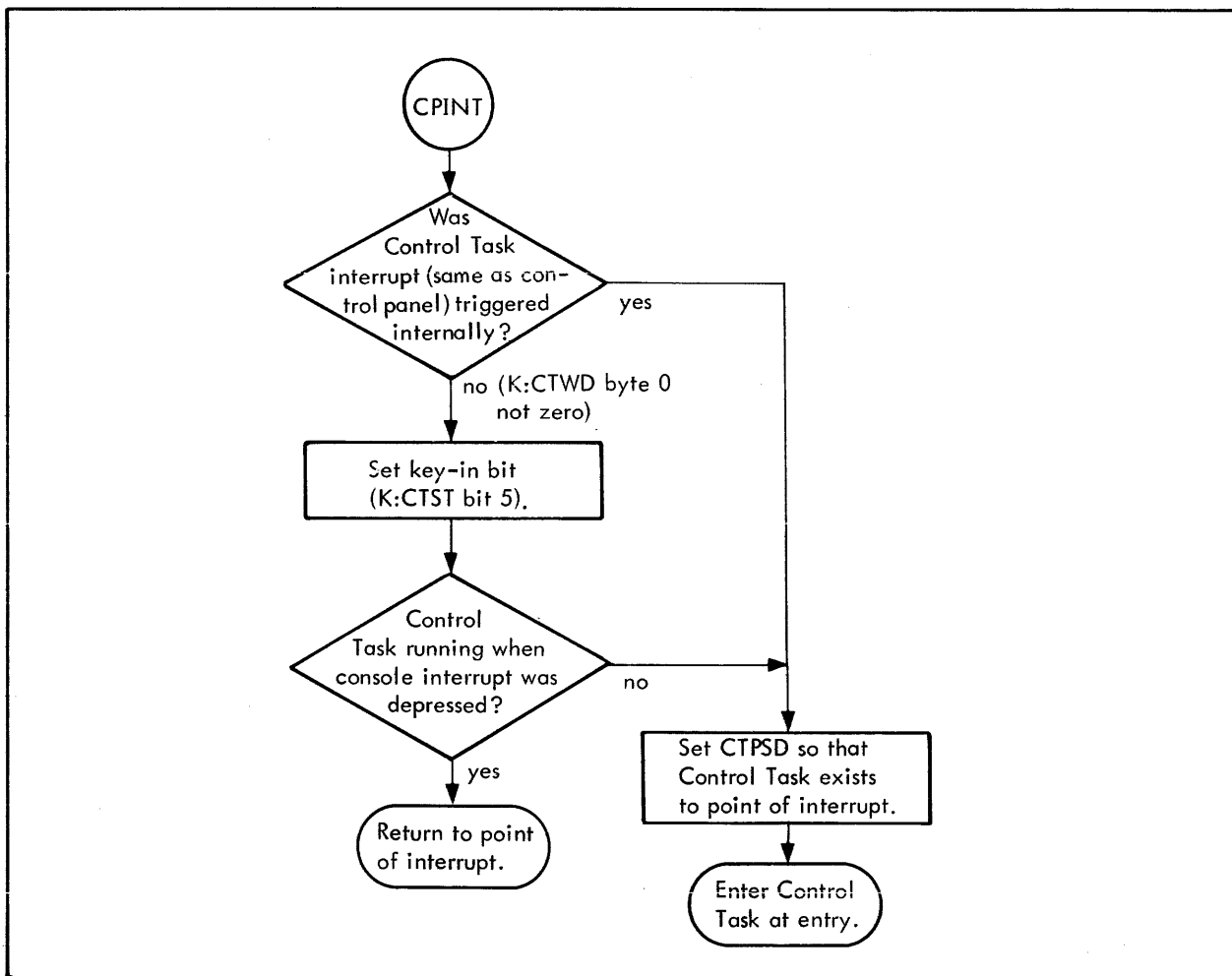


Figure 4. Key-In Processor Flow

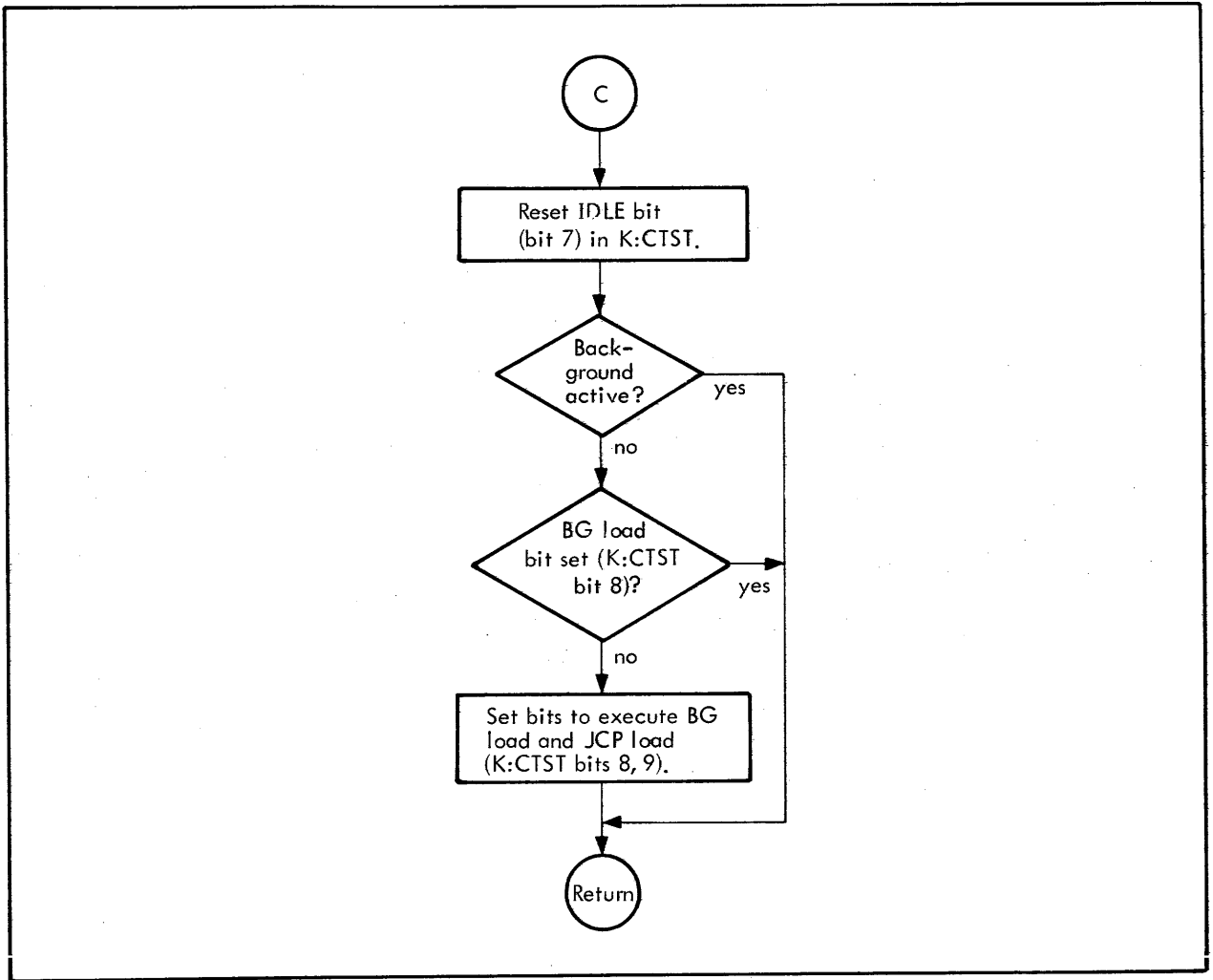


Figure 5. Operator Key-In Flow, "C"

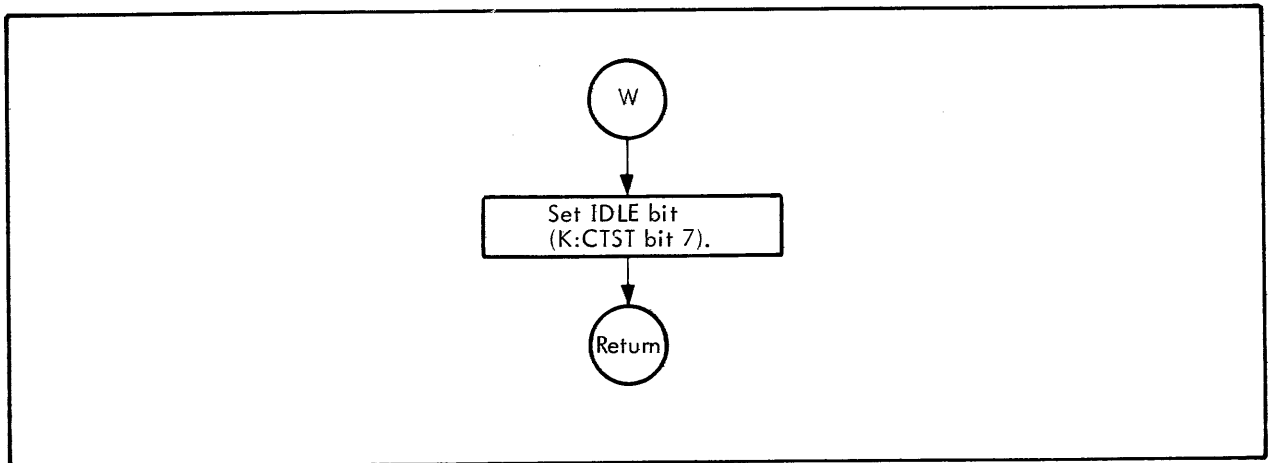


Figure 6. Operator Key-In Flow, "W"

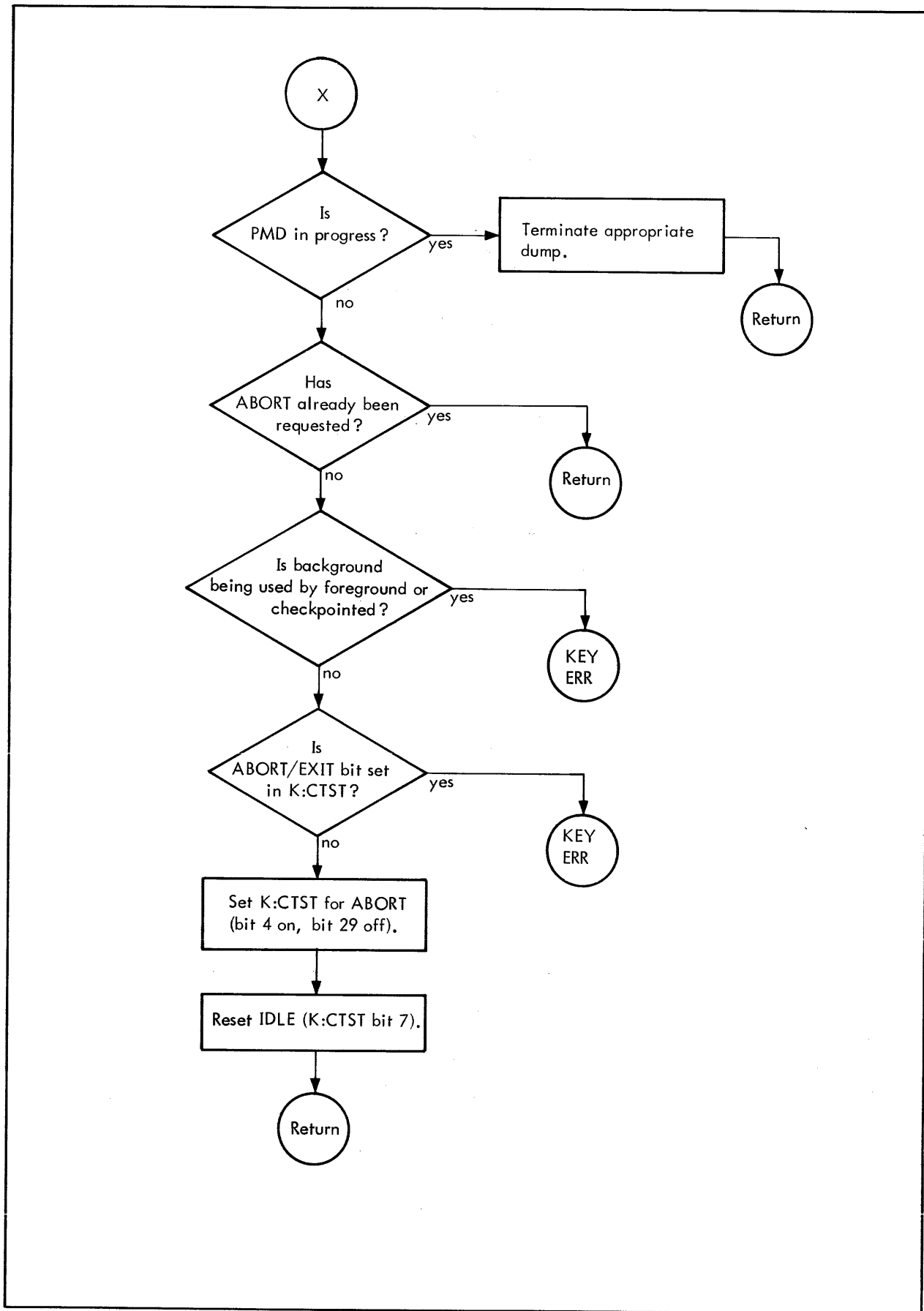


Figure 7. Operator Key-In Flow, "X"

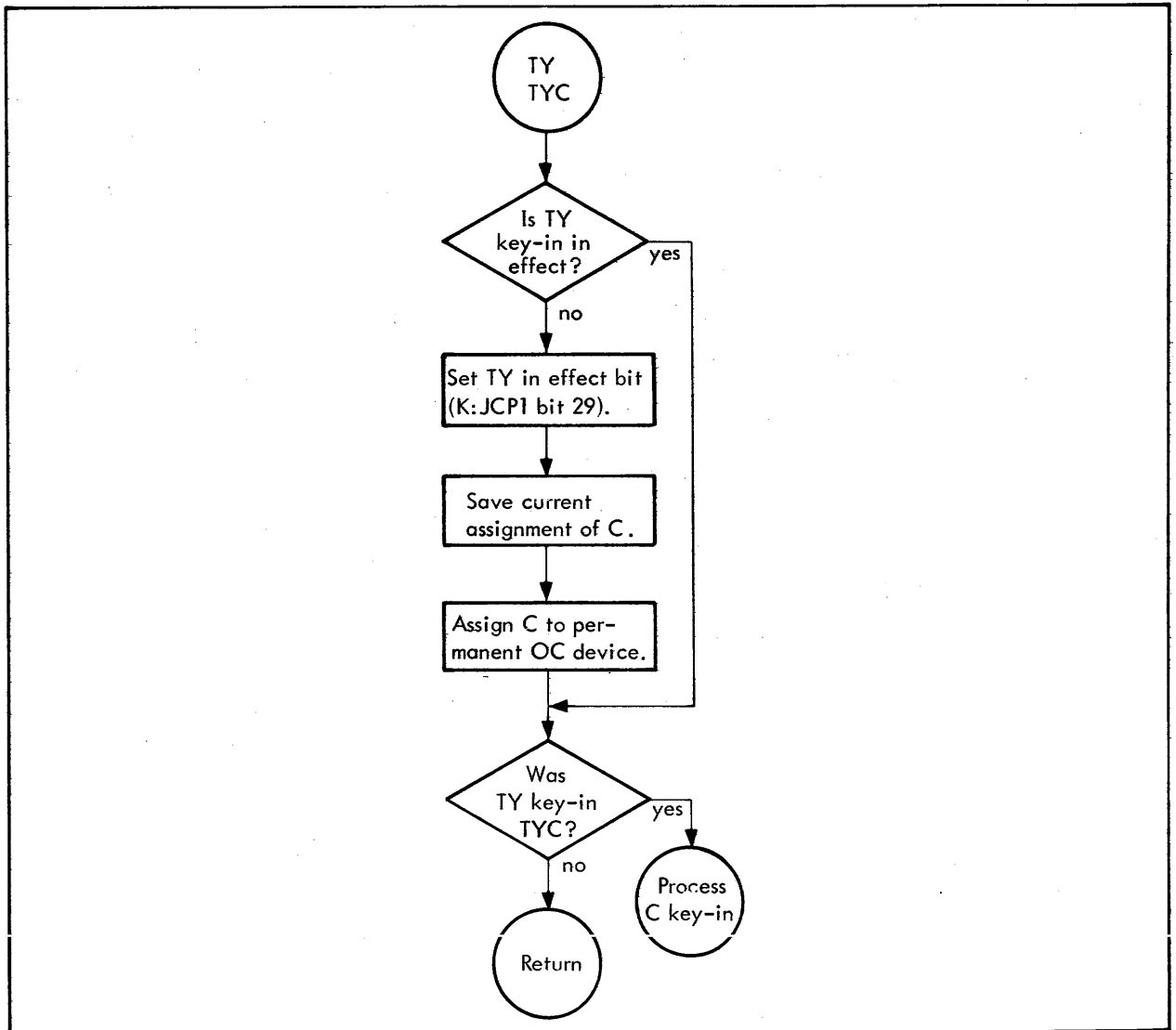


Figure 8. Operator Key-In Flow, "TY" and "TYC"

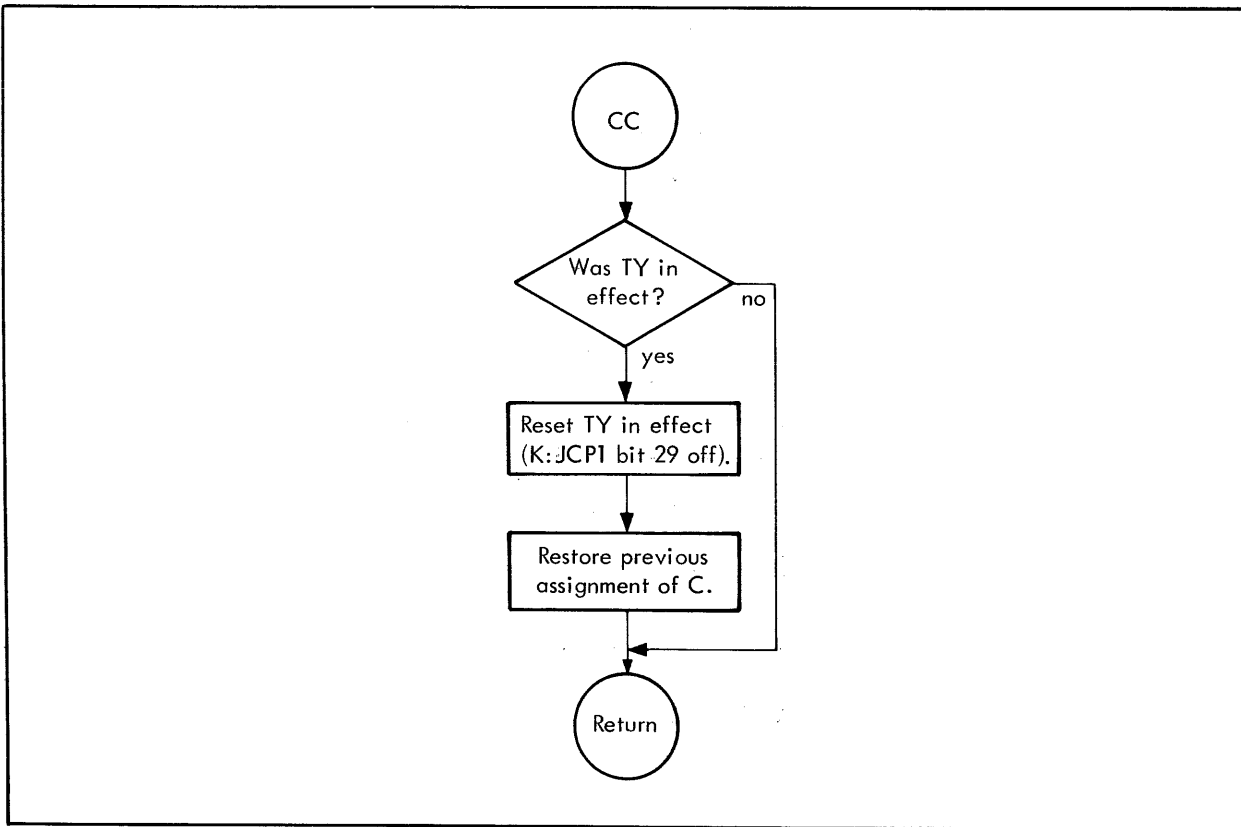


Figure 9. Operator Key-In Flow, "CC"

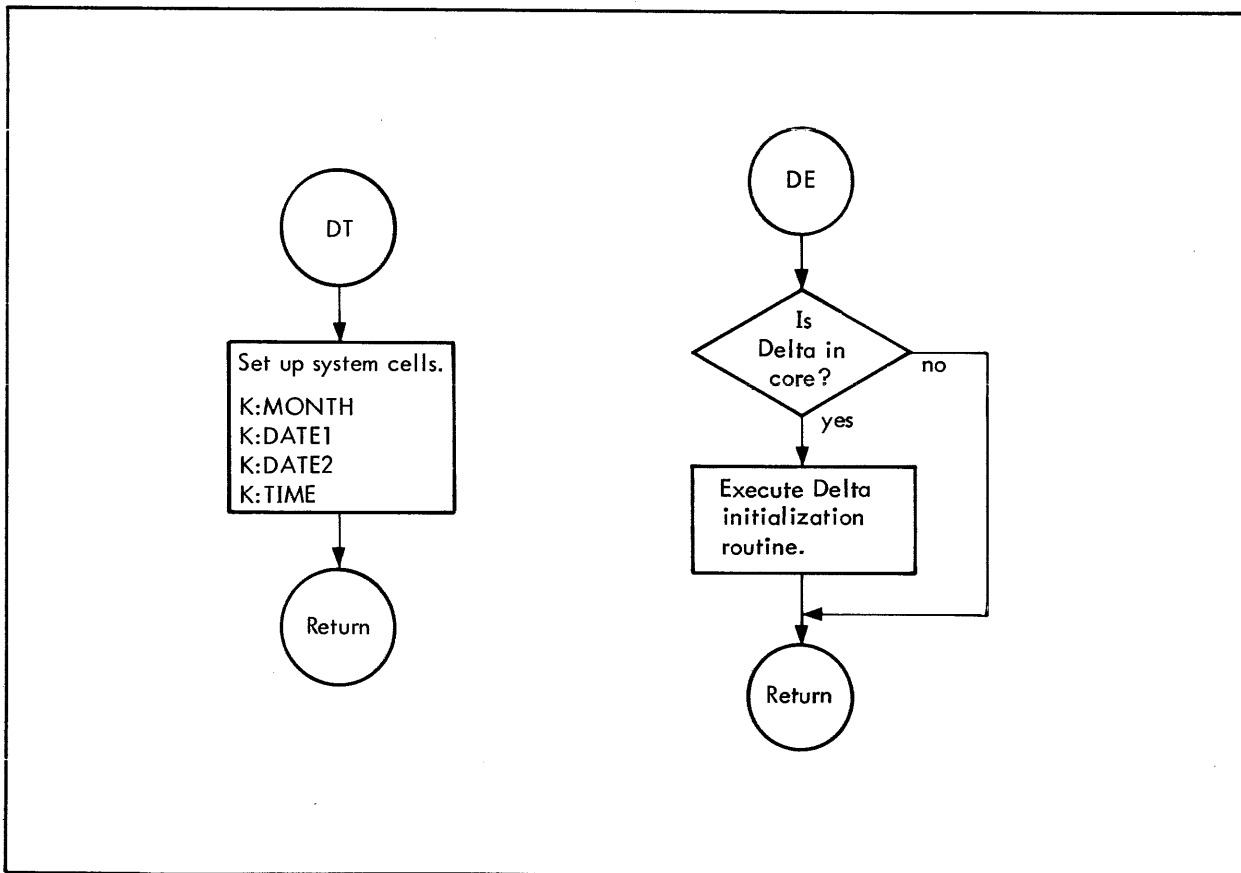


Figure 10. Operator Key-In Flow, "DT" and "DE"

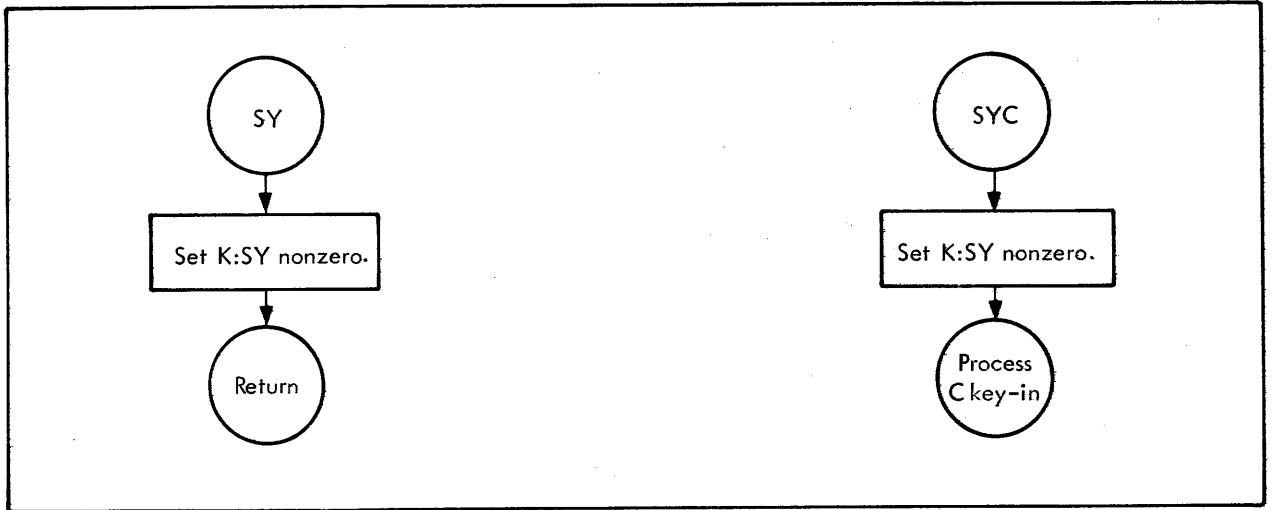


Figure 11. Operator Key-In Flow, "SY" and "SYC"

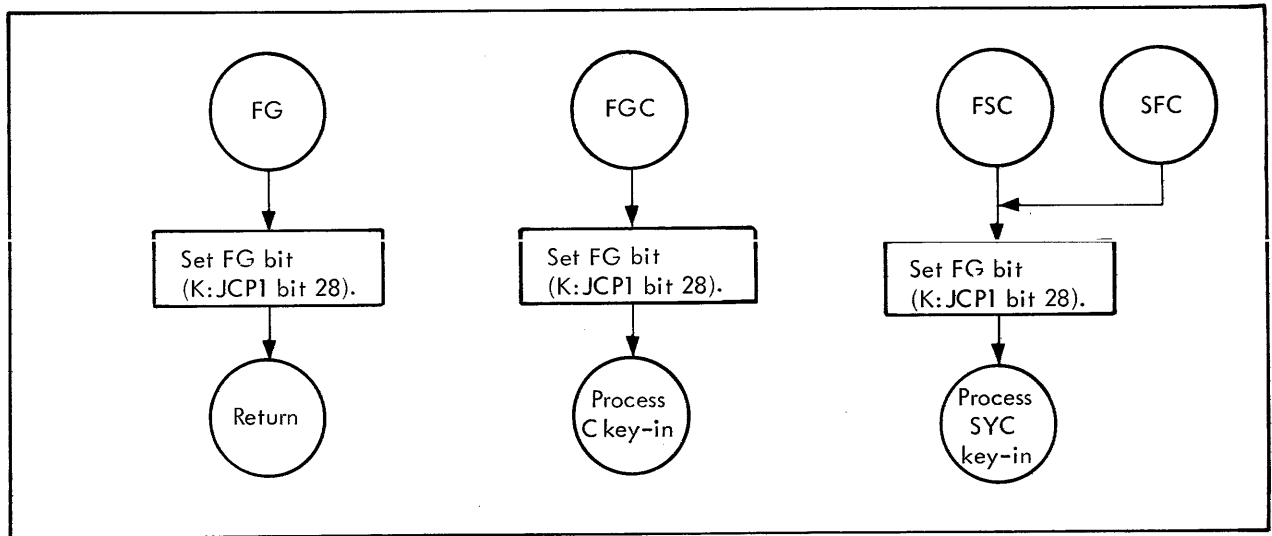


Figure 12. Operator Key-In Flow, "FG", "FGC", "FSC", and "SFC"

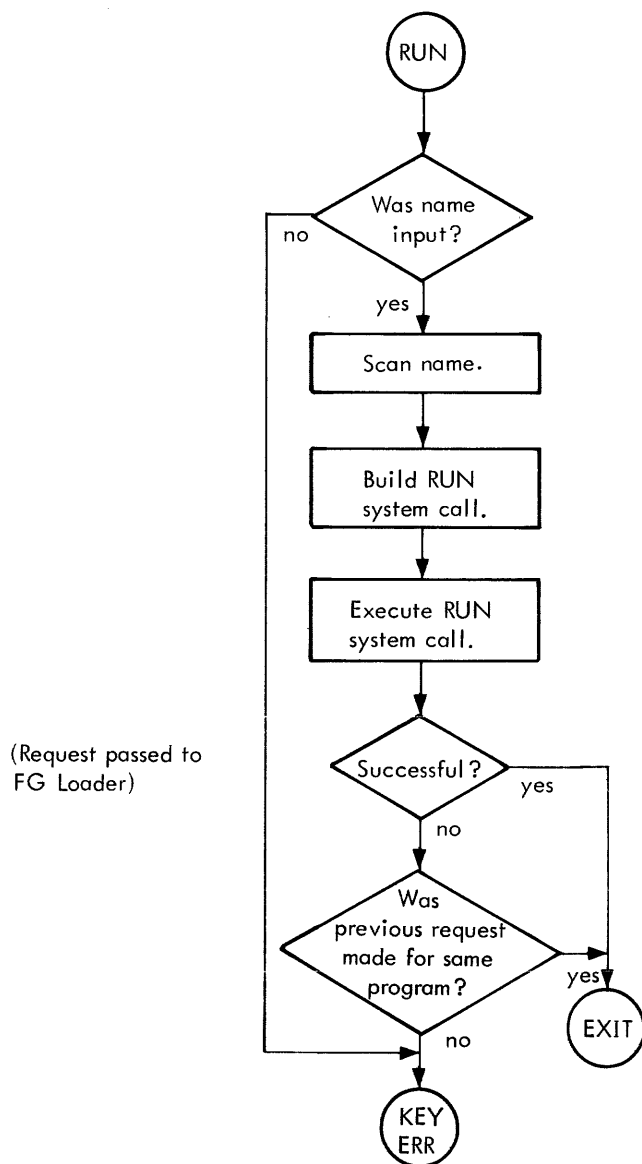


Figure 13. Operator Key-In Flow, "RUN"

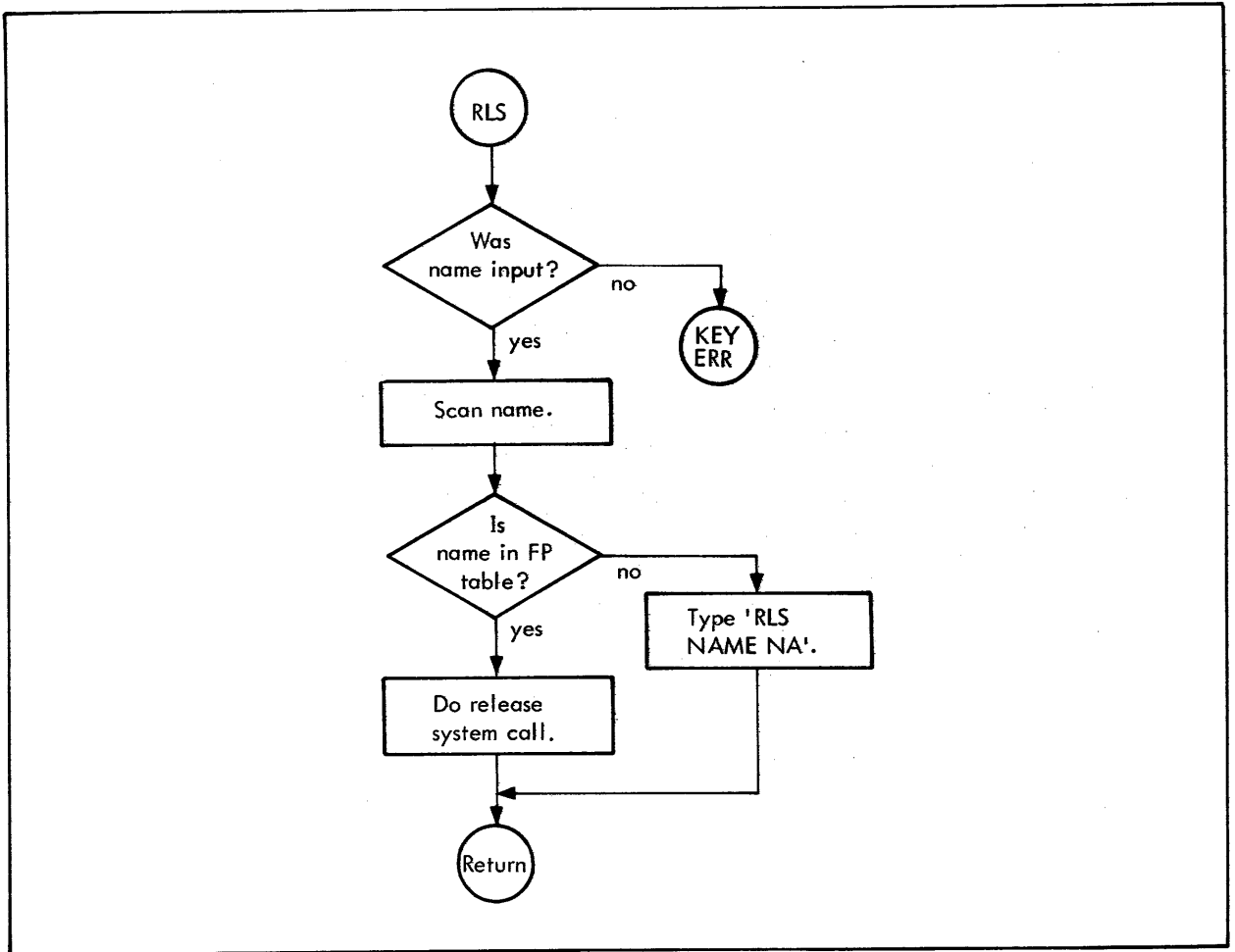


Figure 14. Operator Key-In Flow, "RLS"

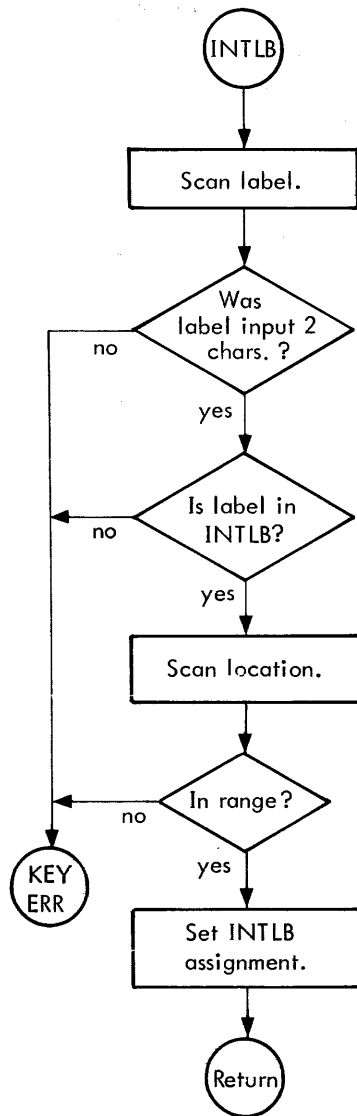


Figure 15. Operator Key-In Flow, "INTLB"

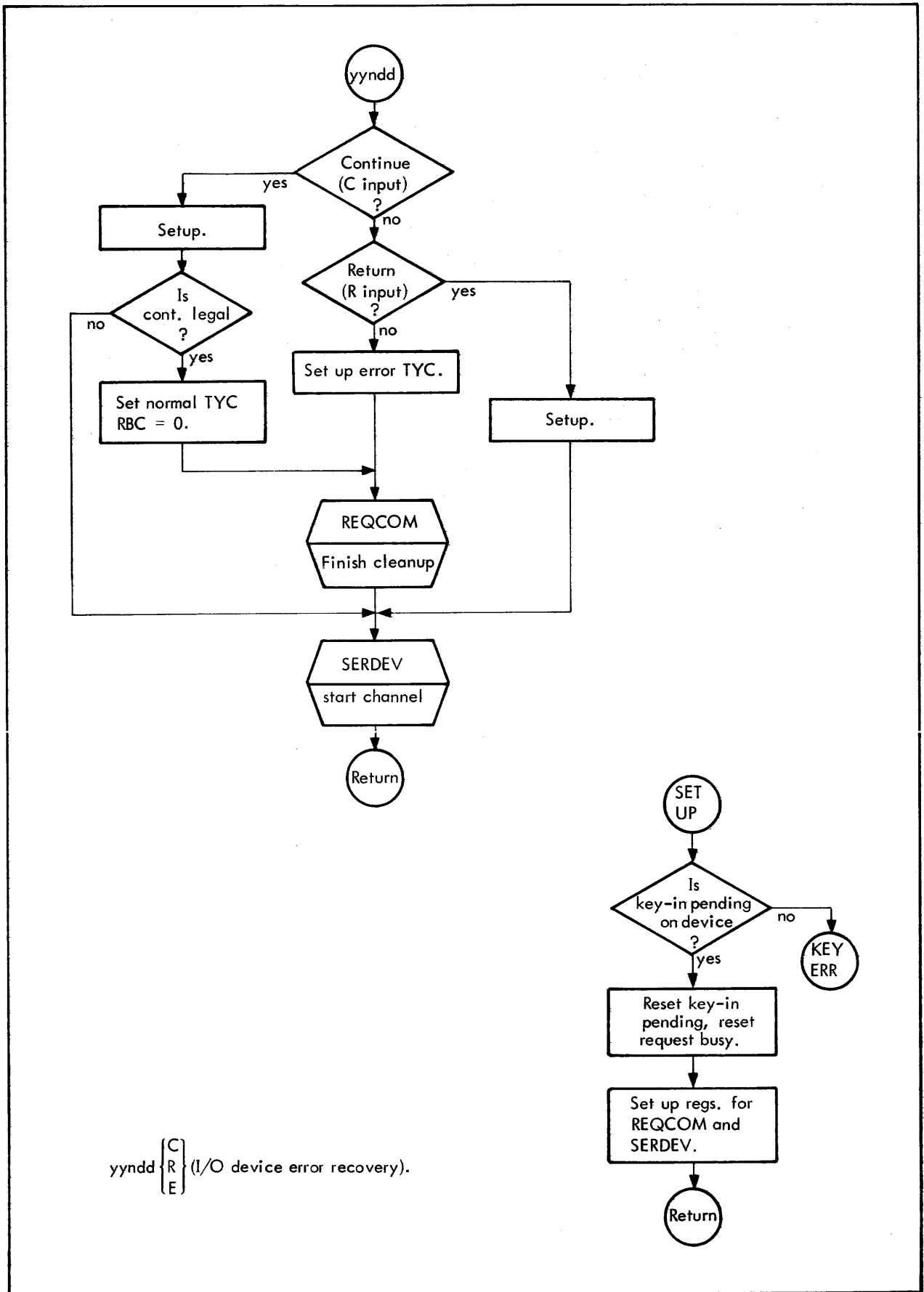


Figure 16. Operator Key-In Flow, "yyndd"

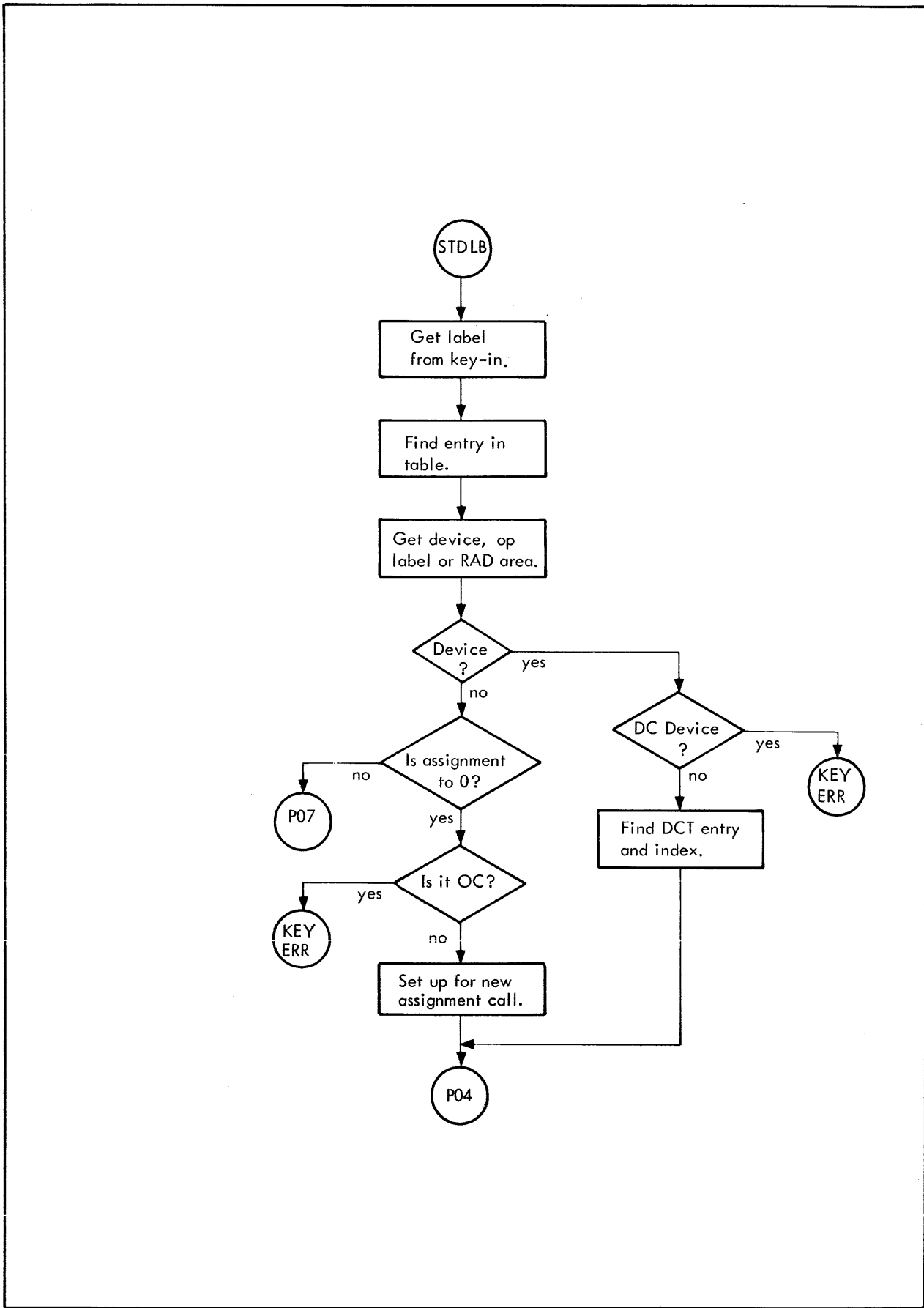


Figure 17. Operator Key-In Flow, "STDLB"

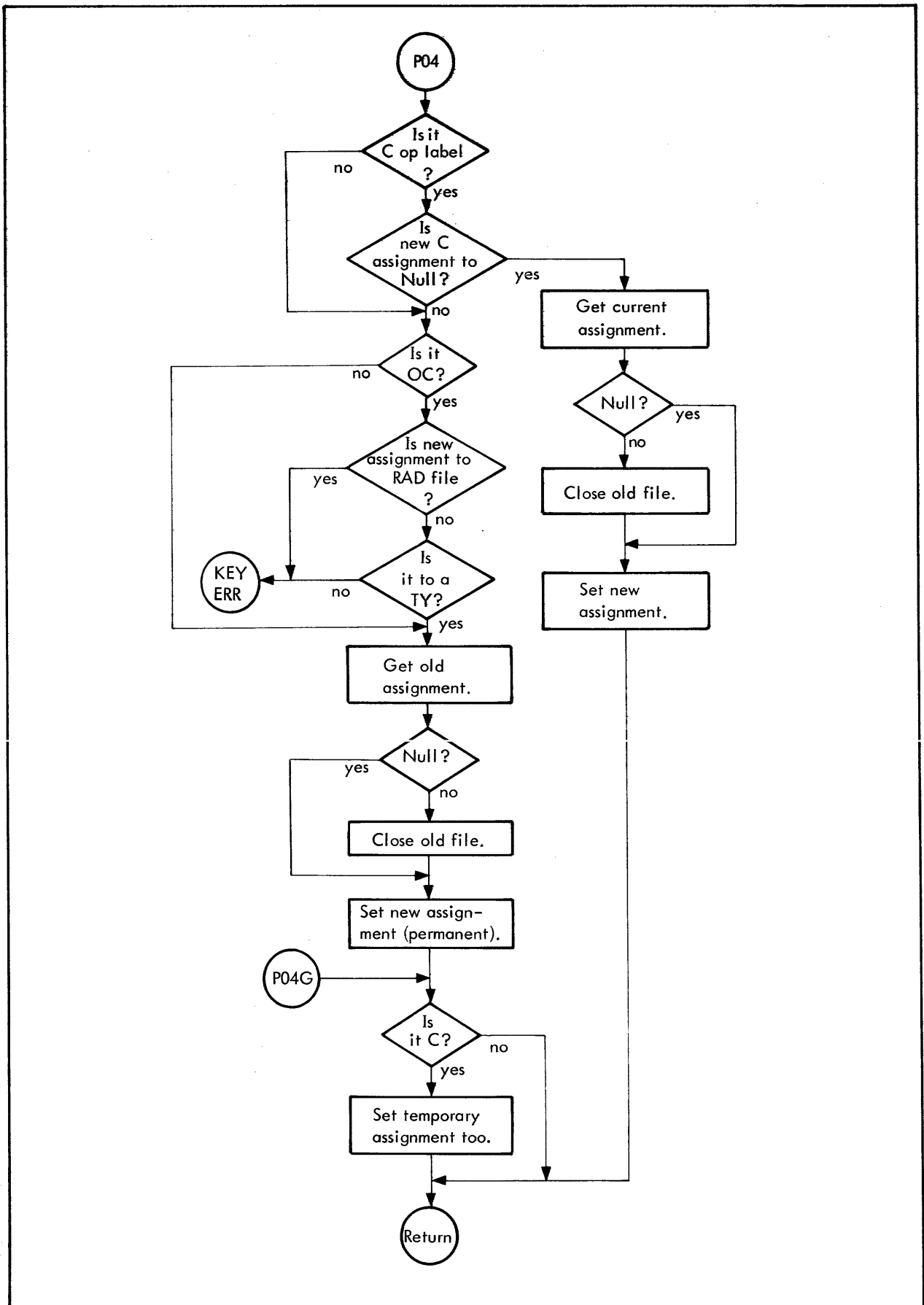


Figure 17. Operator Key-In Flow, "STDLB" (cont.)

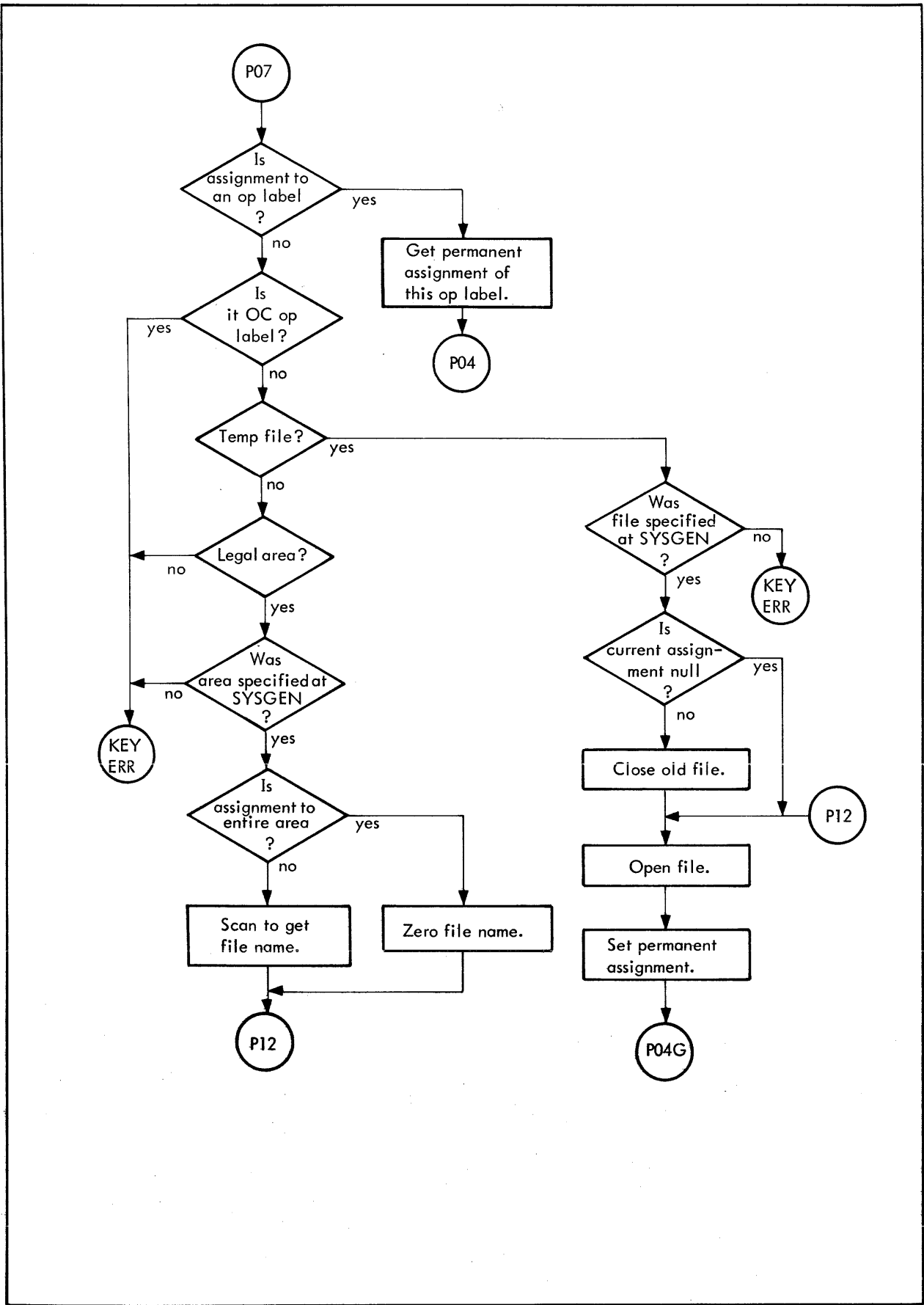


Figure 17. Operator Key-In Flow, "STDLB" (cont.)

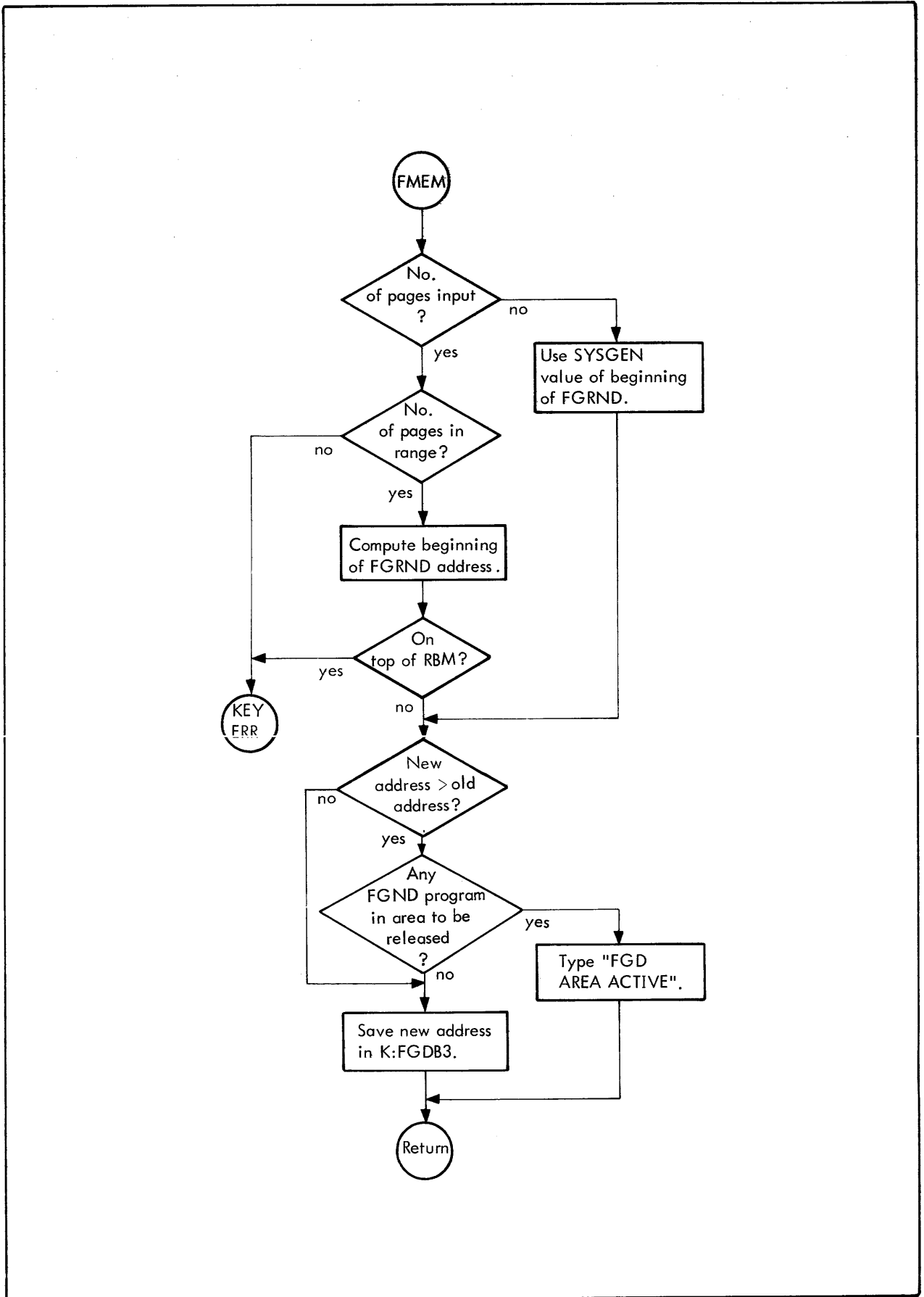


Figure 18. Operator Key-In Flow, "FMEM"

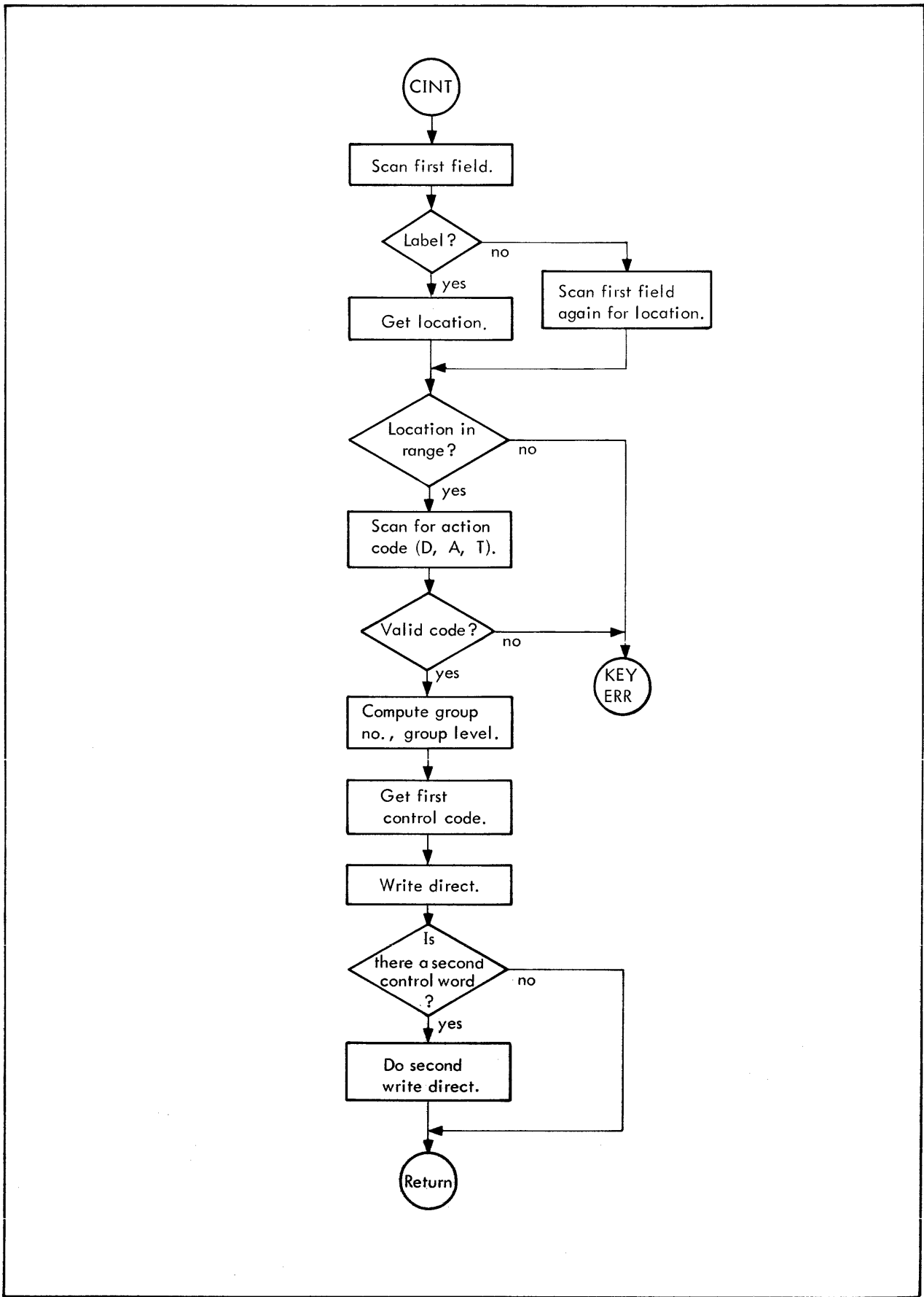


Figure 19. Operator Key-In Flow, "CINT"

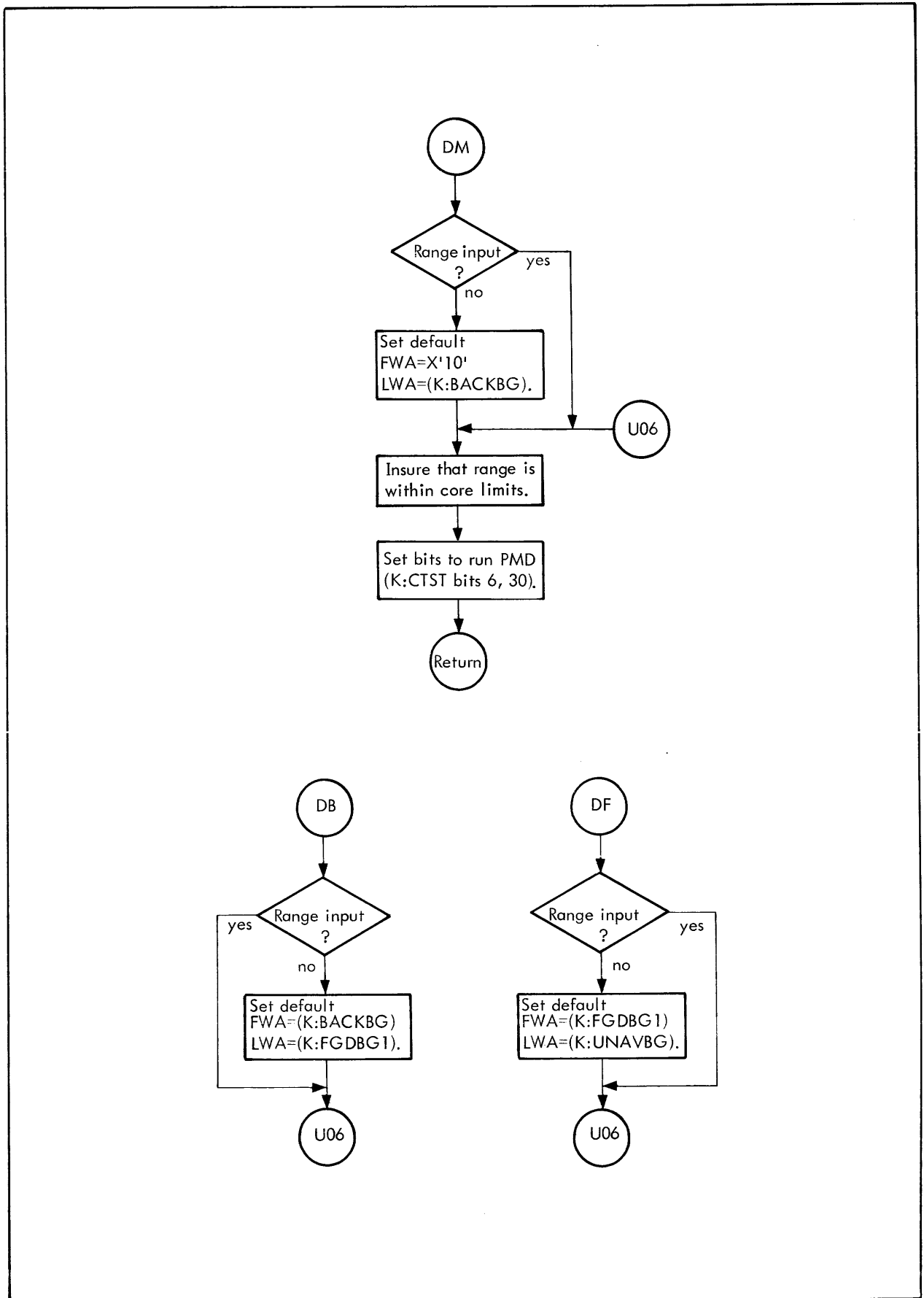


Figure 20. Operator Key-In Flow, "DM", "BB", and "DF"

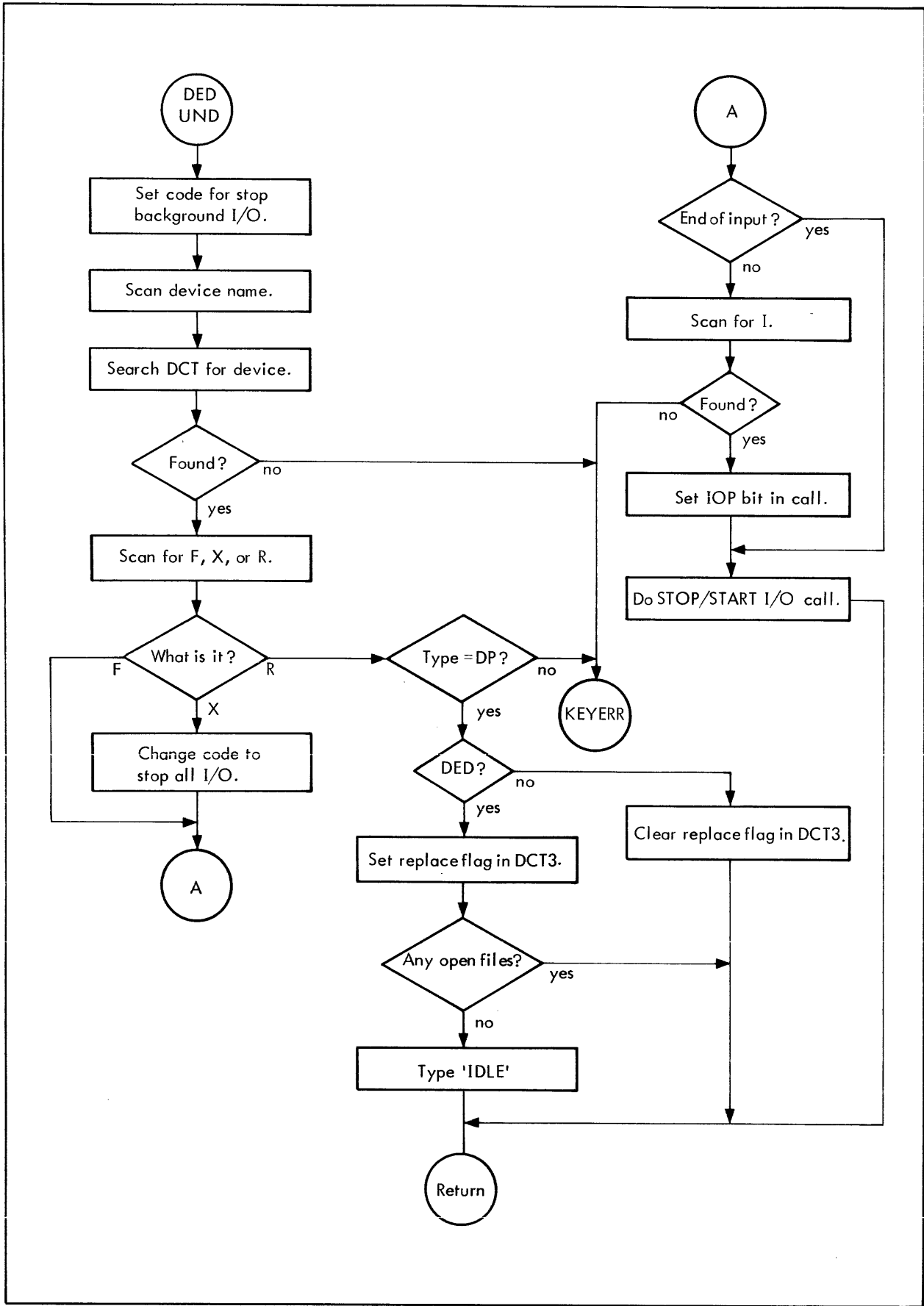


Figure 21. Operator Key-In Flow, "DED" and "UND"

Foreground Release (FGL1)

The primary purpose of this overlay is to release any Public Libraries or foreground programs that are marked "to be released" in FP5 (bit 3). In addition, it tests if the "queue" bit (bit 6 in FP5) is set for any entries; if so, it also sets the "load to be performed" bit (bit 0 in FP5). This overlay also handles the output of operator messages for both FGL1 and FGL2. After performing these functions it returns to the resident Control Task such that FGL2 is loaded and called.

For Public Libraries, a test is performed to ensure that the Public Library is not also in use by the foreground. If not, the FP entry is deleted. For regular foreground programs, the FP entry is deleted, all interrupts in use by the program (identified in the INTTAB) are disarmed and disconnected (MTW, 0 0 is set in interrupt location), and all DCBs are closed. If any Public Libraries were used by the released foreground program, they are released if this is the only foreground program using them. The message

```
!!UNABLE TO CLOSE DCB XXXXXXXX
```

is output to OC if the named DCB (XXXXXXX) cannot be closed. The message

```
!!PROG XXXXXXXX RELEASED
```

is output to OC after the named program is released.

Foreground Loader (FGL2)

Run queuing is an optional feature in RBM. If the feature was assembled into RBM, both "load to be performed" and "queued" bits in FP5 will be set after a RUN request has been made for a particular program. If the feature is not assembled into RBM, only the "load to be performed" bit is set. In the former case, FP4 also contains a priority field in bits 0-14. FGL2 uses this field to determine which queued program it should attempt to load first. In the latter case, FGL2 makes a search only on FP5 to find a program to be loaded. At the conclusion of the attempt to load and initialize all such programs, control is returned to the resident Control Task.

Tests are performed to ensure that the space required for the program is not already in use. If some of the required space is in use by the background, the bit is set (K:CTST bit 0) to cause a checkpoint of the background. All background I/O is then stopped and control is returned to the resident Control Task. At conclusion of the checkpoint, control returns to the Foreground Loader for loading and initialization of the program.

Tests are performed to ensure that the space required for the program is not already in use. If some of the required space is in use by another foreground program, an alarm is printed on OC; furthermore, if the queued bit is not set, the entry will be removed from the FP table. If the queued bit is set, an attempt to load the program will again be made after the next release of any foreground program.

Initialization is accomplished by transferring control to the start address of the program. At conclusion of initialization, the program must perform an EXIT system call. The EXIT processor will recognize that a foreground program initialization was in progress and will return control to the Control Task without performing the other usual functions. As each foreground program is successfully loaded, the message

```
!!LOADED PROG
```

is output to OC and LO, followed by the program name of the next line.

The following error alarms are output by the Foreground Loader:

```
!!FPT FULL, CAN'T LOAD XXXXXXXX
```

```
!!CORE USED, CAN'T LOAD XXXXXXXX
```

```
!!I/O ERR, CAN'T LOAD XXXXXXXX
```

!!NONEXIST. CAN'T LOAD XXXXXXXX

After loading all or any specified foreground programs, a test is performed to determine whether the background may be restarted if it was checkpointed. A restart could occur when one or more programs are released in FGL1 and no program was loaded in FGL2 that used any background space. If the background is to be restarted, the bit is set (K:CTST bit 2) to restart.

Background Loader (BKL1)

The Background Loader controls the loading of all background programs including the JCP, system processors (RAD Editor, language processors, etc.), and user background programs.

The Background Loader tests to determine that the background is to be loaded (K:CTST bits 9 or 10 are on).

If a FMEM key-in has occurred since the last execution of the Background Loader, the background/foreground boundary is moved by setting the proper system cell (K:FGDBG1) and setting the Write locks. This change is made only if no foreground programs are running in any of the core area to be allocated to the background. Should a foreground program be running in this space, the message

!!FGD STILL ACTIVE

is output on OC.

The background program header is read to determine the amount of core memory required. If sufficient core is not available, the message

!!NOT ENUF BCKG SPACE

is output on OC.

If the JCP is to be loaded, the load module is read into core, the Control Task TCB is modified so that the JCP is entered upon exit from the Control Task, and control is returned to the resident Control Task.

If a background program (except the JCP) is to be loaded, the program root is read into core. This may be done with several READ requests if the root is longer than 8191 words. The user's M:SL DCB is then set up if the program is segmented. The user DCBs assignments are made using the data from !ASSIGN control commands (if any) processed by the JCP since the last job step. If any such assignment cannot be performed, the message

!!UNABLE TO DO ASSIGN

is output on OC and LO.

If a !POOL control command was input, the specified number of buffers is determined. If no !POOL control command was input, the number of DCBs assigned to a blocked or compressed file is determined. This number (the number of desired blocking buffers) is passed to the second overlay (BKL2) of the Background Loader. A maximum of two blocking buffers is allocated for all DCBs assigned to scratch files (X1-X9).

Background Loader (BKL2)

This overlay allocates the background blocking buffers, sets up the loading of any needed Public Libraries, moves the control command (!RADEDIT, !PROGRAM, etc.) to the high end of available background, and sets the Control Task TCB so that the background program is entered upon exit from the Control Task.

If the user specified a number of blocking buffers via the !POOL command and there is insufficient space, the message

!!NOT ENUF BCKG SPACE

is output to OC and control is returned to the resident Control Task. If no !POOL control command was input, the desired number are allocated if sufficient space is available. If there is not sufficient space but space for at least one blocking buffer is available, the maximum possible number is allocated.

A RUN system call is built and executed for background programs that use Public Libraries.

Checkpoint/Restart (CKPT)

This overlay performs both the Checkpoint and Restart functions. Checkpoint is accomplished by waiting for outstanding background I/O requests to run to completion and then writing the entire background portion of core to the CK area of the RAD. When the background has been successfully written to the RAD, the message

!!BCKG CKPT

is output on OC. At conclusion of the checkpoint, the background portion of memory is given to the foreground by setting the boundary pointers K:FGDBG1 and K:BCKEND and setting the Write locks appropriately.

The following self-explanatory messages may be output during checkpoint:

!!CKPT WAITING FOR BCKG I/O RUNDOWN

!!BCKG IN USE BY FGD

!!CK AREA TOO SMALL

!!!/O ERR ON CKPT

Restart is accomplished by resetting the boundary pointers K:FDGBG1 and K:BCKEND, and by resetting the Write locks to their precheckpoint settings. The message

!!BCKG RESTART

is output on the OC device and the control bits indicating that the background is checkpointed are reset (K:JCP1 bits 2, 3). Control is then transferred to the resident Control Task, and when all specified subtasks are completed, the Control Task will exit to the proper point in the background.

Abort/Exit

This overlay performs the background Exit and Abort functions. Exit is accomplished by waiting for background I/O rundown, closing all DCBs, and setting the proper indicators so that the next program will be loaded into the background by the Background Loader. If the Exit is from the JCP and no !JOB control command has been read, it is assumed that a !FIN caused the Exit request. The idle indicator (K:CTST bit 7) is set, the Exit/Abort indicator (K:CTST bits 4, 29) are reset, and control is returned to the resident Control Task.

If the Exit is from a background program other than the JCP after I/O rundown and closing the DCBs, the indicators are set to load and execute the JCP (K:CTST bits 8, 9). Control is then returned to the resident Control Task. While waiting for I/O rundown, control is also returned to the resident Control Task whenever the I/O rundown test fails, to permit other higher priority subtasks to be performed in the interim.

Background Abort requests may originate from operator "X" key-in or from a system function call. If the request was from a key-in, a test is performed to determine whether the background was executing when the interrupt occurred. If so, the Abort must be postponed until the background exits from the Monitor. This is accomplished by signaling the system CAL Exit routine that an Abort request was made. The Abort indicators are then reset and control is returned to the resident Control Task. Control eventually returns to the background and when the background exits the Monitor, the CAL Exit routine sets the proper indicators to cause a background Abort and trigger the Control Task. At this execution of the Control Task, conditions 1 and 2 will not be true and the Abort proceeds as it would the first time.

A test for background I/O rundown is made. Any active background requests for devices that are manual and those waiting for operator key-in are dequeued and cleared from the system tables. Eventually, I/O must run down with no further action by the operator. The appropriate message

!!JOB ABORTED AT {XXXXXX}
 {LIMIT }

where XXXXX (a hex location) is then output on OC and LO.

If a PMD (Postmortem Dump) was requested, K:CTST bit 6 is set. K:JCP1 bit 5 is set to cause control commands to be skipped until the next !JOB command is encountered. K:CTST bits 3 and 29 are reset and control is returned to the resident Control Task.

Postmortem Dump (PMD)

This overlay performs core dumps. Any Dump key-in requests in effect at entry are performed first, and when these are exhausted, the background PMD requests are satisfied (maximum of four ranges).

The dump format is either hexadecimal or optionally both hexadecimal and EBCDIC, with the registers being retrieved from their storage area and dumped as locations 0 through X'F'. Subroutines are included in the overlay that perform hexadecimal to EBCDIC conversion and move bytes into the print image.

After queuing each print line, control is returned to the resident Control Task to enable other subtasks to be performed without waiting for total completion of the dump. The resident Control Task returns control to PMD after each line is printed.

3. I/O HANDLING METHODS

Channel Concept

A "channel" is defined as the highest order data path connected to one or more devices, only one of which may be transmitting data (to or from CPU memory) at any given time.

Thus, a magnetic tape controller connected to an MIOP is a channel but one connected to an SIOP is not, since in this case, the SIOP itself fits the definition. Other examples of channels are a card reader on an MIOP, a keyboard/printer on an MIOP, or a RAD controller on an MIOP.

Input/output requests made on the system will be queued by channel to facilitate starting a new request on the channel when the previous one has completed. The single exception to this rule is the "off-line" type of operation, such as the rewinding of magnetic tape or the arm movement of certain moving arm devices. For this type of operation, an attempt is always made to also start a data transfer operation to keep the channel busy if possible.

Handling Devices

The RBM system offers the capability of multiple-step operations by providing an interrupt-to-interrupt mode in addition to the standard single interrupt mode.

Single Interrupt Mode

On the lowest level the I/O handler is supplied a function code and device type. These coordinates are used to access information from tables used by the handler to construct the list of command doublewords necessary to perform the indicated operation. Included will be a dummy (nonexecuted) command containing information pertinent to device identification, recovery procedure, and follow-on operations (see below).

Interrupt-to-Interrupt Mode

A function code for a follow-on operation may be included in the dummy command. This causes the request to be reactivated and resume its normal position in the channel queue, but with a different operation to be performed. It will be started by the scheduler in the normal manner as if it were any other request in the queue. The process may be cascaded indefinitely.

Error recovery may be specified at any point within a series of follow-on operations and will be itself treated by the system as a type of follow-on operation. It should be noted that follow-ons may be intermixed with other operations on the same channel or even on the same device if the situation warrants. Thus, a series of recovery tries on a RAD may be interrupted to honor higher priority requests, or on a tape for higher priority requests on other drives (but not on the same drive).

Note that only one of the follow-on operations may transfer data unless all other parameters are to remain the same (buffer address, byte count, seek address, etc.).

System Tables

Information pertaining to requests, devices, and channels is maintained in a series of parallel tables produced at System Generation time. A definition of these tables is presented here as reference for the remainder of this manual. The first entry (index=0) in each table is reserved for special use by the system. See Chapter 10 for a more complete description of these tables.

IOQ (Request Information)

These tables contain all information necessary to perform an input/output operation on a device. When a request is made on the system, a queue entry is built that completely describes the request. The entry is then linked into the channel queue below other requests of either higher or the same priority.

DCT (Device Control)

The device control tables contain fixed information about each system device (unit level) and variable information about the operation currently being performed on the device.

CIT (Channel Information)

These tables are used primarily to define the "head" and "tail" of entries that represent the queue for given channel at any time. A channel queue may have more than one entry active at any time (e.g., several tapes rewinding while another entry reads or writes).

Handler Tables

Associated with each handler are two tables: the Device Offset Table (DOT), and the Command List Pointer Table (CLST).

The DOT table is a word table that begins on a doubleword boundary and contains:

- Byte 0 A byte offset from the beginning of the DOT table to the corresponding CLST entry.
- Byte 1 The time-out value, which is an integer that represents the number of five-second intervals that are allowed to pass between the SIO and the I/O interrupt before the interrupt is considered lost. The value X'FF' indicates the operation should not be timed out.
- Byte 2 The retry function code. This is the function code to be used for automatic error recovery.
- Byte 3 The continuation function code. This is the function code to be used for multiple interrupt requests. For example, a forward space record on magnetic tape can be performed n times by the basic I/O using the same queued request. Zero is used for no continuation.

The function code is used as the index to reference this table.

The CLST table is a byte table containing the doubleword displacement from the beginning of the corresponding DOT table to the appropriate skeletal command doubleword.

The general method for constructing the command doublewords for an I/O request is to access the DOT table using the function code as index, and then find the skeletal command doubleword offset by using the contents of byte 0 of the DOT entry as index to the CLST table. The skeletal command doubleword has the form

| | | | |
|-------|-----|---|----|
| Order | X | | |
| Flags | 0 | Y | Z |
| 0 | 7 8 | | 31 |

where

- Y = 0 if the command is complete and to be used as is. This implies X is the address and Z is the byte count.
- Y = 1 if a seek address contained in IOQ12 is to be placed in the first word. In this case, the value of X is irrelevant.
- Y = 2 if a regular data transfer is to be performed. In this case, the buffer address is taken from IOQ8 and placed in the first word, and the byte count is taken from IOQ9 and placed in the second word (byte 1).
- Y = 3 if the request represents an I/O error message. This will cause the proper N/L!!yyndd to be chained to the pointed message.
- Y = 4 if a special handler function is to be performed. In this case, X is the address of the entry to the function.

When the building of the command doubleword is completed, a test is performed for command-chaining (command doubleword flag field bits 0 or 2 are on). If another command doubleword is to be chained, it is accomplished by accessing the next successive entry in the CLST table to find the offset of the skeletal command doubleword that is to be used to create the next command doubleword. This command doubleword is constructed in the same fashion as the first, and the process may continue to the limits imposed by the size of the command list area allocated at SYSGEN.

Separation of Priorities and Control Task

All input/output functions are controlled with respect to time by a scheduler called "Service Device" (SERDEV). This routine is considered device-oriented by the calling program, but in reality, the routine takes the necessary steps to keep the applicable channel operating within the constraints of priority.

Every I/O request has a priority associated with it that is the priority of the task making the request. SERDEV causes "I/O Start" and "I/O Cleanup" operations under the following circumstances:

I/O Cleanup

1. When the device is marked as "waiting for cleanup".
2. When either the priority of the request to be cleaned up or the highest priority request in the queue for the channel is not lower than that of the currently active task.

I/O Start

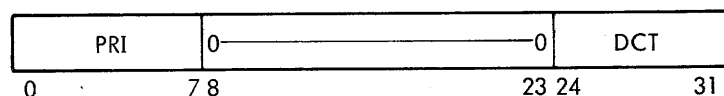
1. The device is marked "not busy".
2. The highest priority request in the queue is of priority not less than that of the currently active task.

Whenever a call to Service Device is made and no processing is performed because of priority considerations, the Control Task is triggered and the device code is entered in the Control Task I/O queue. When the Control Task becomes active it will initiate the deferred processing by calling Service Device.

Service Device is called by

```
BAL,R2 SERDEV
```

R1 must contain:



where

PRI is the priority of the currently active program. The lowest priority is FF and the highest is zero.

DCT is the device index.

When SERDEV has determined that some action can be performed, it will either process an interrupt (cleanup) for a completed operation or start a new operation. On a given device the cleanup must be performed before a new operation can be started. Thus, if a cleanup for a lower priority task is outstanding when a request for the current level is to be started, that cleanup will be performed at the higher level. Follow-on operations will also be processed at a higher level, if necessary, to free the device for higher priority requests.

Generally, Service Device is called whenever an event occurs that may change the status of a given device and/or channel. These events and the reasons for the change of status are as follows:

1. When an I/O request is queued: The device or channel may be stopped either because there were no previous entries in the queue at the time of the request or because of priority consideration. Service Device will make the necessary tests and, if necessary, either start or perform cleanup and then start the device.

2. When an I/O check is requested: Cleanup may be outstanding or the request may have been deferred for priority reasons. In any case, Service Device will perform the necessary action.
3. When an I/O interrupt occurs: The device involved will have cleanup outstanding that may or may not be processed for priority reasons. The channel may be driven if there are requests in the channel queue for the current level or any higher levels.
4. When the Control Task becomes active: The Control Task will call Service Device for each entry in its queue. In effect, it will ignore priority and all deferred operations will be performed.

The general flow of the SERDEV scheduler is illustrated in Figure 22.

INTSIM Routine

This routine simulates the occurrence of the I/O interrupt in cases where an I/O operation has timed out. The logic is illustrated in Figure 23.

CTTEST Routine

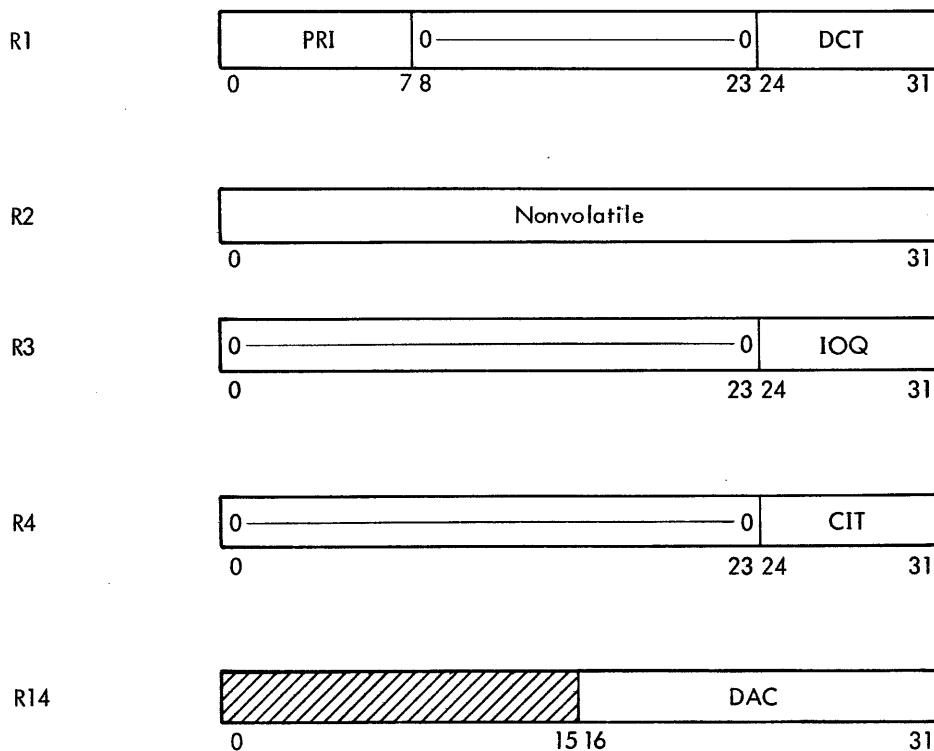
This routine tests to determine whether I/O processing must be deferred to the Control Task because of priority considerations. If so, the Control Task interrupt is triggered and return is +1. If not triggered, return is +2. The logic is illustrated in Figure 24.

Initiating I/O Requests

When Service Device has determined that a request may be initiated, it will make the call

```
BAL,R15 STARTIO
```

with registers set as follows:



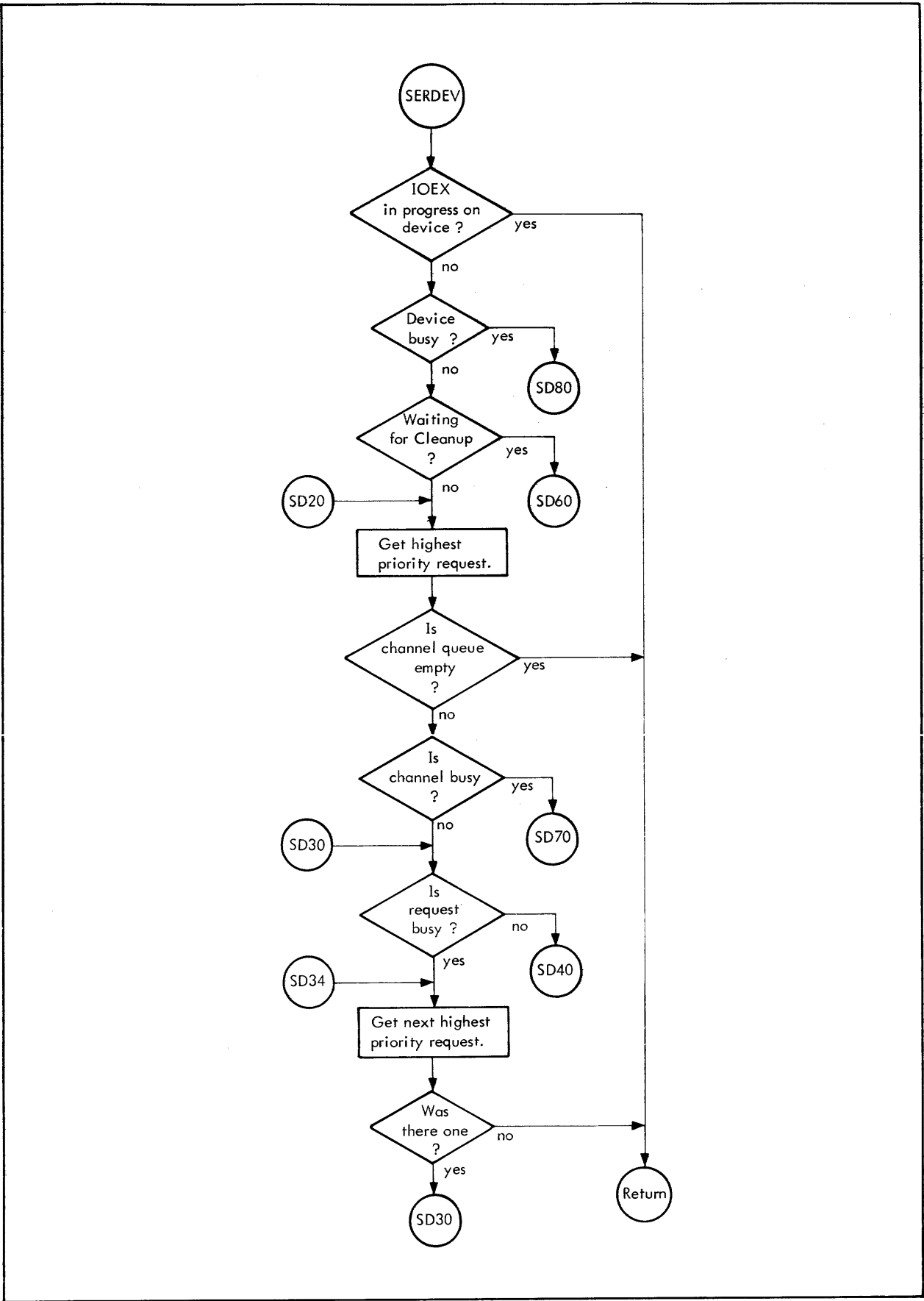


Figure 22. SERDEV Routine Flow

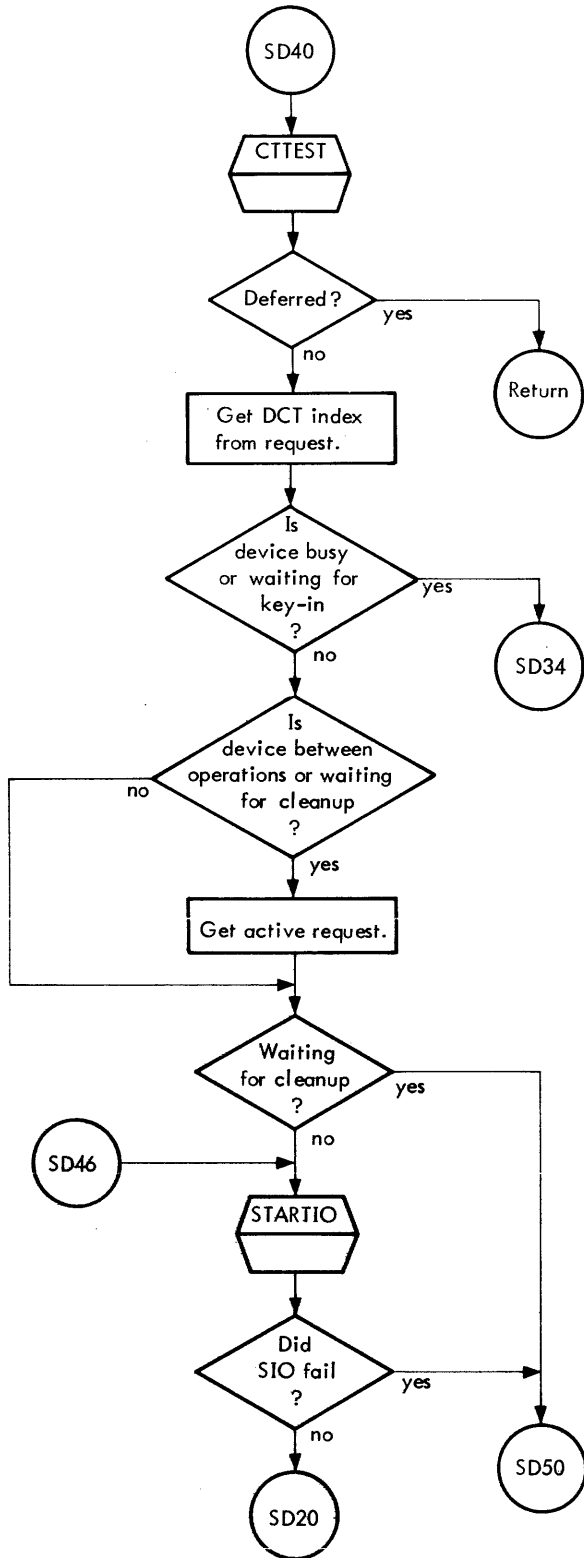


Figure 22. SERDEV Routine Flow (cont.)

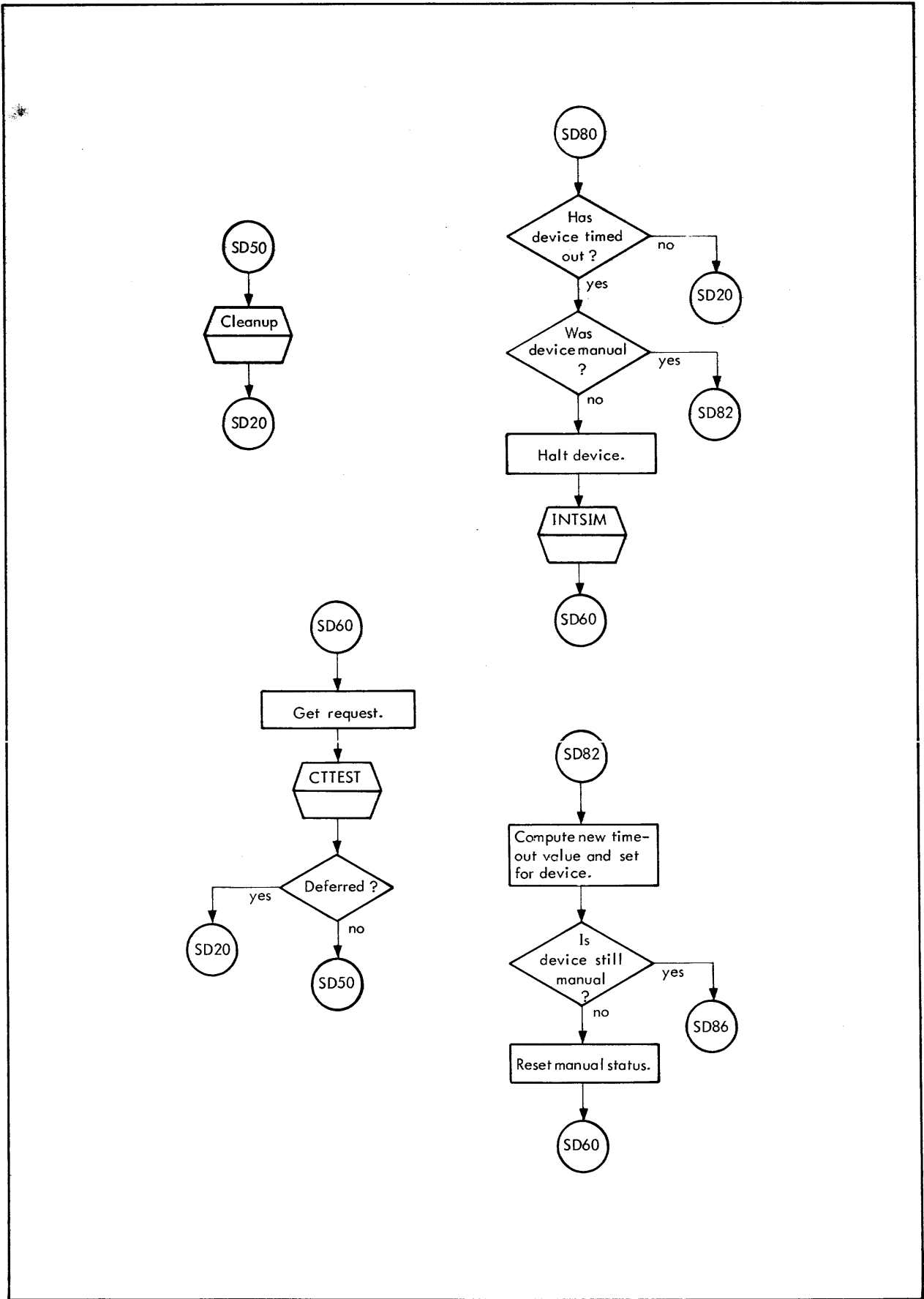


Figure 22. SERDEV Routine Flow (cont.)

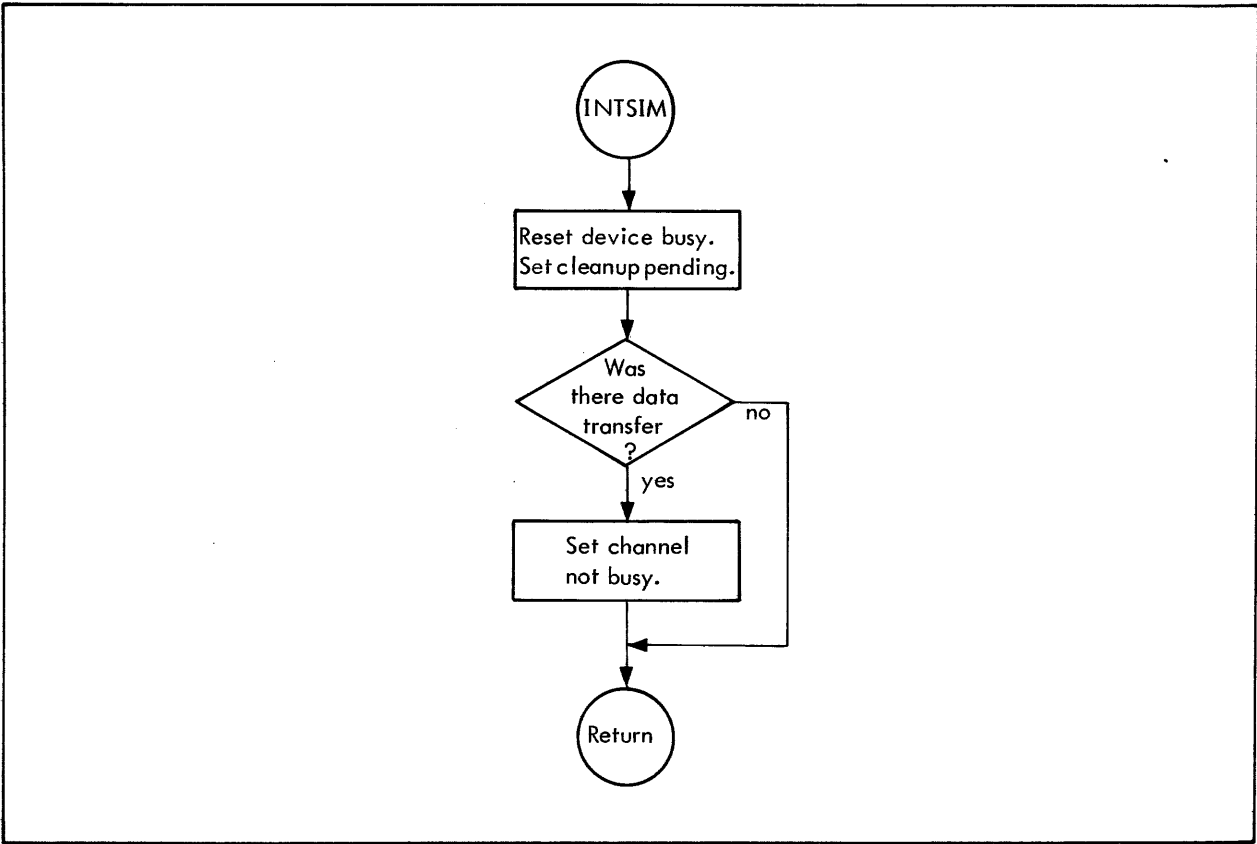


Figure 23. INTSIM Routine Flow

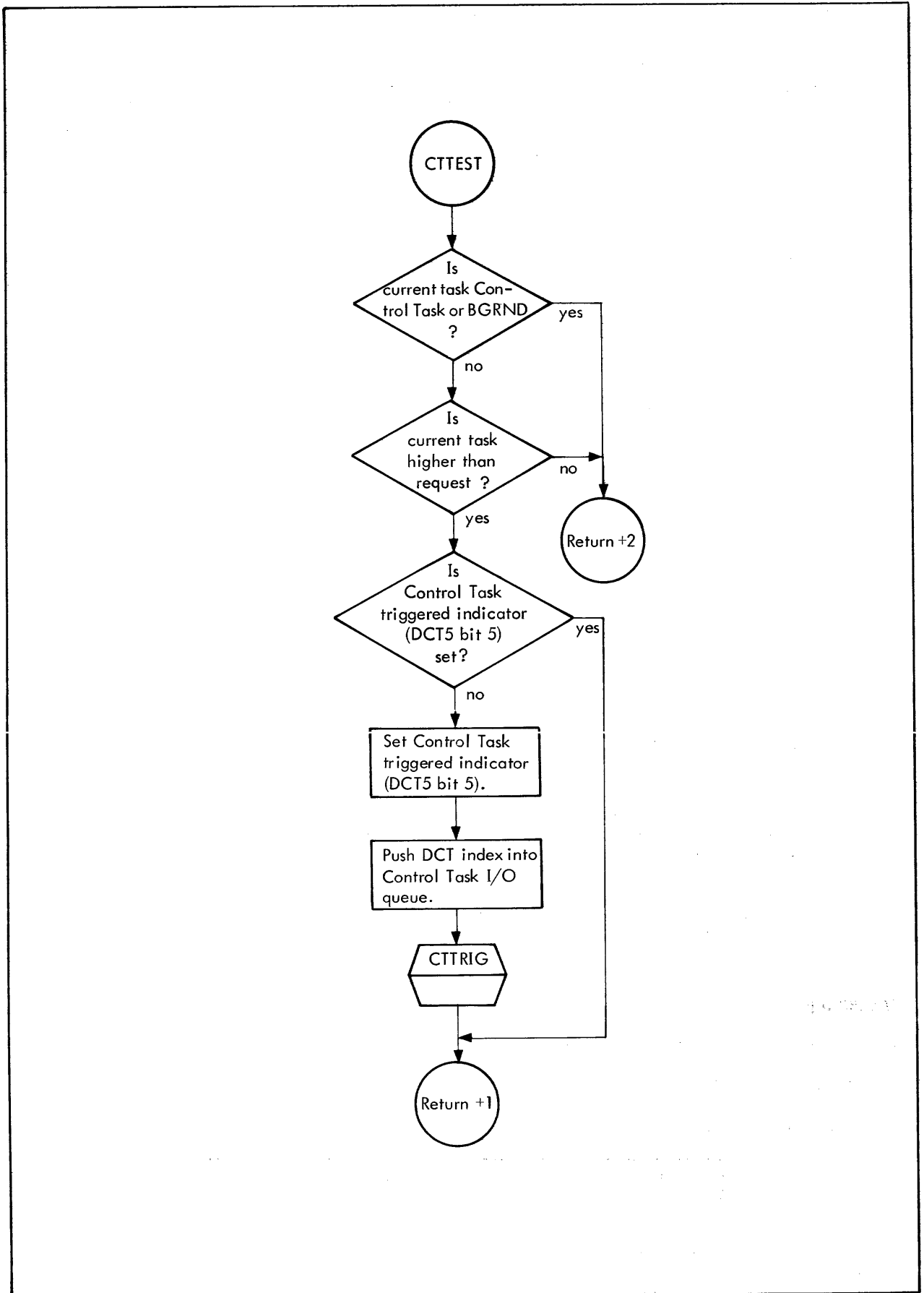
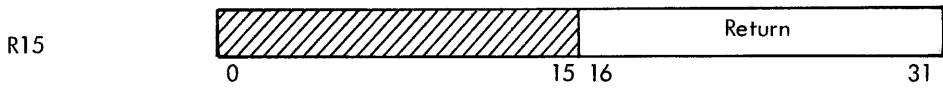


Figure 24. CTTEST Routine Flow



where

PRI is the operating priority.

DCT is the device index.

IOQ is the request index.

CIT is the channel index.

DAC is the Device Activity Counter.

STARTIO will branch to the start address in DCT8 to perform any functions peculiar to the device.

After the command list is created in temporary storage, all interrupts are disabled and a check is made for reentrance. This consists of testing the Device Activity Count (DCT 10) for change. The DAC is incremented when either an operation is started on a device or a cleanup is performed. If no reentrance is in evidence and the channel is not busy, an attempt is made to start the device.

If the SIO is rejected, the request will be treated as if an unrecoverable error condition was present. The cleanup will be performed and the type of complete will be set to indicate the SIO failure. If the SIO is accepted but the status information indicates that the device is in manual mode, an "empty" message is output and no response will be expected from the operator other than setting the device to "automatic". If the operation is timed out by the Watchdog Timer and the device is still manual, the message will be repeated. The flow of the STARTIO routine is illustrated in Figure 25.

I/O Interrupt Processing

When the I/O interrupt occurs, it is necessary to perform certain functions before clearing the level from the active state. The IOINT routine in the I/O interrupt processor is not reentrant. The device that caused the interrupt is determined by searching the hardware address in DCT1.

The device (and channel if applicable) is set to the "not busy" state, and the device is set waiting for cleanup. A check is made to determine whether or not an external interrupt should be triggered. At this point, the I/O interrupt level can be cleared. The system will then be at the priority level of the interrupted task and is subject to interrupt by higher level tasks.

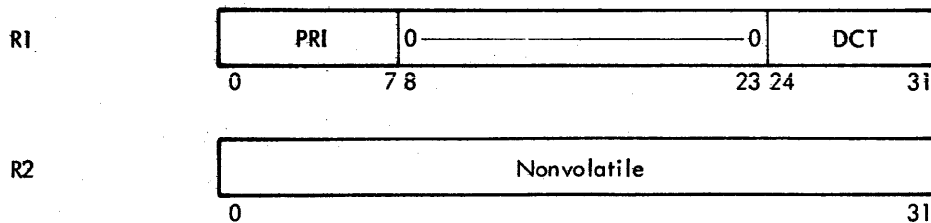
A call is made on the Service Device routine that decides if the cleanup for the just completed operation should be performed now or deferred to the Control Task. The interrupted task suffers only a minimal amount of overhead if the cleanup is deferred. The flow of the IOINT routine is illustrated in Figure 26.

I/O Cleanup

The call to process an interrupt for an I/O request is

BAL,R15 CLEANUP

with registers set as follows:



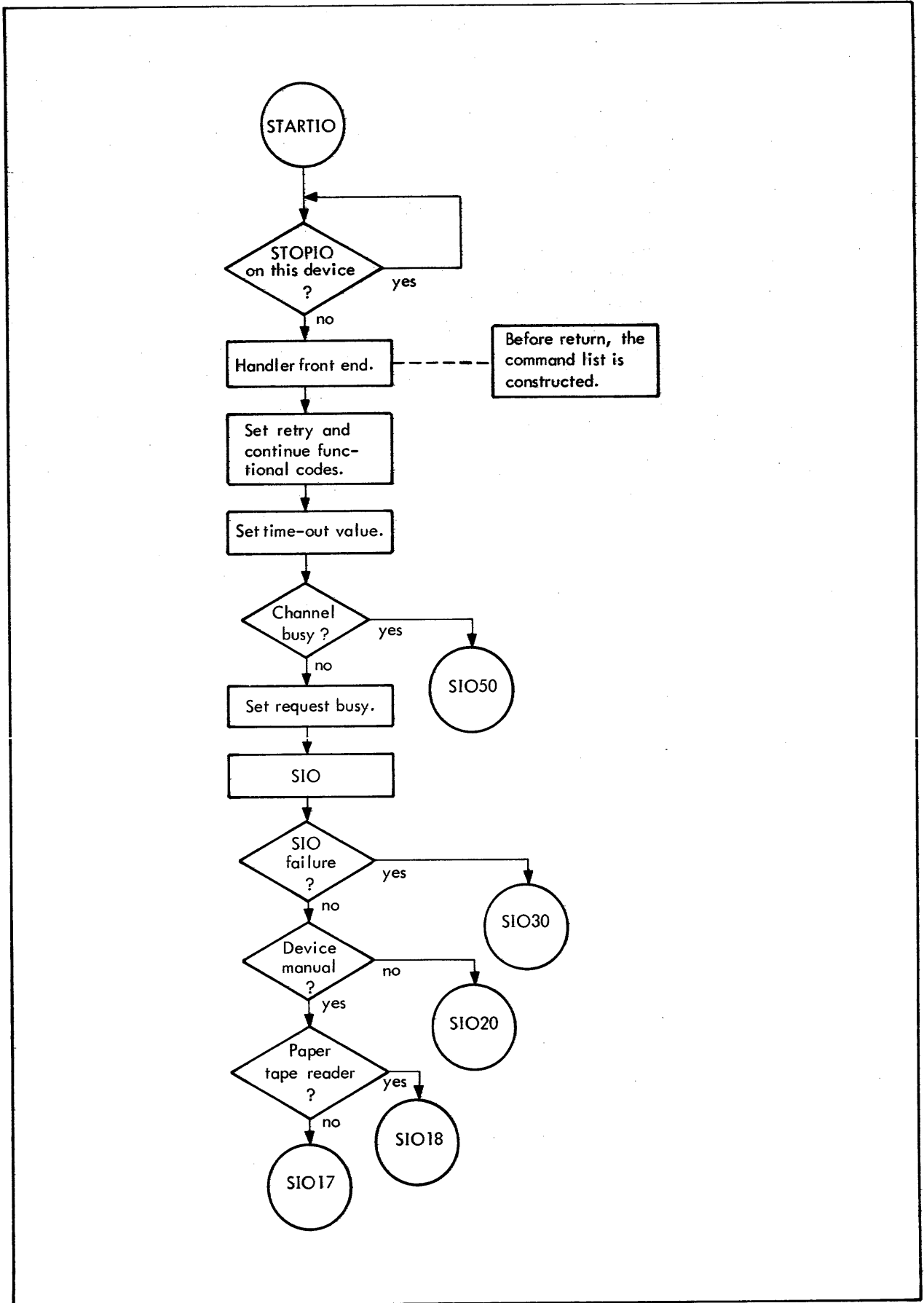


Figure 25. STARTIO Routine Flow

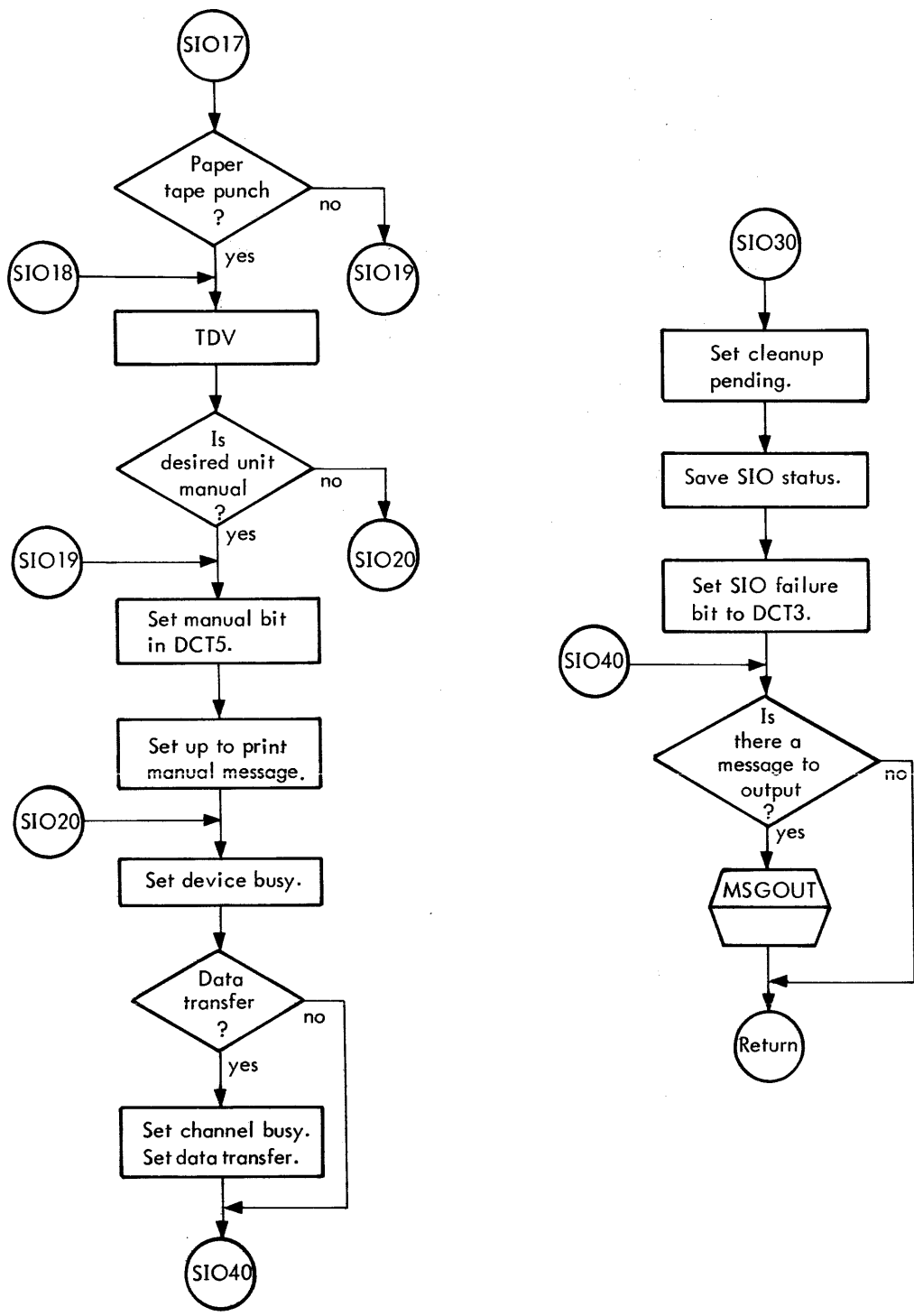


Figure 25. STARTIO Routine Flow (cont.)

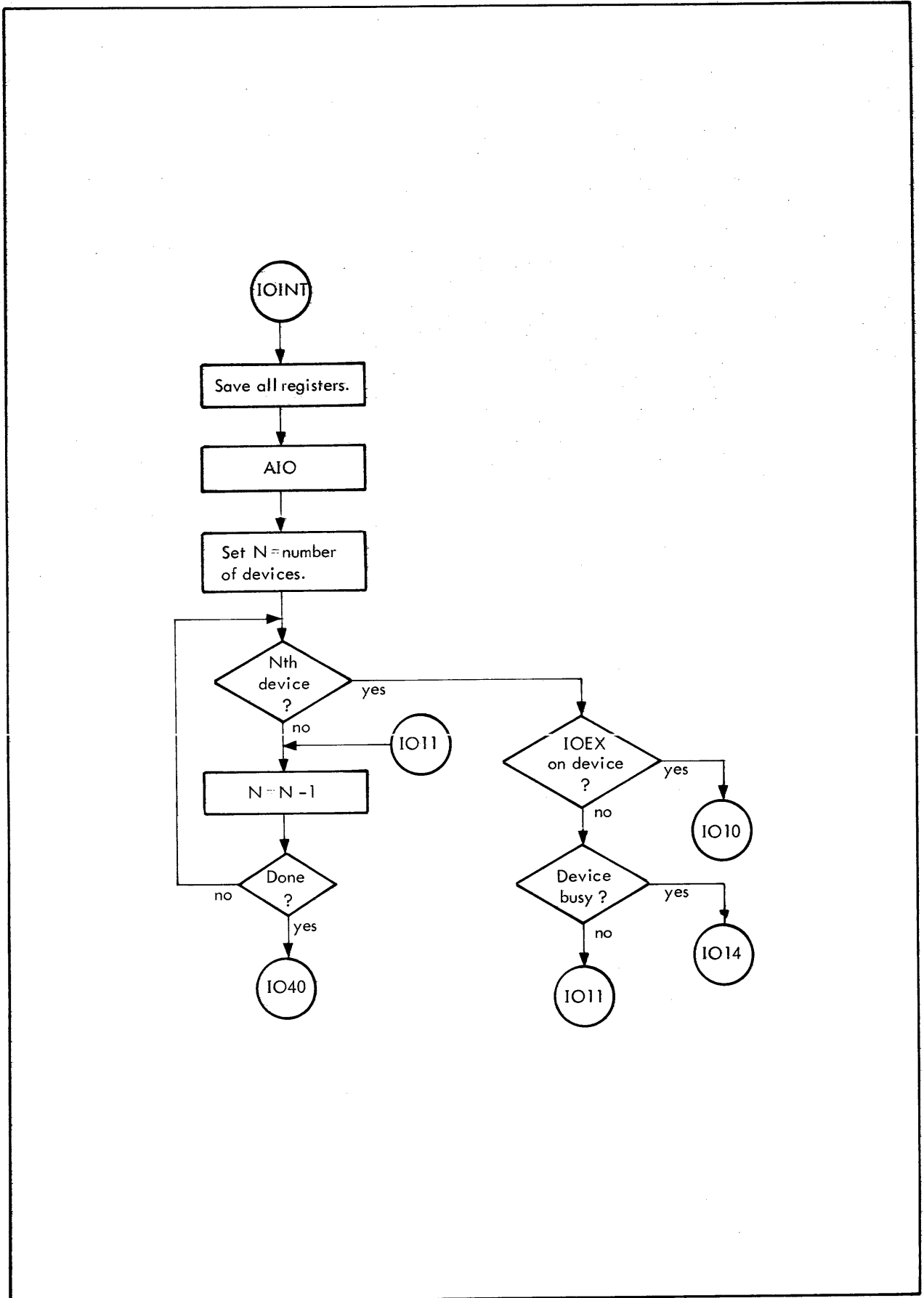


Figure 26. IOINT Routine Flow

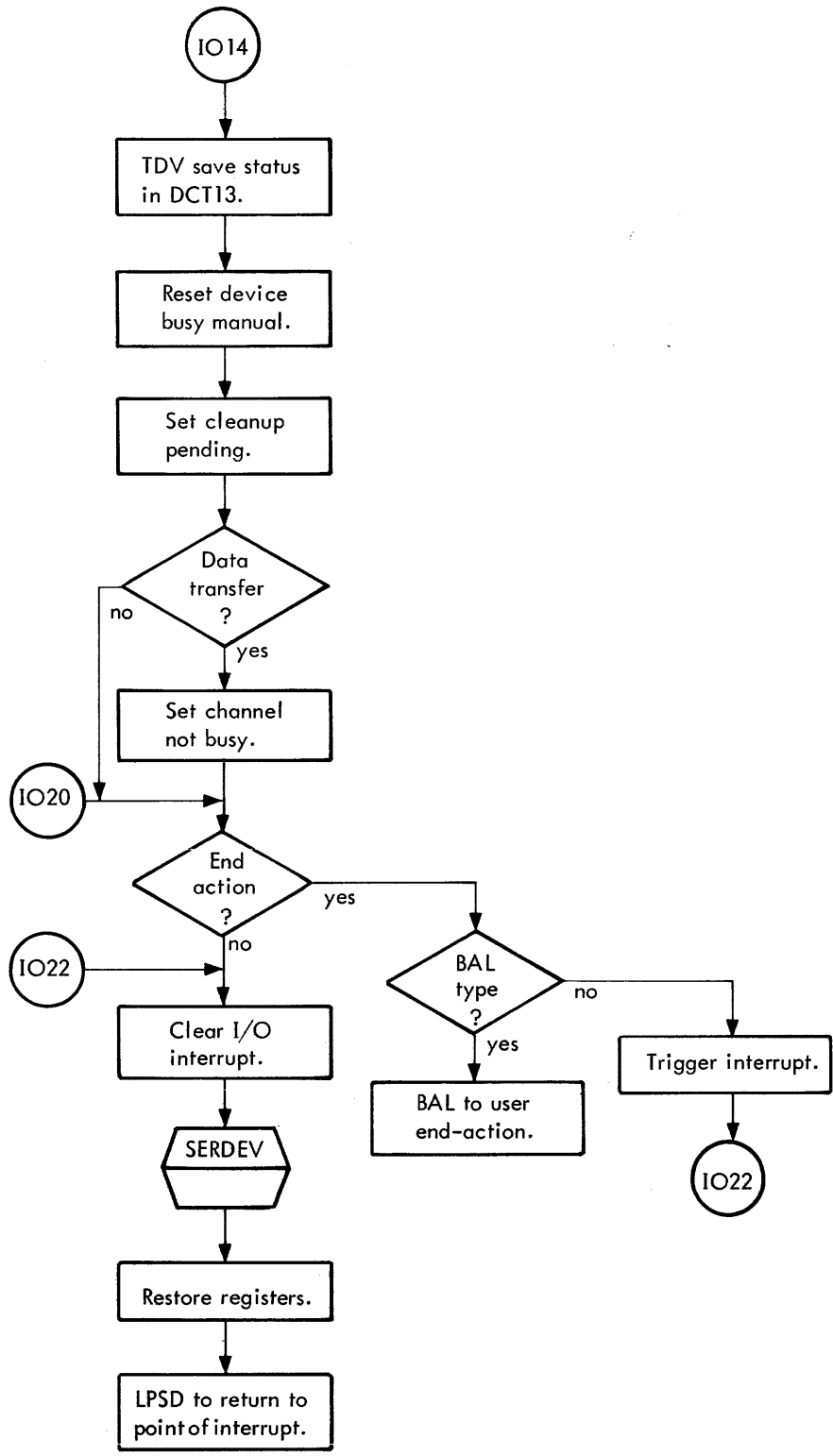
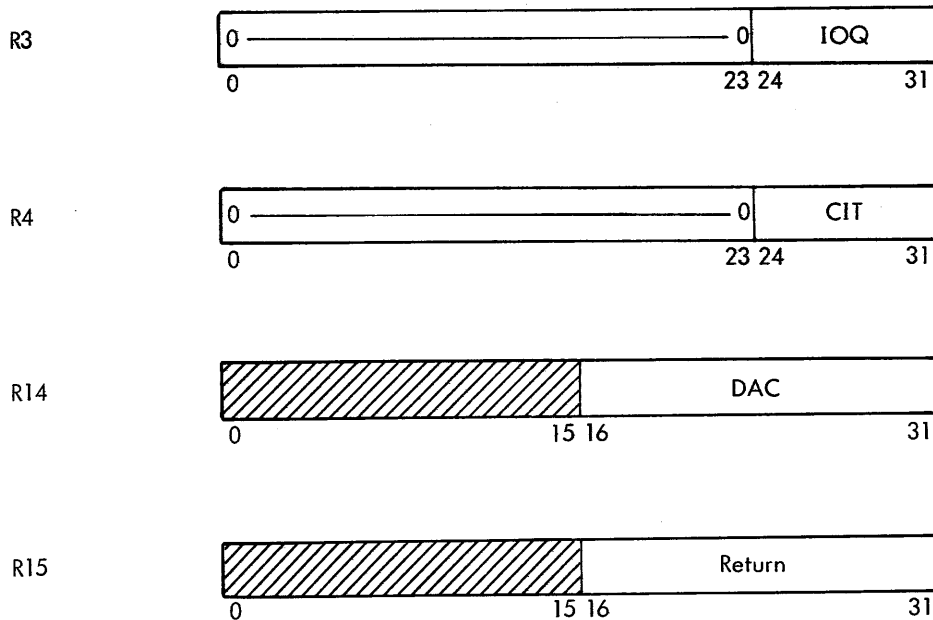


Figure 26. IOINT Routine Flow (cont.)



where the parameters are the same as for STARTIO.

Before the reentrance test is made, CLEANUP must determine if any errors have occurred, what follow-on action is to be taken, and what error messages are to be typed. This must be done prior to disturbing any tables in case the program was actually reentered.

When it has been determined that no reentrance has occurred, the information set up previously will be used to make the necessary changes to the tables. Interrupts will then be enabled so that end-action and error messages may be processed. The general flow of the CLEANUP routine is illustrated in Figure 27.

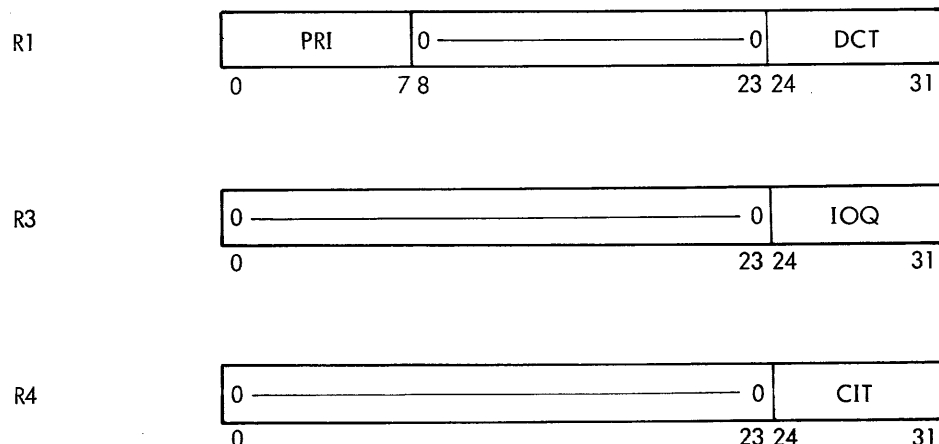
Miscellaneous Basic I/O Subroutines

REQCOM (Request Complete)

This routine dequeues an entry that has been completed. The routine also performs the testing necessary to intercept control commands from the C device. The call to process a cleanup request is

```
BAL,R15 REQCOM
```

with registers set as follows:



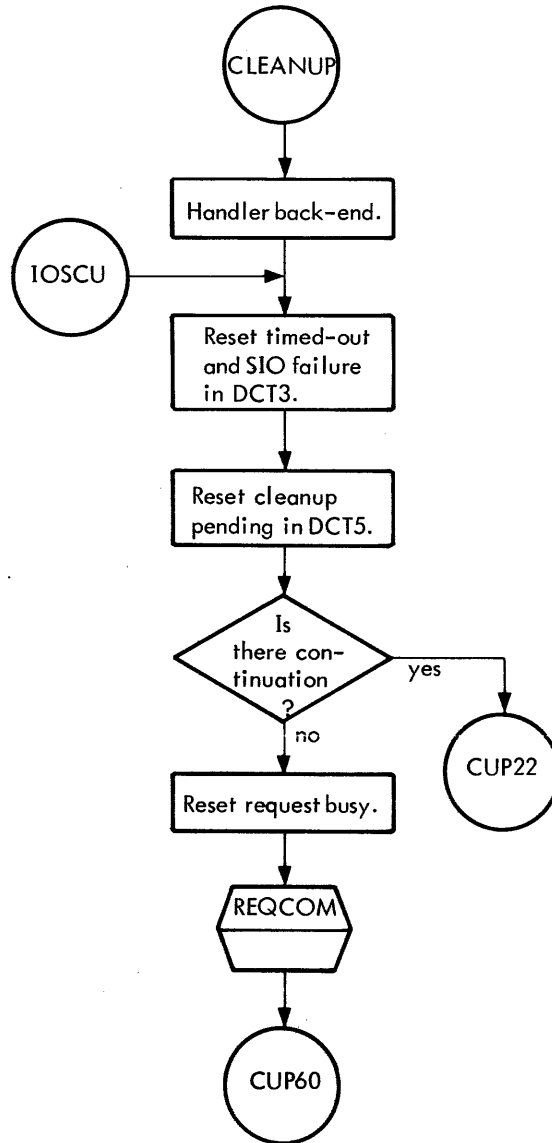


Figure 27. CLEANUP Routine Flow

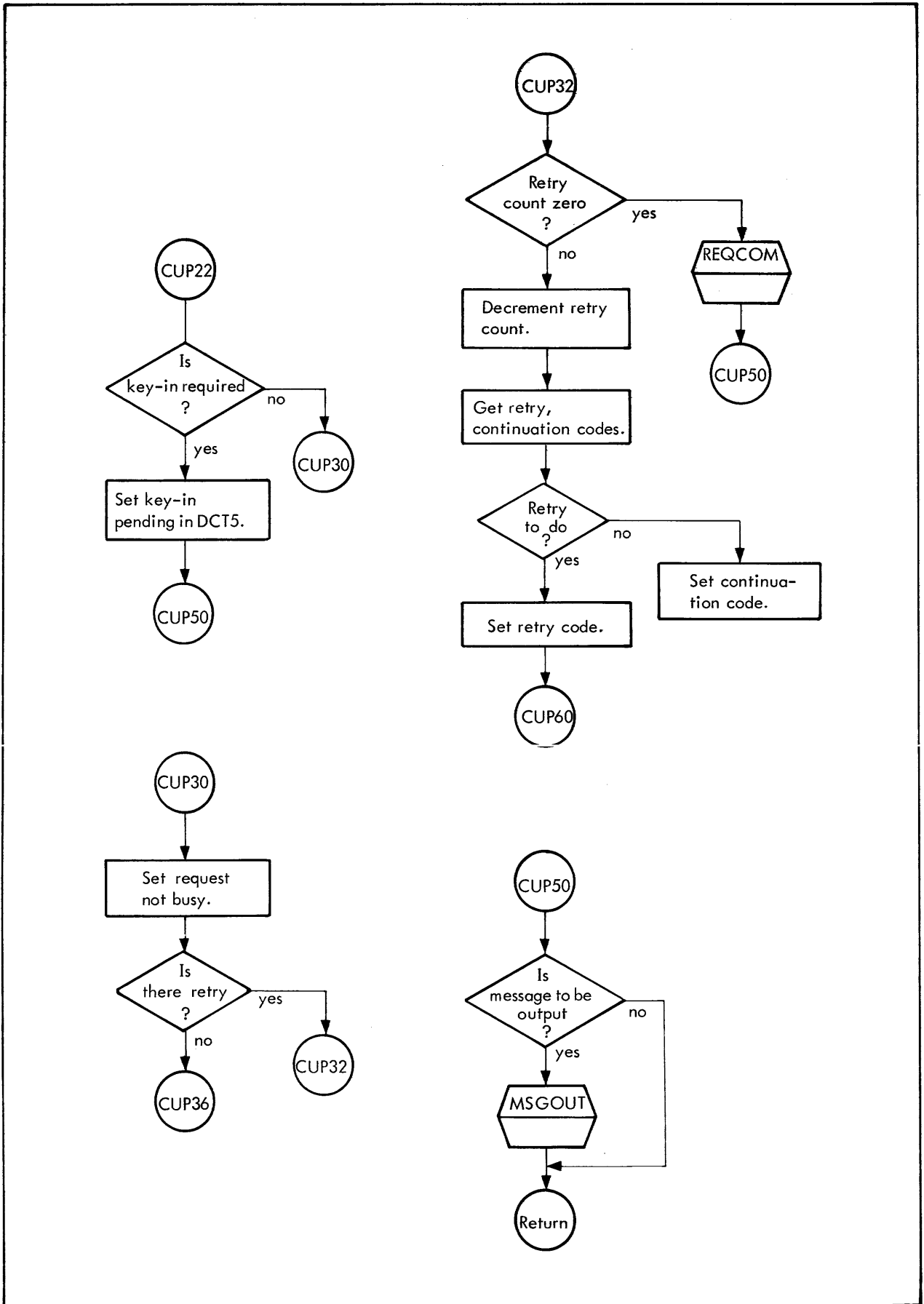
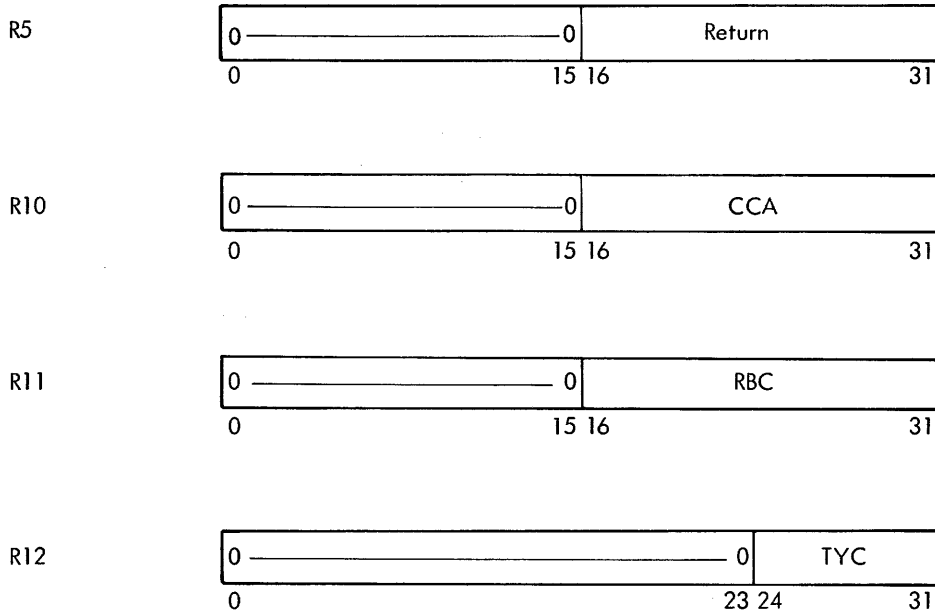


Figure 27. CLEANUP Routine Flow (cont.)



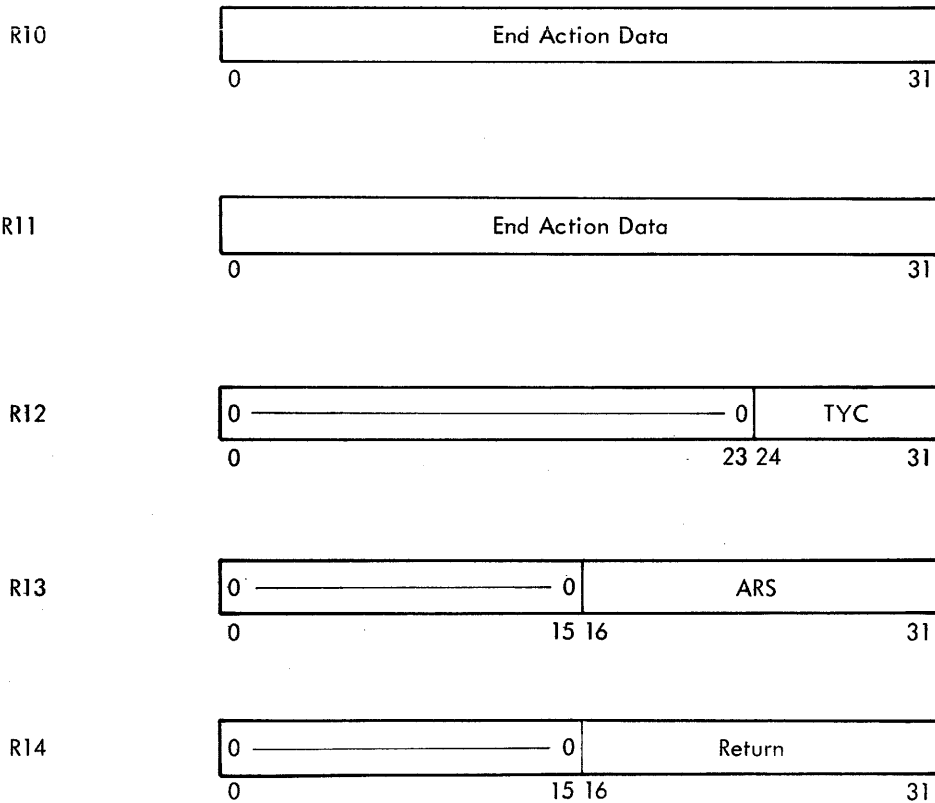
CUPCORE, CUPDCB Cleanup, End-Action Routines

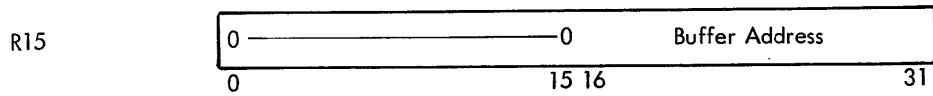
These routines post the TYC and ARS in either a DCB or memory location. When posted in a memory location, the parameters have the format of the FPT status word. The call to perform clean-up end-action is

BAL, R14 CUPCORE

BAL, R14 CUPDCB

with registers set as follows:



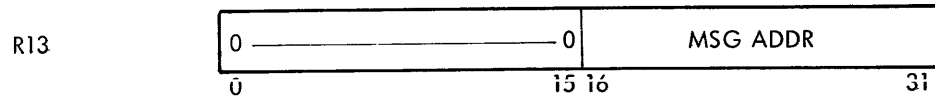
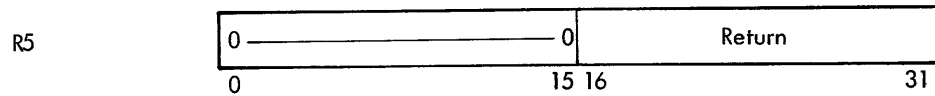
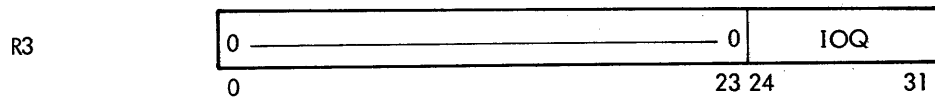
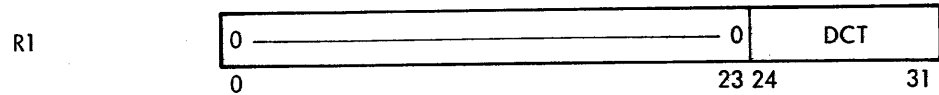


MSGOUT (Message Out)

This routine outputs an I/O error message to OC. The call to output an I/O error message is

BAL, R5 MSGOUT

with registers set as follows:

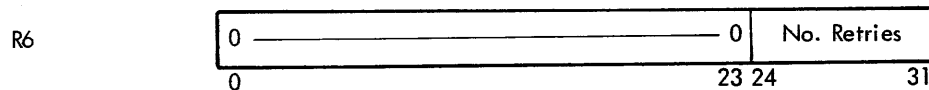
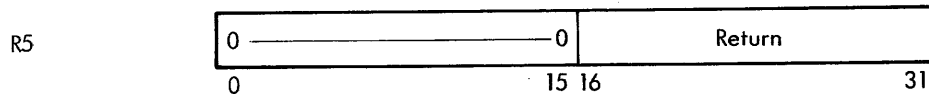
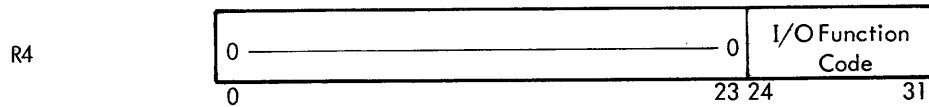


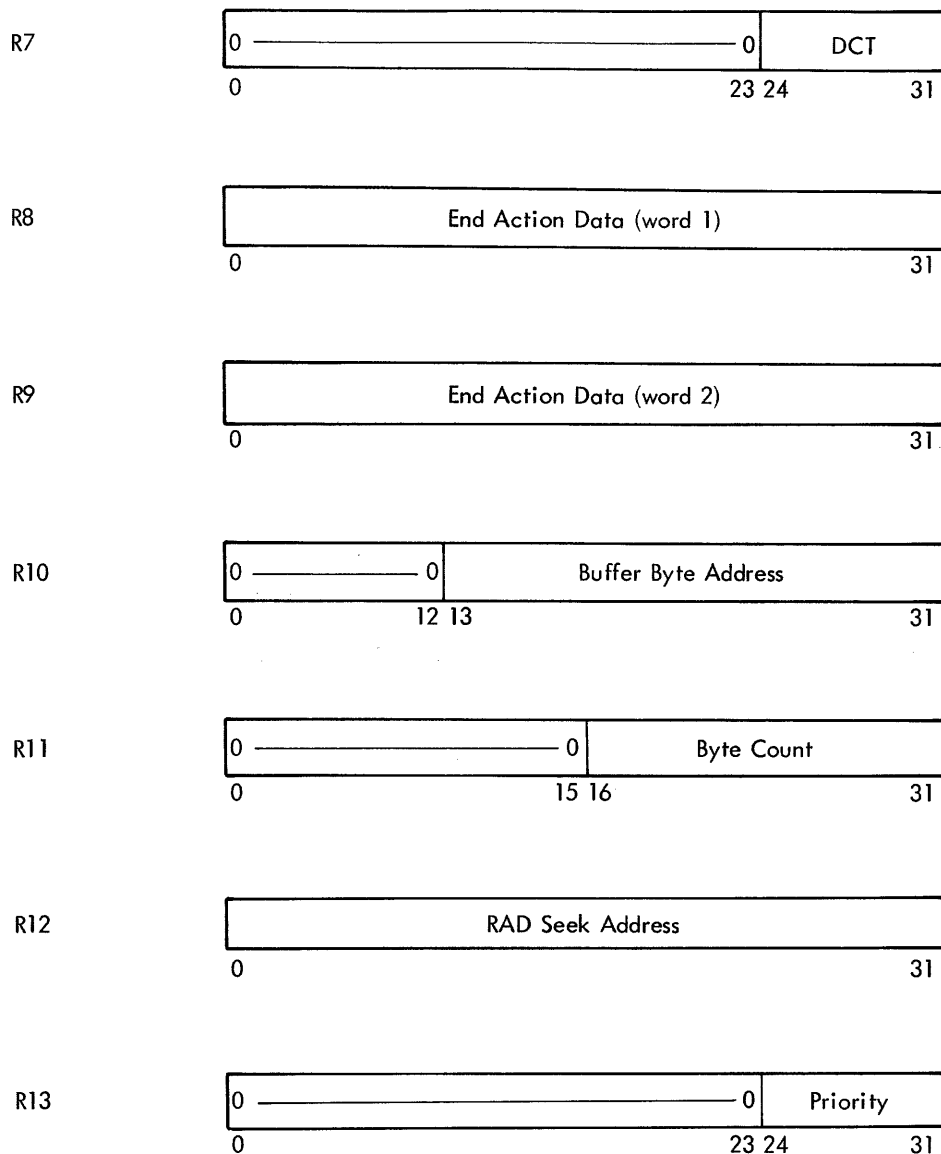
QUEUE

The subroutine labeled QUEUE enqueues I/O requests on a priority basis. A queue entry is constructed that completely defines the I/O operation, and this entry is entered in the channel queue behind (lower in priority) all queued entries of the same or higher priority. The call to enqueue an I/O request is

BAL, R5 QUEUE

with registers set as follows:





The flow of the QUEUE subroutine is illustrated in Figure 28.

User I/O Services

OPEN This function opens a DCB that results in opening a RAD file when the DCB is assigned to a RAD file. If the Error and/or Abnormal address is given in the function call, the addresses are set in the DCB.

Opening a RAD file involves constructing an RFT (RAD File Table) entry for the file. If the file is a permanent file, the area file directory is searched to locate the parameters that describe the file. These parameters are formatted and entered into the RFT. If the "file" is an entire area, the parameters used to construct the RFT entry are taken from the Master Dictionary. If the file is a background temporary file, the RFT entry must already have been constructed by the JCP. If the file is on a disk pack and a DED DPndd,R key-in is in effect, an abnormal code (X'2F') is posted in the DCB.

Blocking buffers or user-provided buffers are used for the directory search. Background requests use background buffers; foreground requests use foreground buffers.

CLOSE This function closes a DCB that may result in the closing of a RAD file. Closing a permanent RAD file involves updating the file directory if any of the directory parameters have been changed by accessing the file. Among such parameters that may change are file size (adding records to the file), record size (by Device File Mode call), etc.

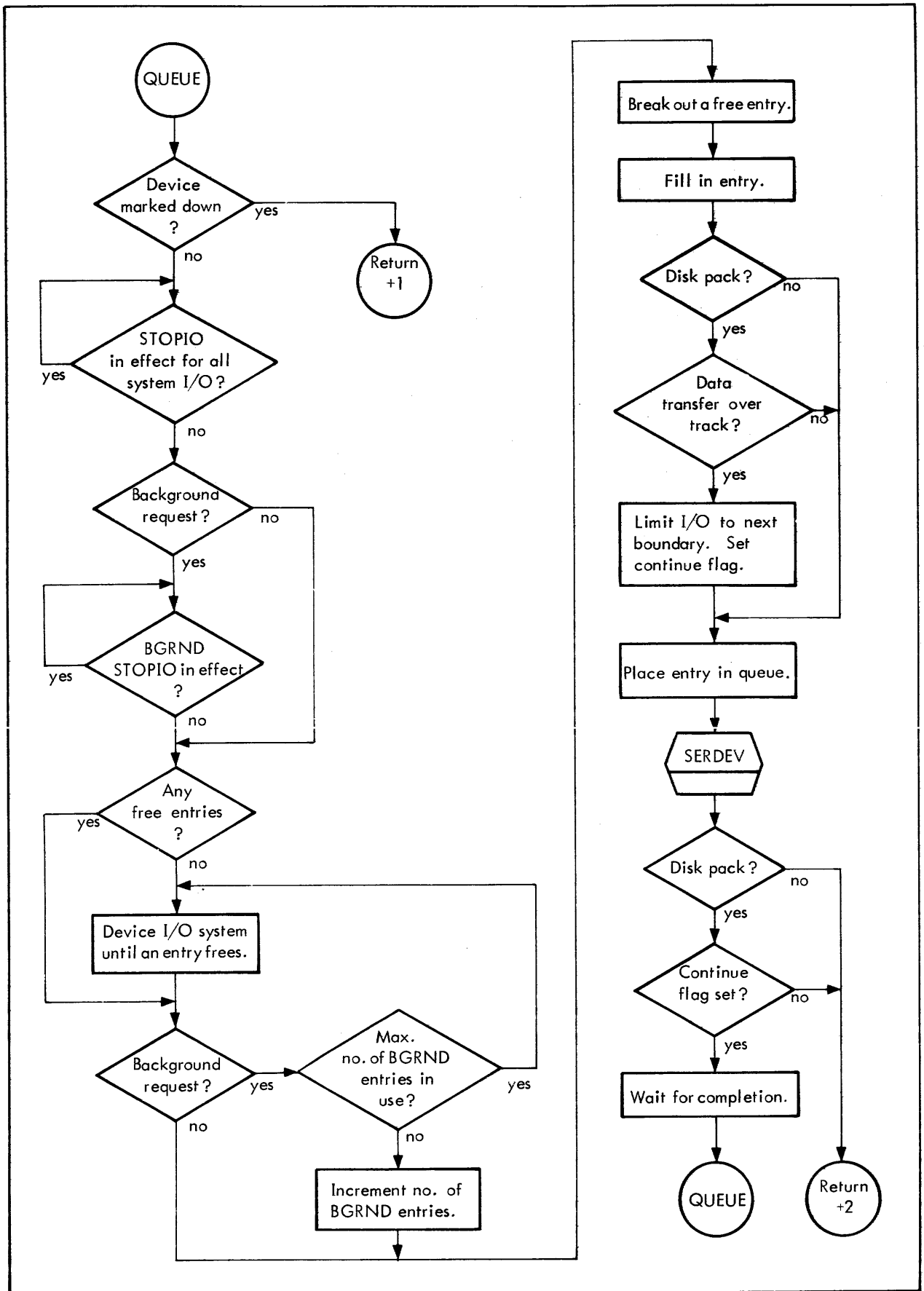


Figure 28. QUEUE Subroutine Flow

RAD files are only closed when the DCB being closed is the last DCB open and assigned to the file. Blocking buffers or user-provided buffers are used for the directory update as in the case of OPEN. If the file being closed is on a disk pack, a DED DPnnd,R key-in is in effect, and this is the last open file on device nnd, the message !!DPnnd IDLE will be output.

READ/WRITE A READ or WRITE function call will cause the addressed DCB to be opened if it is closed. READ/WRITE checks for legitimacy of the request by determining whether or not the following conditions are present:

1. For type I requests, the DCB is not busy with another type I request.
2. The assigned device or op label exists.
3. The user buffer lies in a legitimate region of core memory.
4. The type of operation (input or output) is legitimate on the device (e.g., output to the card reader is not allowed.)

For device I/O, READ/WRITE builds a partial QUEUE calling sequence and calls a device routine that performs device-dependent testing such as:

1. Mode flag in DCB (BIN,AUTO) for devices that recognize it.
2. Testing byte count against carriage size for keyboard/printer.
3. Testing for PACK bit in DCB for 7T magnetic tape.
4. Testing for VFC for line printer or keyboard/printer.

The device routines set up the proper function code in the QUEUE calling sequence and are labeled: RWKP, RW9T, RW7T, WCP, WLP, RCR, RPR, and WPP. These routines transfer control to a routine called GETNRT, which completes the QUEUE calling sequence by setting the number of retries. GETNRT then calls QUEUE. After queueing up the request (and the implicit call on SERDEV), control transfers to the CHECK logic.

For RAD file I/O, READ/WRITE calls the routine labeled RWFILE. RWFILE tests for write protection violation on write requests, end-of-file on sequential read requests, and end-of-tape on all requests. The different types of requests are handled as follows.

Direct Access. The RAD seek address is computed from the granule number provided in the FPT, and a QUEUE calling sequence is constructed that will queue up the request. Control then transfers to the CHECK logic.

Sequential Access (Unblocked). The RAD seek address is computed from the file position parameters and a QUEUE call is made. Control then transfers to the CHECK logic.

Sequential Access (Blocked). The next record is moved from/to the blocking buffer and blocks are read/written as required to allow the record transfer. For example, the first read request results in the first block being read and the first record in the block being deblocked into the user buffer. Successive read requests will not require actual input from the RAD until all records in the blocking buffer have been read. The blocks are always 256 words long and contain an integral number of fixed length records; that is, no record crosses a block boundary.

Background Blocking Buffers are handled dynamically. If a blocked I/O request is made and all allocated Background Blocking Buffers are in use by other files, one of the blocking buffers will be taken from its associated file (after writing the block to the file, if necessary) and used for the current request. This file is now associated with the file that most recently used it. When a request is made for I/O on the original file, the system recognizes that no Background Blocking Buffer is associated with the file and it will locate a buffer for this file by borrowing one from another file if necessary. One Background Blocking Buffer is sufficient for any background program.

Foreground Blocking Buffers are not handled dynamically.

Sequential Access (Compressed Files). Compressed files are treated in a manner similar to blocked files with the following exceptions:

1. The records are compressed/decompressed on the way to/from the blocking buffer.
2. The buffer does not contain a fixed number of records since the records are no longer of fixed length after compression. However, no compressed record crosses a block boundary.

To compress a record, the following EBCDIC codes are used:

| | |
|-------|--------------------|
| X'FA' | End-of-Block code |
| X'FB' | End-of-Record code |
| X'FC' | Blank Flag code |

All occurrences of two or more successive blank codes (X'40') are replaced by a Blank Flag code (X'FC') followed by a byte containing the length of the blank string. An End-of-Record code follows each record, and an End-of-Block code appears after the last record in a block.

When compressing records into the blocking buffer, a length of the compressed record is first computed and a test performed to determine whether the record will fit in the block. If so, it is placed in the buffer. If not, an End-of-Block code is written in the buffer and the buffer is written to the file.

At the conclusion of the file access, the status is posted in the user DCB or FPT and control is transferred to the CHECK logic.

PRINT This function builds the QUEUE calling sequence to perform the output on LL. After calling QUEUE, the routine either waits for completion, if wait was requested in the system call, or returns control to the user.

TYPE This function builds the QUEUE calling sequence by using code contained in the PRINT function. As in PRINT, a wait or return is performed as requested by the user.

DFM This function sets the MOD and PACK indicator in the addressed DCB to values given in the system call. If the DCB is assigned to a RAD file, the record size (RFT5), the organization (RFT7), and/or the granule size (RFT4) are set if requested by the user. The corresponding parameters on the file directory are updated when the file is closed.

DVF This function sets the DVF bit in the addressed DCB to the value (0 or 1) specified by the user.

REWIND This function rewinds magnetic tapes and RAD files. No action is taken if the addressed DCB is assigned to any other type of device.

Magnetic tapes are rewound by building a QUEUE calling sequence with the Rewind function code and calling QUEUE.

RAD files are rewound by zeroing the file position (RFT11), current record number (RFT12), blocking buffer position (RFT10), and blocking buffer control word address (RFT17) parameters.

WEOF This function writes an "end-of-file" on paper tape punch, card punch, magnetic tape, and RAD files. A request addressing a DCB assigned to some other type of device results in no action.

An "end-of-file" is written on paper tape by calling QUEUE with a request to write an EBCDIC '!EOD' record.

An "end-of-file" is written on a card by calling QUEUE with a request to write an EBCDIC '!EOD' record.

An "end-of-file" is written on magnetic tape by calling QUEUE with a request to write a tape mark.

An "end-of-file" on a RAD file is "written" by copying the current record number minus 1 (RFT12) to the file size (RFT6) and setting an indicator so that the file directory will be updated when the file is closed.

PREC This function positions magnetic tapes and RAD files by moving some specified number of records either backward or forward. It does not affect other devices. Positioning is performed as follows:

1. A magnetic tape QUEUE call is constructed that specifies through the function code the direction of the motion, and through the "seek-address" parameter the number of records to move. The basic I/O system then moves the tape.
2. The new position within the file of an unblocked RAD file is computed as a function of the record size and the sector size. File position (RFT11) and current record number (RFT12) parameters are set to indicate the new position.
3. The new position of a blocked RAD file is computed as a function of the current record number, record size, block size, current blocking buffer position, current file position, and RAD sector size. The blocking buffer position (RFT10), file position (RFT11), and current record number (RFT12) are set to indicate the new position.

4. The new current record number of a compressed RAD file is computed and subroutine PCFIL is called. This subroutine positions a compressed RAD file at the specified record by counting records from the beginning of the file until the desired position is found. PCFIL sets the blocking buffer position (RFT10), file position (RFT11), and current record number (RFT12) parameters to indicate the new position.

PFILE This function positions magnetic tape and RAD files at the beginning or end of files. It does not affect other devices. Positioning is performed as follows:

1. A magnetic tape QUEUE call is constructed with function code to "space file" either backwards or forwards. This results in the tape being positioned past the tape mark in the specified direction. If a skip was not requested, the tape is positioned on the other side (near side) of the tape mark through a QUEUE call for a position one record opposite in direction to the space file.
2. The RAD Files Backward file position (RFT11) is set to zero; the blocking buffer position (RFT10) is set to zero; the current record number is set to 1; and the blocking buffer control word address (RFT17) is set to zero.
3. The Unblocked RAD File Forward current file position is computed as a function of the file size, the record size, and the RAD sector size. The current file position (RFT11) and the current record number (RFT12) are set to indicate the new position.
4. The Blocked RAD File Forward current file position (RFT11) and the Blocking Buffer Position (RFT10) are computed as a function of the file size, record size, block size, and RAD sector size. These parameters and the current record number (RFT12) are set to indicate the new position.
5. The Compressed RAD File Forward subroutine PCFIL is called with file size plus one as the record number. This subroutine positions the file at the start of the specified record.

4. JOB CONTROL PROCESSOR

Overview

The Job Control Processor (JCP) is assembled as a Relocatable Object Module (ROM) and is loaded at SYSGEN time by the SYSLOAD phase of SYSGEN. The JCP is absolutized to execute at the start of background and is loaded onto the RBM file on the RAD. The JCP is loaded from RAD for execution by the Background Loader upon the initial "C" key-in; and thereafter, is loaded following the termination of execution of each processor or user program in background memory.

The JCP executes with special privileges since it runs in Master Mode with a skeleton key. Master Mode rather than Slave Mode is essential to the JCP since, at appropriate times, it executes a Write Direct instruction to trigger the RBM Control Task; it also issues an HIO instruction to the "C" device when the "C" device assignment is changed. A skeleton key instead of the background key is also essential to the JCP since it sets flags for itself and the Monitor in the resident Monitor portion of memory. Bit zero of system cell K:JCP1 is set to 1 to inform the Monitor that the JCP is executing.

The JCP controls the execution of background jobs by reading and interpreting control commands. All cards read from the "C" device that contain an exclamation mark in column one (except for an !EOD command), are defined as JCP control commands. The I/O portion of the Monitor will not allow any background program except the JCP to read a JCP control command. The JCP runs until a command is read that requires the execution of a processor or user program, or until a !FIN command is encountered.

The JCP presently requires a minimum of about 5K of core to execute, which means that the smallest possible core space allocated to the background must be at least 5K. Approximately one third of the JCP code consists of the JCP Loader, which is used to load the Overlay Loader at System Generation time.

The flowchart illustrated in Figure 29 depicts the overall flow of the JCP, and Figures 30 through 48 illustrate the JCP commands. The labels used in the flowcharts correspond to the labels in the program listing.

ASSIGN Command Processing

The !ASSIGN commands are read from the "C" device by the JCP, and are primarily used to define or change the I/O devices used by a program. The !ASSIGN command can also be used to change parameters in a DCB. Since all !ASSIGN commands must be input prior to the RUN or Name command (where Name is the name of a processor or user program file in the SP area) to which they apply, the information from each !ASSIGN command is saved in core by the JCP. The JCP builds an ASSIGN table containing the information from each !ASSIGN command. This table consists of ten words for each !ASSIGN, plus one word specifying the number of ten-word entries. The table remains in background memory and is passed to the Background Loader (an RBM overlay that loads a program's root for execution by the JCP). After the Background Loader reads in the program's root, it makes the appropriate changes to the program's DCBs from the information in the ASSIGN table. The ASSIGN table can then be destroyed as the program executes; therefore, !ASSIGN commands take effect only for a job step and not an entire job. The ASSIGN table has the format shown in Table 1.

Table 1. ASSIGN Table

| Words | Contents |
|-------|---|
| 1 | Number of entries in table (each entry of ten words contains data from one !ASSIGN command). This word is always on an odd boundary; K:ASSIGN contains the address of word 1. |
| 2,3 | Name of DCB to change in EBCDIC. This pair of words and the next four pairs of words are on a doubleword boundary. |
| 4 | This word contains changes to the items in word 0 of the DCB. |
| 5 | Mask for items being changed in word 0. The Background Loader does an STS instruction (using words 4 and 5) to change the items in word 0 of the DCB. |
| 6 | Changes for word 1 of DCB. |
| 7 | Mask for items being changed in word 1. |
| 8 | Changes for word 3 of DCB. |
| 9 | Mask for items being changed in word 3. |
| 10,11 | File name in EBCDIC if DCB is assigned to a RAD file; otherwise, these words equal zero. |

Words 2 through 11 contain one entry in the ASSIGN table and are repeated for each !ASSIGN command.

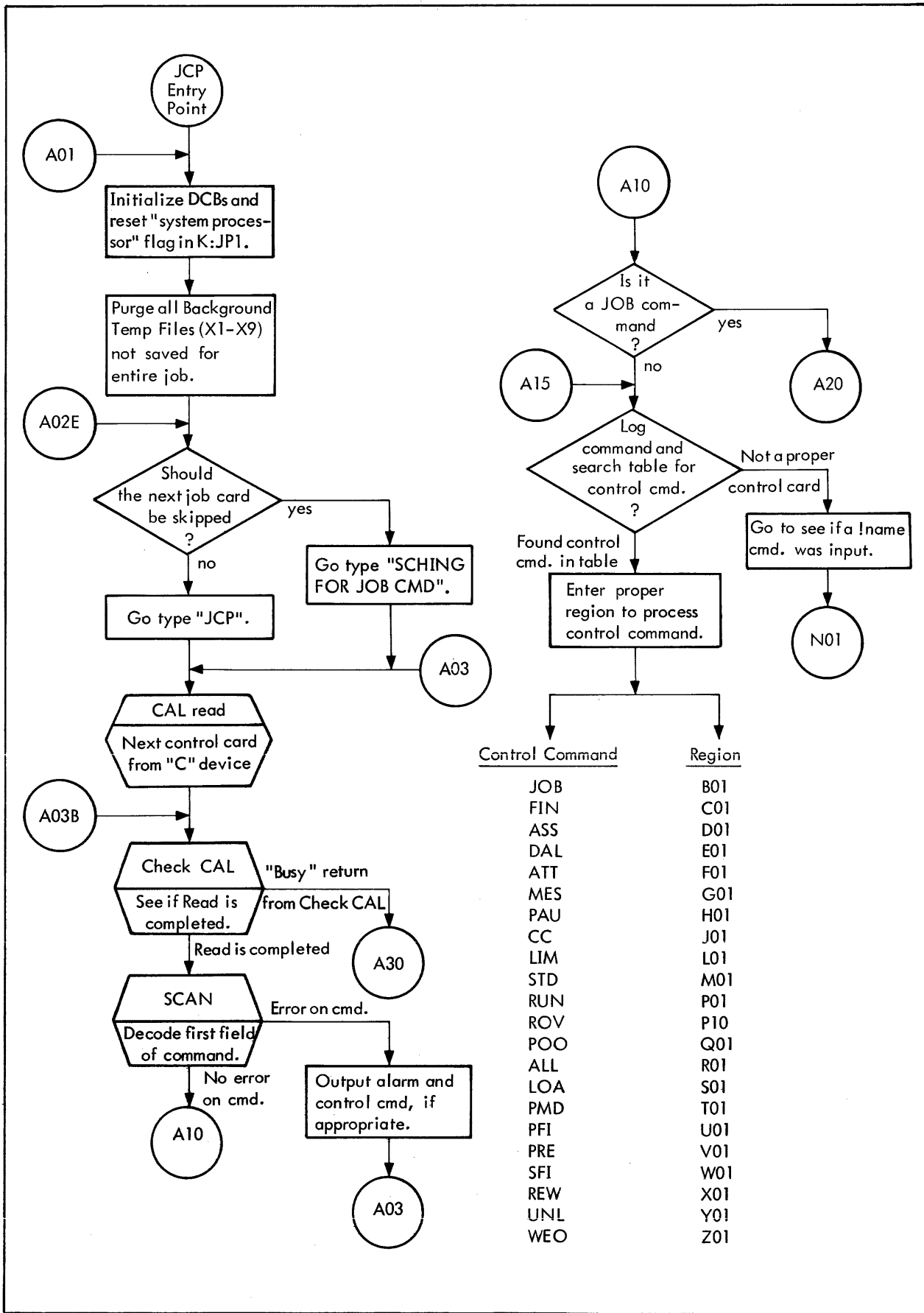


Figure 29. JCP General Flow

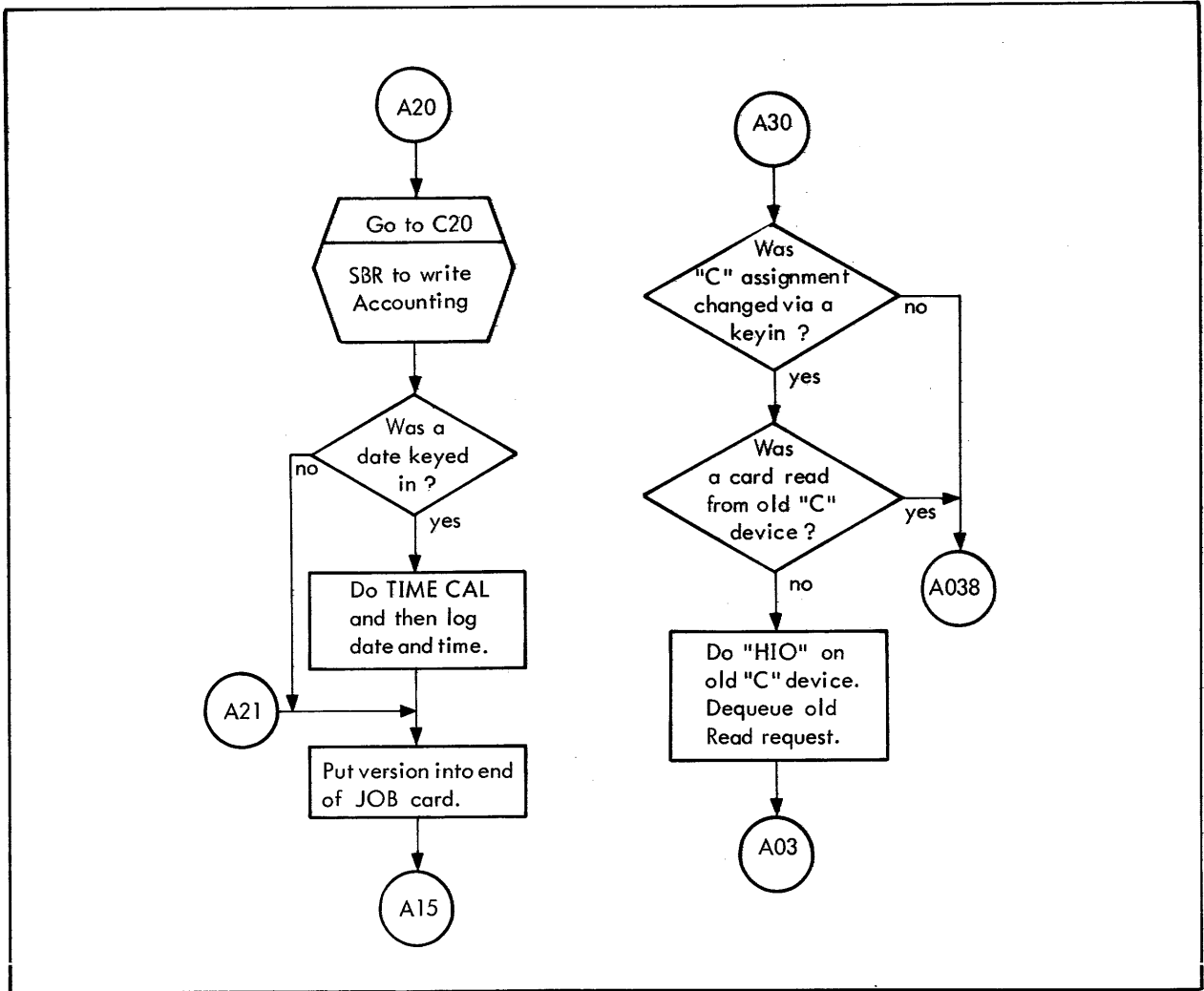


Figure 29. JCP General Flow (cont.)

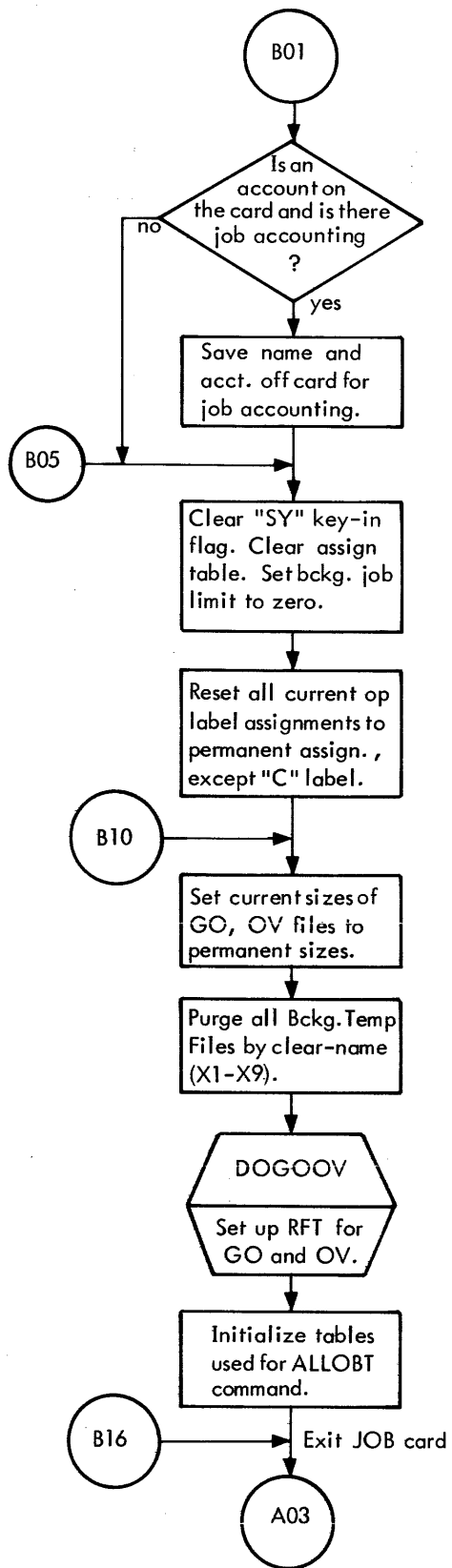


Figure 30. JOB Command Flow

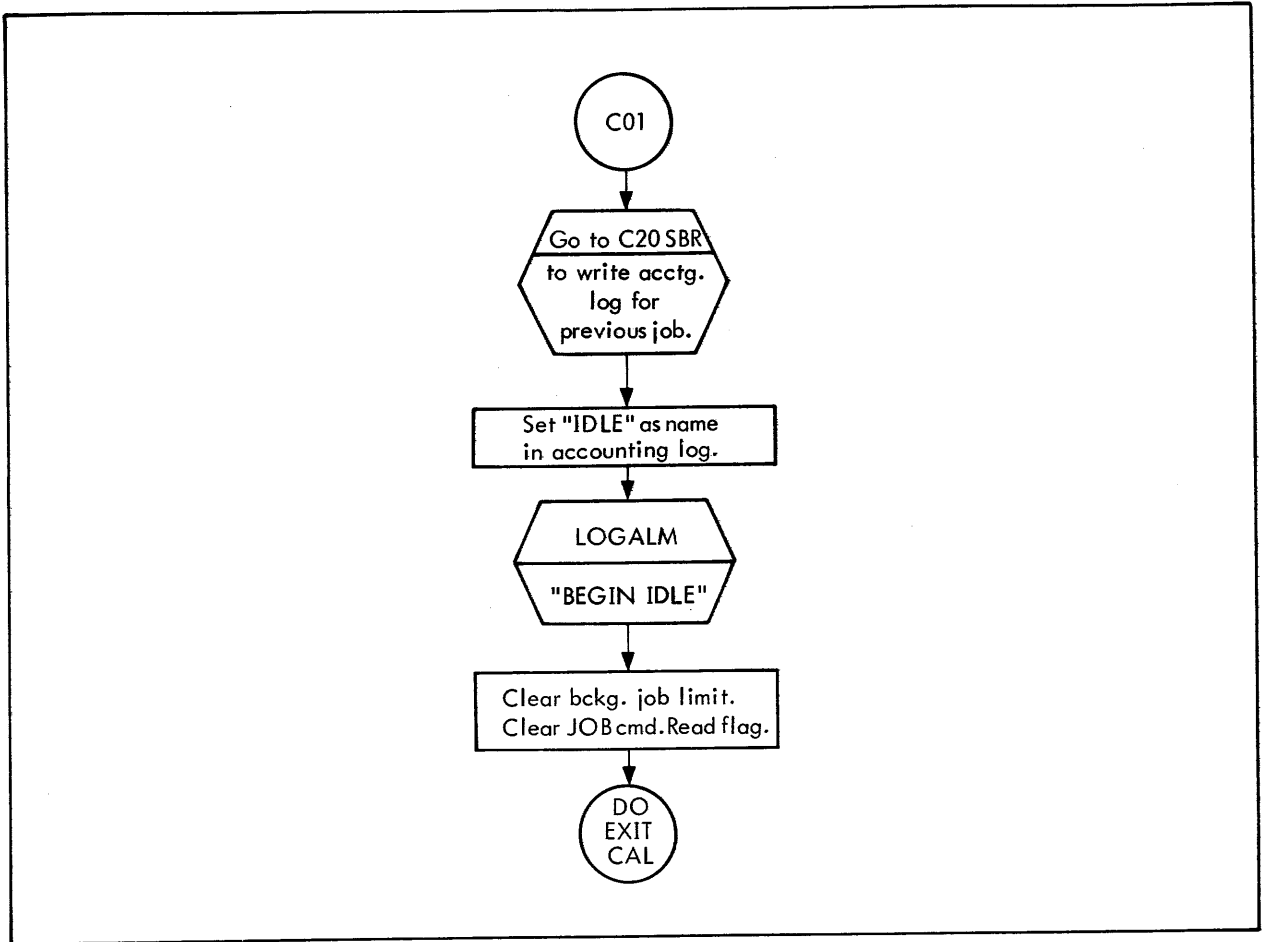


Figure 31. FIN Command Flow

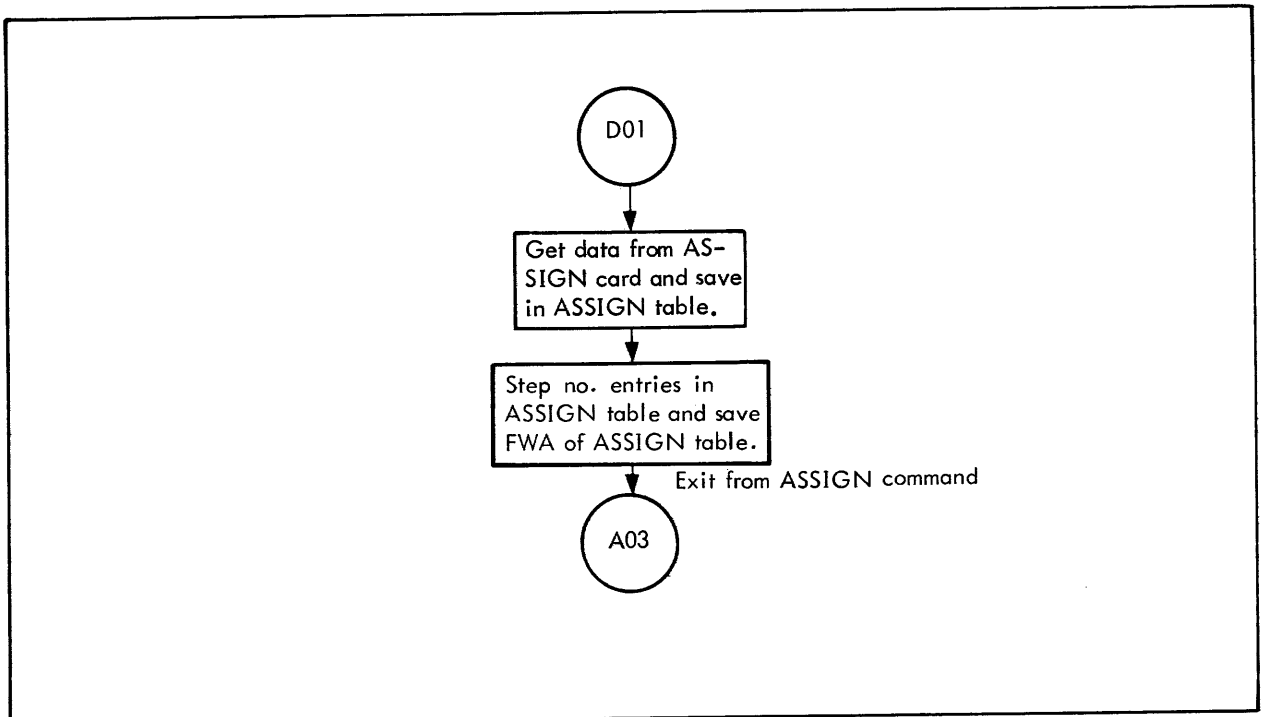


Figure 32. ASSIGN Command Flow

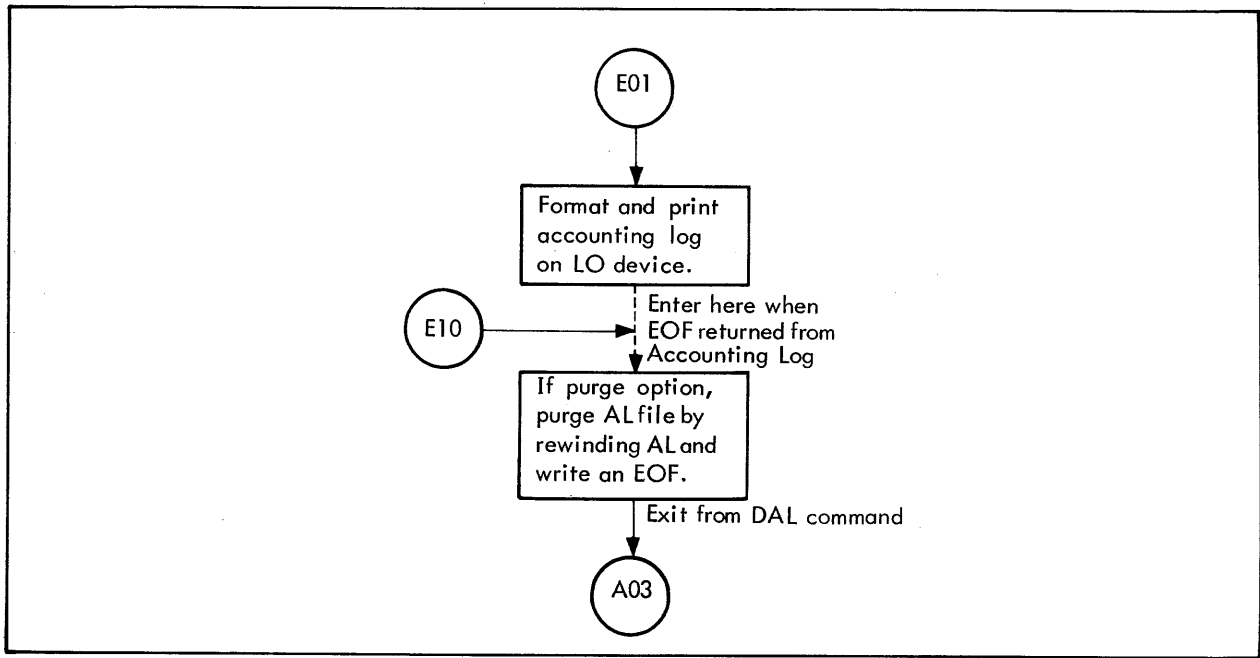


Figure 33. DAL Command Flow

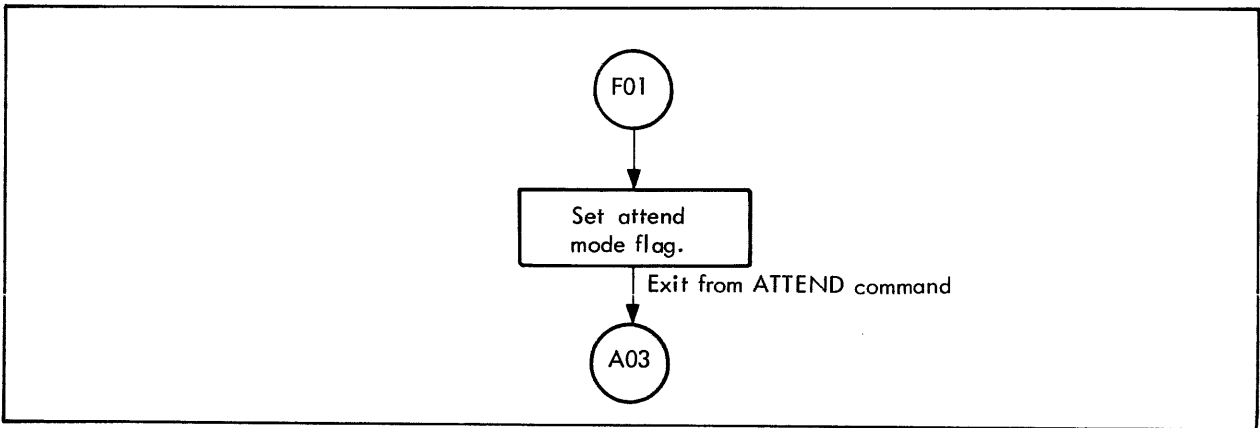


Figure 34. ATTEND Command Flow

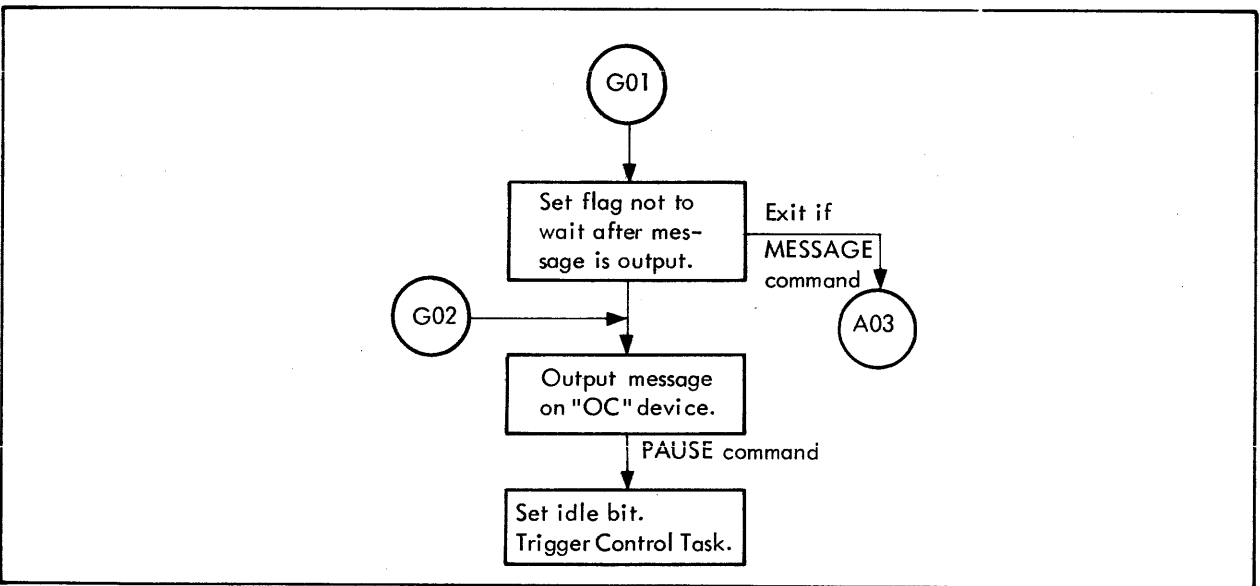


Figure 35. MESSAGE Command Flow

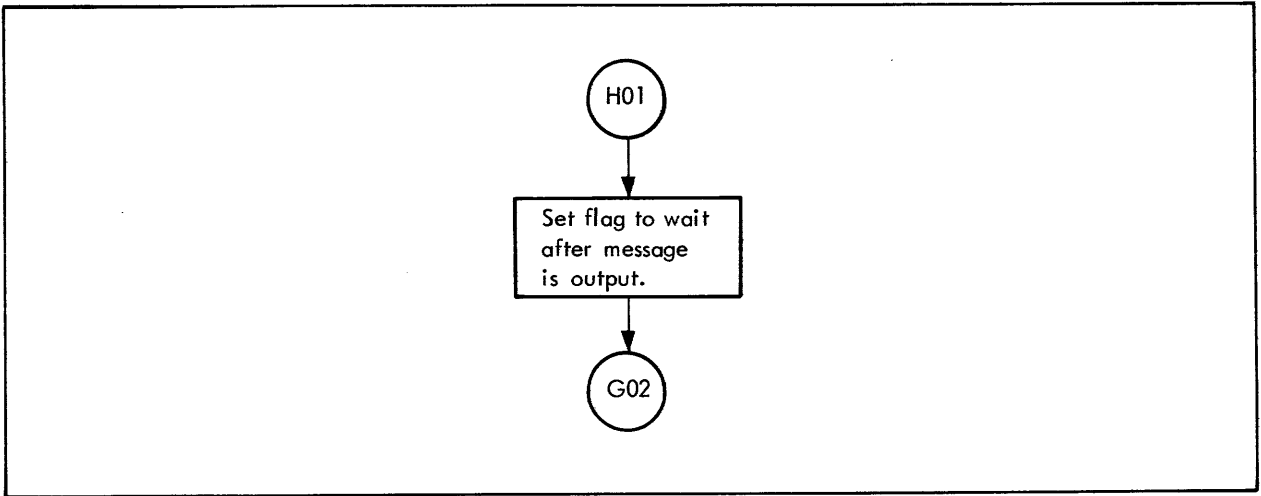


Figure 36. PAUSE Command Flow

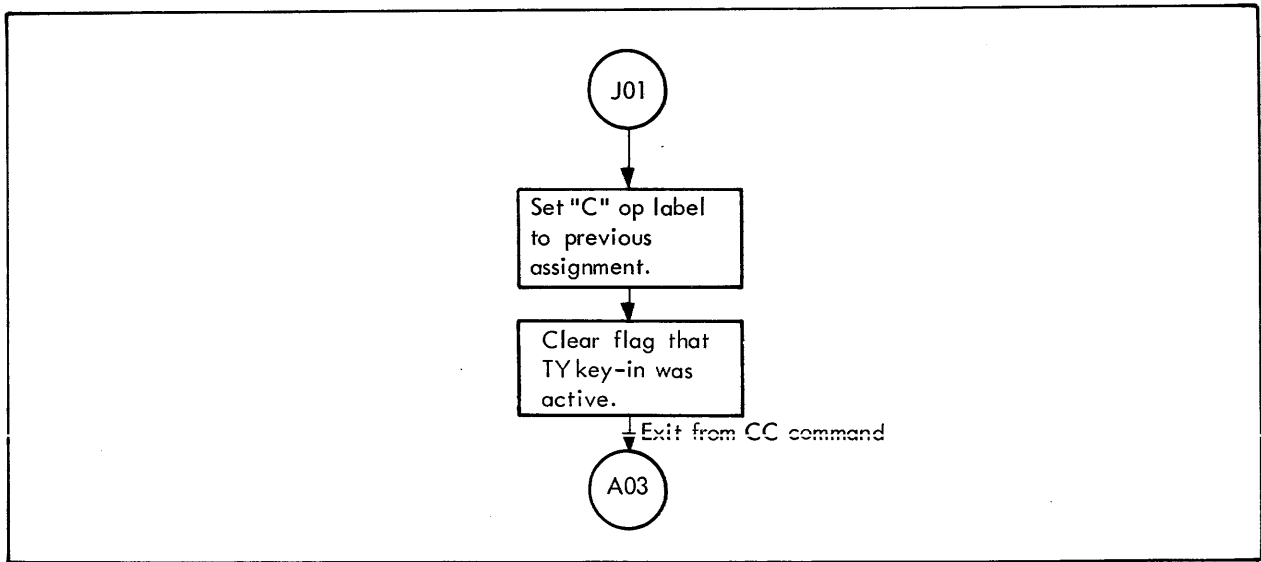


Figure 37. CC Command Flow

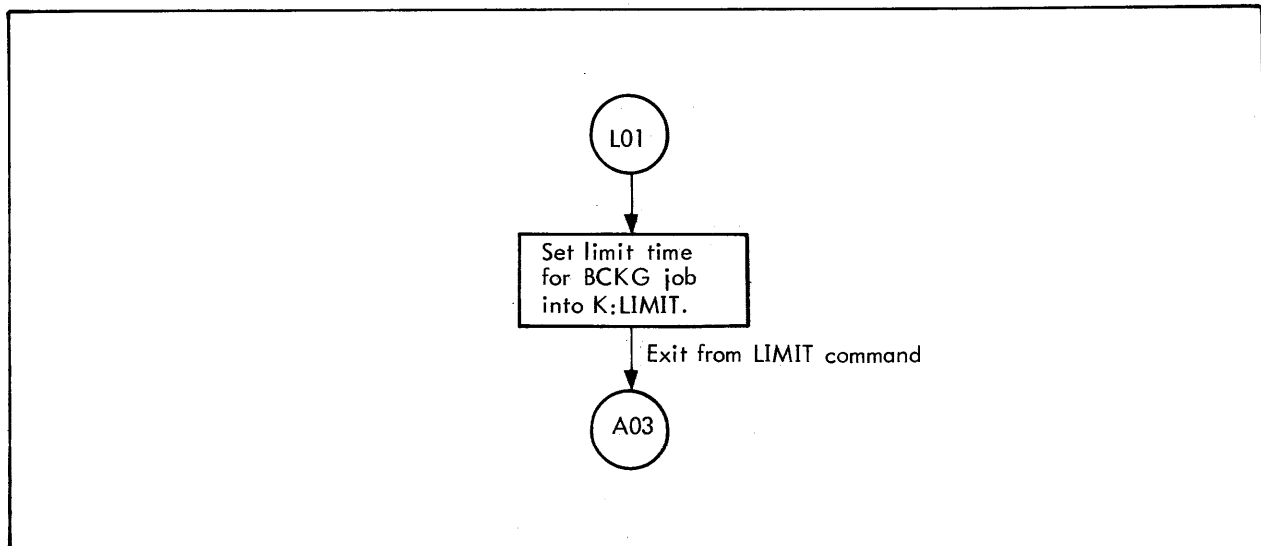


Figure 38. LIMIT Command Flow

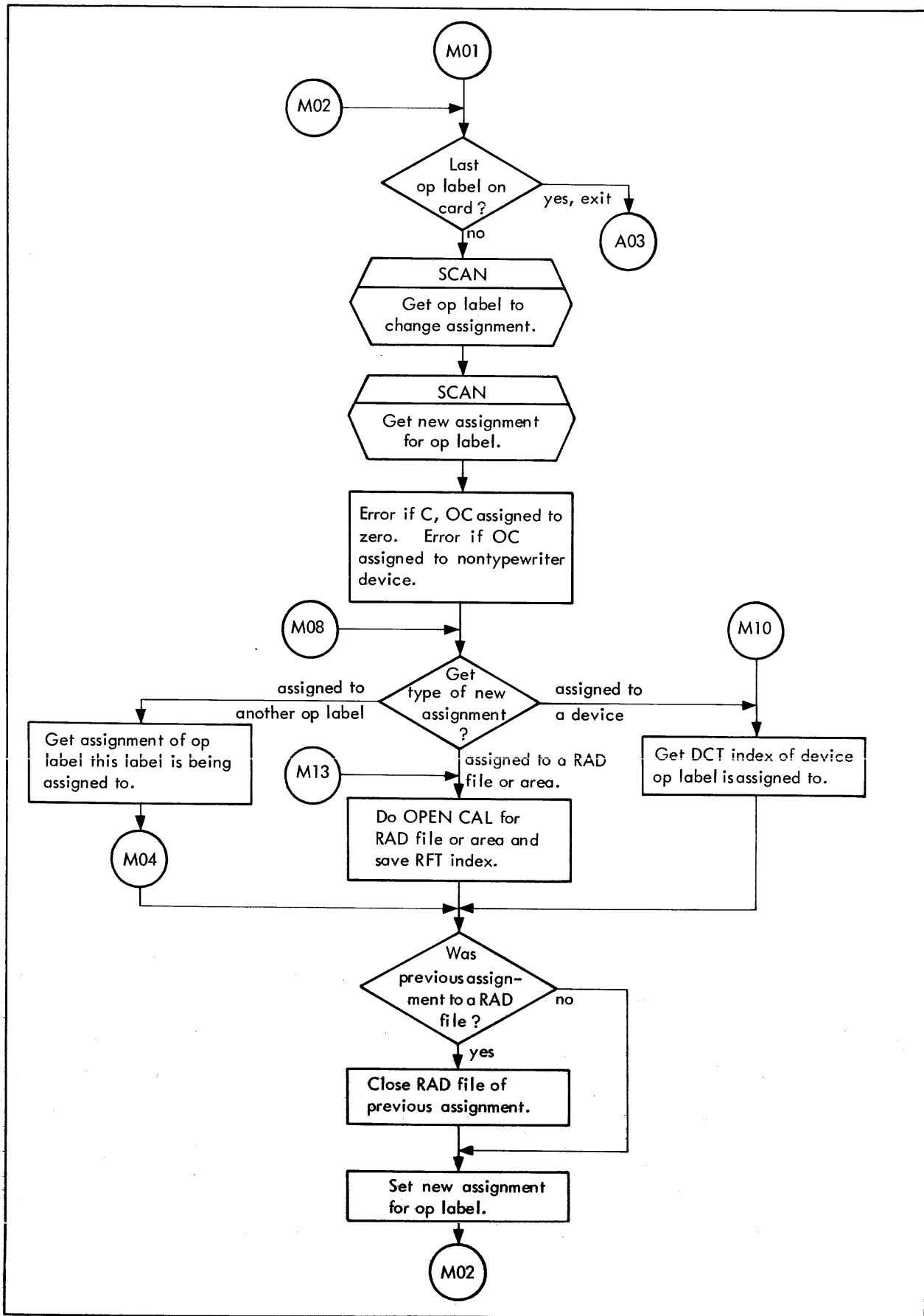


Figure 39. STDLB Command Flow

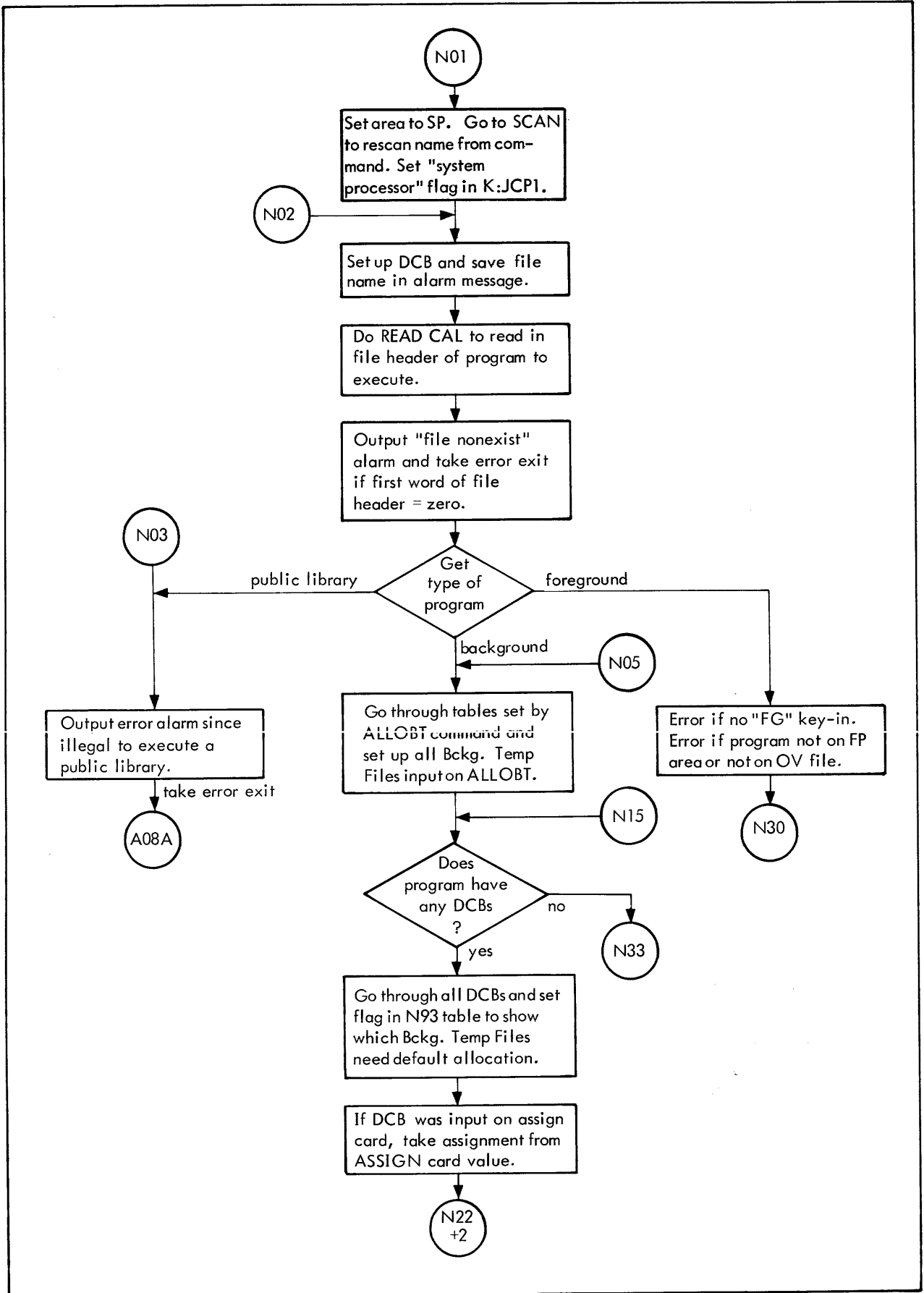


Figure 40. NAME Command Flow

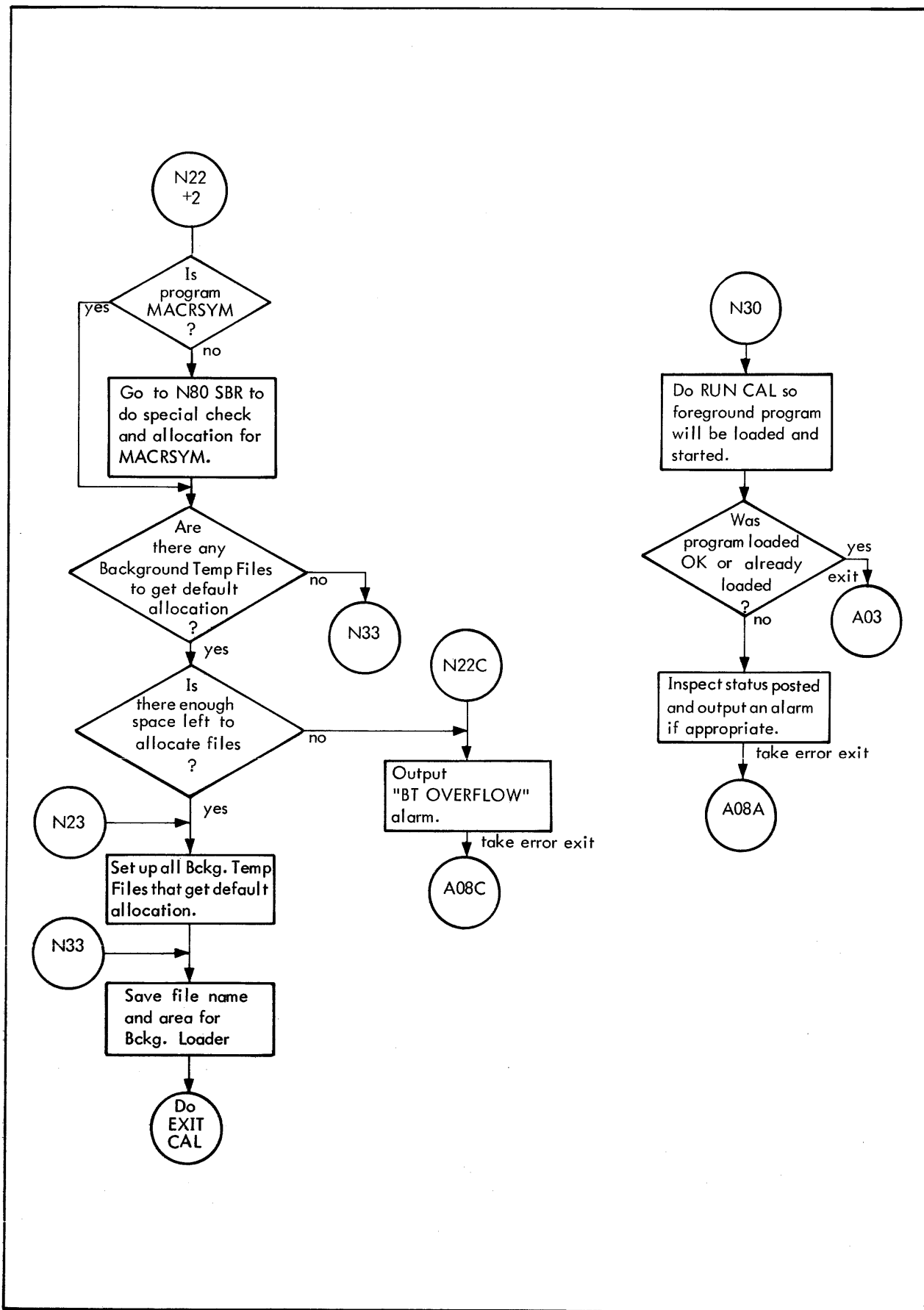


Figure 40. NAME Command Flow (cont.)

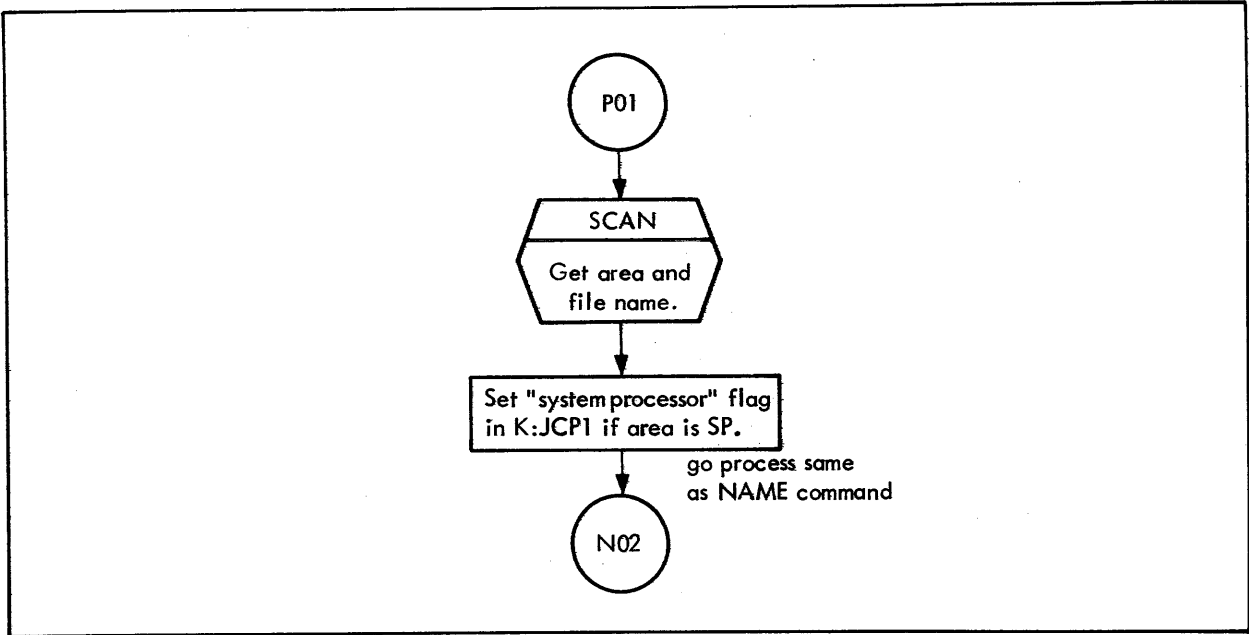


Figure 41. RUN Command Flow

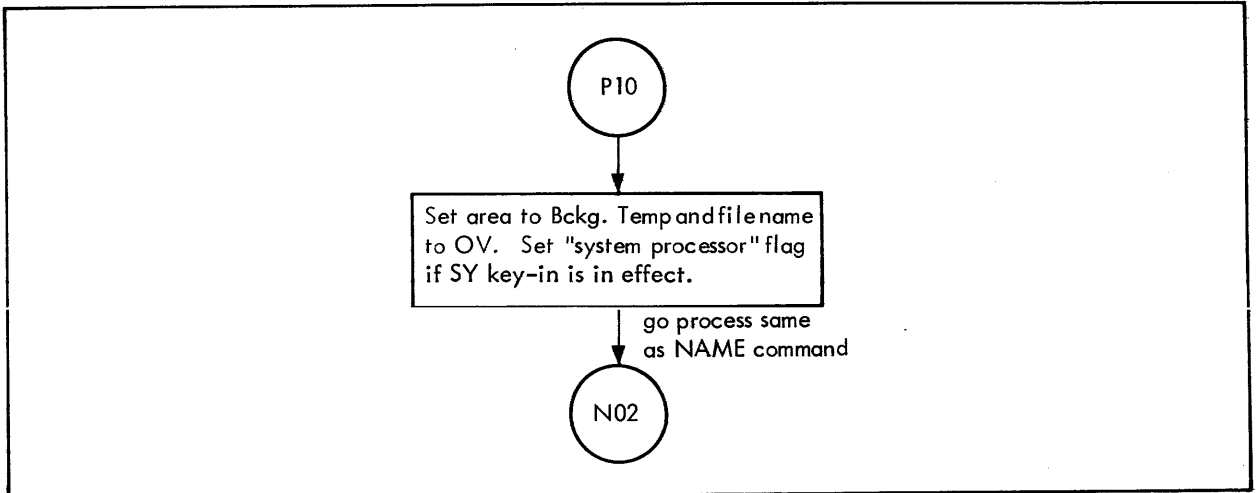


Figure 42. ROV Command Flow

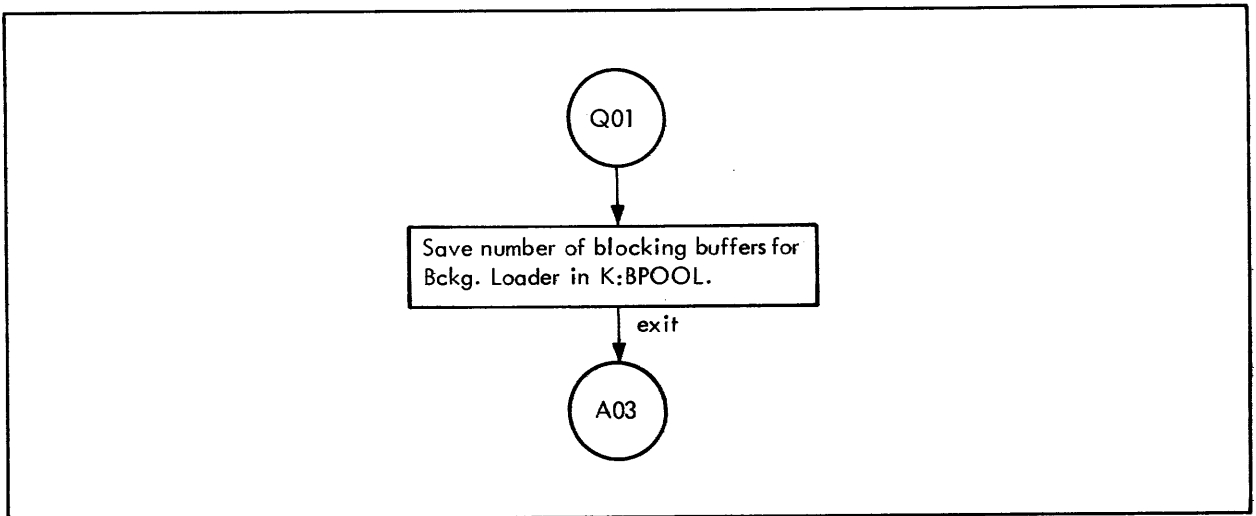


Figure 43. POOL Command Flow

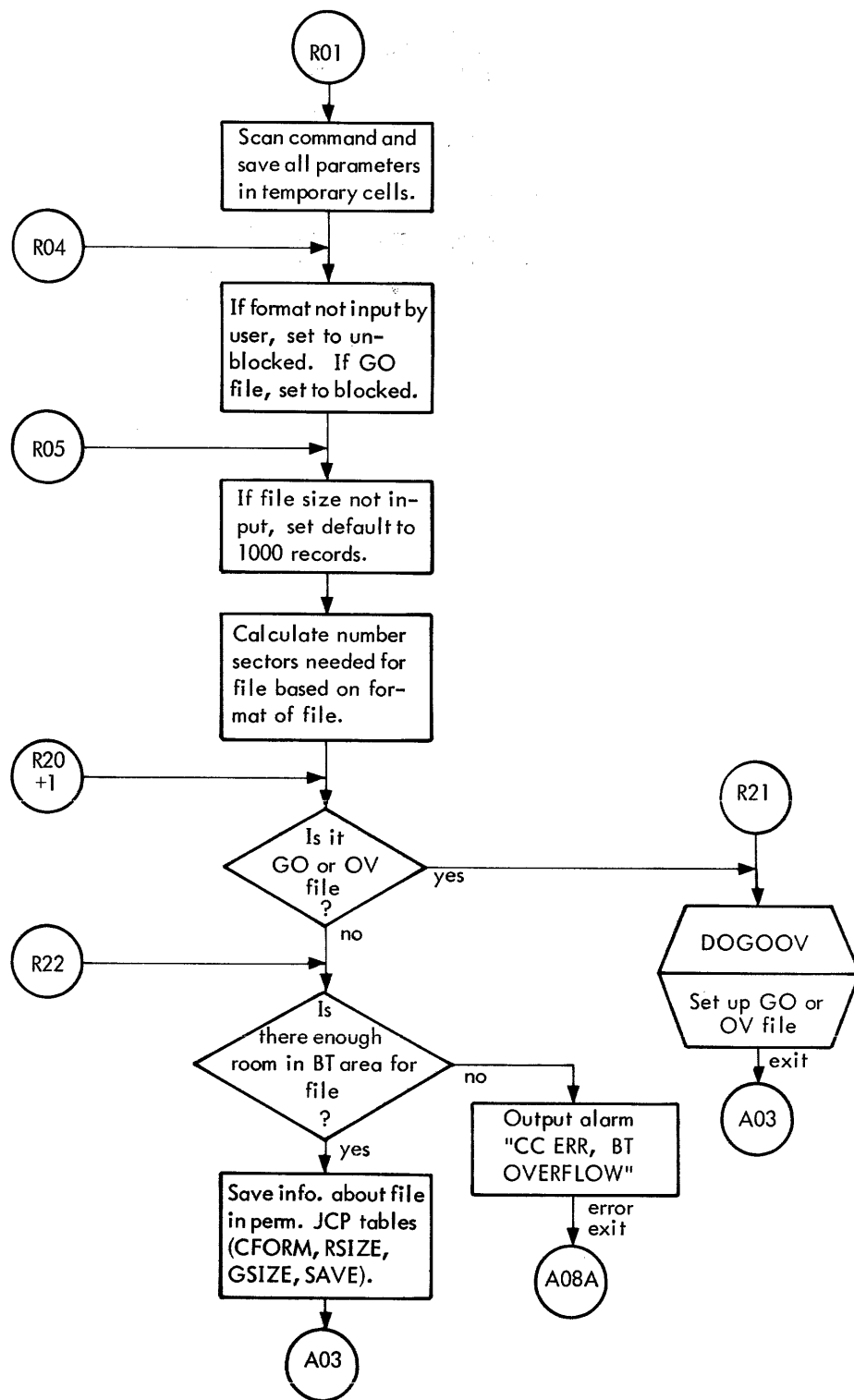


Figure 44. ALLOBT Command Flow

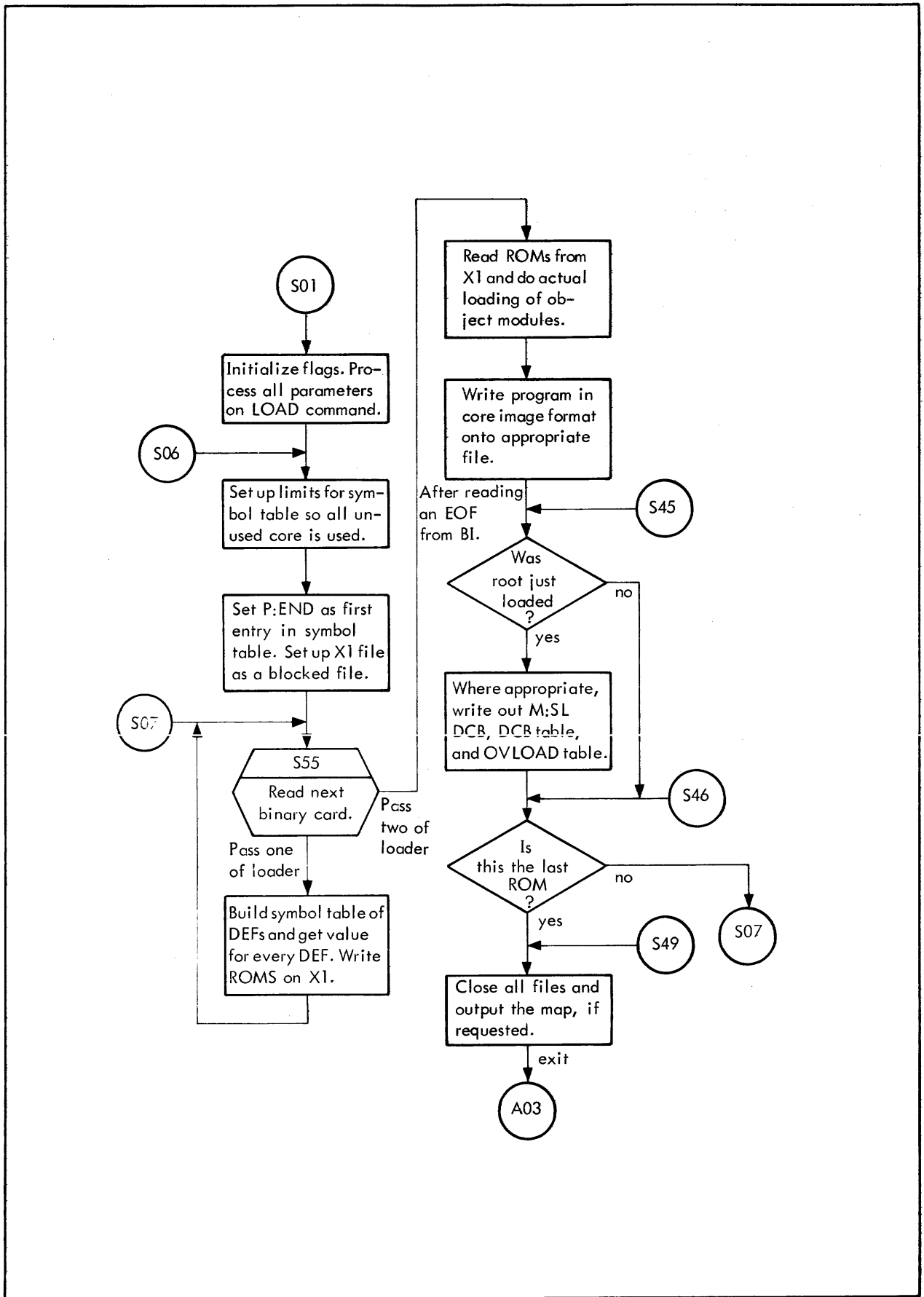


Figure 45. LOAD Command Flow

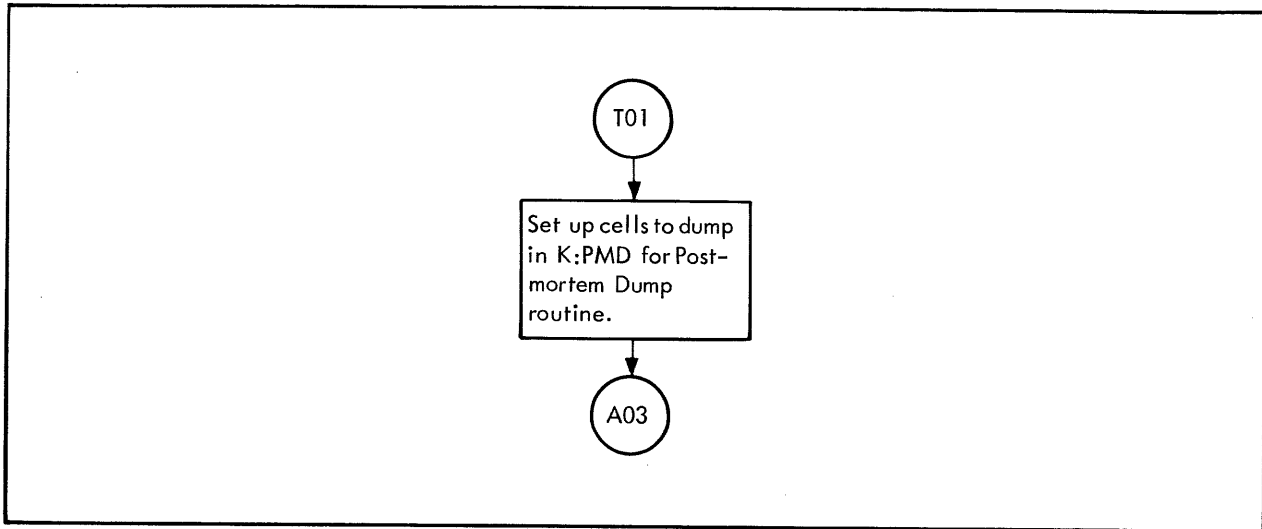


Figure 46. PMD Command Flow

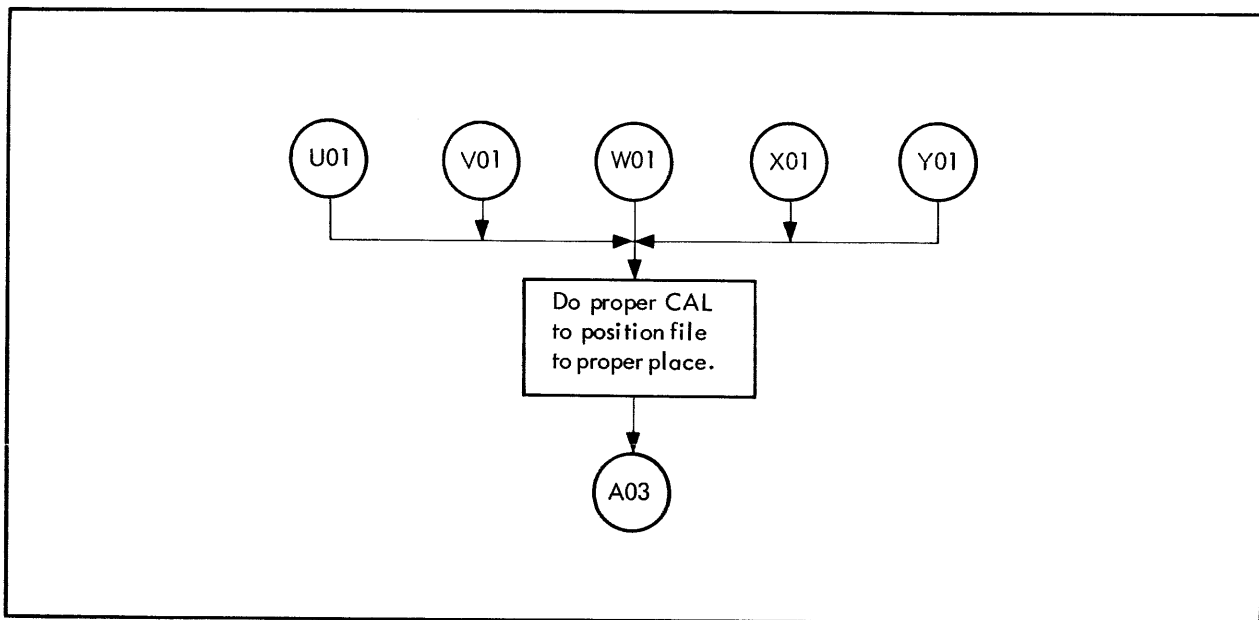


Figure 47. PFIL, PREC, SFIL, REWIND, and UNLOAD Command Flows

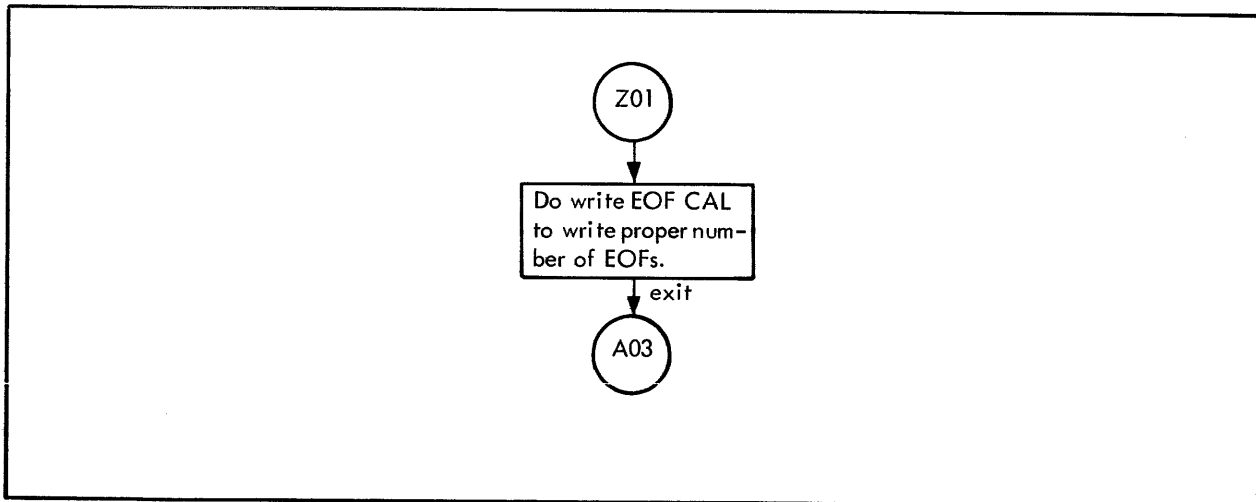


Figure 48. WEOF Command Flow

The diagram in Figure 49 depicts the core layout as the JCP executes.

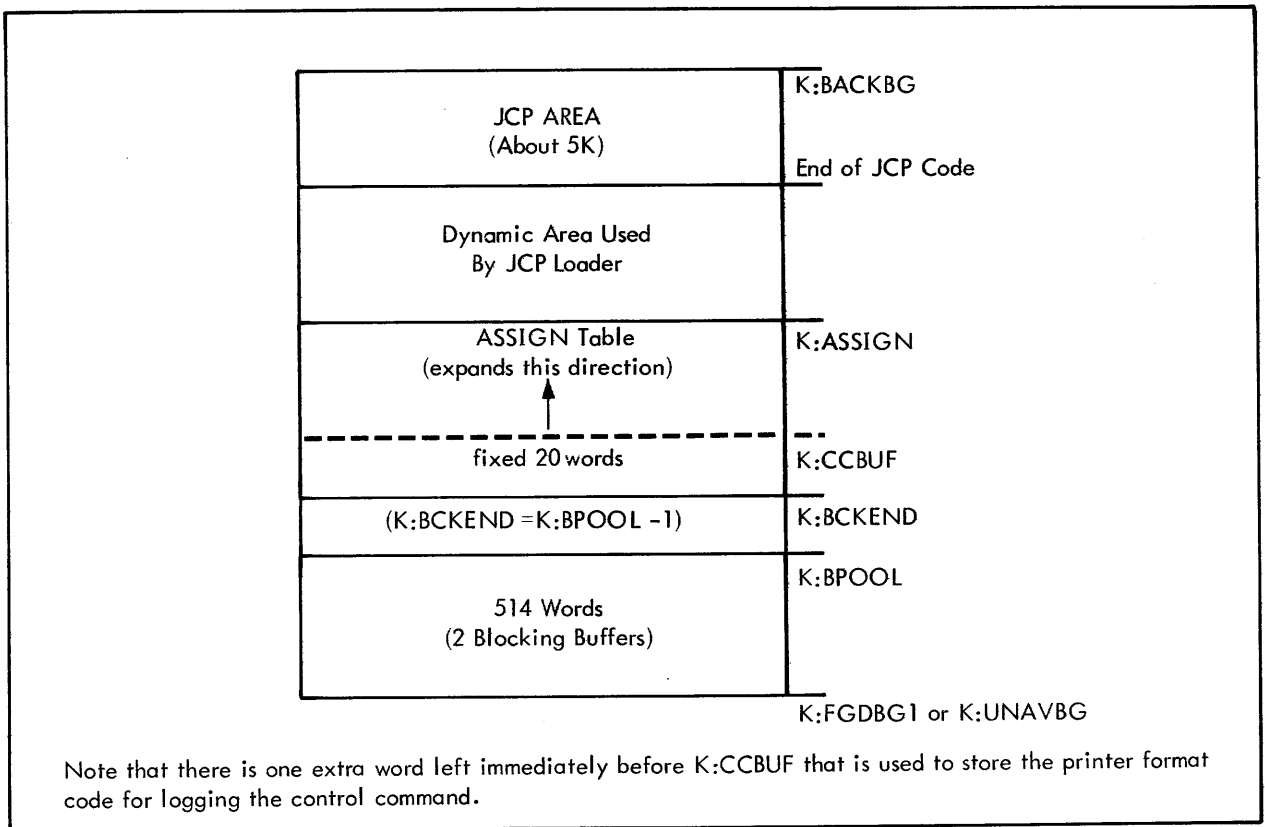


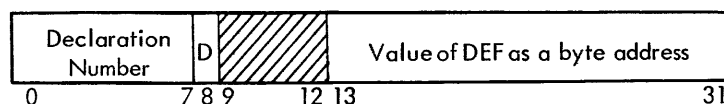
Figure 49. Core Layout During JCP Execution

JCP Loader

The JCP Loader loads Relocatable Object Modules (ROMs) or groups of object modules that use a subset of the XDS Sigma 5/7 Object Language. Initially, the Loader processes all parameters on the !LOAD command and sets up the appropriate DCBs and flags. If the program being loaded has overlays, space is reserved for the program's OVLOAD table at the end of the JCP. The OVLOAD table contains 11 words for each overlay; the first word of OVLOAD contains the number of entries in the table. The exact format of the OVLOAD table is given later in this chapter. Note that words 2 through 10 of the OVLOAD table have the same format as the Read FPT that is needed to read an overlay into core. Next, the first word addresses of the Symbol table (SYMT1 and SYMT2) are set up. The diagram in Figure 50 depicts the core layout before PASS1 of the JCP Loader.

The JCP Loader is a two-pass loader. In Pass1, the ROMs are input from the BI op label and copied onto the X1 file on the RAD. The X1 file is set up to use all of the Background Temp area of the RAD that is available for scratch storage. The main function of PASS1 is to build the symbol table (SYMT1 and SYMT2) containing all DEF items, and to assign a value to each DEF. The symbol table has the following format:

- SYMT1 a doubleword table containing the name, in EBCDIC, of each DEF item in the program being loaded. The first entry is not used.
- SYMT2 a word table in the format shown below. The first entry contains the total number of DEFs in the table as



where bit 8 = 1 if this is a duplicate DEF.

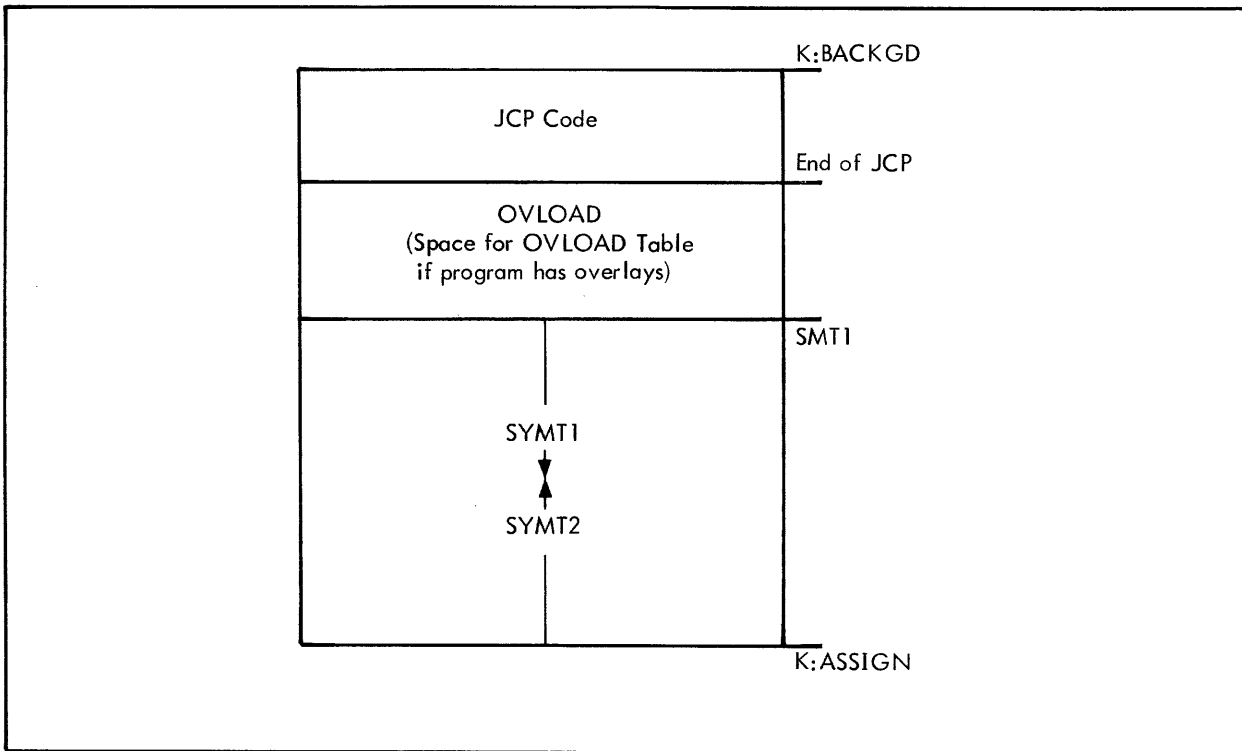
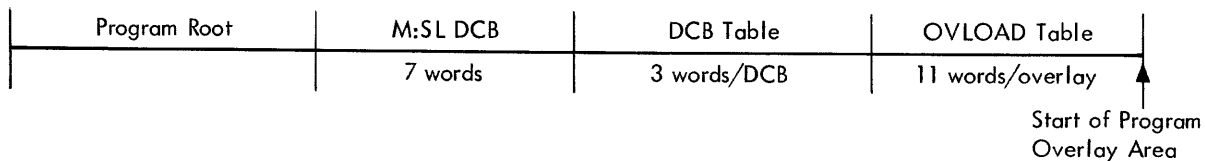


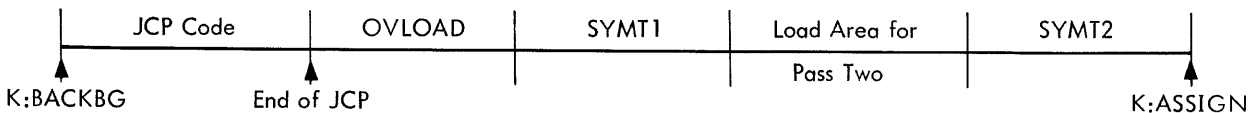
Figure 50. Pre-PASS1 Core Layout

At the end of PASS1, the size of the symbol table is fixed so the remainder of core can be used as a load area in PASS2. After loading the program root in PASS1, space is allocated for the M:SL DCB (if the program has overlays), the DCB table, and the OVLOAD table (if the program has overlays). These items are allocated in the following order:



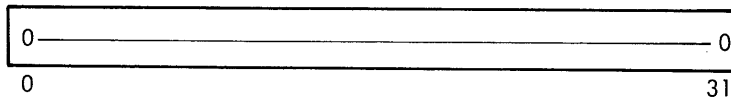
The DCB table is built in an internal table in the JCP in PASS1 after loading the program root. The DCB table is made up of all M: and F: DEFs in the root, including the value of each DEF. The complete OVLOAD table is also built during PASS1; each overlay's entry being made after the overlay is loaded. Hence, PASS1 completely allocates all space for the program.

After the last ROM is loaded at the end of PASS1, the file header is written to the appropriate RAD file. The remainder of core not used by the Symbol table is then rounded down to an even multiple of RAD granules and set up as the load area for PASS2. There must be enough room to hold at least one RAD granule, plus 12 extra words, or the load will be aborted at this point. The X1 file is then rewound and PASS2 commences. The following diagram depicts the core setup at the start of PASS2:

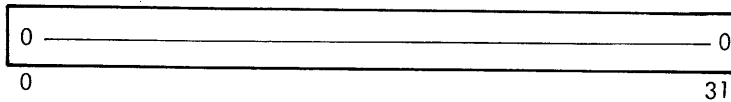


PASS2 inputs the ROMs from the X1 file, satisfies all external REFs by finding the value of the corresponding DEF in the Symbol table, and then writes the program in core image format to the proper RAD file in a multiple of granules at a time. Between 8 and 12 extra words are loaded each time at the end of the load area in case a define field load item requires that the load location be backed up a maximum of 8 words. This prevents having to read a granule back into core after it has been written in the event a word has to be changed because of a define field item.

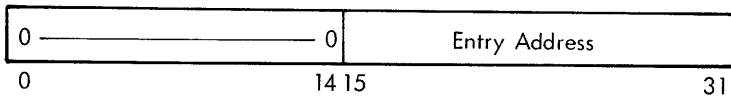
9



10



11



Job Accounting

Job accounting is an option selected at SYSGEN time. An accounting file will be kept on the RAD by the JCP if the accounting option was chosen. The file must be defined by the user; must have the name "AL"; and must be in the D1 area of the RAD.

Whenever a !JOB or !FIN command is read by the JCP, the JCP will update the AL file for the previous job. The format and record size of the AL file is automatically set by the JCP via a File Mode CAL. The JCP defines the AL file as a blocked file with a record size of 32 bytes. The AL file on the RAD consists of a series of eight-word records, where a new eight-word record is added for each job. The first record in the file is reserved for the IDLE account and is the only record that is ever rewritten. The elapsed time in the IDLE account is incremented by the appropriate amount anytime a !JOB command is input after a prior !FIN command, and the IDLE entry is then rewritten on the RAD. The format of each record in the AL file is as follows:

| <u>Word</u> | <u>Description</u> |
|-------------|---|
| 1,2 | Account number in EBCDIC |
| 3,4,5 | Name in EBCDIC |
| 6 | Left halfword = (year - 1900) in binary, Right halfword = date as day of year (1 - 365) |
| 7 | Start time of job in seconds (0 - 86399) |
| 8 | Elapsed time of job in seconds |

The IDLE account has an account number of "IDLE" and a name consisting of all EBCDIC blanks.

Whenever an entry is added to the AL file, the file is opened and a file skip performed so that the new entry can be made at the end of the existing entries. No attempt is made to combine entries in any way. The contents of the AL file can be listed via the !DAL command, (Dump Accounting Log), and the option exists for the user to purge the file after the dump is completed. The AL file is purged by rewinding it and writing an EOF.

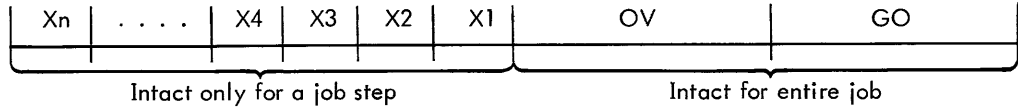
Background TEMP Area Allocation

The JCP allocates and sets up the files in the Background Temp (BT) area (X1-X9, GO, OV) before exiting to the Background Loader to load a processor or user program. The BT files needed by the user are defined either via !ALLOBT commands or through default by the JCP from inspection of the user's DCBs. The GO and OV files are set up at the start of each job and remain intact for an entire job; the required files X1 through X9 are normally set up for each job step only.

Information for files X1-X9 read in from !ALLOBT commands is stored in tables (Gsize, Fsize, FORM, SAVE, Rsize) that are internal to the JCP. If the GO or OV file is changed via an !ALLOBT command, the file is redefined at the time the command is processed.

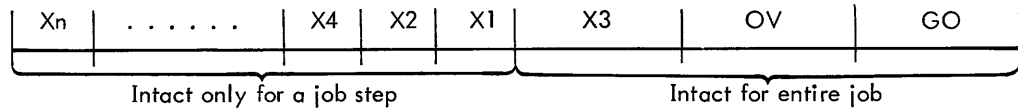
The files in the BT area are allocated so that files remaining intact only for that job step are allocated at the front of the BT area. Files that remain intact for the entire job are allocated at the back of the BT area. Normally, this means that X1 through X9 are allocated at the front of the BT area, and GO and OV at the opposite end. If the SAVE option is used on an !ALLOBT command for an Xi file, the Xi file will be allocated at the opposite end of the BT area, as will GO and OV. The following diagrams illustrate the BT allocation:

BT allocation without !ALLOBT Commands:



The proper Xi file is allocated for each M:Xi DCB in the user program. The remainder of the BT area after GO and OV have been allocated is evenly divided among the Xi files.

BT allocation with !ALLOBT Command:



The above diagram illustrates how BT would be allocated if an !ALLOBT command was input to save the X3 file. Note that X3 is allocated at the opposite end of the area with OV and GO.

-

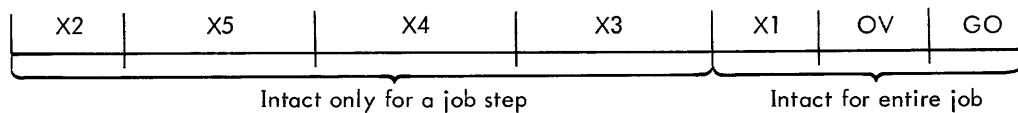
Allocation of the Xi (1 ≤ i ≤ 9) files is performed in the following sequence: First, any files input on an ALLOBT command are allocated at the proper end of the BT area. Next a search is made of all user M:Xi DCBs, and any Xi files that were not input on an ALLOBT command are allocated by default in the remaining area. Note that if the "ALL" option is used for file size in the ALLOBT command, there will be no room remaining for default allocations and if a M:Xi DCB is found for which a file has not been allocated, a "BT OVERFLOW" alarm will be output and the job aborted.

The following examples depict the allocation of BT as previously described:

Example 1:

1. An !ALLOBT command for X1 file with SAVE option.
2. An !ALLOBT command for X2 file.
3. A user program with M:X1, M:X2, M:X3, M:X4, and M:X5 DCBs.

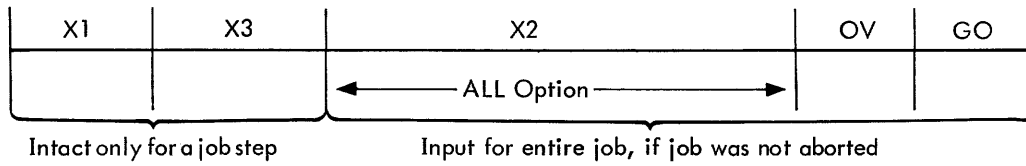
In this case, the BT area would be allocated as



In this example, the X1 and X2 files would receive the sizes input on the !ALLOBT command, while the X3, X4, and X5 files would be evenly distributed over the remaining area.

3. A user program with M:X1, M:X2, M:X3, and M:X4 DCBs.

The BT area in this case would be allocated as



In this example, the job would be aborted because there is no remaining room to allocate the M:X4 DCB, since the "ALL" option was used for the X2 file. If the "ALL" option is used for file size, all Xi files used by the program must be allocated via the !ALLOBT command.

The JCP does special allocation of the BT area for MACRSYM, since MACRSYM may or may not need an X2 file, depending upon the parameters on the !MACRSYM control command. Also, MACRSYM requires that the area for its BT files be divided unevenly between the X1, X2, and X3 files. The N95 table in the JCP contains the ratios to use in dividing the BT area for MACRSYM. If special allocation had to be done for another processor, it would only require the addition of another table similar to N95 and a special check for that processor's name. The N96 table is used for all processors with no special allocation requirements.

5. FOREGROUND SERVICES

Foreground services are those service functions restricted to foreground utilization. In general, they are associated with the control of system interrupts, the handling of foreground tasks, and direct I/O (IOEX). The following service functions fall in this category:

RUN
RLS
MASTER/SLAVE
STOPIO/STARTIO
IOEX
TRIGGER
ENABLE/DISABLE
ARM/DISARM
CONNECT

In terms of the functions as part of the resident RBM, the resident function sets indicators for RUN and RLS, and the Control Task actually performs the function.

Implementation

RUN If an entry for the specified program does not already exist in the FP table, an entry is built. The FP subtables are set as follows:

| | |
|-----|--|
| FP1 | Program name |
| FP2 | Group code for interrupt to be triggered at conclusion of initialization by Control Task |
| FP3 | Group level for said interrupt |
| FP4 | Signal address and (optionally) priority |
| FP5 | Switches |

K:FGLD is set nonzero, the Control Task is entered, and control is returned to the user program.

If an entry does exist in the table for the program, a code is placed in the signal address. The codes used are

| | |
|---|------------------------------|
| 3 | Program already loaded |
| 4 | Program waiting to be loaded |

If no entry exists for the program and there are no free entries in the FP table, a code of 5 is placed in the signal address. Sufficient reentrance testing is performed (for details, see the program listing).

RLS If an FP entry does not exist for the specified program, control is returned to the user.

If an entry exists and the program is not loaded, FP1 and FP5 are zeroed, and control is returned to the user.

If an entry exists and the program is loaded, bit 3 in FP5 is set, K:FGLD is set nonzero, the Control Task is triggered, and control is returned to the user (for details of reentrance testing, see the program listing).

MASTER/SLAVE The mode bit in the PSD saved in the user Temp Stack is set to the proper state and control is returned to the user. When returning control, CALEXIT executes an LPSD that establishes the proper mode for the user.

STOPIO/STARTIO The specified device is determined and all other devices associated with it (all other devices on a multidevice controller or all devices on the IOP if the call so requests) have their proper STOPIO counts incremented or decremented. The count is either in DCT14 or DCT15 as specified by the call.

An HIO is performed on these devices if requested by the call.

If a DCT15 count goes to zero as a result of a decrement, the IOEX busy bit in DCT5 (bit 7) is reset for the device.

IOEX For HIO, TIO, and TDV instructions, the instruction is executed and the status is placed in the copies of R8, and R9. The condition code field of the saved PSD is placed in the Temp Stack. Then at CALEXIT, these copies are placed in R8, R9, and the PSD, and returned to the user.

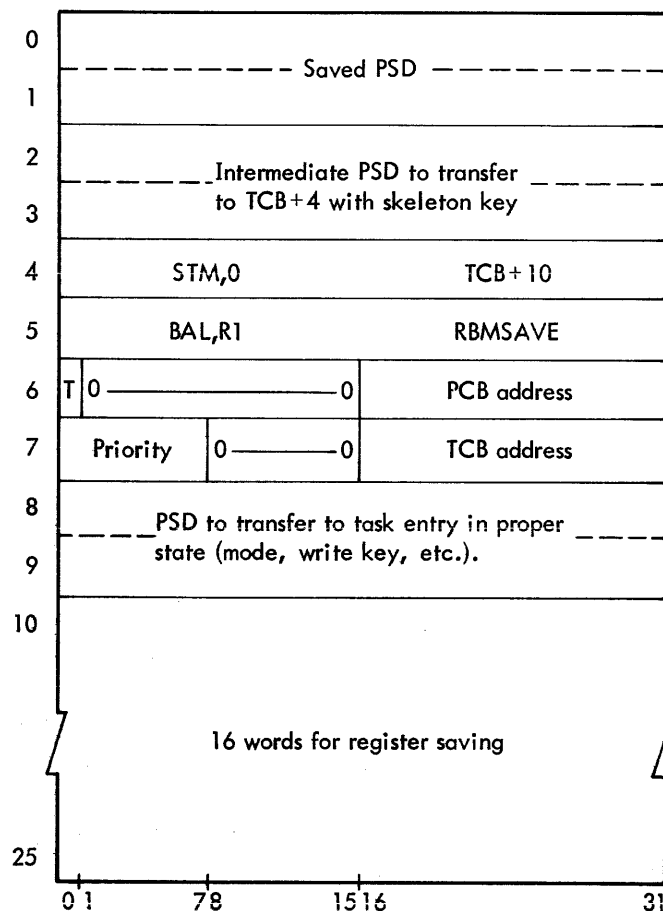
For SIO, the system waits until the device is not busy with regular system I/O (DCT5, bit 0); it then sets the IOEX busy bit (DCT5, bit 1), executes the SIO, and returns the status to the user.

TRIGGER, DISABLE, ENABLE, ARM, DISARM, CONNECT These functions are similar in that they involve the execution of a Write Direct after determining the group code and group level of the specified interrupt.

In addition, a task connection is performed if requested by ARM, DISARM, and CONNECT requests. Note that the CONNECT call is a special case of the ARM call. The logic for ARM, DISARM, and for CONNECT functions is illustrated in Figure 51.

Task Control Block (TCB)

The CONNECT function initializes words 2-9 of the user-allocated TCB for interrupts and CALs that are to be centrally connected. The format of the TCB is shown below:



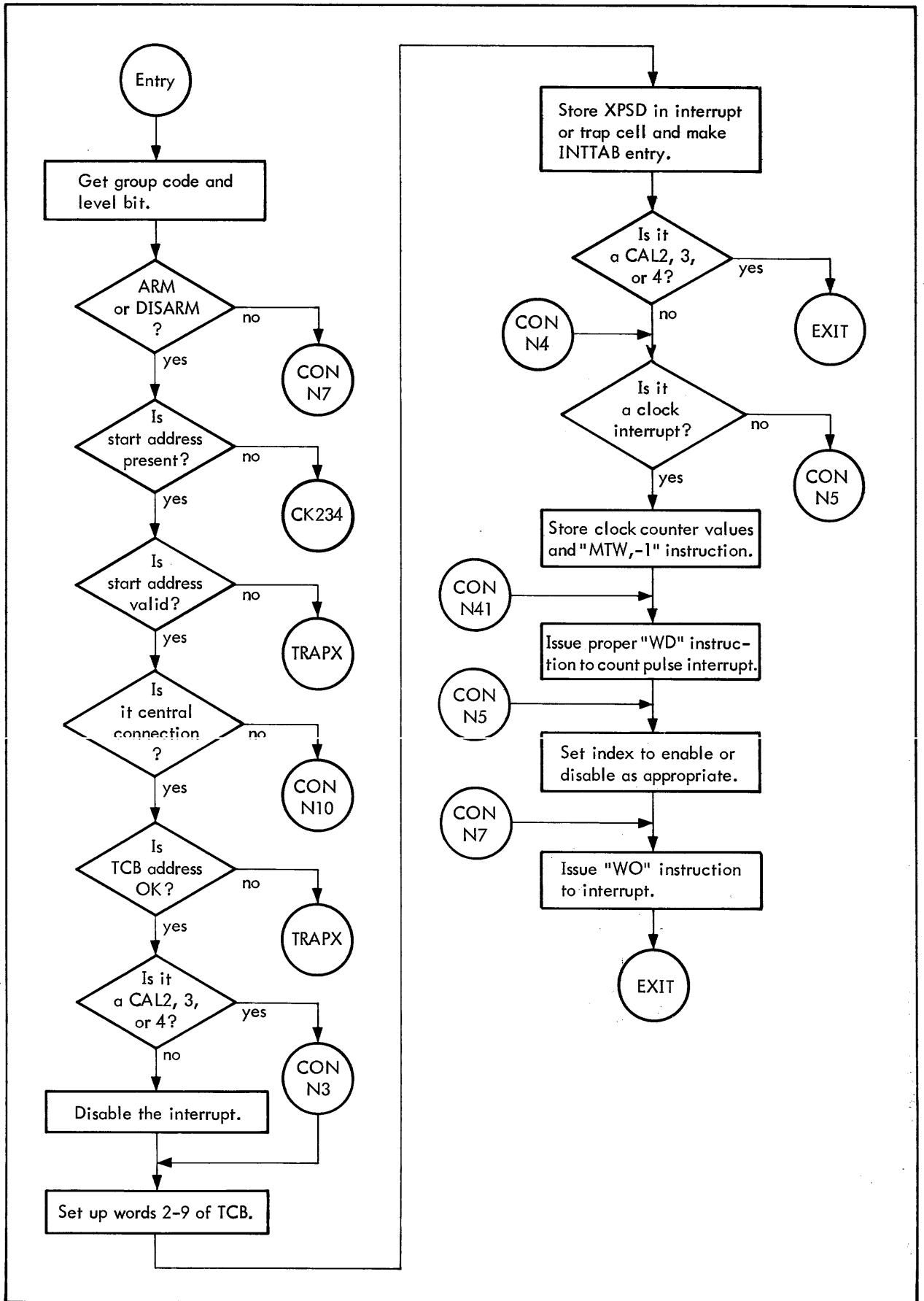


Figure 51. ARM, DISARM, and CONNECT Function Flow

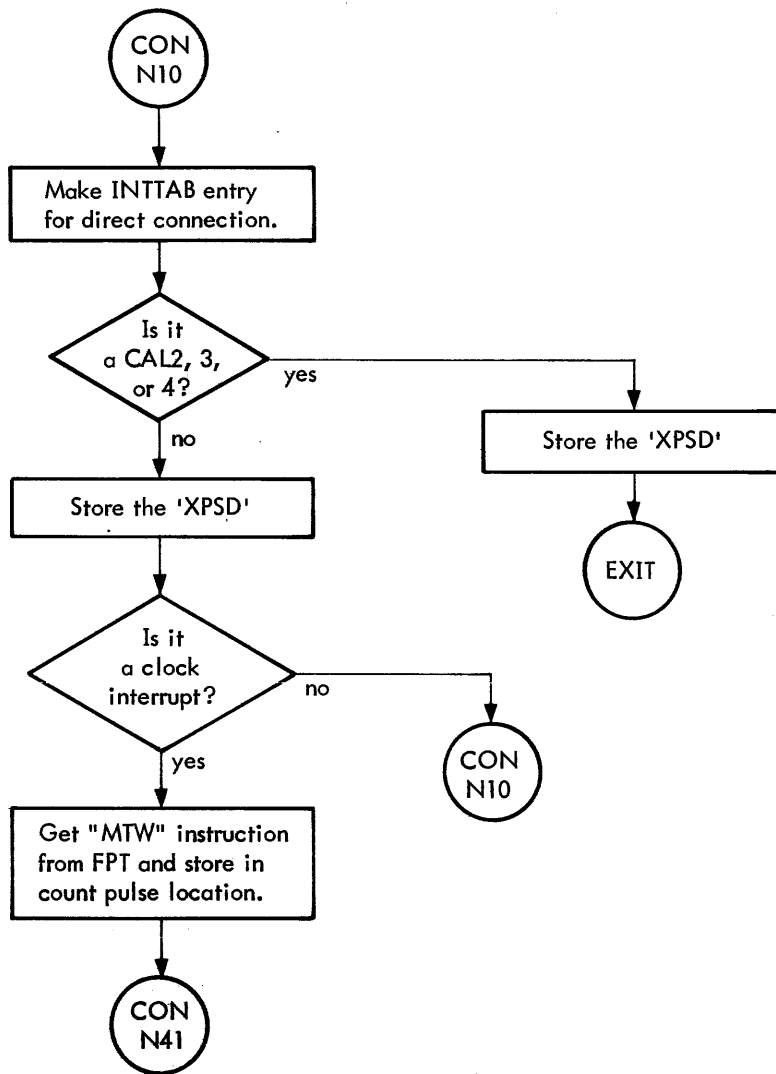


Figure 51. ARM, DISARM, and CONNECT Function Flow (cont.)

6. MISCELLANEOUS SERVICES

Miscellaneous services are functions available to both foreground and background programs but which do not directly involve I/O services.

SEGLOAD

This function loads explicitly requested overlay segments of a program into memory for execution. The user's M:SL DCB (allocated by the Overlay Loader) is used to perform the input operation.

For an FPT for READWRIT, the system uses the entry in the program OVLOAD table that corresponds to the segment. The OVLOAD table is constructed by the Overlay Loader.

The function locates the proper entry in the OVLOAD table and places the user-provided error address in both the OVLOAD entry (FPT) and in the M:SL DCB. If end-action was requested, the FPT is set to cause end-action at conclusion of the segment input.

If the calling program has requested that the segment be entered (at its entry point), the PSD at the top of the user Temp Stack is altered so that upon CALEXIT, control goes to the segment entry address.

The function then sets R0 to point at the FPT in the OVLOAD table and transfers to READWRIT. The segment input is then treated as a READ request with possible end-action, and at the user's option, control is returned either following the SEGLOAD CALL, or to the segment entry address.

Trap Handling

Trap CAL

This function sets up the trap control field and TRAPADD field in a user's PCB and sets the Decimal Mask (DM) and Arithmetic Mask (AM) bits in the user PSD to mask out occurrences of these traps. PSD bits are modified by changing them in the user PSD at the top of the Temp Stack and in the PSD contained in the user's TCB.

If the user-provided trap address is invalid (not in background for background program, or in foreground for foreground user), or if the user specifies that he is to receive occurrences of some trap and no trap address is provided, control is transferred to TRAPX. This results in the message

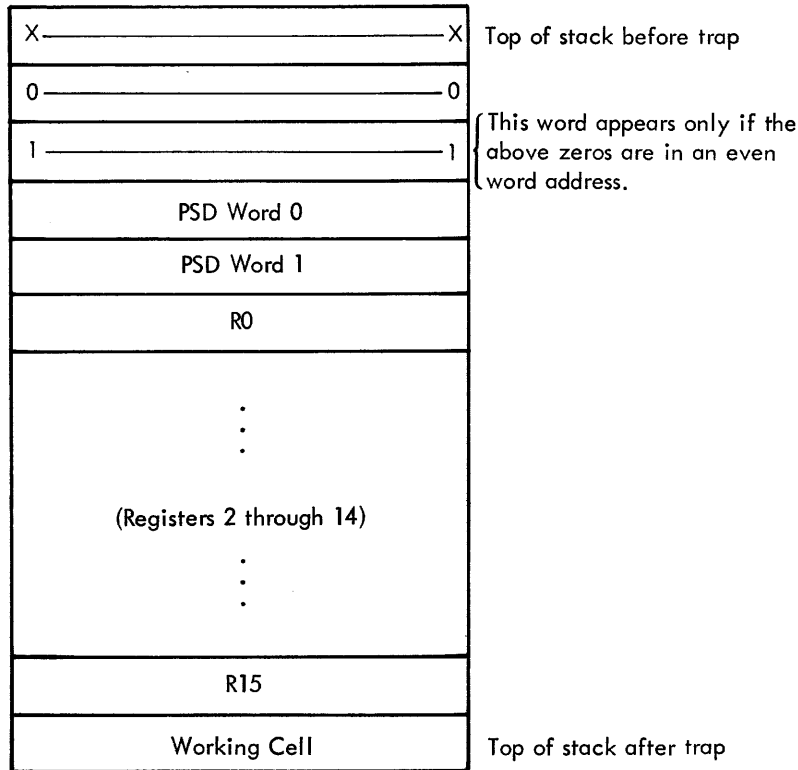
ILL PARAM., CAL AT XXXXX

being output on OC and LL.

Trap Processing

Traps are either handled by the user, cause simulation of the instruction where possible, or result in an abort condition.

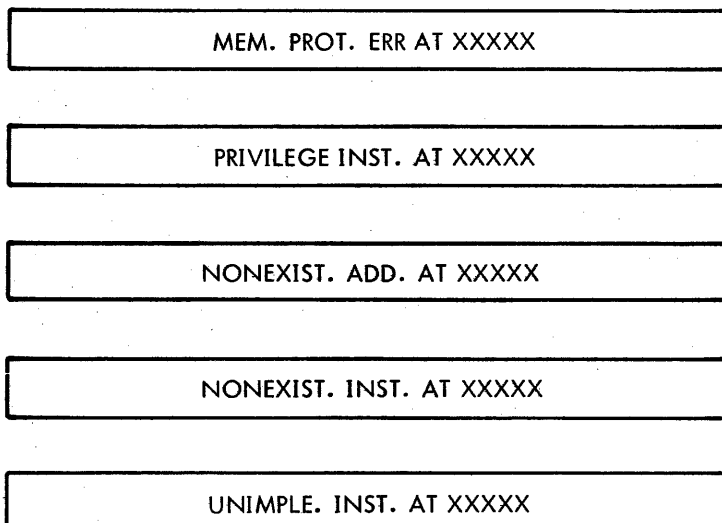
The registers and PSD are saved in the user Temp Stack in the following format:



If the trap is either a nonexistent instruction or unimplemented instruction, the instruction causing the trap is analyzed to determine whether the proper simulation package (if any) is in the system. If so, the simulation is called; if not, it is treated like any other trap.

A test is performed to determine whether the user is to process this particular trap. If so, the trap address (X'40', X'41', etc.) is placed in the top word of the stack and the user's trap handling routine is entered by LPSD, eight of the user PSD, with the trap handler substituted for the address where the trap occurred.

Traps not handled by instruction simulation or by the user result in one of the following messages being output to OC and LL:



STACK OVERFLOW AT XXXXX

ARITH. FAULT AT XXXXX

WDOG TIMER RUNOUT AT XXXXX

MEM. PARITY ERR AT XXXXX

ILL. PARAM., CAT AT XXXXX

Note that the last message results from the simulation of a trap (called Trap X'50'). This is done by the system when a system call cannot be processed due to incorrect parameters being input. After the message is output, a test is performed to determine whether the trap occurred in the background or foreground. If background, an ABORT function is performed; if foreground, the current task is exited.

TRTN (Trap Return)

This function returns control following the instruction which caused a trap and is employed by the user to return control after processing a trap.

At the time of the TRTN call, the user Temp Stack is set as described previously under "Trap Processing". The TRTN function strips the stack of the context placed there by the CAL processing (from the TRTN CAL). It then clears the stack by the Trap processor and returns control to the instruction that follows the one causing the trap.

7. RBM SIZES

The total size of RBM, including all handlers and excluding the initialization routine, is 5170 words. The size of the minimum RBM (card reader and RAD handler options only) is 3823 words. The length of the RBM overlay area is 512 words.

The space required for each optional feature and optional routine is:

| <u>Option</u> | <u>Words Required</u> |
|---------------------------------|-----------------------|
| Patch | 0 |
| Compressed Files | 158 |
| IOEX | 54 |
| Job Accounting | 65 |
| Card Reader | 23 |
| Card Punch | 82 |
| Card Punch (Low Cost) | 21 |
| Magnetic Tape | 240 |
| Paper Tape | 225 |
| Line Printer | 18 |
| Line Printer (Low Cost) | 24 |
| Plotter | 13 |
| RAD | 43 |
| Disk Pack | 278 |
| SYSPROC | 66 |
| Run Queuing | 35 |
| Sigma 9 Compatibility | 25 |
| GDL Compatibility | 30 |
| Instruction Simulator Interface | 121 |
| Floating Point Simulator | 246 |
| Decimal Simulator | 538 |
| Byte String Simulator | 102 |
| Convert Simulator | 56 |
| Delta | 3196 |

8. RBM TABLE FORMATS

RAD File Table (RFT)

Parameters describing the file are taken from the directory entry for the file. These parameters include:

- File name
- Beginning sector address (relative to beginning of the area)
- Ending sector address (relative to beginning of the area)
- Granule size
- Record size
- File size (number of records)
- Organization (blocked, unblocked, compressed)

The parameters specifying the physical characteristics of the RAD, the boundaries of the RAD area, and the Write Protection key are in the Master Dictionary. To enable access to these, the RFT contains a Master Dictionary Index (specifying the area).

For manipulation of the file, the RFT contains the following items:

- Blocking buffer control word address
- Blocking buffer position
- Position within the file (sector last accessed – used for blocked and unblocked)
- Current record number
- Number of DCBs open to the file.

These parameters are entered in the RFT by the OPEN function. The parallel table concept is used for the RFT, and the tables are allocated and initialized as given in Table 2.

In Table 2, below

| | |
|-----------------|--|
| File name all 0 | Signifies entry not in use. |
| RFT2 index 0 | Entry contains the total number of RFT entries. |
| RFT3 index 0 | Entry contains the maximum number of RFT entries allowed for background use. |
| RFT4 index 0 | Entry contains the current number of background file entries. |
| RFT5 index 0 | Entry is used as the RFT activity count for reentrance tests. |
| RFT6 index 0 | Entry contains the number of temp files allocated. |
| Other index 0 | Entries are not used. |

The Job Control Processor builds the RFT entries for the Background Temp Files. These entries are the first $n + 2$ in the table (n is the number of X_i files), where entry 1 is for the OV file, entry 2 is for the GO file, entry 3 is for the XI file, etc.

Table 2. RAD File Table Allocation

| Address | Contents | Initial Value | Length |
|---------|---|---------------|------------|
| RFT1 | File Name | 0 | Doubleword |
| RFT2 | Beginning Sector Address | X | Halfword |
| RFT3 | Ending Sector Address | X | Halfword |
| RFT4 | Granule size (in bytes) | X | Halfword |
| RFT5 | Record size (in bytes) | X | Halfword |
| RFT6 | File Size (in records) | X | Halfword |
| RFT7 | <u>Switches</u> where Bit 0 = 1 means sequentially written Bit 1 = 1 means directly written Bit 3 = 1 means compressed Bit 7 = 1 means blocked | X | Byte |
| RFT8 | Master Dictionary Index | X | Byte |
| RFT9 | Not used | X | Byte |
| RFT10 | Blocking Buffer Position (in bytes) | X | Halfword |
| RFT11 | File Position (in sectors) | X | Halfword |
| RFT12 | Current Record Number | X | Halfword |
| RFT13 | Number of Open DCBs (total) | X | Byte |
| RFT14 | Function | X | Byte |
| RFT15 | Number of BGND DCBs | X | Byte |
| RFT16 | Status (bit 0 on for sequential write, bit 1 on for direct access write) | X | Byte |
| RFT17 | Blocking Buffer Control Word Address | X | Word |

Device Control Table (DCT)

DCT Format

The Device Control Table (DCT) is composed of several parallel subtables (see Table 3, below). The various entries associated with a given device are accessed using the DCT index of the device and addressing the tables DCT1 through DCT19. For example, DCT1 would be accessed by

LH, R DCT1, X

DCT2 would be accessed by

LB, R DCT2, X

where Register X contains the DCT index value for the device.

Table 3. DCT Subtable Formats

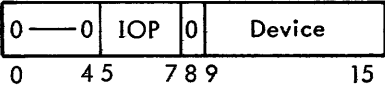
| Subtable Address | Contents | Length |
|------------------|---|----------|
| DCT1 | I/O Address of the device  | Halfword |
| DCT2 | Channel Information Table Index - A pointer to the CIT entry for the channel associated with the device. | Byte |
| DCT3 | Bit 0 = 1 means output is legal for this device. Bit 1 = 1 means input is legal for this device. Bit 2 = 1 means device has been marked down and is inoperative. Bit 3 = 1 means device timed out. Bit 4 = 1 means SIO has failed. Bit 5 = unused Bit 6 = DED DPnnd, R key-in in effect Bit 7 = unused | Byte |
| DCT4 | <u>Device Type</u> 0 = NO (IOEX) 1 = TY 2 = PR 3 = PP 4 = CR 5 = CP 6 = LP 7 = DC 8 = 9T 9 = 7T 10 = CP (Low Cost) 11 = LP (Low Cost) 12 = DP 13 = PL | Byte |
| DCT5 | <u>Status Switches</u> Bit 0 = device busy. Bit 1 = waiting for cleanup. Bit 2 = between inseparable operations. Bit 3 = data being transferred. | Byte |

Table 3. DCT Subtable Formats (cont.)

| Subtable Address | Contents | Length |
|------------------|--|------------|
| DCT5 (cont.) | <p>Bit 4 = error message given (key-in pending).</p> <p>Bit 5 = control task notified of deferral of processing for this device.</p> <p>Bit 6 = SIO was given while device was in manual mode.</p> <p>Bit 7 = IOEX on this device.</p> | |
| DCT6 | Pointer to queue entry representing current request. | Byte |
| DCT7 | Command list doubleword address. | Halfword |
| DCT8 | Handler start address. | Word |
| DCT9 | Handler cleanup address. | Word |
| DCT10 | Device activity count (used for I/O Service reentrance testing). | Halfword |
| DCT11 | Time-out value (used to abort request when no interrupt occurs). | Word |
| DCT12 | Cleanup word 2 before I/O interrupt, AIO status after the interrupt. | Word |
| DCT13 | TDV status. (Halfword 2 has type completion if cleanup requires key-in.) | Doubleword |
| DCT14 | STOPIO (background only) count. | Byte |
| DCT15 | STOPIO (all system I/O) count. | Byte |
| DCT16 | The first eight characters of the operator message. It contains the five-character device name (CRA03) preceded by the three characters "N/L!!". | Doubleword |
| DCT17 | Retry function code (for error recovery) and continuation. | Halfword |
| DCT18 | Open DCB count (total). | Byte |
| DCT19 | Open DCB count (background). | Byte |

SYSGEN DCT Consideration

System Generation allocates the space for the subtables DCT1-DCT19. Initial values are defined for these entries as follows:

- DCT1 As specified by :DEVICE command
- DCT2 As specified by :DEVICE and :SIOP commands
- DCT3 As specified by :DEVICE command
- DCT4 As specified by :DEVICE command
- DCT5 Zero
- DCT6 Zero

| | |
|-------|--|
| DCT7 | Pointer to SYSGEN allocated space for command list |
| DCT8 | Zero |
| DCT9 | Zero |
| DCT10 | Zero |
| DCT11 | Zero |
| DCT12 | Zero |
| DCT13 | Zero |
| DCT14 | 1 if (DEDICATE, F); otherwise, zero |
| DCT15 | 1 if (DEDICATE, X); otherwise, zero |
| DCT16 | "N/L!!YYNDD" where YYNDD came from the :DEVICE command |
| DCT17 | Zero |
| DCT18 | Zero |
| DCT19 | Zero |

The index 0 entry of each subtable is not used as a true table entry because of the nature of the BDR instruction.

DCT7 points to the space allocated by SYSGEN for the command list for the device. The area must begin on a doubleword boundary and have a word length as follows:

| | |
|---------------------------|----------|
| Magnetic Tape (7T and 9T) | 6 words |
| Keyboard/Printer | 4 words |
| Card Reader | 2 words |
| Card Punch (7160) | 74 words |
| Card Punch (7165) | 2 words |
| RAD | 4 words |
| Disk Pack | 6 words |
| Paper Tape | 8 words |
| Other Devices | 8 words |
| Line Printer (7440, 7445) | 2 words |
| Line Printer (7450) | 4 words |
| Plotter | 2 words |

Halfword 0 of DCT1 is set by SYSGEN to contain the number of devices (DCT entries) in the DCT table.

Channel Information Table (CIT)

The Channel Information Table consists of parallel subtables, each with an entry per channel. There is one channel per controller connected to a MIOP, and one channel per SIOP. The "channel" concept is used since there cannot be more than one data transfer operation in process per channel. I/O device requests are queued on a per-channel basis. System Generation allocates and initializes these subtables as shown below:

| <u>Address</u> | <u>Contents</u> | <u>Size</u> |
|----------------|--|-------------|
| CIT1 | Queue Head set to 0 by SYSGEN | Byte |
| CIT2 | Queue Tail set to 0 by SYSGEN | Byte |
| CIT3 | Switches set to 0 by SYSGEN; Bit 0 Channel busy | Byte |

The CIT subtable entries are accessed by using

LB, R CITN, X

where Register X contains the index (1-N).

The index 0 entry is not used because of the nature of the BDR instruction.

I/O Queue Table (IOQ)

The I/O Queue Table consists of parallel subtables each with an entry per queue entry. These tables are accessed in the same manner as DCT and CIT by using an index. As is true for DCT and CIT, the index 0 entry of each subtable is not used as a true queue entry.

System Generation allocates and initializes the IOQ tables as given in Table 4.

Notice that IOQ2 index 0 is initialized by SYSGEN. This byte is used and maintained by the I/O system as the "free entry pool" pointer. By initializing IOQ2 as shown, SYSGEN links all entries into this pool.

IOQ1 index 0 is initialized by SYSGEN to the maximum number of queue entries allowed to the background.

IOQ3 index 0 is initialized to 0, since this byte is used and maintained by the I/O system as the current number of queue entries in use by background. IOQ4 (index 0) is the total number of IOQ entries.

Table 4. IOQ Allocation and Initialization

| Address | Contents | Initial Value | Length |
|---------|---|---|--------|
| IOQ1 | Backward Link | 0 | Byte |
| IOQ2 | Forward Link | Entry M contains M + 1 for N > M ≥ 0. Entry N contains 0. N is the number of queue entries. | Byte |
| IOQ3 | <u>Switches</u> Bit 0 = 1 means request busy Bit 5 = 1 means continued operation Bit 6 = 1 means reuse queue entry Bit 7 = 1 means operation complete | 0 | Byte |
| IOQ4 | Function Code (:DOT table index) | 0 | Byte |
| IOQ5 | Current Function Step | 0 | Byte |
| IOQ6 | Not used | 0 | Word |
| IOQ7 | Device Index | 0 | Byte |
| IOQ8 | Byte Address of Buffer | 0 | Word |

Table 4. IOQ Allocation and Initialization (cont.)

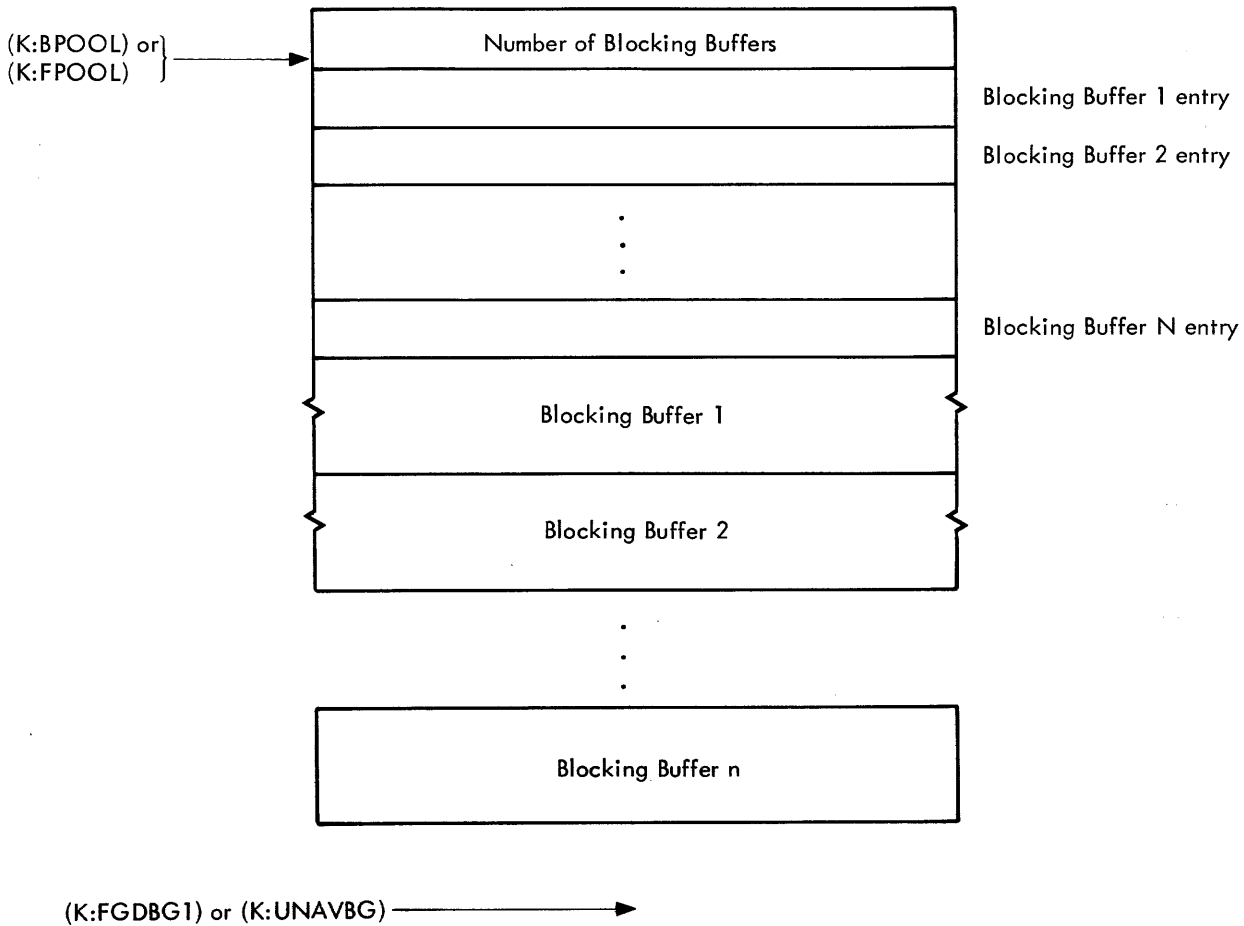
| Address | Contents | Initial Value | Length |
|---------|---|---------------|------------|
| IOQ9 | Byte Count | 0 | Halfword |
| IOQ10 | Maximum retry count | 0 | Byte |
| IOQ11 | Retry count | 0 | Byte |
| IOQ12 | Seek Address | 0 | Word |
| IOQ13 | End-Action data <u>Word 1</u> byte 0 is cleanup code where value: 1 = Post status in FPT 2 = Post status in DCB 3 = Not used 4 = No end action (only available to the monitor) bit 8 = control device read bit 9 = end action data in word 2 bit 15-31 = FPT or DCB address for cleanup code 1 or 2 <u>Word 2</u> If word 2 = 0, parameter not present If byte 0 = X'FF', bits 15-31 are user's end-action address. If word 2 = 0, and byte 0 = X'FF', byte 0 = end-action interrupt group code, byte 1 = interrupt address - X'4F', bits 15-31 contain level bit for interrupt. | 0 | Doubleword |
| IOQ14 | Priority | 0 | Byte |

Since the 0th entry is never used in subtables whose entries are words or doublewords, it is not necessary to allocate space for this entry. If the 2N words for IOQ13 are allocated beginning at location ALPHA, IOQ13 is given value ALPHA-2. Thus, IOQ13 may actually point into another table but presents no problem because IOQ13 will never be accessed with index 0.

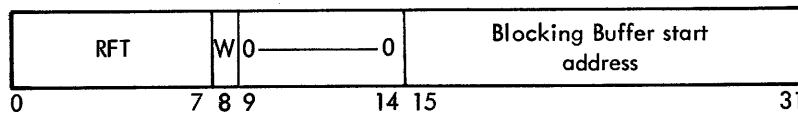
It should be noted that none of the subtables need be positioned in any particular relationship to each other. They may be allocated anywhere in core with the restriction that Doubleword Tables begin on doubleword boundaries.

Blocking Buffers

To facilitate control of blocking buffers, a control table is generated by the buffer allocation mechanism. This table will contain a word entry for each blocking buffer and has the format



where each entry is of the form



RFT is the index of the RFT entry for the file currently using this buffer.
0 signifies that the buffer is not in use.

W is set if the blocking buffer has been written in.

Foreground Program Table (FGT)

The Foreground Program Table contains an entry for each foreground program in memory at a given time. This table consists of parallel subtables allocated by System Generation and is maintained by the system RUN and RELEASE functions (system calls and/or key-ins). The format of the subtables is given in Table 5.

Table 5. Foreground Program Subtables

| Subtable Address | Contents | Initial Value | Size |
|------------------|--|---------------|------------|
| FP1 | Program Name. | 0 | Doubleword |
| FP2 | The interrupt group code and interrupt address for interrupt to be triggered before the program is loaded. The first core location (DW address) after loading. Entry 0 contains the number of table entries. | X | Halfword |
| FP3 | The group level for the interrupt to be triggered before the program is loaded. The last core location (DW address) after loading. | X | Halfword |
| FP4 | Before the program is loaded, bits 0-14 contain the priority-sequence field, and bits 15-31 contain the signal address. After the program is loaded, this word contains up to three indexes (into the FGT table) of public libraries used. For public libraries, this word contains the number of programs using the library (when 0, library is unloaded). | X | Word |
| FP5 | <u>Status Flags</u> Bit 0 = 1 Load is to be performed Bit 1 = 1 Public Library used by FGRND Bit 2 = 1 Public Library used by BGRND Bit 3 = 1 Release is to be performed Bit 4 = 1 Release this Public Library used by BORND Bit 5 = 1 Program is loaded Bit 6 = 1 Program run request queued Bit 7 = 1 Run with Delta | 0 | Byte |

Master Dictionary

K:MASTD (location X'14A'), contains the address of the Master Dictionary. This serial table is indexed by area number where:

| <u>Area</u> | <u>DW Index Value</u> | <u>Write Protect Code (WP shown below)</u> |
|-------------|-----------------------|--|
| SP | 0 | 4 |
| FP | 1 | 4 |
| BP | 2 | 4 |
| BT | 3 | 2 |
| XA | 4 | 5 |
| CK | 5 | 3 |
| D1 | 6 | 1 or 2 (specified during SYSGEN) |
| D2 | 7 | 1 or 2 |
| ⋮ | ⋮ | ⋮ |
| DF | 20 | 1 or 2 |

The format of the Master Dictionary (2 words/entry) is

| | | | | | | | |
|---|-----------------------------------|----------------------|---------------------------------|-----|----|-------------|----|
| 1 | No. sectors per track | No. words per sector | A | 0—0 | WP | DCT Index | |
| 2 | Starting RAD Address [†] | | Ending RAD Address [†] | | | | |
| | 0 | 67 | 15 | 16 | 17 | 20 21 23 24 | 31 |

where

A = 0 this area is not allocated.

A = 1 area is allocated.

WP = 1 only foreground can write in this area (unless SY key-in).

WP = 2 only background can write in this area (unless SY key-in).

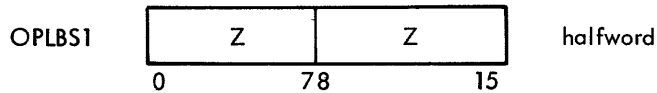
WP = 3 only the Monitor can write in this area.

WP = 4 no one can write in this area unless SY key-in.

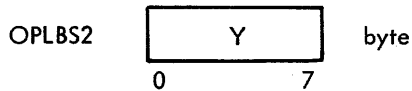
WP = 5 only IOEX can write in this area

Operational Label Table (OPLBS)

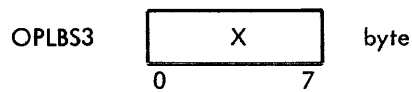
The Operational Label Table is a parallel table with the format



where ZZ is the operational label in EBCDIC



where Y is DCT or RFT index of the permanent assignment (bit 0 = 0 if DCT index; bit 0 = 1 if RFT index).

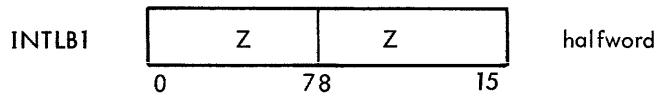


where X is DCT or RFT index of current assignment.

Number of entries in OPLBS is in first halfword of first entry in OPLBS1.

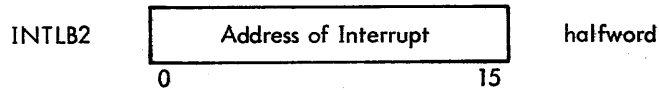
Interrupt Label Table (INTLB)

The Interrupt Label table is a parallel table with the format



where ZZ is the interrupt label in EBCDIC.

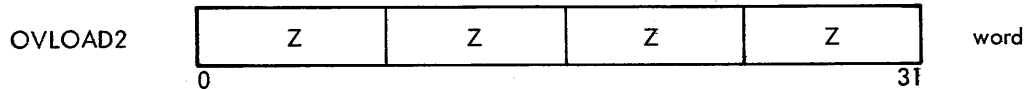
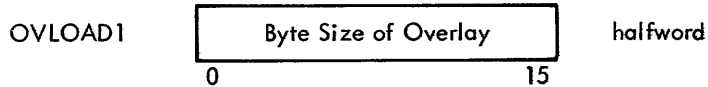
[†] Starting and ending RAD address is given as a sector number.



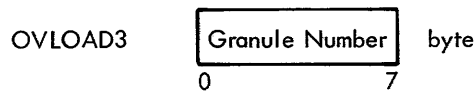
The number of entries in INTLB is in the first halfword of the first entry in INTLB1.

OVLOAD Table (for RBM Overlays Only)

The OVLOAD Table is a parallel table with the format



where ZZ = first four characters of name of overlay in EBCDIC

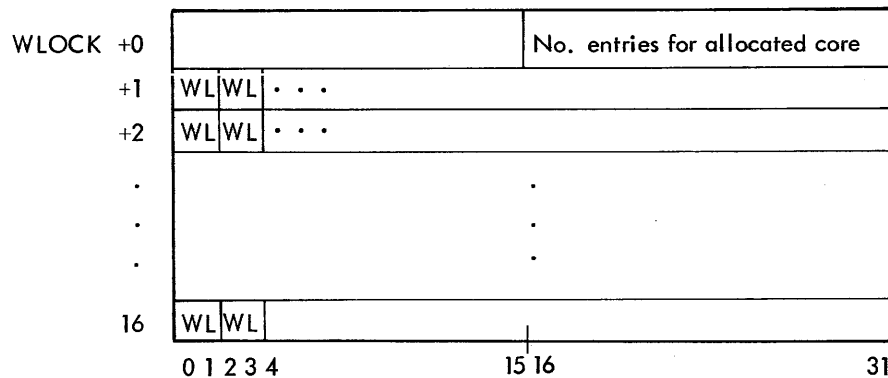


where the specified Granule Number is in the file RBM.

The number of entries in OVLOAD is in first halfword of OVLOAD1.

Write Lock Table (WLOCK)

Assuming no checkpoint, WLOCK contains write locks for the current core allocation. After a checkpoint the write locks will be restored from this table.



WLOCK + 1 always contains the write locks for the first 8K of memory. The table is always 17 words in length but the first word reflects the number of registers that must be output following a checkpoint.

9. OVERLAY LOADER

Overlay Structure

The Overlay Loader is itself an overlaid program, with a root and the six segments illustrated in Figure 52.

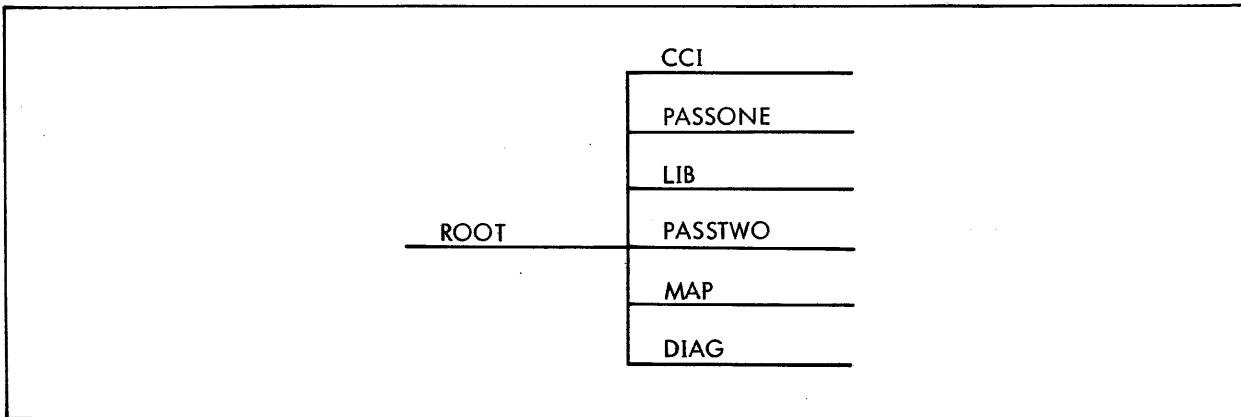


Figure 52. Overlay Structure of the Overlay Loader

The functions of the Root and segments is given in Table 6.

Table 6. Overlay Loader Segment Functions

| Segment | Function |
|---------|---|
| ROOT | Calls in the first segment (CCI) but thereafter, the segments call in other segments. ROOT is a collection of subroutines, tables, buffers, FPTs, DCBs, flags, pointers, variables, and temp storage cells. Root is resident at all times. |
| CCI | Reads and interprets all Loader control commands. |
| PASSONE | Makes the first pass over the Relocatable Object Modules, satisfies DEF/REF linkages between ROMs in the same path, links references to Public Library routines, and allocates the loaded program's control and dummy sections (e.g., assigns absolute core addresses). |
| LIB | Searches the library tables for routines to satisfy primary references left unsatisfied at segment end. |
| PASSTWO | Makes the second pass over the ROMs, creates absolute core images of segments, provides the necessary RBM interface (PCB, Temp Stack, REFd DCBs, DCBTAB, INITTAB, and OVLOAD), and writes the absolute load module on the output file. |
| MAP | Outputs the requested information about the loaded program. |
| DIAG | Outputs all Loader diagnostic messages. |

Overlay Loader Execution

The Root of the Overlay Loader is read into the background when the Job Control Processor (JCP) encounters an !OLOAD control command on the "C" Device. The JCP allocates six scratch files (X1, X2, X3, X4, X5, and X6) in the Background Temp area of the RAD unless otherwise specified on a Monitor !ALLOBT command, and three blocking buffers unless otherwise specified on a Monitor !POOL command. The core layout of the Overlay Loader is illustrated in Figure 53.

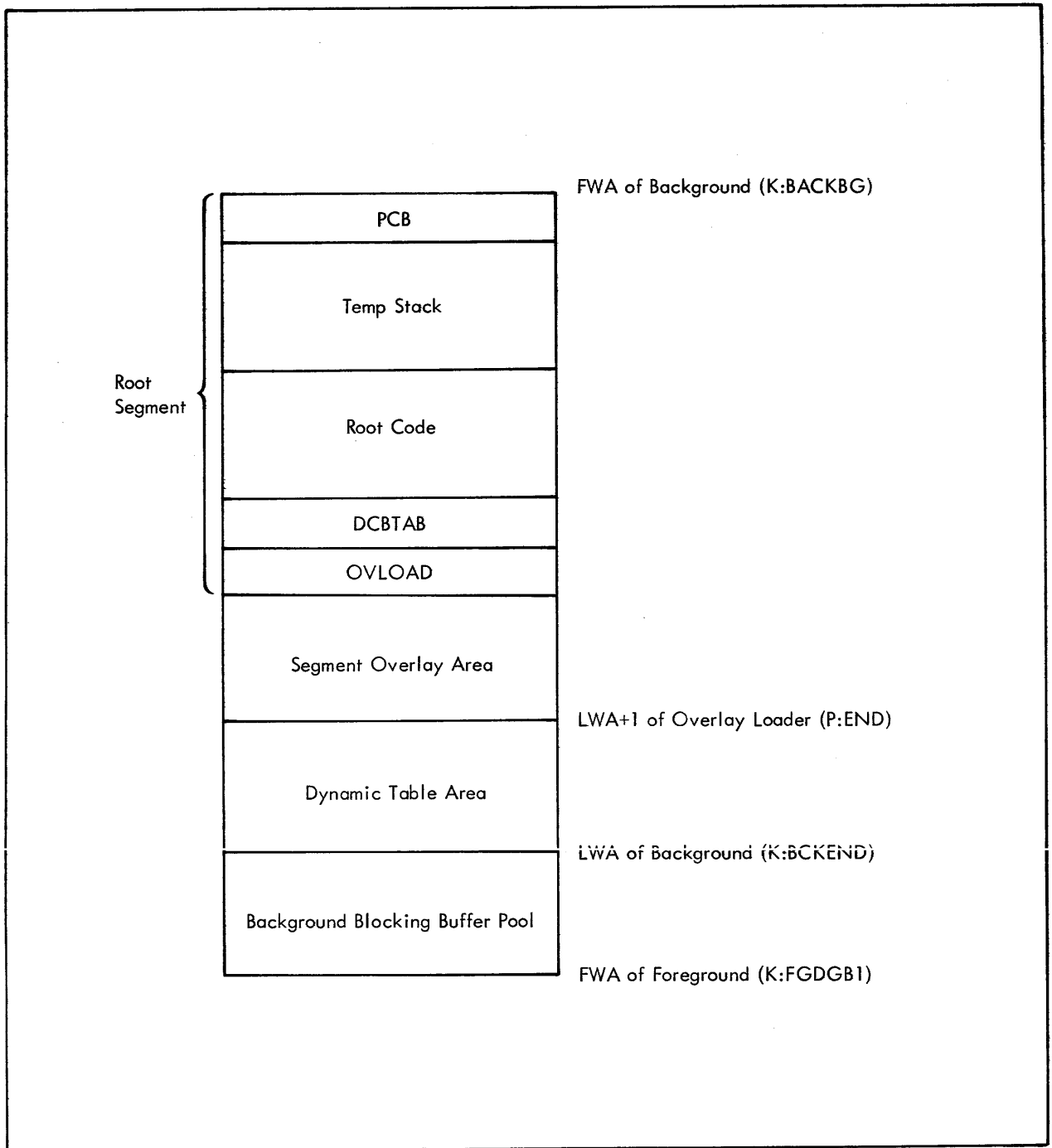


Figure 53. Overlay Loader Core Layout

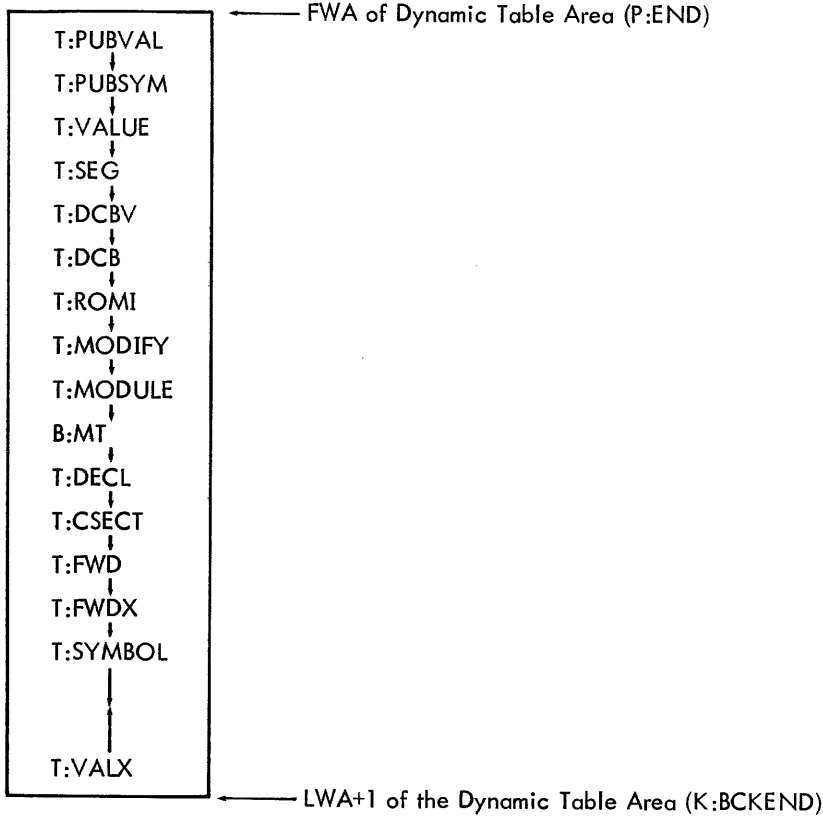
Dynamic Table Area

The Dynamic Table Area is an area of core beginning at the LWA+1 of the Overlay Loader's code and extending to the beginning of the background blocking buffer pool. That is, the Loader uses the remaining core in background for a work area.

The Dynamic Table Area is divided into 16 table areas with boundaries that can change, subject to the length of the tables. The tables are built by CCI and PASSONE from information on the control commands and ROMs, and are therefore only dynamic until the beginning of PASSTWO, when the table areas are fixed. Since these tables are an essential part of the load process, it is important to understand the function of the tables.

Dynamic Table Order

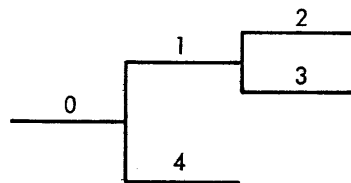
During the first pass over the object modules, the 16 table areas have a fixed order as follows:



For better reader comprehension, the table area descriptions given below are given in a logical order rather than the program listing sequence.

T:SYMBOL and T:VALUE

The program's external table is a collection of DEFs, PREFs, SREFs, and DSECTs (excluding DCBs). The external table is divided into two parts: one containing the EBCDIC name of the external (T:SYMBOL), and the other containing the value (T:VALUE). Each table is divided into segment subtables that overlay each other in core in the same way that the segments themselves are overlaid. For example, the external tables of a program with the overlay structure



would exist in core (for both PASSONE and PASSTWO) as follows:

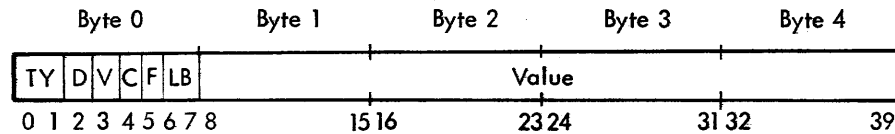
| For Root | For Seg 1 | For Seg 2 | For Seg 3 | For Seg 4 |
|----------|-----------|-----------|-----------|-----------|
| 0 | 0 | 0 | 0 | 0 |
| | — 1 | — 1 | — 1 | — 4 |
| | | — 2 | — 3 | |

Segments in different paths cannot communicate (i.e., the subtables of segments in different paths are never in core at the same time). A segment's T:SYMBOL and T:VALUE subtables are built by CCI and PASSONE and saved on a RAD scratch file at path end (i.e., when the next segment starts a new path). However, only tables overlayed by the new segment at path end get written out. For example, at the end of path (0,1,2), segment 2 would be written out; at the end of path 0,1,3), segments 3 and 1 would get written out; and at the end of the program, segments 4 and 0 would get written out.

A segment's subtable consists of all DEFs in the segment, DSECTs not allocated in a previous segment of the path, and any REFs not satisfied by DEFs in a previous segment of the path. Since the DEF/REF links are all satisfied by PASSONE, T:SYMBOL is not used by PASSTWO.

T:VALUE ENTRY FORMATS

T:VALUE entries are numbered from 1 to n and have a fixed size of bytes, with the format



where

TY is the entry type

TY = 00 DEF

TY = 01 DSECT

TY = 10 SREF

TY = 11 PREF

D is a flag specifying whether or not the external is defined/allocated/satisfied.

D = 1 external has been defined/allocated/satisfied.

D = 0 external is undefined/unallocated/unsatisfied.

V is a flag specifying the type of value (meaningful only if D = 1).

V = 1 value is the value of the external.

V = 0 value is the byte address of the expression defining or satisfying the external in T:VALX.

C is a constant (meaningful only if V = 1).

C = 1 value is a 32-bit constant.

C = 0 value is a positive or negative address with byte resolution.

F is a flag specifying whether the external is a duplicate or an original.

F = 1 external is a duplicate.

F = 0 external is an original.

LB specifies source of external.

LB = 00 external from input ROM or CC.

LB = 01 external from System Library.

LB = 10 external from User Library.

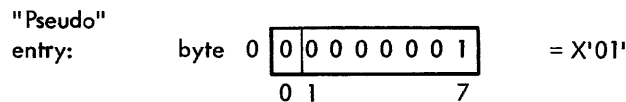
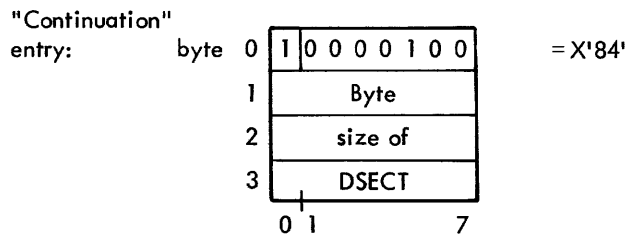
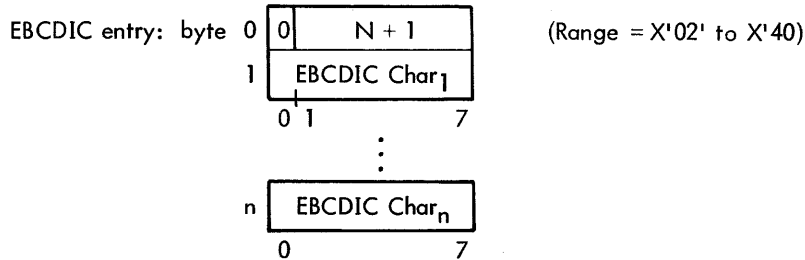
Value is initially set to zero; usage is dependent upon D, V, and C flags.

Since the T:VALUE entries are kept as small as possible, unused bit combinations are reserved to define the following two intermediate external types:

1. If TY = PREF, C = 0, and V = 1, the external is an "excluded pref" which means that the PREF will cause neither library loading nor linkage (including the Public Library). Instead, the PREF will be satisfied by a DEF in a segment further up the path.
2. If TY = DSECT, D = 1, and V = 0, the external was input from the :RES control command and is to be allocated at the end of the segment.

T:SYMBOL ENTRY FORMATS

T:SYMBOL is a byte table with variable sized entries that are numbered from 1 to n. There are three types of entries: EBCDIC, "continuation", and "pseudo". The EBCDIC entry contains the name of the external. The "continuation" entry contains the size of a DSECT and only follows a DSECT entry. The "pseudo" entry is a FWD or CSECT entry that has been added to T:SYMBOL because the entry was referenced in a T:VALX expression that could not be resolved at "module end". The entry formats are as follows:



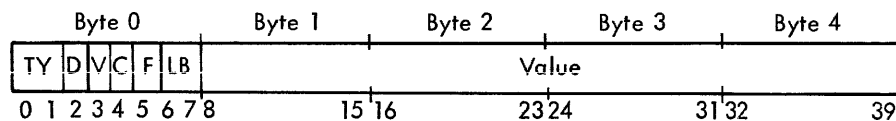
Note that the first byte contains the byte count of the entry (in bits 1-7).

T:PUBVAL and T:PUBSYM

Each Public Library file has an external table of DEFs (there are no DSECTs or unsatisfied REFs in a Public Library) that is divided into two parts; VALUE and SYMBOL. T:PUBVAL contains the VALUE tables for each public library specified in the PUBLIB option of the IOLOAD control command, and T:PUBSYM contains the corresponding SYMBOL tables. Since the sizes of the table areas are fixed once T:PUBVAL and T:PUBSYM have been input, there are only 14 dynamic table areas.

T:PUBVAL ENTRY FORMATS

T:PUBVAL entries are numbered from 1 to n and have a fixed size of five bytes. Since the size of T:PUBVAL does not change, T:PUBSYM is located at the next doubleword boundary following T:PUBVAL. T:PUBVAL entries have the format



where

TY = 00 = DEF

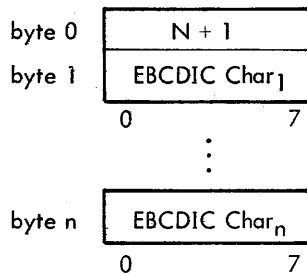
D = 1 the DEF has been defined.

- V = 1 value is the value of the DEF.
- C = 1 value is a 32-bit constant.
- C = 0 value is a positive or negative address with byte resolution.
- F = 0 not a duplicate DEF.
- LB = 11 PUBLIB

Note that the T:VALUE and T:PUBVAL entries have the same formats even though the T:PUBVAL entries are a subset of the T:VALUE format.

T:PUBSYM ENTRY FORMATS

T:PUBSYM is a byte table with variable sized entries that are numbered from 1 to n. Since the size of T:PUBSYM does not change, the table following is located at the next doubleword boundary after T:PUBSYM. T:PUBSYM entries have the format



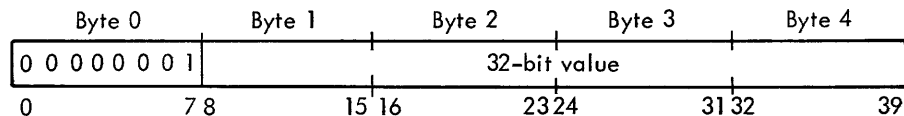
T:VALX

External definitions are defined with expressions. If the expression can be resolved, its value is stored in the DEFs T:VALUE entry. If the expression cannot be resolved, it is saved in T:VALX and the byte address of the expression is stored in the DEFs T:VALUE entry.

Once an expression is resolved, its entry is zeroed out. The T:VALX entries cannot be packed to regain space, since the T:VALUE entries contain address pointers, however, empty entries are reused where possible.

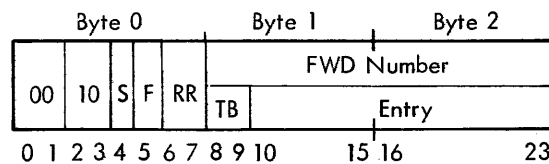
Expressions have a variable size and are made up of expression bytes, combined in any order. The formats for the T:VALX expression bytes (slightly different than the object language) are

Add Constant (X'01')



This item causes the specified four-byte constant to be added to the Loader's expression accumulator. Negative constants are represented in two's complement form:

Add/Subt Value (X'2N')



where

- S = 1 subtract value.
- S = 0 add value.

F = 1 add/subtract value of T:FWD entry where the FWD number is in bytes 1 and 2.

F = 0 add/subtract value of TABLE entry where

TB = 00 Entry points to T:DCB.

TB = 01 Entry points to T:VALUE/T:SYMBOL.

TB = 10 Entry points to T:CSECT.

TB = 11 Entry points to T:PUBVAL/T:PUBSYM.

RR = 00 byte address resolution.

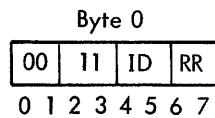
RR = 01 halfword address resolution.

RR = 10 word address resolution.

RR = 11 doubleword address resolution.

This item causes the value of the FWD or TABLE entry to be converted to the specified address resolution (only if the value is an address) and added to the Loader's expression accumulator. Note that expressions involving T:FWD and T:CSECT entries point to the current ROM's FWD and CSECT tables. If these expressions are not resolved at module end, the Loader creates dummy T:SYMBOL and T:VALUE entries from the FWD or CSECT entry and changes the pointer in the expression to point to the dummy entry in T:VALUE. However, unresolved expressions rarely happen.

Address Resolution (X'3N')



where

ID = 00 changes the partially resolved expression (if an address) to the specified resolution.

ID = 01 identifies the expression as a positive absolute address with the specified resolution (add absolute section).

ID = 10 identifies the expression as a negative absolute address with the specified resolution (subtract absolute section).

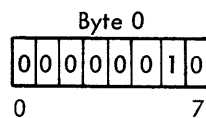
RR = 00 byte address resolution.

RR = 01 halfword address resolution.

RR = 10 word address resolution.

RR = 11 doubleword address resolution.

Expression End (X'02')

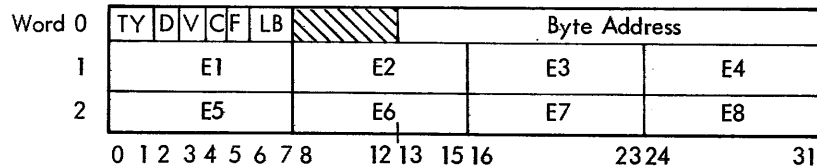


This item identifies the end of an expression (the value of which is contained in the Loader's expression accumulator).

T:DCB

T:DCB contains the DEFs and REFs that are recognized as either system (M:) or user (F:) DCBs. DCBs declared as external definitions must exist in the Root segment. The Loader allocates space in part two of the Root for DCBs

that are declared external references, and supplies default copies of system DCBs. T:DCB is resident at all times. Entries have a fixed size of three words and have the format



where

Word 0

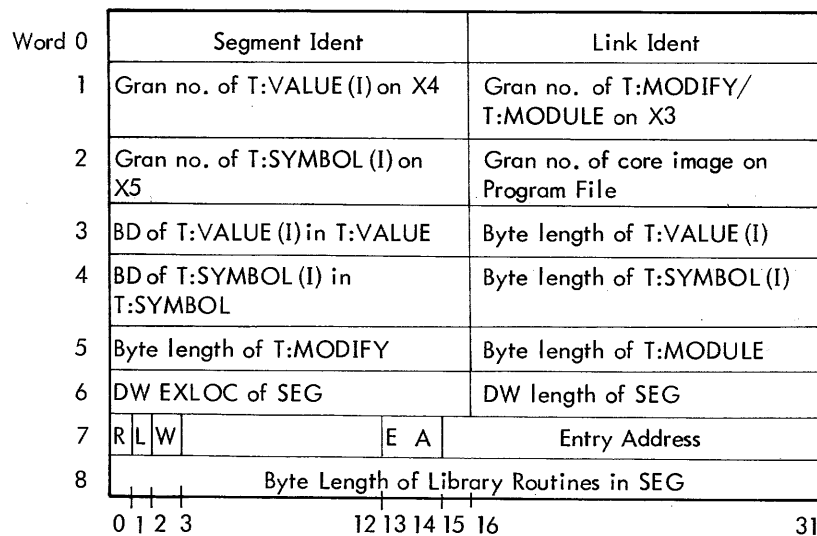
- TY = 00 DEF (coded in the Root by the user).
- TY = 11 PREF (allocated in Root part 2 by Loader)
- D = 1 defined or allocated.
- D = 0 undefined/unallocated.
- V = 1 address is the byte value of the DCB, only meaningful if D = 1.
- V = 0 address points to an expression in T:VALX, only meaningful if D = 1.
- C = 1 the DCB was defined with a value that is either a constant or an illegal address (i.e., negative or mixed resolution), only meaningful if V = 1.
- C = 0 the value of the DCB is an address, only meaningful if V = 1.
- F = 0 DCB cannot be a duplicate (duplicates are put in T:SYMBOL/T:VALUE).
- LB = 00 the DCB was input from a nonlibrary ROM.
- LB = 01 the DCB was input from the System Library.
- LB = 10 the DCB was input from the User Library.

Word 1,2

- E1 - E8 is the EBCDIC name of the DCB, padded with blanks if necessary.

T:SEG

T:SEG contains information about the program's segments and is resident at all times. One entry is allocated per segment. Entries have a fixed size of nine words and have the format



where

- Gran no. the granule number in the RAD file where the table begins. If the RAD file overflows, Gran no. will equal X'FFFF'. Granules are numbered from 0 to n.

- (I) segment's subtable.
- BD byte displacement.
- EXLOC execution location.
- DW doubleword.
- R = 1 error severity level set on at least one ROM in the segment.
- R = 0 error severity level reset on every ROM in the segment.
- L = 1 load error (duplicate DEFs, unsatisfied REFs, etc.).
- L = 0 no loading errors in SEG.
- W = 1 T:VALUE (I) and T:SYMBOL (I) output on X4, X5.
- W = 0 T:VALUE (I) and T:SYMBOL (I) not output on X4, X5.
- EA = 00 value in bits 15–31 (if nonzero) is last entry address (in words) encountered on non-Lib ROM.
- EA = 01 unused.
- EA = 10 SEG's entry address input from CC and value in bits 15–31 is the entry address (in words).
- EA = 11 SEG's entry address input from CC and value in bits 15–31 is the entry number of the T:SYMBOL/T:VALUE DEF specified on the CC.

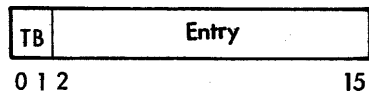
B:MT

There are four tables associated with each ROM loaded (including library ROMs): T:DECL, T:CSECT, T:FWD, and T:FDX. The size of these tables can be extremely large or small, depending upon which processor produced the ROM and the content of the program. To conserve time and space, these tables are packed into the Module Tables buffer (B:MT) at module end, and output to the X2 TempFile on the RAD only when either the buffer is full or at segment end. The size allocated for B:MT is dependent upon the size of the Dynamic Tables area and is made a multiple of the sector size of the X2 RAD file.

T:DECL

DEFs, PREFs, SREFs, DSECTs, and CSECTs are referenced in the object language by declaration number. Therefore, associated with each ROM is a table of declarations whose entries point to DEF, REF, DSECT, and CSECT entries in other tables.

According to the object language convention, entry zero points to the standard control section declaration. Entries are numbered from 0 to n; have a fixed size of two bytes; and have the format

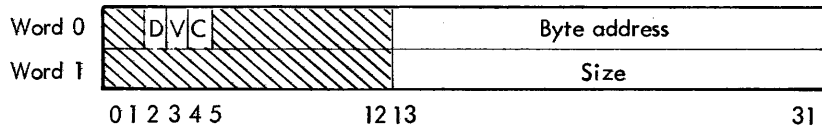


where

- TB = 00 Entry points to T:DCB.
- TB = 01 Entry points to T:SYMBOL/T:VALUE.
- TB = 10 Entry points to T:CSECT (associated with current ROM).
- TB = 11 Entry points to T:PUBSYM/T:PUBVAL
- Entry Table entry number. The range is 1 through 16,383.

T:CSECT

Associated with each ROM is a table of standard and nonstandard control sections. A nonstandard control section is allocated by the Loader when the declaration is encountered. The standard control section is allocated when the first reference to declaration 0 is encountered in an expression defining the origin load item. T:CSECT entries are numbered from 1 to n ; have a fixed size of two words; and have the format



where

Word 0

D = 1 allocated.

V = 1 value.

C = 0 address.

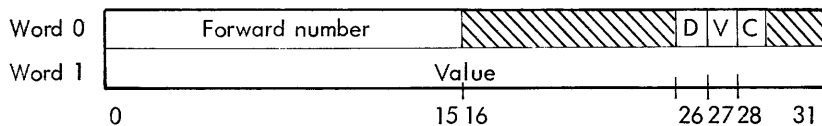
Byte address first byte address of the control section.

Word 1

Size Number of bytes in the control section.

T:FWD

Associated with each ROM is a table of forward definitions (forwards). Each forward is identified by a random two-byte reference number. Thus, when a forward is referenced in an expression, the T:FWD table for that ROM must be searched for a matching number. T:FWD entries have a fixed size of two words with the format



where

D = 1 defined.

V = 1 value is the value of the resolved expression.

V = 0 value is a byte displacement pointer to the expression in T:FWDX.

C = 1 value is a constant (only meaningful if V = 1).

C = 0 value is a positive or negative address with byte resolution (only meaningful if V = 1).

T:FWDX

Forwards are defined with expressions and are of two types: the first is defined with an expression that can be resolved by module end; the second type is defined with an expression that involves an external DEF, REF, or DSECT (many of these cannot be resolved at module end). Associated with each ROM is a table containing all unresolved expressions defining FWDs. When a T:FWDX expression is resolved, its entry is zeroed out and the space reused, if possible. T:FWDX entries have the same format as T:VALX entries.

T:MODULE

Each segment has a T:MODULE table. T:MODULE contains information about a segment's Relocatable Object Modules (ROMs). One entry is allocated per ROM. Entries have a fixed size of five words and have the format

| Word 0 | V | Entry no. | G | L | B | Record displacement in file |
|--------|---|---|---|---|---|-----------------------------|
| 1 | | Gran no. of B:MT on X2, or BD of T:DECL (J) in B:MT | | | | Byte length of T:DECL (J) |
| 2 | | BD of T:CSECT (J) in B:MT | | | | Byte length of T:CSECT (J) |
| 3 | | BD of T:FWD (J) in B:MT | | | | Byte length of T:FWD (J) |
| 4 | | BD of T:FWDX (J) in B:MT | | | | Byte length of T:FWDX (J) |

0 1 7 8 14 15 16 31

where

V = 1 Entry no. in bits 1-7 points to T:DCBV.

V = 0 Entry no. in bits 1-7 points to T:DCBF.

Entry no. the entry number of the DCB (in either T:DCBV or T:DCBF) that points to the RAD file where the ROM is located.

G = 1 T:DECL (J) begins at byte zero in B:MT and HWO (halfword zero) in word 1 contains the granule no. of B:MT on X2. If the Granule no. equals X'FFFF', X2 has overflowed and B:MT did not get saved on the RAD.

G = 0 T:DECL (J) is located in B:MT at the byte displacement specified in HWO of word 1.

LB = 00 not Library ROM.

LB = 01 ROM from System Library (SP area of RAD).

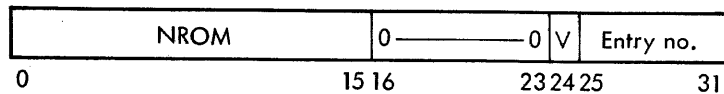
LB = 10 ROM from User Library (FP area of RAD).

Record displacement in the MODULE file (only meaningful for library ROMs.)

T:ROMI

T:ROMI contains the information necessary for PASSONE to load a segment's ROMs. T:ROMI is built by CCI from the input options specified on the segment's :ROOT, :SEG, or :PUBLIB control command, or by :LIB to point to the library routines required for the segment. At the beginning of PASSTWO, the area size for T:ROMI is set to zero. There are three types of T:ROMI entries, as illustrated below, and entries have a fixed size of one word.

Entry for ROMs input from RAD files (built by CCI):



where

NROM is the number of ROMs to input or contains -5, which means to input until !EOD is encountered. This halfword is used as a decreasing counter by PASSONE and eventually equals zero.

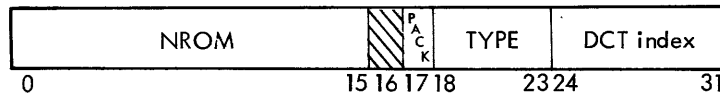
Bits 16-23 always equal zero to specify entry type.

V = 1 Entry no. in bits 25-31 points to T:DCBV.

V = 0 Entry no. in bits 25-31 points to T:DCBF.

Entry no. is the entry number of the DCB (in either T:DCBV or T:DCBF) that points to the RAD file where the ROM is located.

Entry for ROMs input from a specified device or OPLB (built by CCI):



where

Bits 16-23 always equal nonzero to specify entry type.

NROM is described above.

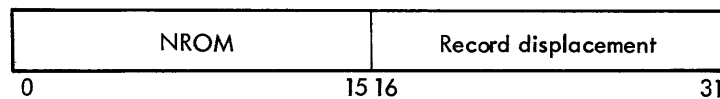
PACK is the PACK flag (bit 22 of word 0) in DCB.

TYPE is the device type code (bits 18-23 of word 1) in DCB.

DCT index is the DCT index of the device (bits 24-31 of word 1) in DCB.

PASSONE will store the information in F:DEVICE and input the ROMs via that DCB. Note that OPLBs are converted to their assigned devices.

Entry for ROMs input from the System or User Library (built by LIB):



where

NROM is described above.

Record displacement is the record displacement of the ROM in the MODULE file of the area specified by FL:LBLD.

Library ROM entries are distinguished from the other two entry types by the Loader flag FL:LBLD. The flag is always reset when the other entry types are in T:ROMI.

T:DCBV

T:DCBV is a table of DCBs assigned to the various RAD files specified (other than GO) on the input options of the :ROOT and :SEG, or :PUBLIB control commands. One DCB is created for each unique file name specified. T:DCB is resident at all times. T:DCBV entries are numbered from 1 to n, and have the standard seven-word DCB format.

T:MODIFY

Each segment's :MODIFY commands are translated into object language load items and stored in the segment's T:MODIFY table, and each :MODIFY command is translated into a T:MODULE entry. Entries begin with an "origin" load item and are terminated by either the next "origin" load item or a "module end" load item. Entries are made up of the load items described below and expressions in the T:VALX/T:FWDX format:

Origin (X'04')

This one-byte item sets the load-location counter to the value designated by the expression (in T:VALX format) immediately following the origin control byte. The value of the expression equals the location specified on the :MODIFY command.

Load Absolute (X'44')

This one-byte item causes the next four bytes to be loaded absolutely and the load-location counter advanced appropriately.

Define Field (X'07') (X'FF') (field length)

This three-byte item defines an expression value to be added to the address field of the previously loaded four-byte word. The expression is in T:VALX format and immediately follows the 'field length' byte.

Load Expression (X'60')

This one-byte item causes an expression value to be loaded absolutely and the load-location counter advanced appropriately. The expression to be loaded is in T:VALX format and immediately follows the 'load expression' control byte.

Module End (X'0E')

This one-byte item terminates the load items in T:MODIFY.

Use of the Dynamic Table Area During LIB

During the library search, LIB temporarily reorganizes the Dynamic Table area by packing the 16 tables together at the top of the area. LIB uses the remaining space for its tables. The core layout of these tables and their formats are illustrated in Figure 54.

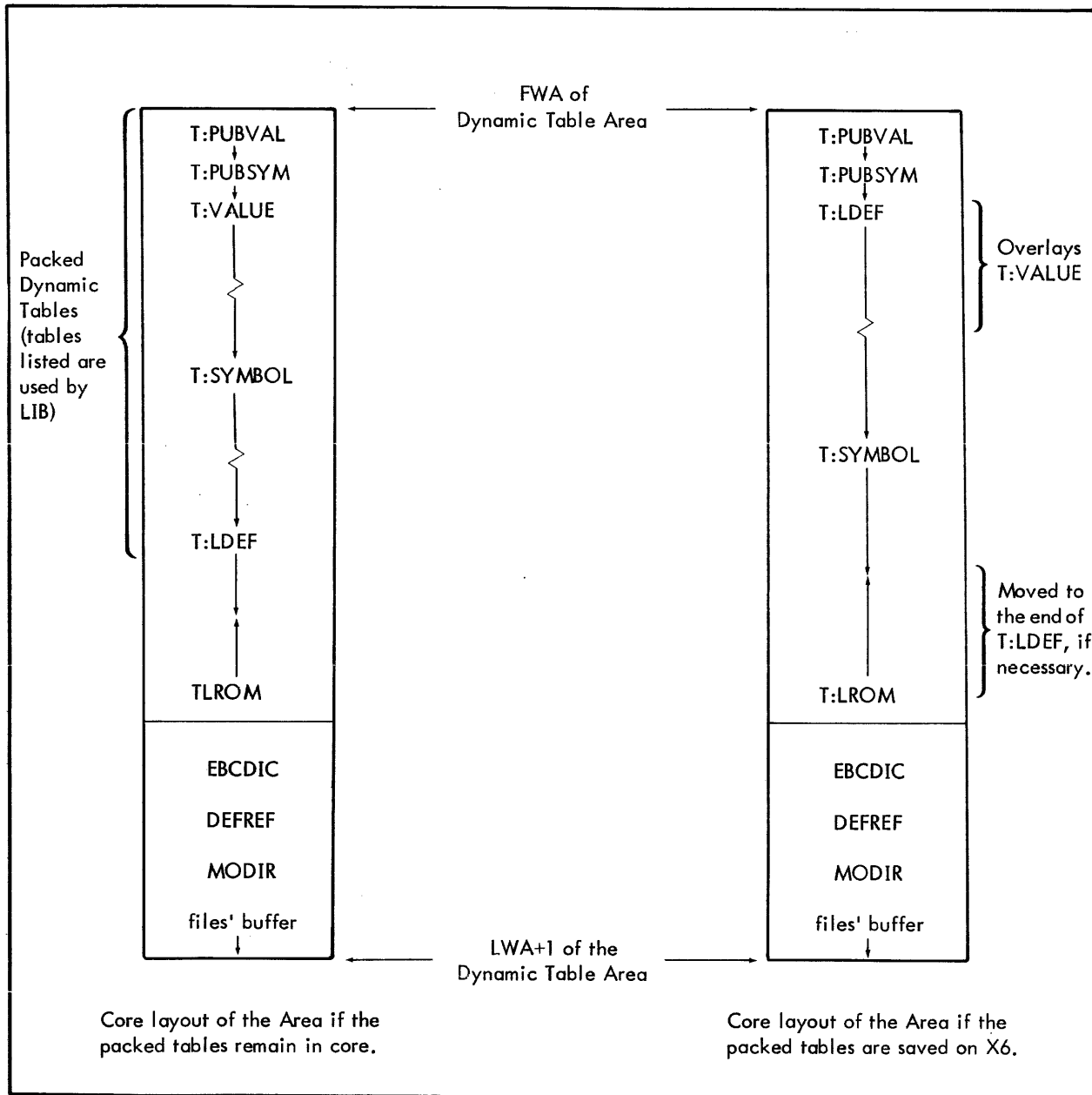


Figure 54. LIB Reorganization of Dynamic Table Area

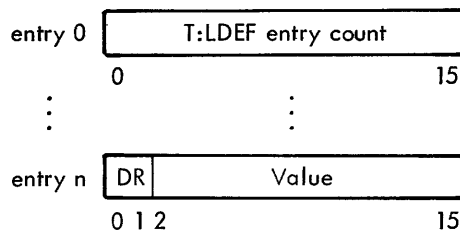
T:LDEF

T:LDEF is located in the Dynamic Table area only when the LIB segment is executing and is used by LIB to satisfy REFs to library routines. Initially, T:LDEF contains the following items:

1. All unsatisfied REFs from the current segment's T:VALUE subtable.
2. All excluded PREFs from the current segment's T:VALUE subtable.
3. All DEFs in the path T:VALUE table that are from the same library as the one being searched.
4. All Public Library (T:PUBVAL) DEFs.

The Library DEFs are included so that library routines loaded in previous segments of the Public Library will not be duplicated. The excluded PREFs (that inhibit library loading) are treated as DEFs. Since library routines may themselves reference other library routines, the set of DEFs and REFs associated with a library routine are included in T:LDEF if, and only if, at least one of the DEFs satisfies a REF in T:LDEF. When a REF is satisfied it is changed to a DEF. Eventually, T:LDEF contains library DEFs, any REFs that cannot be satisfied in the Library, and the excluded PREFs.

T:LDEF has a variable number of entries with the count kept in entry 0. Entries have a fixed size of two bytes with the format



where

DR = 00 null entry.

DR = 01 DEF.

DR = 10 unsatisfied PREF.

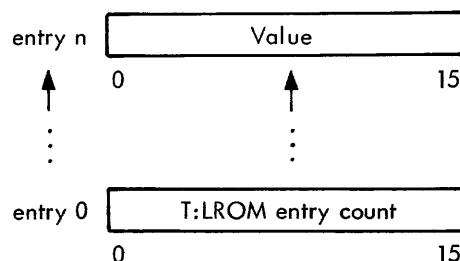
DR = 11 excluded PREF.

Value entry number in T:SYMBOL, that is later changed to the corresponding entry's byte offset in the EBCDIC file.

T:LROM

T:LROM is located in the Dynamic Table area only when the LIB segment is executing and contains pointers to library routines whose DEFs have satisfied REFs in T:LDEF. That is, T:LROM points to the library routines that are to be loaded along with the segment.

T:LROM entries initially point to a library ROM's entry in the MODIR file and then get changed to point to the corresponding ROM's location in the MODULE file. T:LROM has a variable number of entries, with the count kept in entry 0. T:LROM is built backwards but has forward entries. Entries have a fixed size of two bytes with the format



where

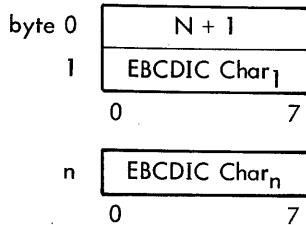
value halfword offset of the library ROM's entry in the MODIR file, which is later changed to the starting record number of the ROM in the MODULE file.

MODULE File

The MODULE file is a blocked sequential file, with 120 bytes per record, that contains the Library's ROMs.

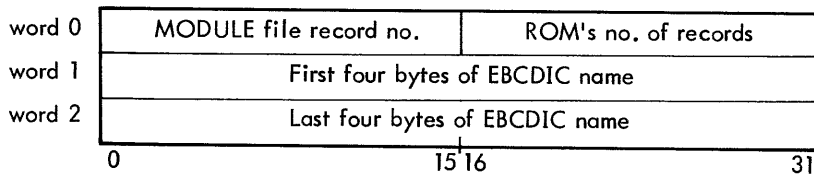
EBCDIC File

The EBCDIC file is an unblocked sequential file consisting of one variable length record. The EBCDIC file contains the unique EBCDIC names of all DEFs and REFs declared in the ROMs in the MODULE file. Entries have a variable number of bytes with the format



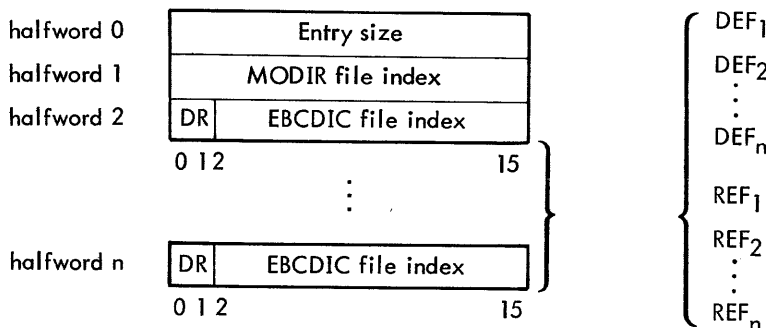
MODIR File

The MODIR file is an unblocked sequential file consisting of one variable length record. Each MODIR file entry corresponds to a ROM on the MODULE file and contains the name of the ROM, its location on the MODULE file, and the number of records in the ROM. Entries have a fixed size of three words with the format



DEFREF File

The DEFREF file is an unblocked sequential file consisting of one variable length record. Each entry in the DEFREF file corresponds to a ROM in the MODULE file and contains all the external DEFs and REFs declared in the ROM, plus a pointer to the ROM's entry in the MODIR file. Entries have a variable number of halfwords with the format



where

Entry size number of halfwords in the entry (including itself).

MODIR file index relative halfword of the ROM's corresponding entry in the MODIR file. X'FFFF' means that the entry has been deleted.

DR = 00 not used.

DR = 01 DEF.

DR = 10 PREF.

DR = 11 not used.

EBCDIC file index relative byte of the external name entry in the EBCDIC file.

Use of Dynamic Table Area During PASSTWO

PASSTWO reorganizes the Dynamic Table area by moving the resident tables T:SEG, T:DCBV, and T:DCB to the end of T:PUBVAL. PASSTWO uses the remaining space to read in the necessary tables built during PASSONE to build its own tables and to create the core image of the segment. The core layout of these tables and their format is illustrated in Figure 55.

T:GRAN

Since the Work area has a finite size that varies according to the size of B:MT, it may not be large enough to contain a segment's total core image at all times. Therefore, before a segment is created, its core image length is divided into granule size partitions, where the granule size equals the sector size of the program file. T:GRAN

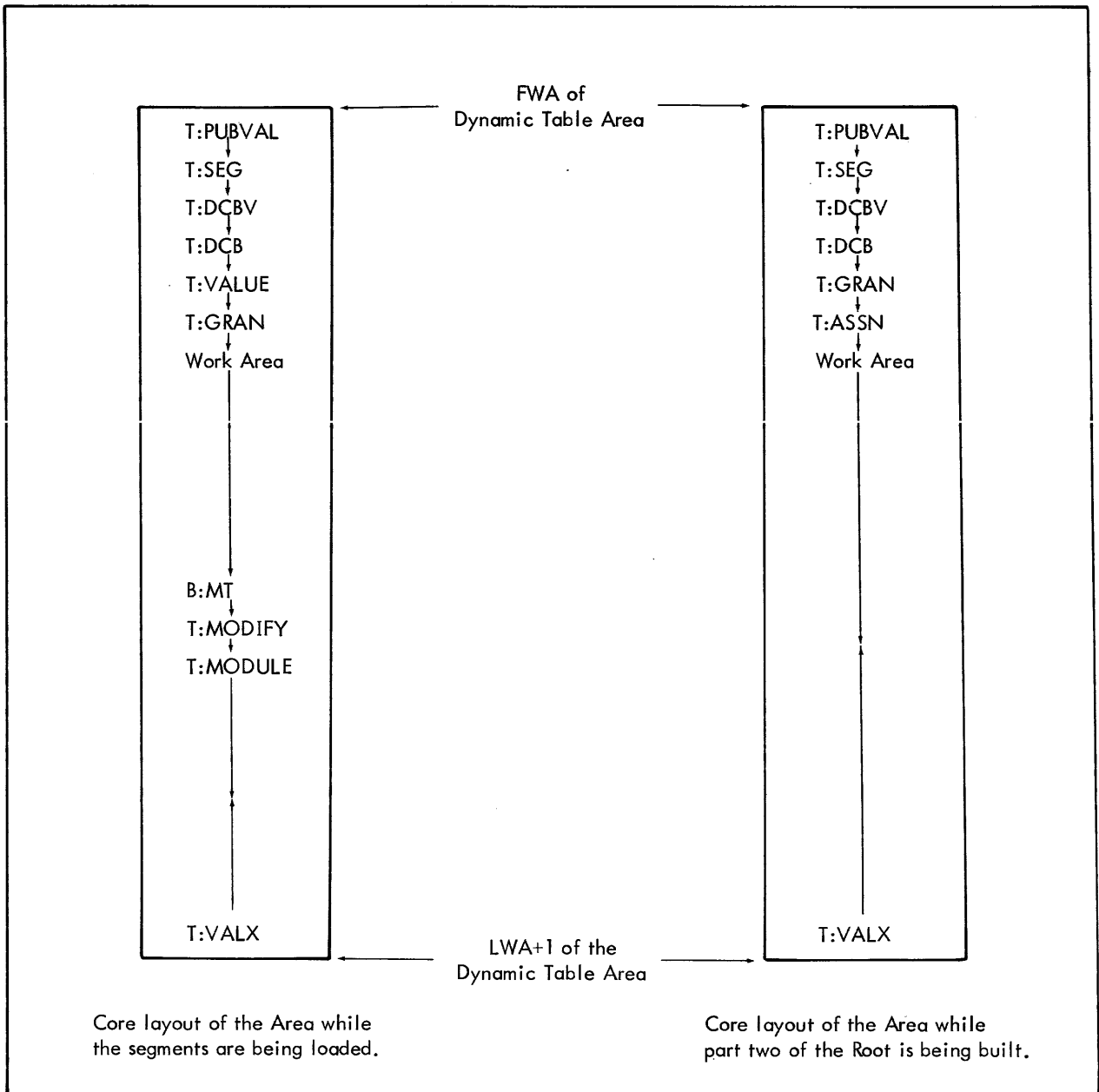
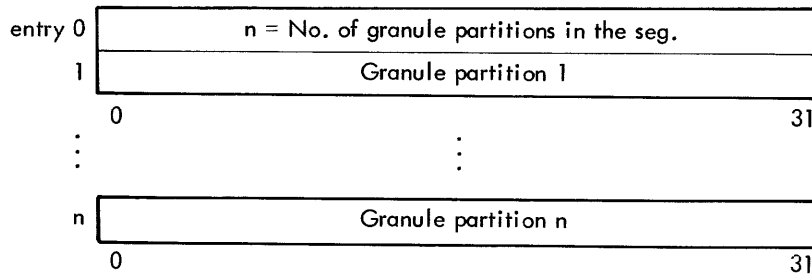


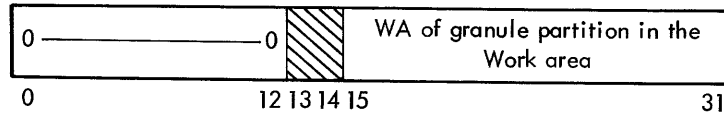
Figure 55. PASSTWO Reorganization of Dynamic Table Area

entries point to the location of a segment's partition (if created) either in core or on the program file. T:GRAN has the following format:

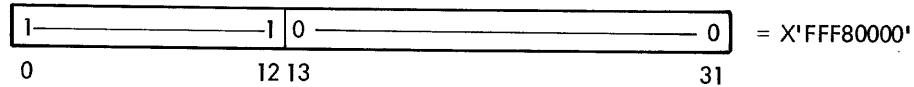


T:GRAN entries have a fixed size of one word with three different formats.

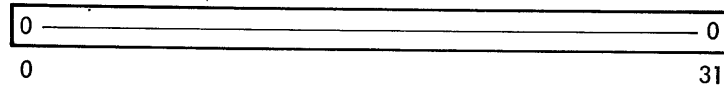
If the granule partition exists in the Work Area:



If the granule partition exists on its corresponding granule in the Program File:



If the granule partition has not been allocated; and data has not yet been loaded into that area of the segment:



T:ASSN

T:ASSN contains the information necessary to reassign DCBs as specified on :ASSIGN commands. T:ASSN is located in the Dynamic Table area during PASSTWO (after all the segments have been loaded) and is built by CCI. Each :ASSIGN command is translated into a T:ASSN entry. Entries have a fixed size of ten words with the format

| | |
|--------|--|
| Word 0 | Byte address of DCB's execution location |
| 1 | Word address of DCB's entry in T:DCB |
| 2 | Changes for word 0 of DCB |
| 3 | Mask for word 0 of DCB |
| 4 | Changes for word 1 of DCB |
| 5 | Mask for word 1 of DCB |
| 6 | Changes for word 3 of DCB |
| 7 | Mask for word 3 of DCB |
| 8 | First four EBCDIC bytes of file name or zero |
| 9 | Last four EBCDIC bytes of file name or zero |
| | 0 31 |

MAP Use of Dynamic Table Area

MAP moves the resident tables T:SEG and T:DCB to the top of the area, and uses the remaining space to read in and reference the tables necessary for the MAP output. MAP does not build any tables. The core layout of the table referenced by MAP is illustrated in Figure 56.

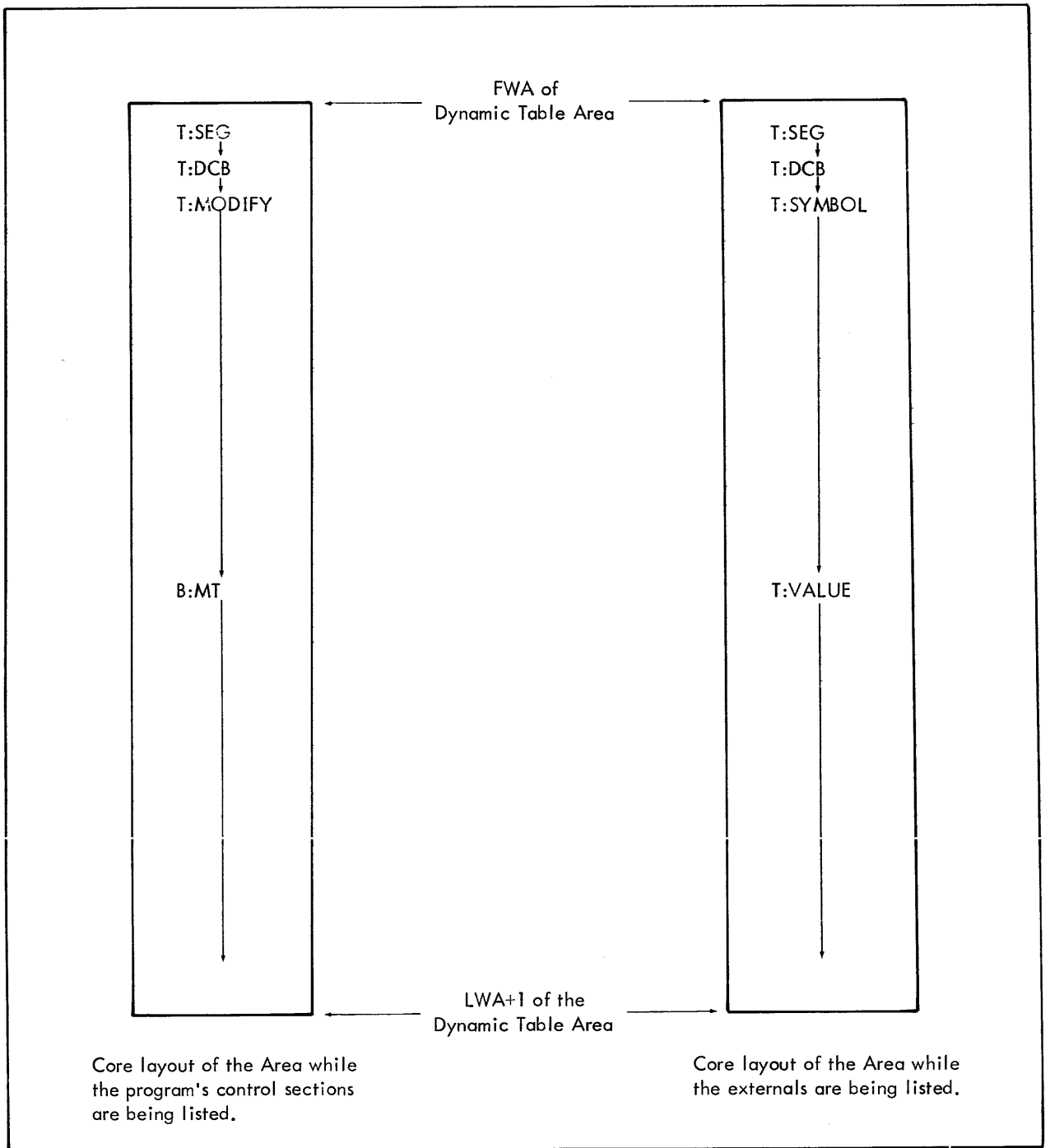


Figure 56. MAP Table Reference

DIAG Use of Dynamic Table Area

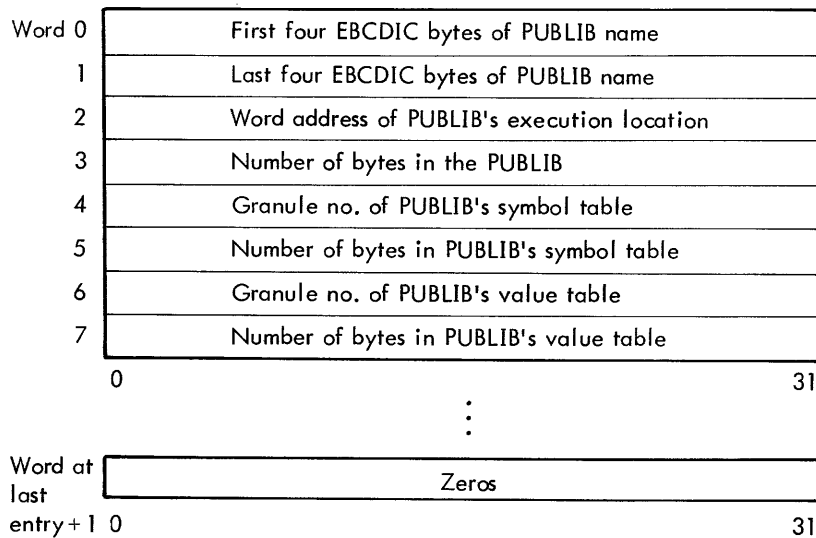
DIAG only uses the Dynamic Table area to reference T:SEG and T:MODULE.

Root Tables

Two tables in the Root, T:PL and T:DCBF, have a fixed size and are referenced by other tables. Their format and use is given below. The usage and format of other tables in the Root are well documented in the Overlay Loader's listing and are not detailed in this manual.

T:PL

T:PL contains the information necessary to create T:PUBSYM and T:PUBVAL and to load the Public Libraries specified on the !OLOAD control command. T:PL exists in the Root and has a maximum of three entries. Table end is indicated by a word of zeros. Entries have a fixed size of eight words with the format



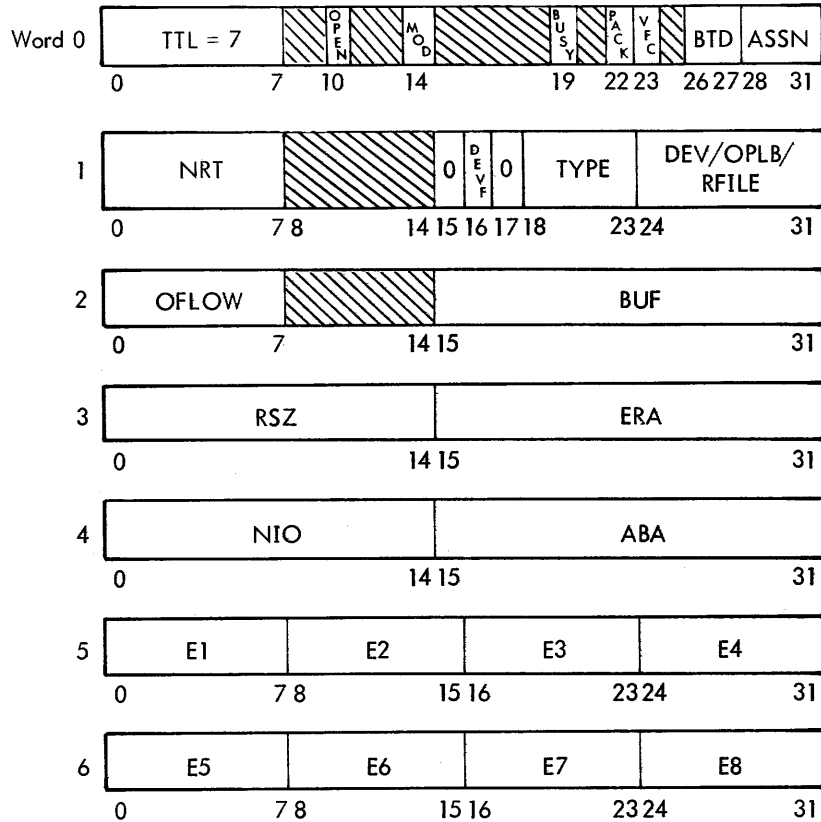
T:DCBF

T:DCBF contains the set of fixed DCBs that are required by the Loader. Each entry contains one DCB. T:DCBF has a fixed number of entries and exists in the Root. T:DCBF entries are numbered from 1 to 18, and have the fixed order given in Table 7.

Table 7. T:DCBF Entries

| Entry | Mnemonic | Pointer To |
|-------|----------|---|
| 1 | F:PUBL | Files specified in the PUBLIB option of !OLOAD. |
| 2 | F:DEVICE | Devices specified in the DEVICE and OPLB input options. |
| 3 | M:GO | GO file in the Background Temp area. |
| 4 | M:OV | Either OV or the file specified in the FILE option of !OLOAD. |
| 5 | M:X1 | X1 in the Background Temp area. |
| 6 | M:X2 | X2 in the Background Temp area. |
| 7 | M:X3 | X3 in the Background Temp area. |
| 8 | M:X4 | X4 in the Background Temp area. |
| 9 | M:X5 | X5 in the Background Temp area. |
| 10 | M:X6 | X6 in the Background Temp area. |
| 11 | F:MODIR | MODIR file in either the SP or FP area. |
| 12 | F:EBCDIC | EBCDIC file in either the SP or FP area. |
| 13 | F:DEFREF | DEFREF file in either the SP or FP area. |
| 14 | F:MODULE | MODULE file in either the SP or FP area. |
| 15 | M:C | C operational label. |
| 16 | M:LL | LL operational label. |
| 17 | M:OC | OC operational label. |
| 18 | M:LO | LO operational label. |

All T:DCBF entries have the standard seven-word DCB format, with two exceptions: OFLOW and NIO, that are used only for the M:OV, M:X1, M:X2, M:X3, M:X4, M:X5, and M:X6 DCBs. The seven-word DCB format is



where

- OFLOW = 0 EOT not encountered.
- OFLOW = 1 EOT encountered.
- NIO number of records (for X1) or granules required.

Scratch Files

The six scratch files in the Background Temp area of the RAD are used by the Loader as temporary storage and are written during the first pass over the object modules. The number of granules required by each scratch file is calculated (whether the file overflows or not) and saved in the DCB assigned to the file. If any of these files overflows (e.g., if the EOT is encountered during a Write operation), the Loader continues PASSONE, skips PASSTWO, then calls the MAP to communicate the number of granules required for each scratch file to the user. The Loader's use of these files is defined in Table 8.

Table 8. Background Scratch Files

| File Name | Loader Use |
|-----------|---|
| X1 | A sequential file with blocked record format. Record size equals 120 bytes; granule size equals 256 words. ROMs input from non-RAD devices are copied onto X1. |
| X2 | A direct access file with the granule size set equal to the sector size. The module's tables (T:DECL, T:CSECT, T:FWD, and T:WDX) are output on X2 when either B:MT is full or at segment end. |
| X3 | A direct access file with the granule size set equal to the sector size. A segment's T:MODIFY and T:MODULE tables are packed together at segment end and output on X3. |

Table 8. Background Scratch Files (cont.)

| File Name | Loader Use |
|-----------|--|
| X4 | A direct access file with the granule size set equal to the sector size. A segment's T:VALUE subtable is output on X4 when the end of a path is encountered and the segment is being overlayed by another segment. |
| X5 | A direct access file with the granule size set equal to the sector size. A segment's T:SYMBOL subtable is output on X5 when the end of a path is encountered and the segment is being overlayed by another segment. |
| X6 | A direct access file with the granule size set equal to the sector size. The LIB overlay packs the 16 Dynamic Tables at the top of the Dynamic Table area and outputs the "pack" on X6 only if the remaining area will not contain the tables required for the library search. |

Program File Format

The format for the Program File is illustrated in Figure 57.

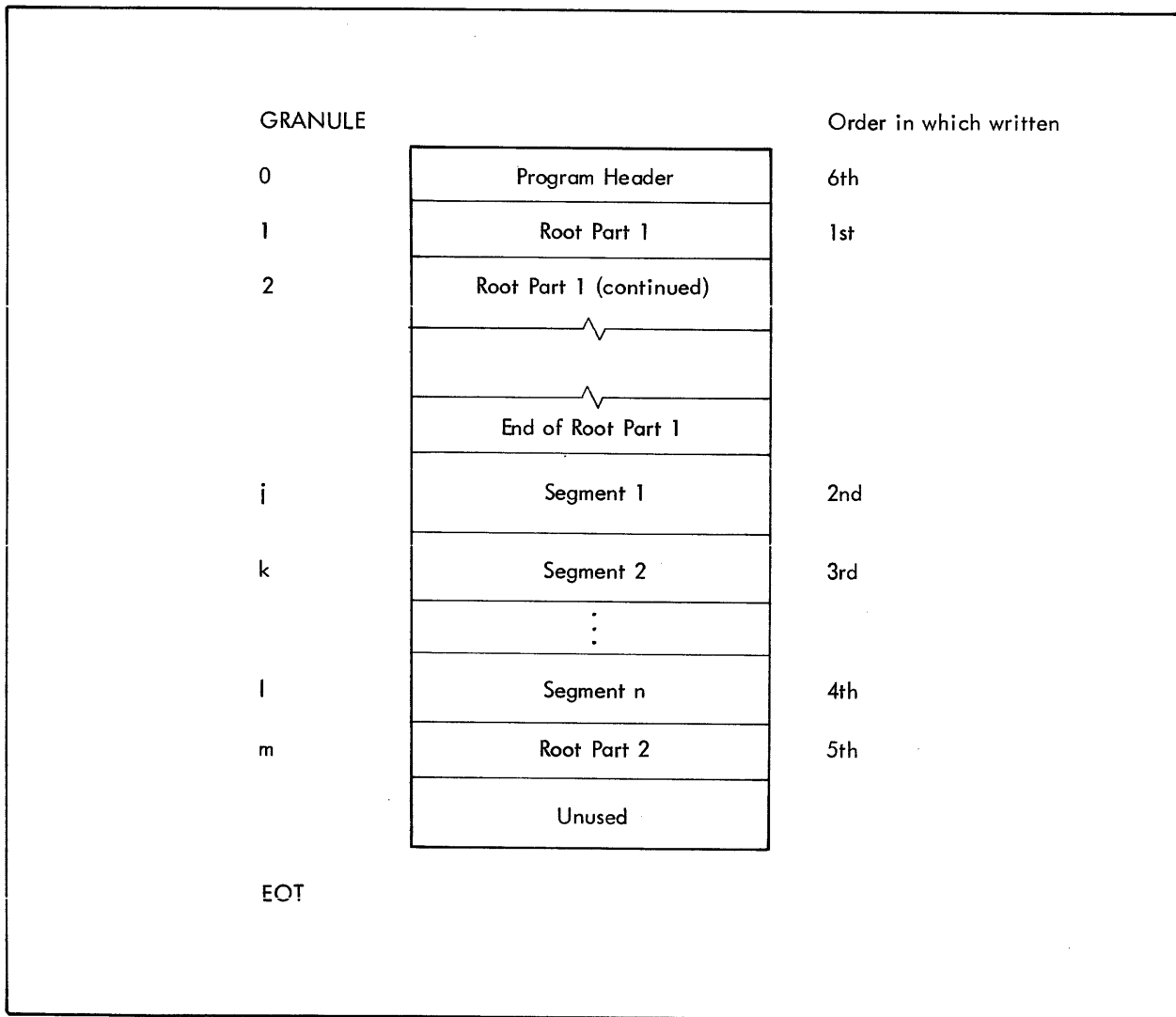


Figure 57. Program File Format

PL = 01 Public Library.
EXLOC execution location.

Logical Flow of the Overlay Loader

After the Root segment has been loaded by the JCP, the Root calls the Monitor SEGLOAD function to read CCI into the overlay area and then transfers control to CCI to process the IOLOAD control command.

Logical Flow of CCI

When CCI is called, there is usually a control command in the control command buffer (B:C). If not, CCI reads the next command into B:C and logs it onto LO. If the command terminates a :ROOT, :SEG, or :MODIFY substack, PASSONE is called; if it terminates an :ASSIGN substack, PASSTWO is called. If the command does not terminate a substack, CCI scans the options specified and performs the following functions for the different control commands.

IOLOAD Command. CCI sets flags; puts the program file name in M:OV DCB; builds T:PL, T:PUBVAL, and T:PUBSYM from files specified in the PUBLIB option; allocates the 14 remaining Dynamic Table areas; and if the GO option has been specified, builds T:ROMI.

:ROOT, :SEG, and :PUBLIB Commands. CCI creates an entry in T:SEG; builds T:ROMI and T:DCBV entries from the specified input options; allocates space for the PCB in the Root segment; and for the :SEG command, calls the PATHEND subroutine. PATHEND determines if the segment exists in a different path; if so, writes out any overlaid segment's T:SYMBOL and T:VALUE subtables on the RAD scratch files; and sets the byte displacement points for the new segment's T:SYMBOL and T:VALUE subtables.

Logical Flow of PASSONE

PASSONE branches to process T:MODIFY if CCI has just been previously called by PASSONE to input :MODIFY commands. Otherwise, PASSONE processes T:ROMI which has been built by either CCI or LIB. PASSONE inputs the ROMs from the devices specified in T:ROMI; builds T:MODULE entries for each ROM input; saves ROMs input from non-RAD devices onto the X1 scratch file; and scans the ROMs for pass-one type load items. It then builds the following entries:

1. Parallel T:SYMBOL and T:VALUE entries from external DEF, PREF, SREF, and DSECT declarations. Entries in T:VALX are built when expressions defining DEFs cannot be resolved. Except for blank COMMON, a DSECT is allocated when first encountered, and its address is stored in the T:VALUE entry.
2. T:DCB entries from external DEF and REF declarations that begin with either M: or F:. The address of the DCB is either defined with an expression (for DEFs), or allocated by PASSTWO (for REFs) and stored in the T:DCB entry.
3. T:CSECT entries and allocates CSECTs when encountered.
4. T:FWD entries when FWDs are defined. Entries in T:FWDX are built when expressions defining FWDs cannot be resolved.
5. Entries in T:DECL whenever a DEF, REF, SREF, CSECT, or DSECT declaration is encountered.

At module end, the four module tables (T:DECL, T:CSECT, T:FWD, and T:FWDX) are packed together and moved to B:MT. If the buffer is full, the tables are output on X2.

When all the entries in T:ROMI have been processed, PASSONE determines whether the libraries specified have been searched. If not, PASSONE calls LIB to search the library specified. Note that the library is searched and the ROMs from the library are loaded before the next library is searched.

If there are any :MODIFY commands for the segment, PASSONE calls CCI. After CCI recalls PASSONE, control is returned to this point where T:MODIFY and T:MODULE are packed together and output on X3.

If there is a :SEG command in B:C, PASSONE calls CCI. Otherwise, the end of PASSONE is signaled. Blank COMMON is allocated at the end of the longest path (if not allocated previously) and the remaining T:SYMBOL, T:VALUE subtables are output. The resident table areas (T:DCB, T:SEG, T:DCBV, T:VALX) are set equal to the

actual lengths of the data in the tables. The T:ROMI area length is set to zero (since it is not used by PASSTWO) and an end-of-file is written on X1. If any of the six scratch files overflowed, MAP is called; otherwise, PASSTWO is called.

Logical Flow of LIB

The LIB segment first packs the 16 Dynamic Tables together at the top of the Dynamic Table area. The remaining space will be used for the LIB's tables. (Whenever enough room does not exist for the LIB's tables, the "pack" is written on the RAD scratch file, X6.) LIB then creates T:LDEF, starting from the end of the "pack".

The FWA of the EBCDIC, DEFREF, and MODIR files' buffer is calculated by subtracting the length of the longest file from the end of the Dynamic Table area. The EBCDIC file is read into the buffer and the entries in T:LDEF are converted to point from T:SYMBOL to entries in the EBCDIC file. T:LDEF entries not having corresponding EBCDIC entries are changed to null entries.

The DEFREF file is then read into the buffer. LIB uses the DEFREF file to satisfy PREFs in T:LDEF. All the DEFs and REFs from an entry in the DEFREF file are added to T:LDEF if at least one of the DEFs satisfies a PREF in T:LDEF. The pointer to the ROM's MODIR file entry is saved in T:LROM, which is built backwards, beginning from the top of the DEFREF buffer. The DEFREF search is finished when all the PREFs in T:LDEF, that can be, are satisfied. T:LROM now contains pointers to all the library ROMs, and T:LDEF is no longer required.

The MODIR file is read into the buffer and the T:LROM entries are changed to point to the ROM's starting record number in the MODULE file.

The packed tables are read from the RAD (if they were saved in X6), and T:LROM is moved to the temporary buffer (TEMPBUF) inside the LIB overlay while the Dynamic Tables are being unpacked. Note that if the DIAG segment were to be called at this point, TEMPBUF would be destroyed. T:LROM entries are converted into T:ROMI format and added to T:ROMI in the Dynamic Table area. PASSONE is then called to input the ROMs specified in T:ROMI.

Logical Flow of PASSTWO

PASSTWO branches to process T:ASSIGN if CCI has just been previously called by PASSTWO to input :ASSIGN commands. Otherwise, it reorganizes the Dynamic Table area and moves the resident tables T:SEG, T:DCBV, and T:DCB to the end of T:PUBVAL and locates T:VALUE at the end of T:DCB. PASSTWO then allocates part two of the Root either at the end of the longest path or at the end of blank COMMON, whichever is greater.

PASSTWO is now ready to process the segments. It points to the first/next T:SEG entry; reads the segment's T:VALUE subtable into T:VALUE; calculates the number of granules required for the segment on the Program File; creates T:GRAN at the end of T:VALUE; reads the segment's T:MODIFY and T:MODULE tables at the top of T:VALX; and allocates the Work area (which is divided into granule partitions and contains all or part of the segment's partitioned core image) at the end of T:GRAN. The Work area extends to the Module Tables Buffer (B:MT), which varies in size, and is allocated backwards from the top of T:MODIFY. The Work area is dynamic and changes in size either when tables in B:MT are no longer required, or when another set of Module Tables is input.

PASSTWO is now ready to process the segment's ROMs. It points to the first/next T:MODULE entry; reads in the first/next set of Module Tables into B:MT if necessary; points to the current module's T:DECL, T:CSECT, T:FWD, and T:FWDX table; inputs the ROM; scans the load items; creates the absolute core image in the Work area using T:GRAN to locate the granules; and if the Work area gets full, outputs the necessary granules to the Program File.

PASSTWO repeats this cycle until all the modules in the segment have been input and then writes the granules remaining in core onto the program file. It then points to the next T:SEG entry and repeats the outer cycle until all the segments in the program have been created.

If a Public Library is not being created, PASSTWO builds T:GRAN for part two of the Root, located at the end of T:DCB. If there is an :ASSIGN command in B:C, PASSTWO allocates T:ASSN from the end of T:GRAN to the beginning of T:VALX and calls CCI to build T:ASSN. After CCI recalls PASSTWO, control is returned to this point. PASSTWO allocates the Work area at the end of T:ASSN (which may be of zero length); creates OVLOAD, DCBTAB, INTTAB, and the referenced DCBs; reassigns DCBs referenced in T:ASSN; writes part two of the Root on the Program File; creates the program header; and writes it on the Program File. If a Public Library is being created, T:SYMBOL and T:VALUE are output on the Program File. PASSTWO then exits by calling the MAP.

Logical Flow of MAP

MAP moves T:SEG and T:DCB to the top of the Dynamic Table area, and unless "no MAP" was specified, outputs the program header information.

MAP points to the first/next T:SEG entry, and unless "no MAP" was specified, outputs the segment's header information. If either the PROGRAM or ALL option was specified, MAP reads the segment's T:MODIFY and T:MODULE tables into core at the end of T:DCB; locates B:MT at the end of T:MODULE; uses T:MODULE to read in the Module Tables associated with the segment; maps the segment's control sections (including Library CSECTs if ALL specified); and if this is the Root segment, lists T:DCB.

Regardless of the option specified, MAP reads the segment's T:SYMBOL and T:VALUE subtables into core at the end of T:DCB. If the ALL option was specified, MAP reads T:PUBSYM and T:PUBVAL in as part of the root's external table and lists all the symbols in the external table. If the PROGRAM option was specified, MAP lists all the non-library symbols in the external table. If either the SHORT or "no MAP" option was specified, MAP lists only the duplicate DEFs, undefined DEFs, unsatisfied REFs, and duplicate REFs.

This cycle is repeated until all the entries in T:SEG have been mapped. If a RAD file used by the Loader overflowed, the number of granules used or needed for all files is listed. Otherwise, this information is output only if either the PROGRAM or ALL option was specified.

MAP terminates the Overlay Loader by either calling the Monitor EXIT function or ABORT function. MAP aborts and destroys the Program File if either a RAD file overflowed or there were loading errors when a Public Library was being created.

Logical Flow of DIAG

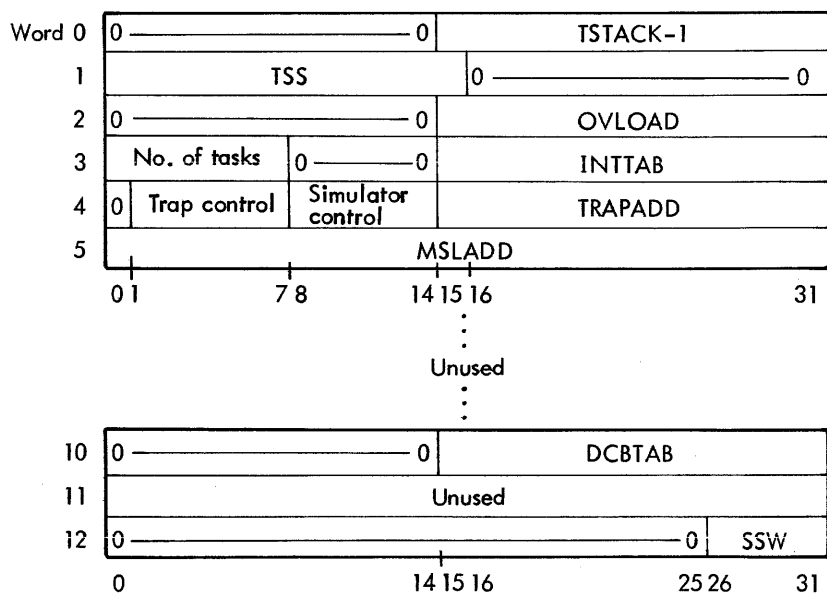
When the DIAG overlay is called, the environment of the calling program is unchanged. Since the DIAG segment overlays the calling segment, all the temporary and permanent storage cells used by the calling segment are located in either the Root or the Dynamic Table area. DIAG is called by the RDIAG subroutine which exists in the Root. When RDIAG is called, it saves the 16 registers and then calls in DIAG via the Monitor SEGLOAD function. DIAG outputs the specified diagnostic and depending upon the exit code associated with the diagnostic, either aborts, returns to RDIAG, or calls the Monitor WAIT function. If control is returned from the WAIT function, DIAG returns to RDIAG. RDIAG then reloads the calling segment via the Monitor SEGLOAD function, restores the 16 registers, and returns to the calling segment at the address following the RDIAG call.

Loader-Generated Table Formats

The Loader creates the program's Program Control Block (PCB), DCB Table (DCBTAB), Interrupt Table (INTTAB), and Segment Loading Table (OVLOAD).

PCB

The PCB exists as part of the Root segment and is initialized (except for words 4 and 12) by PASSTWO, when the Root segment is created. The PCB has the format



where

TSTACK is the address of the current top of the user's Temp Stack.

TSS indicates the size, in words, of the user's Temp Stack.

OVLOAD is the address of the table used by the SEGLOAD function to read in overlay segments or zero.

No. of tasks is the number of tasks in the program. This is also the number of entries in INTTAB.

INTTAB is the address of the interrupt table associated with the program or zero. This table is maintained by the CONNECT function. The format of this table is shown below.

Trap control Bits 1-7 specify how the various traps are to be handled.

Simulator control is used by the unimplemented/nonexistent instruction trap handler.

TRAPADD is the address of the user's routine that processes the various traps.

MSLADD is the address of the M:SL DCB used to load overlay segments.

DCBTAB is the address of a table of names and addresses of all of the user's DCBs. This table has the form given below.

SSW contains the user's sense switch settings. Bit 26 contains the setting of switch 1, etc.

DCBTAB

DCBTAB is built from T:DCB, and is located in part two of the Root. DCBTAB has the format

| | | | | | | |
|---------|---|---------------------------------|-----|-------|-------|----|
| Word | 0 | Total number of entries | | | | |
| Entry n | 1 | E1 | E2 | E3 | E4 | |
| | 2 | E5 | E6 | E7 | E8 | |
| | 3 | FWA of DCB's execution location | | | | |
| | | 0 | 7 8 | 15 16 | 23 24 | 31 |

where

E1-E8 is the EBCDIC name of the DCB (left-justified with trailing blanks).

INTTAB

INTTAB is built only if the program has at least one task connected to an interrupt. INTTAB is located in part two of the Root and has the format

| | | | |
|------|------|------|------|
| I | | | |
| ⋮ | | | |
| INT4 | INT3 | INT2 | INT1 |

where

I is the index value used to access the next available entry in the table $0 < I \leq 4N - 1$.

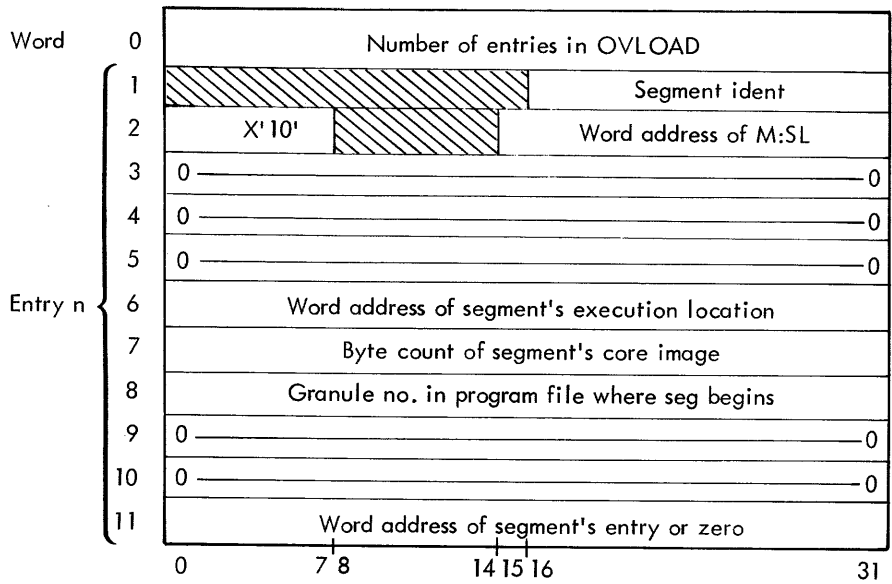
N is the number of words allocated by the Loader for the table. The table is maintained by the CONNECT system call. The initial value of I is set by PASSTWO.

Each byte represents the priority of the interrupt, where 1 represents the highest priority, and corresponds to interrupt location X'50'.

OVLOAD

The OVLOAD table contains the information necessary for the Monitor SEGLOAD function to read in overlay segments at execution time. One entry is created for each overlay segment. Thus, a program consisting only of a Root would not have an OVLOAD Table.

OVLOAD is located in part two of the Root. The format of an entry is such that it can be used as an FPT by SEGLOAD to read in the requested segment. OVLOAD has the format



Loading Overlay Loader

Before the Overlay Loader can be loaded, the OLOAD file in the SP area must be previously allocated by the RAD Editor. It is loaded by the JCP Loader with the !LOAD command. It is critical that the ROMs of the Overlay Loader's segments be ordered correctly, so that the segment's idents assigned by the JCP Loader coincide with the idents used within the program. The segment idents are listed below:

| SEG | IDENT |
|---------|-------|
| ROOT | 0 |
| CCI | 1 |
| PASSONE | 2 |
| PASSTWO | 3 |
| MAP | 4 |
| DIAG | 5 |
| LIB | 6 |

The overall flow of the Overlay Loader is illustrated in Figures 58 through 65.

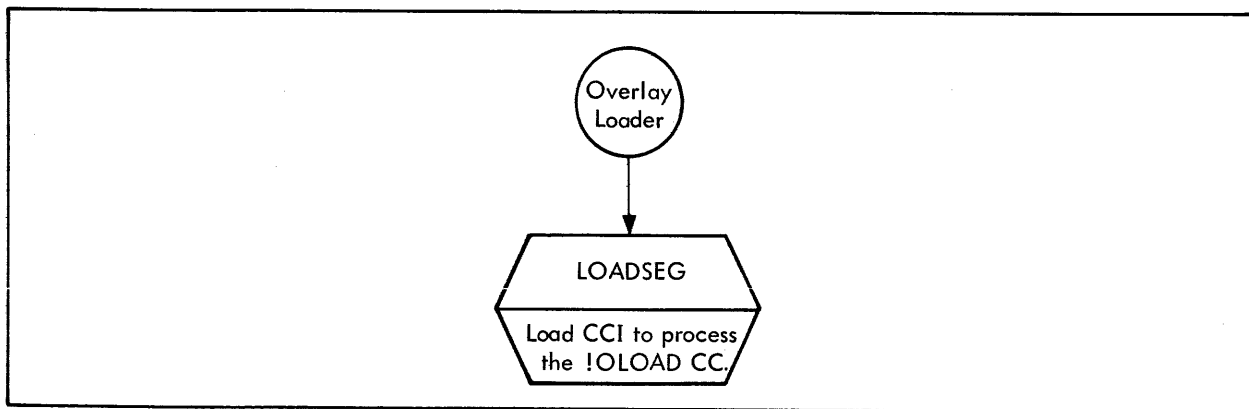


Figure 58. Overlay Loader Flow, !OLOAD

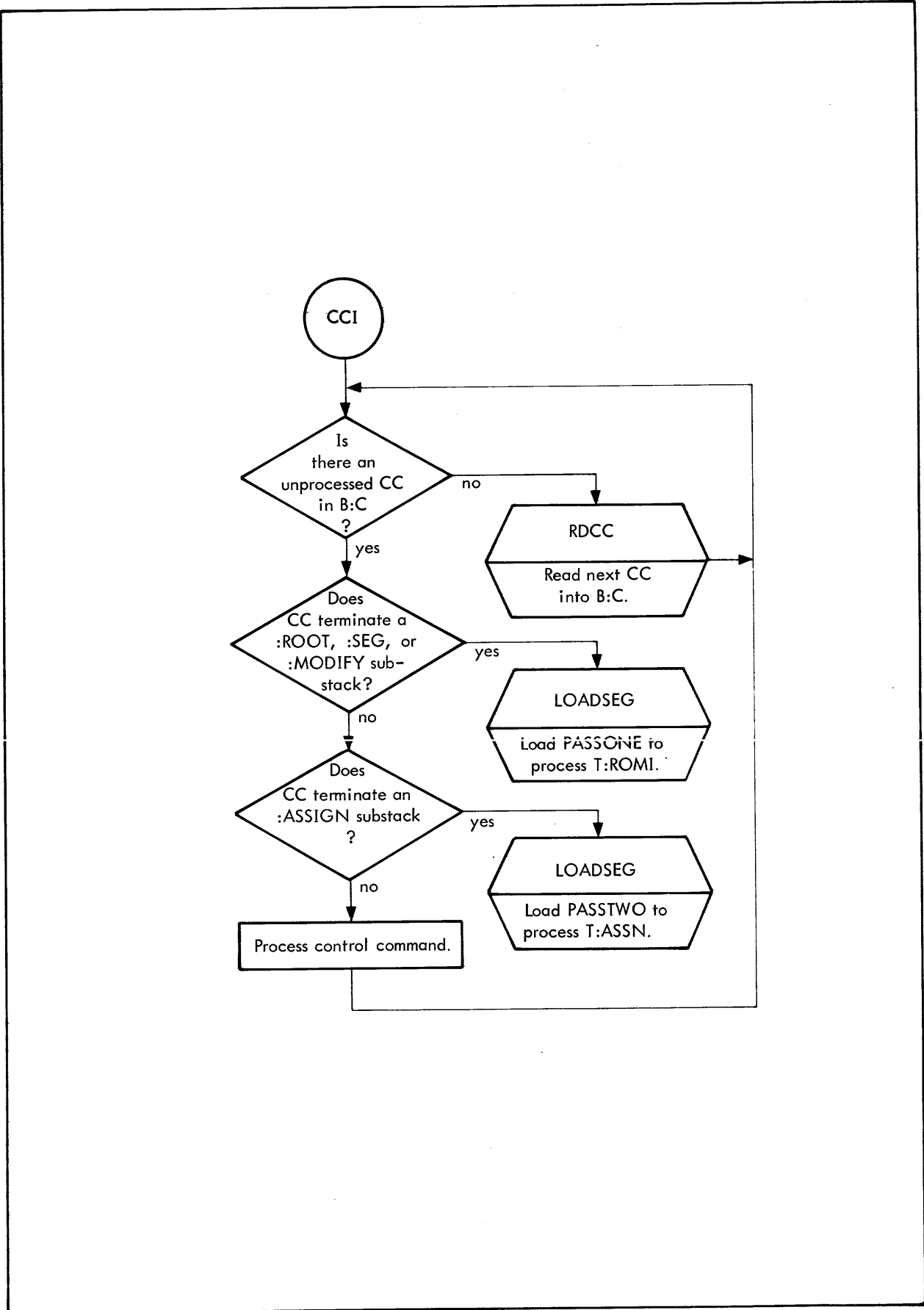


Figure 59. Overlay Loader Flow, CCI

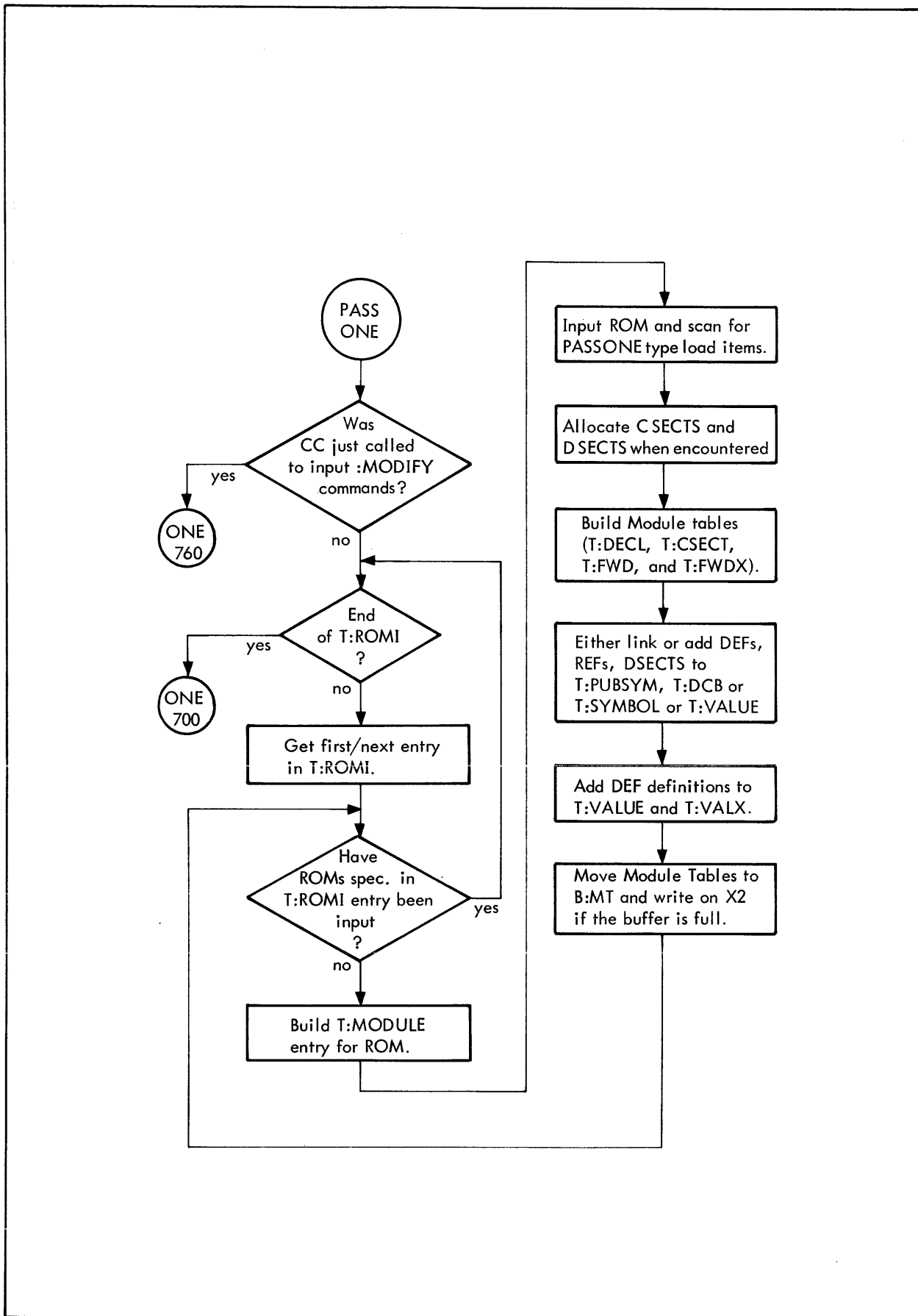


Figure 60. Overlay Loader Flow, PASSONE

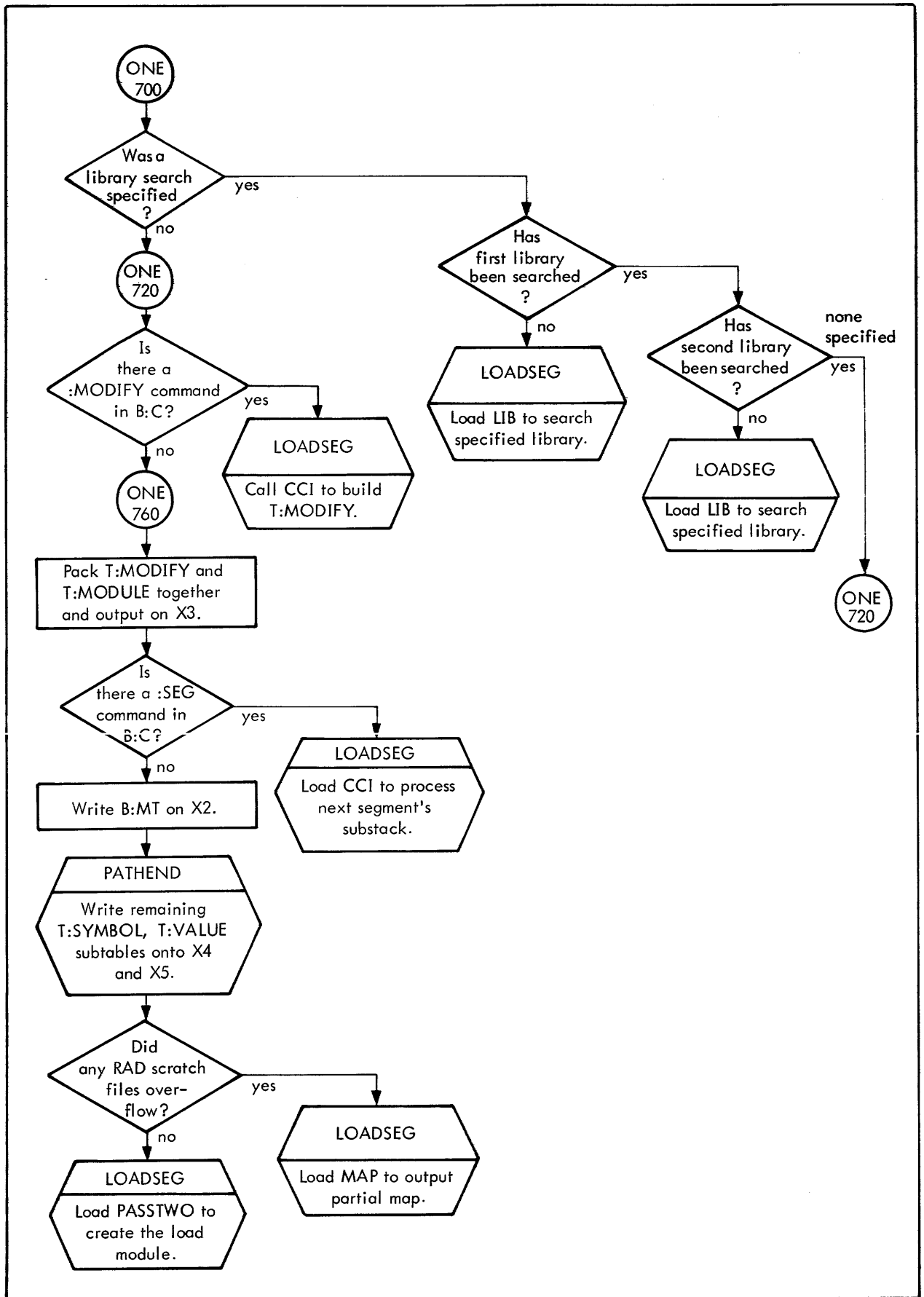


Figure 60. Overlay Loader Flow, PASSONE (cont.)

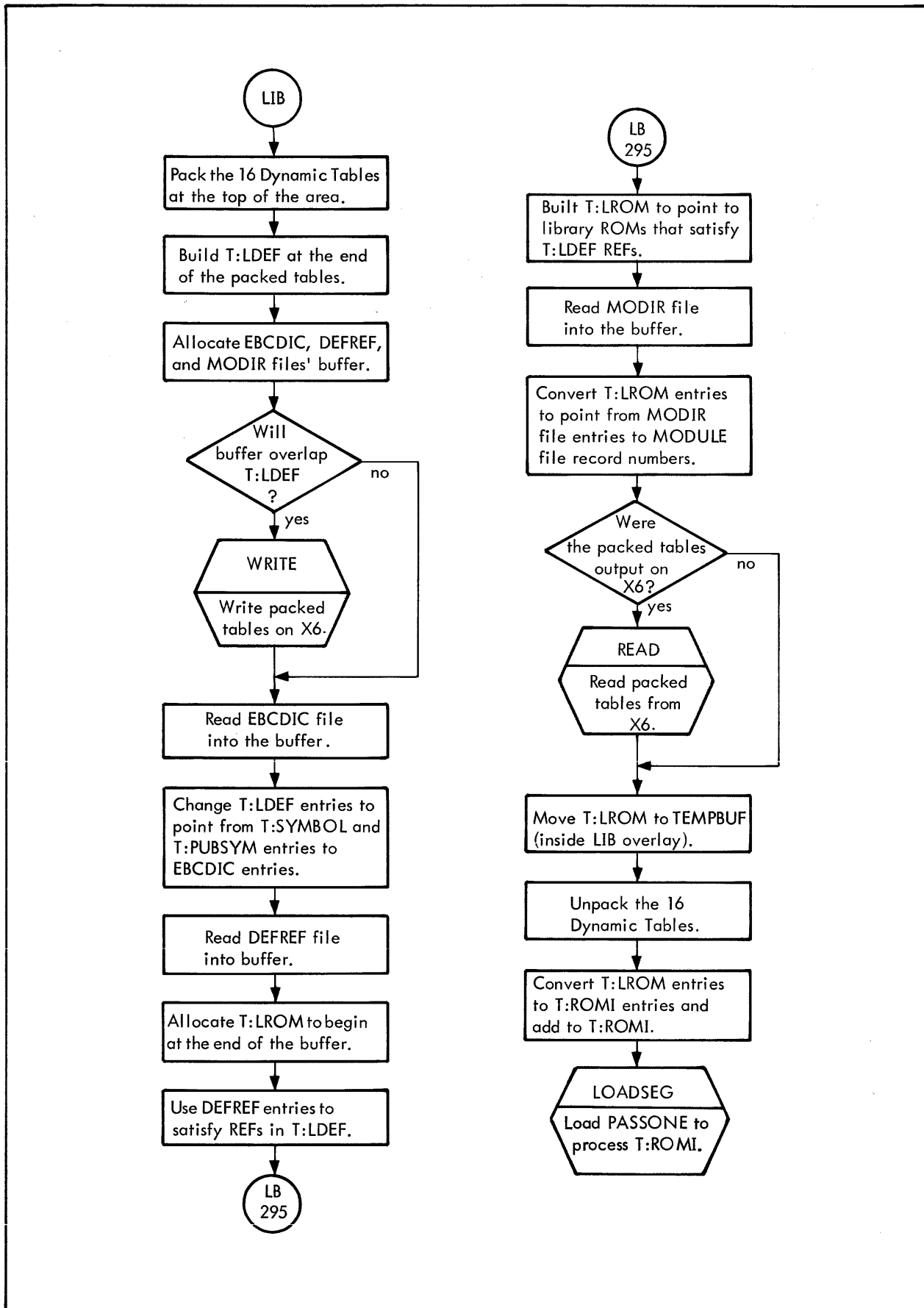


Figure 60. Overlay Loader Flow, PASSONE (cont.)

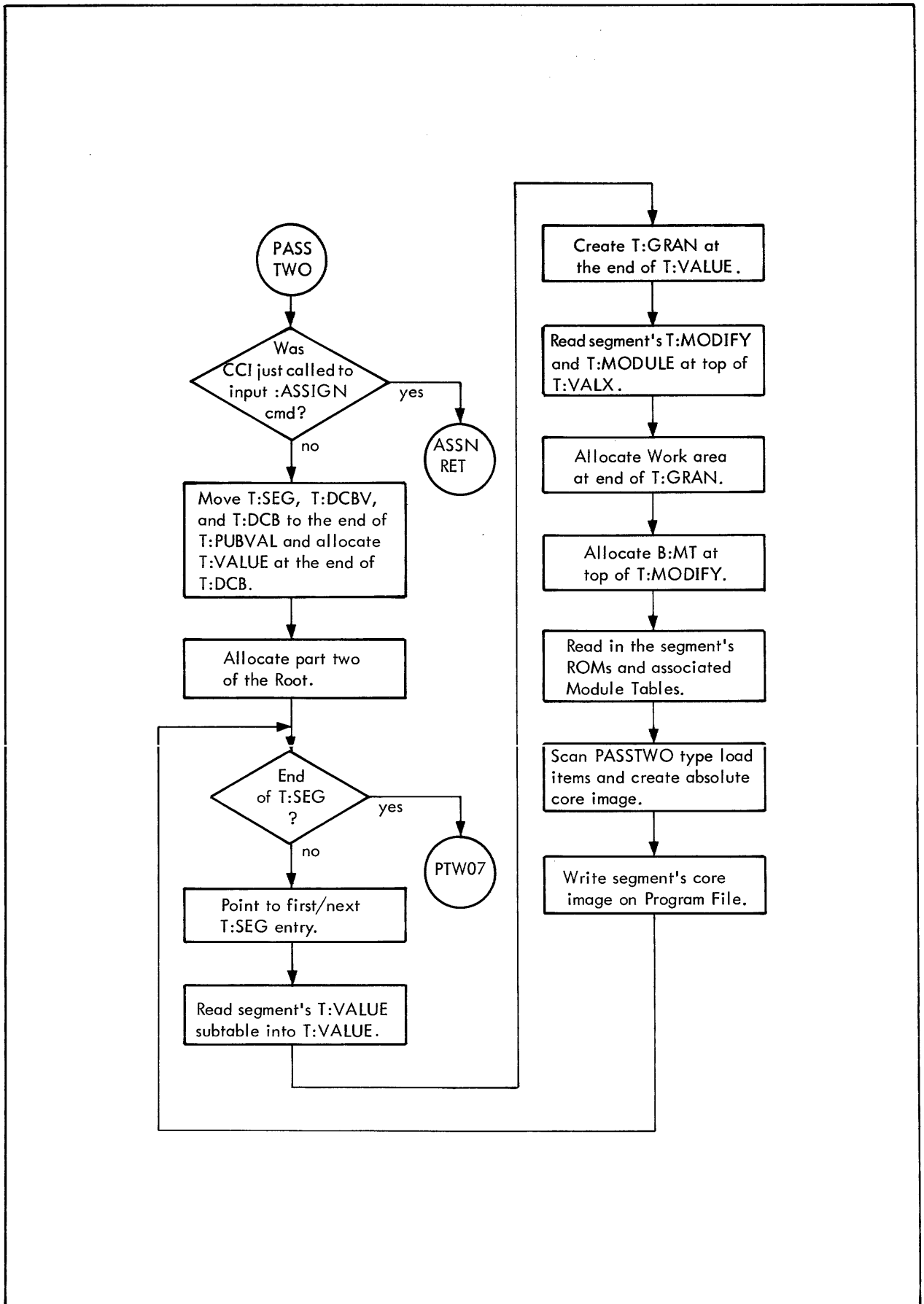


Figure 61. Overlay Loader Flow, PASSTWO

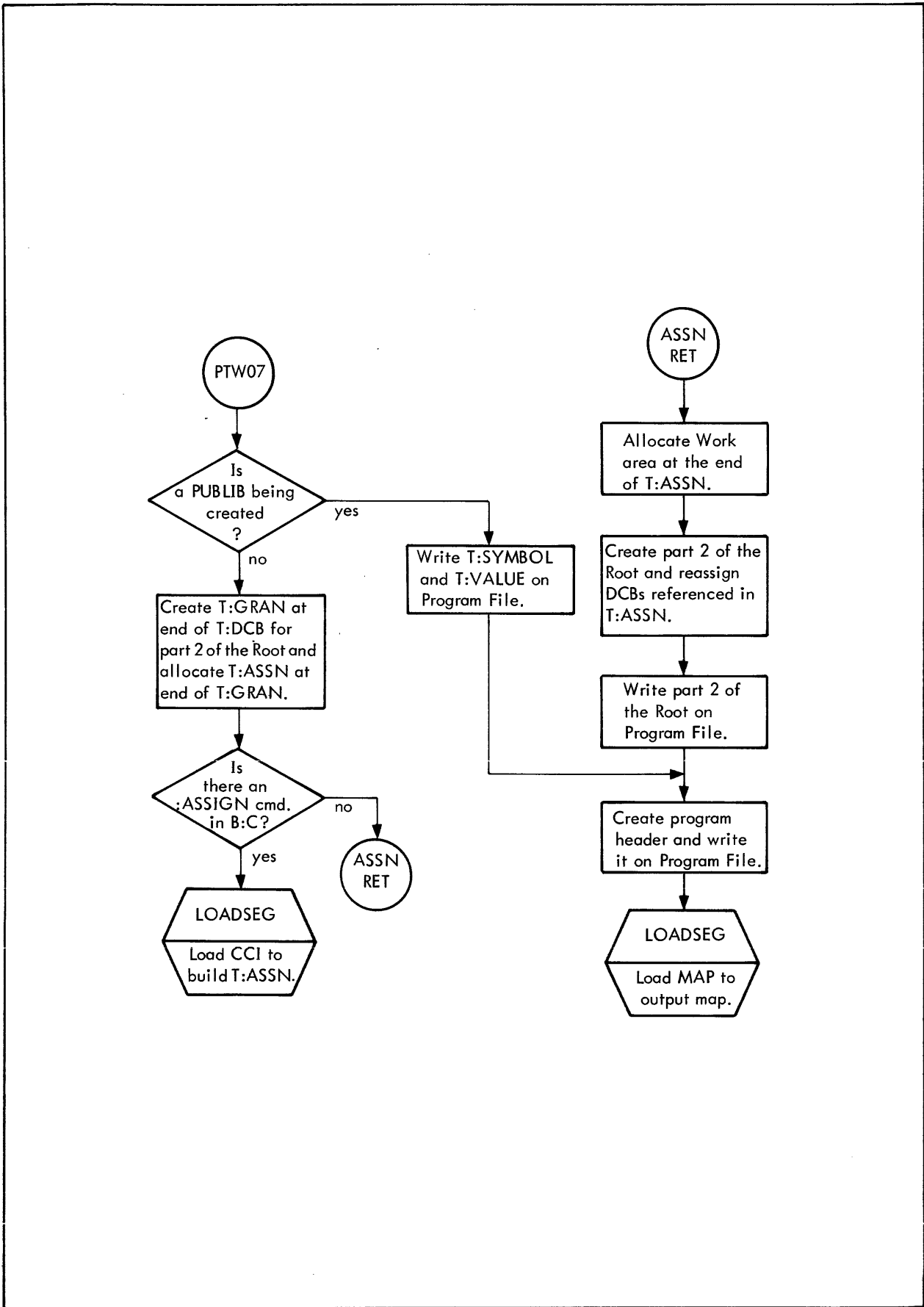


Figure 61. Overlay Loader Flow, PASSTWO (cont.)

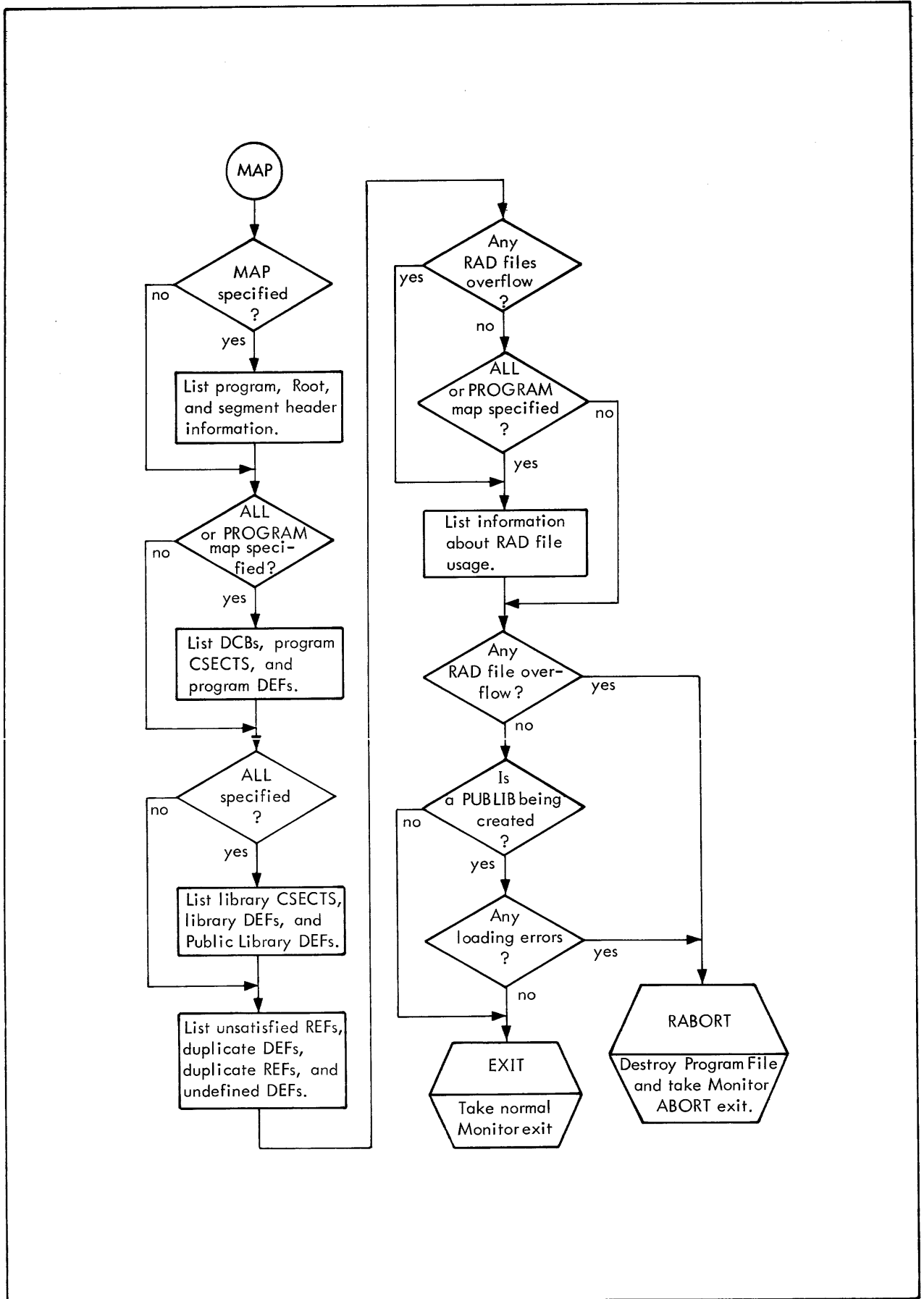


Figure 62. Overlay Loader Flow, MAP

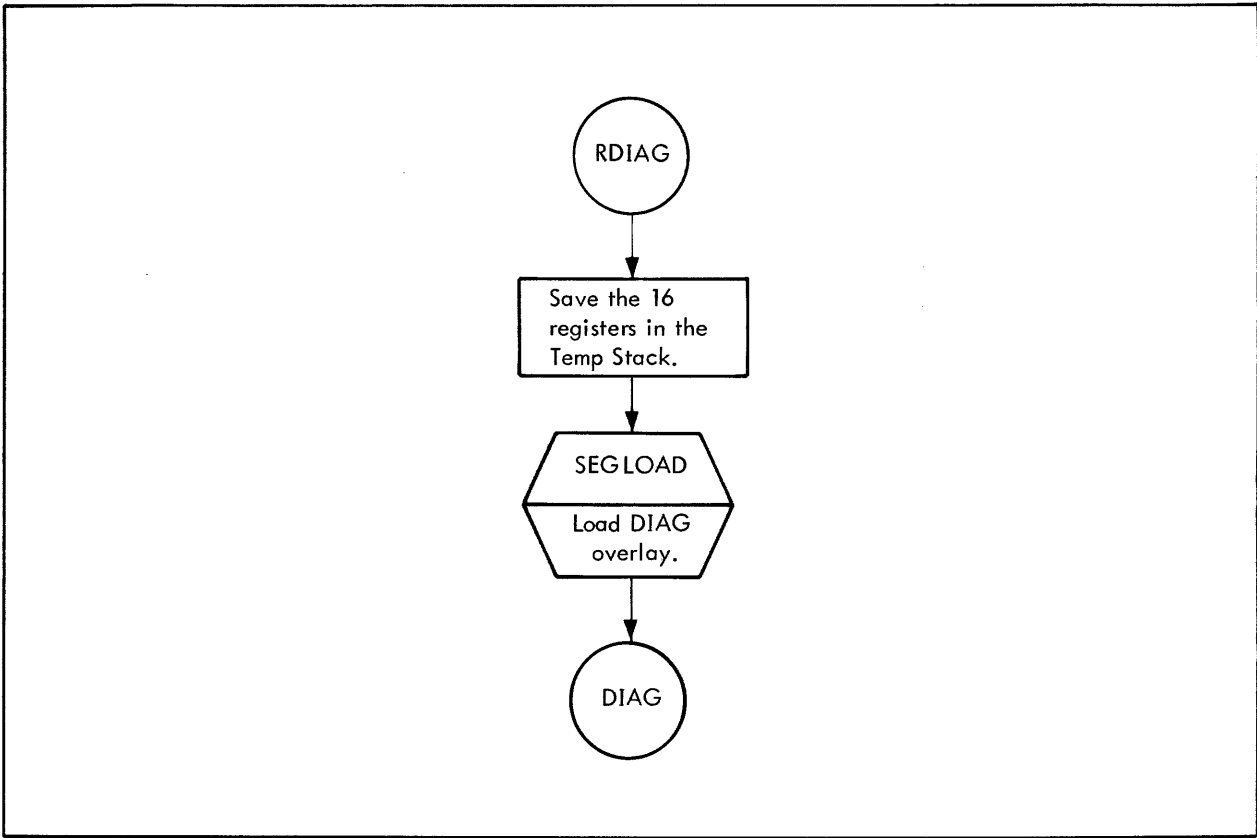


Figure 63. Overlay Loader Flow, RDIAG

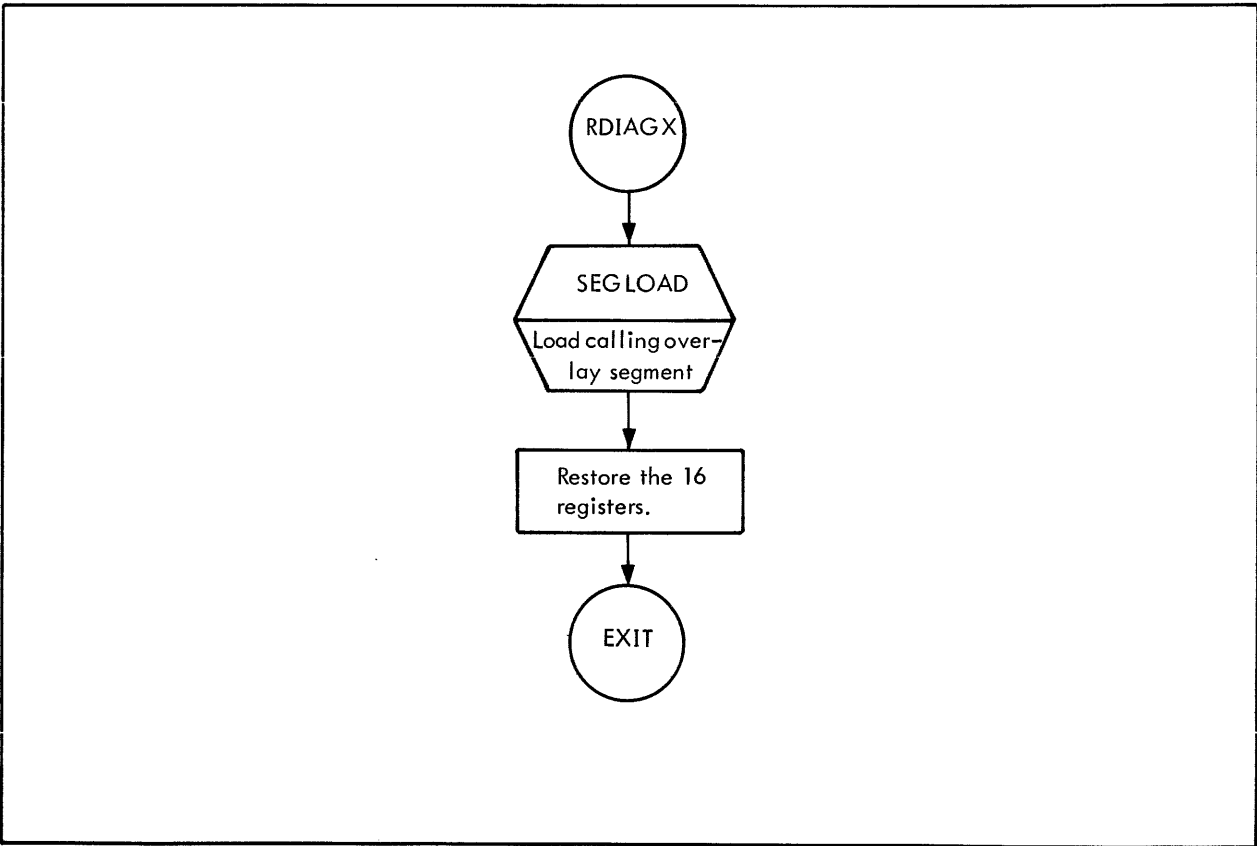


Figure 64. Overlay Loader Flow, RDIAGX

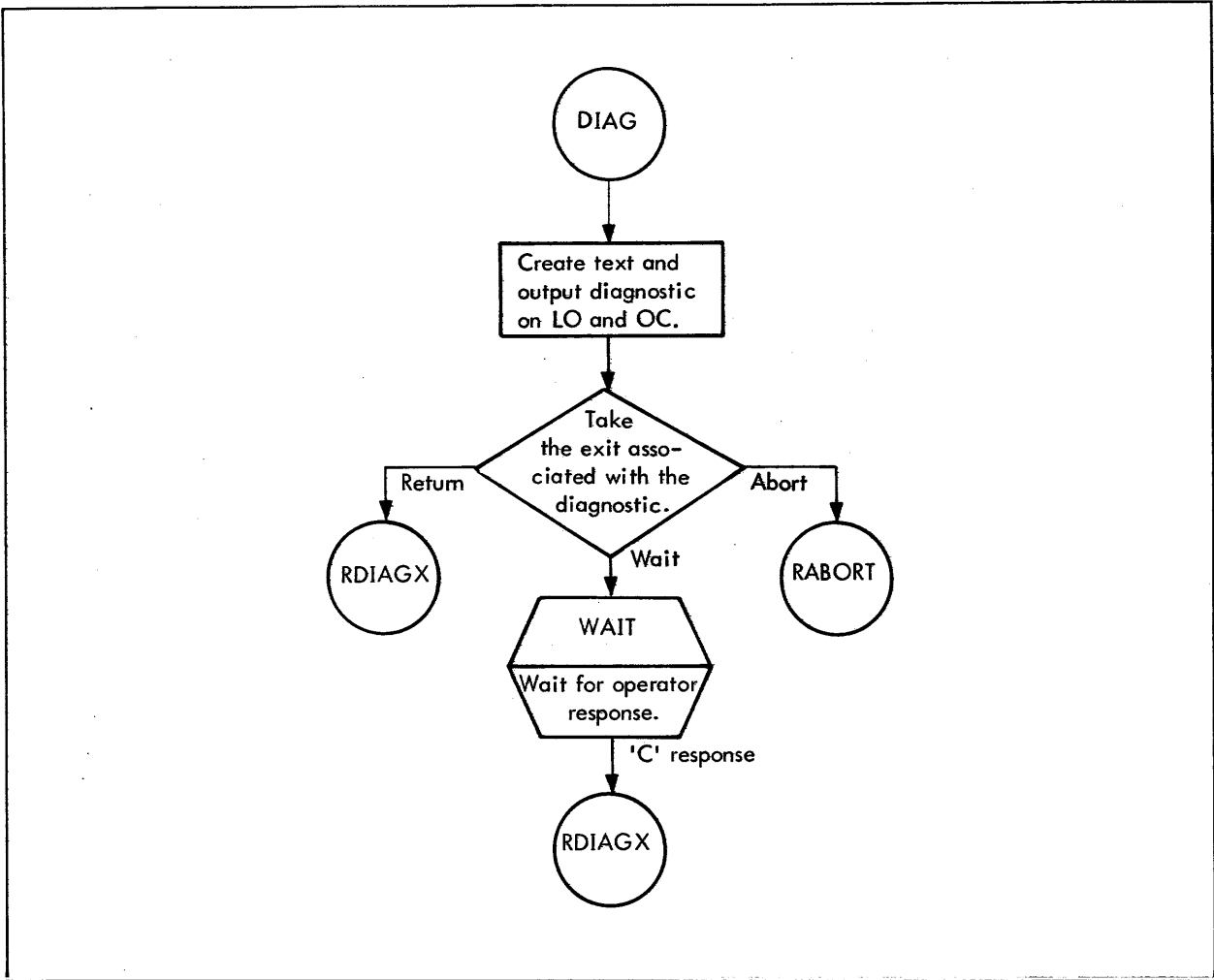


Figure 65. Overlay Loader Flow, DIAG

10. RAD EDITOR

The program listing for each RAD Editor subroutine is prefaced with a description that includes both the purpose of the routine and the calling sequence. Therefore, to avoid duplication, this information is not included in this manual.

Functional Flow

RBM loads and transfers control to the RAD Editor upon reading a !RADEDIT control command from the C device. The Executive routine of the RAD Editor initializes DCBs and the Scan routine parameters, scans the command, loads the required segment if not already in core, and transfers to the proper routine. The RAD Editor is exited when a command with an exclamation character (!) in column one is encountered (with the exception of !EOD). An !EOD is used to indicate an end-of-data to the RAD Editor when data is input via the :COPY command.

A functional flow diagram is shown in Figure 66.

Permanent RAD Area Maintenance

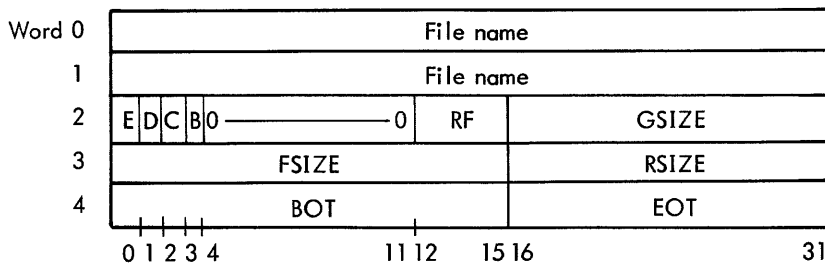
A Master Directory, located in the resident portion of RBM, is set up at System Generation. The format of the Master Directory is illustrated in Chapter 8 of this manual.

The Master Directory contains an entry for each permanent RAD area, extending over the area indicated by the starting and ending addresses. Within each permanent RAD area, the RAD Editor allocates and maintains files by means of a file directory. The RAD Editor maintains permanent file directories for the following permanent RAD areas:

- Foreground Program Area
- System Program Area
- Background Program Area
- Data Areas

Permanent File Directory

The RAD Editor controls file allocation by generating and maintaining a permanent file directory for each area. The file directory begins in the first sector of a permanent RAD area, and each entry in the file directory defines a file in an area and describes the format of the file. It has the form



where

File name is to a file name with a maximum length of eight alphanumeric characters. If File name - 0, it indicates a deleted entry; -1 indicates a bad track entry.

- E If E = 0, sequentially written file; if E = 1, not sequentially written file.
 - D If D = 0, not directly written file; if D = 1, directly written file.
- } Maintained by the Monitor; initially set to 0.

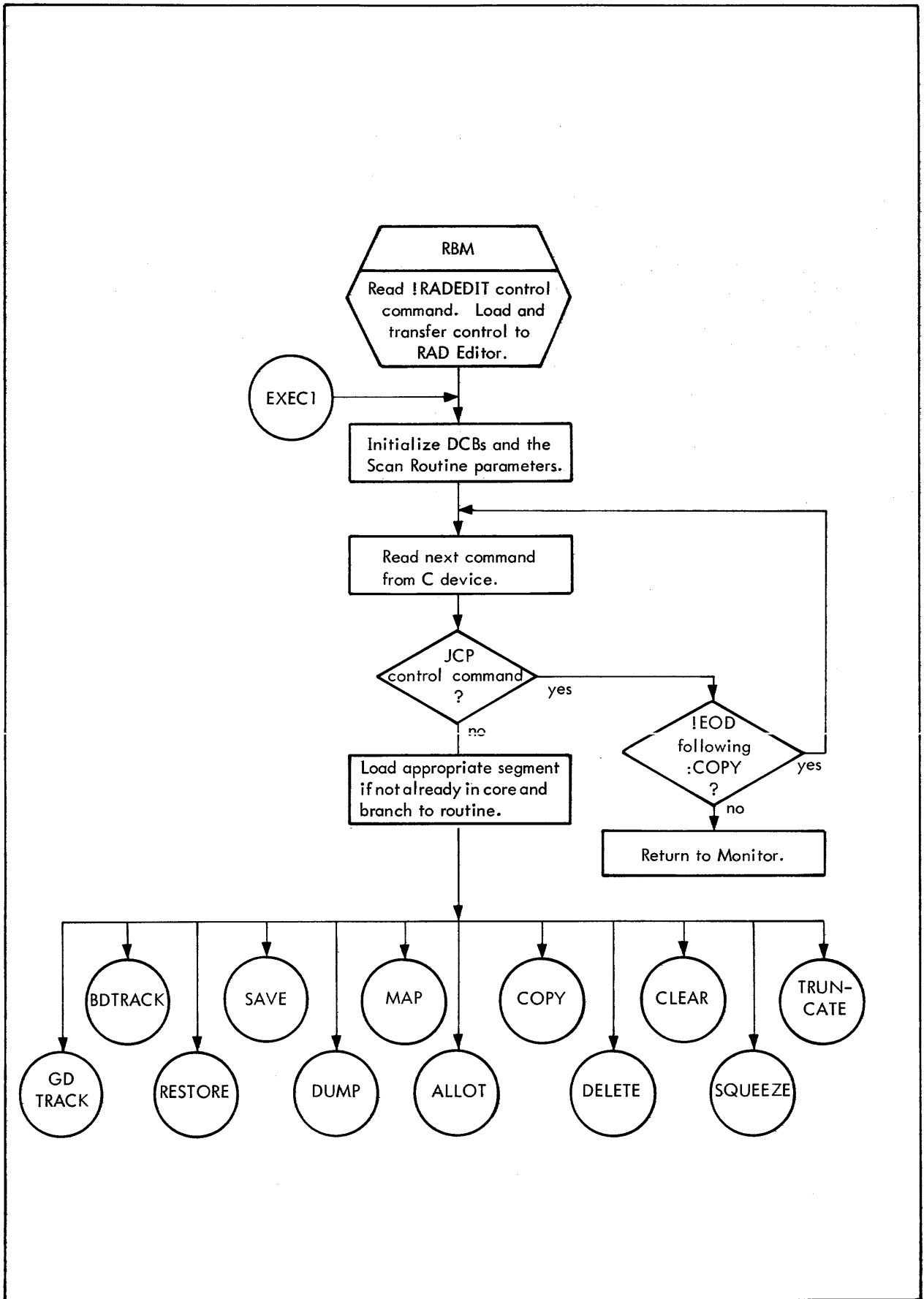
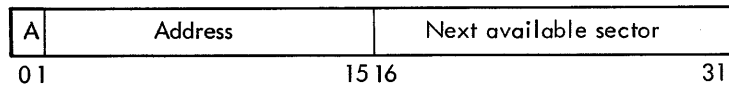


Figure 66. RAD Editor Functional Flow

- C If C = 0, not compressed records; if C = 1, compressed records.
- B If B = 0, unblocked records; if B = 1, blocked records.
- RF If RF = 0, background or nonresident foreground program; if RF = 1, resident foreground program.
- GSIZE is the granule size, in bytes, to be used for direct accessing.
- FSIZE is the current number of records in file.
- RSIZE is the number of bytes per logical record.
- BOT is the relative RAD address of first sector defined for the file.
- EOT is the relative RAD address of last sector defined for the file.

No entry extends over a sector boundary. After a sector of directory is filled, the next available sector within the permanent RAD area is allocated as a continuation of the directory. Sectors of a directory are linked by means of a one-word identification entry which is the first word of every sector of the directory. It has the form



where

- A If A = 0, the directory ends in this sector; if A = 1, the directory is continued on another sector.
- Address If A = 0, "Address" contains the relative location within the sector available for the next entry; if A = 1, "Address" is the relative RAD address of the sector where the directory is continued.
- Next available sector is the relative RAD address of the first unused sector in the area. This word is meaningful only for the last sector of directory.

Space within the permanent RAD area is allocated sequentially. The first file in an area, which corresponds to the first entry in the sector of directory, begins in the second sector and extends over an integral number of sectors. Every file begins and ends on a sector boundary.

Control Commands

The permanent RAD areas are maintained through the execution of :ALLOT, :DELETE, :TRUNCATE, and :SQUEEZE commands.

The permanent file directories are maintained so that the directory entry defining a file is always contained in a sector of directory that has a lower sector address than the file it defines. To facilitate maintenance, files always appear in the same order as the entries in the file directory.

:ALLOT The permanent RAD area specified on the command determines the area in which a file is to be allocated. The FILE, FORMAT, FSIZE, RSIZE, GSIZE, and RF parameters are used to form a new directory entry.

The new entry is added to the current sector of directory (identification entry with A = 0) at the location specified by "Address" in the identification entry. The BOT of the new entry is set equal to the "Next available sector". The EOT is computed, using the FSIZE, RSIZE, and FORMAT parameters. The identification entry is updated to reflect the new entry. The "Next available sector" is set = EOT of the new entry + 1, and the "Address" is incremented by 5.

If there is insufficient space in the current sector of directory for another entry, "A" in the identification entry is set to 1; "Address" is set = "Next available sector" and that sector address is used for the new sector of directory. A new identification entry is built by setting "A" = 0; "Address" = 6; and "Next available sector" = EOT of the new entry + 1.

If there is insufficient space to allocate for a file, the file directory is searched for deleted entries (file name = 0). The deleted entry that allocates sufficient space and the least amount of space is selected for the new entry. RAD space is lost if the deleted entry allocates more area than is required by the entry. This space can be made available for allocating by executing a :SQUEEZE command. The area allocated by a new entry is zeroed out.

The number of sectors to allocate for a file is calculated using the formulas

$$C = \left(\frac{FSIZE}{25} + r \right) * \left(\frac{256}{s} + r \right)$$

$$B = \left(\left(\frac{FSIZE}{RSIZE} + r \right) * \frac{256}{s} \right)$$

$$U = ((RSIZE/s)+r)*FSIZE$$

where

r = 1 if remainder \neq 0, and 0 if remainder = 0.

x equal RAD sector size in words.

:DELETE The permanent RAD area specified on the command determines the area in which a file is to be deleted, and the file name is used to search the file directory for the entry to be deleted, with the first four words of the file directory entry being zeroed out. The last word of the entry (BOT and EOT) remains unaltered. The space formerly allocated by the entry becomes unused until either a :SQUEEZE command is executed, or an :ALLOT command is executed with insufficient space on the end of an area to allocate. Space is then allocated by using a deleted entry.

:TRUNCATE The permanent RAD area specified on the command determines the area in which a file(s) is to be truncated, with the file name specified being used to search the file directory for the entry to be truncated. The actual size of the file is calculated and the EOT of the file directory entry is updated accordingly.

The actual file size for blocked and unblocked files is determined by using the FSIZE and RSIZE of an entry; for compressed files, an RFT entry (K:RFT11) containing the current record number is used. The space formerly allocated between the EOT of an entry and the BOT of the next entry becomes unused and is not reallocated until a :SQUEEZE command is executed.

:SQUEEZE The parameters on the :SQUEEZE command determine which permanent RAD area to squeeze. Truncating or deleting a file that is subsequently reallocated may cause a loss of space that cannot then be allocated. That is, the current permanent file directory entry allocates less space than allocated by the original entry. Executing a :SQUEEZE regains all unused space. The directories are compacted and the files themselves are moved to regain the unused space. The BOT and EOT entries (of the permanent file directory) are updated as they are compacted to indicate the area occupied by the moved file. Figure 67 illustrates the permanent RAD area before and after squeezing.

Library File Maintenance

Both the System Library files residing in the SP area and the User Library files residing in the FP area have the same file structure. Each library consists of one blocked Module File (MODULE) and three unblocked files: the Module Directory File (MODIR), EBCDIC File (EBCDIC), and DEFREF File (DEFREF).

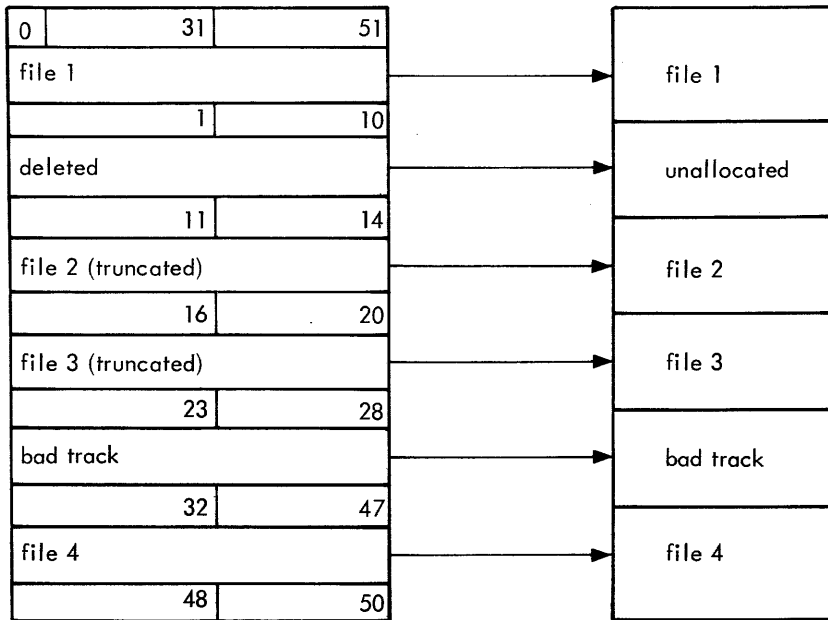
The MODIR File contains general information about each library module, including its name, where in the MODULE File it is located, and its size. The MODULE File contains the object modules. The EBCDIC File contains only the DEFs and REFs of the library modules. The DEFREF File contains indices to the DEFs and REFs in the EBCDIC File for each module. These files must be defined via the :ALLOT command before attempting to generate them via the :COPY command.

Algorithms for Computing Library File Lengths

The following algorithms may be used to determine the approximate lengths of the four files in a library. It is not crucial that the file lengths be exact, since any unused space can be recovered via the :TRUNCATE

Permanent RAD Area Before Squeezing

Identification
Entry



Permanent RAD Area After Squeezing

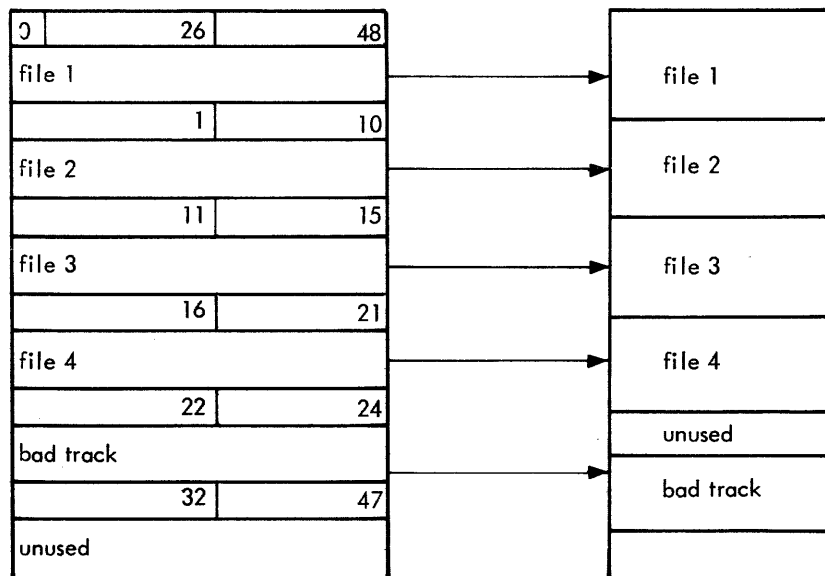


Figure 67. Permanent RAD Area

command. The approximate number of sectors (n_{MODIR}) required in the MODIR File is

$$n_{\text{MODIR}} = \frac{3(i)}{s}$$

where

i is the number of modules to be placed in the library.

s is the RAD sector size in words.

3 words is the length of a MODIR File entry.

The approximate number of sectors (n_{EBCDIC}) = $\frac{2(d)}{s}$

where

d is the unique number of DEFs in the library.

s is the RAD sector size in words.

2 words is the average length of an EBCDIC File entry.

The approximate number of records (n_{MODULE}) required in the MODULE File is

$$n_{\text{MODULE}} = \sum_{i=1}^n C_i$$

where

n is the total number of modules in the library.

C_i is the number of card images in the i th library routine.

The approximate number of sectors (n_{DEFREF}) required in the DEFREF File is

$$n_{\text{DEFREF}} = \frac{\sum_{i=1}^n 1 + \frac{d_i + r_i}{2}}{s}$$

where

n is the total number of routines in the library.

d is the number of DEFs in the i th library routine.

r is the number of REFs in the i th library routine.

s is the RAD sector size in words.

Library File Formats

The library file formats are described below. These files are generated from object modules read in via the :COPY command.

MODIR File

The MODIR File is an unblocked, sequential access file and acts as a directory to the MODULE File. The file always consists of one variable length record that increases in size as object modules are added to the library. There is one entry in the MODIR File for each object module, with each entry consisting of three words.

| | | |
|---------|-------------------------|--------------------|
| Words 0 | MODULE File record no. | Records per module |
| 1 | Module name (first DEF) | |
| 2 | Module name | |
| 3 | MODULE File record no. | Records per module |
| 4 | Module name | |
| 5 | Module name | |
| 6 | | |
| 7 | : | |
| 8 | | |
| 9 | | |
| 10 | . | |
| 11 | . | |
| 12 | | |
| | 0 | 15 16 31 |

where

MODULE File record no. is the relative record within the MODULE File where the object module (corresponding to this entry) begins.

records per module is the number of records in the object module.

module name is the name of the object module that is the first DEF in an object module.

A deleted entry contains zeros in all three words.

MODULE File

The MODULE File is a blocked, sequential access file and contains the object modules. The location of the object module within the file and the size is indicated by the MODIR File entry.

EBCDIC File

The EBCDIC File is an unblocked, sequential access file. The file always consists of one variable length record that increases in size as object modules are added to the library. The EBCDIC File contains all the unique DEFs and REFs in the library object modules.

| | | | | |
|---|---|---|---|---|
| 0 | n | e | e | e |
| 1 | e | n | e | e |
| 2 | e | e | e | e |
| 3 | e | e | | |

where

- n is the number of bytes in entry (including itself).
- e is an external definition or reference in EBCDIC.

DEFREF File

The DEFREF File is an unblocked, sequential access file. The file always consists of one variable length record that increases in size as object modules are added to the library. For each module there is one entry that varies in size according to the number of DEFs and REFs. DEFs always precede the REFs in the entry.

| Entry size (no. 1) | | | | MODIR File index | | | | | |
|--------------------|-------|---|--|--------------------|-------|----|----|---|----|
| d | DEF 1 | | | d | DEF 2 | | | | |
| d | DEF 3 | | | r | REF 1 | | | | |
| r | REF 2 | | | Entry size (no. 2) | | | | | |
| MODIR File index | | | | d | DEF 1 | | | | |
| r | REF 1 | | | r | REF 2 | | | | |
| 0 | 1 | ⋮ | | 15 | 16 | 17 | 18 | ⋮ | 31 |

where

- entry size is the number of halfword entries (including itself) for the object module. $3 \leq \text{entry size} \leq 32,767$.
- MODIR File index is the relative halfword in the MODIR File that identifies the object module. $0 \leq \text{MODIR File index} \leq 32,767$. -1 means a deleted entry.
- d if d = 1, means a DEF entry.
- r if r = 1, means a REF entry.
- def n is the byte index of an external definition in the EBCDIC File.
- ref n is the byte index of an extended reference in the EBCDIC File.

A deleted DEFREF entry contains a MODIR File index of -1, with the rest of the entry remaining the same.

Command Execution

The library files are maintained through the execution of :ALLOT, :COPY, :DELETE, and :SQUEEZE commands. The entries in the MODIR File, MODULE File, and DEFREF File are in the same sequential order. The ith entry in the MODIR File identifies the ith object module in the MODULE File, and corresponds to the ith entry in the DEFREF File. The ordering of these files is always preserved.

:ALLOT Library files are allocated in the same general manner as other files described previously, but with certain specific differences. When area SP or FP is specified, a check is made to determine if the file name is MODIR, MODULE, DEFREF, or EBCDIC. If MODULE is specified, RSIZE is required to be 30 words and FORMAT must be blocked. If MODIR, DEFREF or EBCDIC is specified, FORMAT must be unblocked. RSIZE can be any value for the unblocked files and is used solely for calculating the amount of space to allocate for the file. The record size for these three files is set to 0 when allocated. GSIZE on all library files is ignored, and is always set equal to RAD sector size by the RAD Editor.

:COPY The permanent RAD area specified on the :COPY command determines which library a module(s) is to be added to. For each object module added, the following procedure is followed:

1. An object module is read from the input device specified on the command. The module is added to the end of the MODULE File as it is being scanned for external definitions and references. The MODULE File record number for the MODIR File is obtained from RFT12 (current record no. of file). The MODIR File index is obtained from RFT5 (record length).
2. As DEFs and REFs are encountered, they are added as entries to the end of the EBCDIC File. The first DEF encountered is used as the MODULE File name. However, REFs are added to the EBCDIC File if they are not in duplicate.
3. The indices to the EBCDIC File entries are saved to create the DEF n and REF n words of the entry to the DEFREF File.
4. The addition of the object module to the library is completed by updating the "records per module" in the MODIR File entry; "entry size" in the DEFREF File entry; and writing the MODULE, DEFREF, and EBCDIC Files to the RAD.

:DELETE The permanent RAD area on the :DELETE command is used to determine which area contains the library object module to be deleted. The MODIR File entry containing the same module name as that appearing on the command is zeroed out. The corresponding DEFREF File entry is located and the halfword containing the MODIR File index is set to -1. No other changes are made to the EBCDIC and MODULE Files as a result of the :DELETE command.

All unused space resulting from a module deletion is recovered when a :SQUEEZE command is executed.

:SQUEEZE The permanent RAD area on the :SQUEEZE command is used to determine the library to be squeezed. Permanent RAD areas containing libraries are squeezed in the same way as other areas with the following exception: after the permanent file directories are compacted and files are moved to regain the unused space, a search is made of the MODIR File. All existing modules are copied from the MODULE File to the Temporary File X1. Using X1 as the source of input, the library files MODIR, EBCDIC, and DEFREF are regenerated.

Bad Track Handling

Bad tracks within permanent file areas on a RAD are removed from use by making special entries to the appropriate file directory. All bad tracks can be handled in this manner except those that contain a sector of the file directory. These cannot be removed from use as it would make accessing of certain files impossible. All bad tracks on a disk pack are removed from use by flawing the bad track(s) and using alternate tracks if available. Otherwise, they are handled the same as for a RAD.

Command Execution

Bad tracks are handled through execution of :BDTRACK and :GDTRACK commands. The :BDTRACK command removes the track from use by allocating or flawing the track. The :GDTRACK command returns the track for use by deleting the entry made by :BDTRACK or removing the flaw marks.

:BDTRACK The permanent file area that encompasses the bad track is determined by the RAD or disk pack (DP) and bad track specified on the command. A check is made to determine if a sector of directory falls within the bad track. If it does, the bad track is not eliminated from use. A search of the file directory is made to determine if the bad track is allocated. If it is, the entry(s) that allocates the track is eliminated and replaced by a bad track entry. If it is not allocated, a bad track entry is added to the end of the file directory. A bad track entry consists of the "file name" being set to -1, and the BOT and EOT being set to the starting and ending sector of the bad track. The appearance of files in the same order as the entries in the file directory is maintained.

If the bad track is on a disk pack, a search is made for the available alternate track (alternate is all 1's). When found, the cylinder, head, and sector addresses of the alternate track headers are inserted. The alternate cylinder and head address fields are updated to contain the flawed track address. The header of the specified bad track is updated to contain the flawed track address. The header of the specified bad track is updated by inserting the alternate cylinder and head addresses, and setting the flaw mark bits to 1's. If there are no alternate tracks available, bad tracks are handled the same as on a RAD; that is, by putting an entry in the file directory.

:GDTRACK The permanent file area that encompasses the good track is determined by the RAD or disk pack (DP) and bad track specified on the command. A search of the file directory is made for the entry that allocates the track specified on the command. The entry is deleted (file name set = 0), making the track available for allocating.

If a good track is on a disk pack, the flaw bits in the headers are checked to see if they are set; if so, the headers are altered by clearing the flaw bits and setting the alternate track field to all 0's, and the headers in the alternate track are altered by setting the alternate track field to all 1's. If the flaw bits are not set, the good track specified is handled the same as for a good track on a RAD (by deleting the appropriate file directory entry).

Use of IOEX for Disk Pack

The flawing of bad tracks is performed with a call to IOEX. The RBM assembly option #SYSPROC must be included to correctly perform this operation. If #SYSPROC is not included in RBM, the flawing is not performed and the disk pack is treated exactly as a RAD (i. e., a bad track entry is placed in the file directory).

Utility Functions

The following utility functions are performed by the RAD Editor:

- Maps permanent RAD areas.
- Clears permanent RAD areas.
- Enters data onto permanent RAD files.
- Appends records to the end of an existing permanent RAD file.
- Copies permanent RAD files.
- Copies library object modules.
- Dumps the contents of RAD files or entire RAD areas.
- Saves the contents of RAD areas in self-reloadable form.
- Restores RAD areas previously saved.

:MAP The permanent RAD area(s) to be mapped is indicated on the :MAP Command, with the map information being output to the device assigned to the M:LO DCB. Whenever the sequence yyndd is encountered, all following area mnemonics are processed if the area resides on the specified device. In addition, if yy = DP, a list of flawed tracks and alternates are output as follows:

```
FLAWED TRACKS AND ALTERNATES
FFFF          AAAA
```

where

FFFF is the flawed track number (decimal).

AAAA is the allocated alternate track number (decimal).

Each map consists of up to three sections: one section when RAD areas CK, XA, or BT are mapped; two sections if RAD areas without libraries are mapped; three sections if RAD areas containing libraries are mapped. The three sections of the map are as follows:

1. Information from the Master Directory identifying the permanent RAD area, starting and ending RAD addresses, write protection, and device number of the RAD from the Device Control Tables.
2. Information obtained from the permanent file directories concerning each file in the area; its name, format, granule size, record size, file size, beginning of file, and ending of file.

3. Information about object modules in the library files; consisting of the permanent RAD areas, module name, and the definitions and references in the module.

Section 1 of the map has the format

```
RAD AREA ZZ RAD yyndd BOA bbbbbb EOA eeeee WP w
```

where

ZZ identifies the permanent RAD area.
 yyndd is the RAD that contains the permanent RAD area.
 bbbbbb is the absolute RAD address of the first sector of the area in decimal.
 eeeee is the absolute RAD address of the last sector of the area in decimal.
 w is the write protection for the file.
 F is write-permitted by foreground only unless SY key-in.
 B is write-permitted by background only unless SY key-in.
 M is write-permitted by the Monitor only.
 N is write-permitted only if SY key-in.
 X is write-permitted by IOEX only.

Section 2 of the map has the format

```
NAME FORMAT GSIZE RSIZE FSIZE BOF EOF
nnnnnnnn f g r l s t
```

where

nnnnnnnn is the name of a file in the permanent RAD area.
 f is the file format:
 U specifies unblocked
 B specifies blocked
 C specifies compressed
 g is the granule size in bytes in decimal.
 r is the record size in bytes in decimal.
 l is the number of records in file in decimal.
 s is the relative RAD address of the first sector defined for the file in decimal.
 t is the relative RAD address of the last sector defined for the file in decimal.

Section 3 of the map has the format

```
MAP OF LIBRARY IN AA AREA
MODULE NAME LOCATION DEFS REFS
mmmmmmmm IIII dddddddd dddddddd rrrrrrr rrrrrrr
```

where

AA is the permanent RAD area that contains the library.
 mmmmmmm is the object module name.
 IIII is the relative sector address of the first sector of the object module.
 dddddddd is the name of an external definition.
 rrrrrrr is the name of an external reference.

The mapping of an area is performed as follows:

1. Information is obtained from the Master Directory for Section 1 of the map and output to the LO device. If an area is not allocated, the mapping of that area is ignored.
2. Information is then obtained from the permanent file directory for Section 2 and output to the LO device. If an area other than CK, XA, or BT does not contain files, a message will be output to that effect. When a bad track entry is encountered, "BADTRACK" is printed as the name of the file.
3. If the permanent RAD area is either FP or SP and contains libraries, information is obtained for Section 3. The MODULE NAME is obtained from the MODULE File, the module record number from the MODIR File, and the definitions and references are obtained by scanning the DEFREF File for the indexes to the EBCDIC located in EBCDIC File.

:CLEAR The permanent RAD area on the :CLEAR command is used to determine the area to be cleared (set to zero). The area is cleared using the direct access method. The granule size is set equal to the amount of unused background space available, which is zeroed out and written to the RAD.

:COPY The parameters on the :COPY command are used to set up the F:SI and F:SO DCBs. Files are copied sequentially. When an !EOD, :EOD, or EOT is encountered, the COPY is terminated. When an object module is copied to an output device, the COPY is terminated when the module end load item is encountered.

:DUMP The permanent RAD area or file to be dumped is indicated on the :DUMP command. The information is dumped to the device assigned to the M:LO DCB. The file dump has the format

```
DUMP OF FILE nnnnnnnn IN AREA AA
RECORD rrrr
WD 0000 dddddddd dddddddd ... dddddddd
WD 0008      .      .      .
WD 0016      .      .      .
```

where

nnnnnnnn is the name of the file.

AA identifies the permanent RAD area (area BT inclusive).

rrrr is the relative record number and begins with 1.

ddddddd is a data word in hexadecimal.

The area dump has the format

```
DUMP OF AREA ZZ
SECTOR ssss
WD0000 dddddddd dddddddd ... dddddddd
WD0008      .      .      .
WD0016      .      .      .
```

where

ZZ identifies the RAD area.

ssss is the relative sector number, and begins with 0.

ddddddd is a data word in hexadecimal.

The dumping of an area or file is performed as follows:

1. The directive is scanned to determine whether an area or file is to be dumped. If a value for SREC is not specified, 0 is assumed. If a value for EREC is not specified, the last record of the file or area is assumed.
2. The record(s) to be dumped is accessed sequentially. Within a record, if a word is duplicated more than sixteen times in order, it is output only once in the message

'WDxxx THRU xxx CONTAIN xxxxxxxx'

If records are duplicated, the message

'RECORDxxx THRU xxx CONTAIN xxxxxxxx'

is output.

If sectors are duplicated, the message

'SECTOR xxx THRU xxx CONTAIN xxxxxxxx'

is output.

3. The dump is terminated when the specified number of records have been dumped or when a complete file or area has been dumped.

:SAVE The area(s) to be saved is specified on the :SAVE command. The data is dumped to the device assigned to the M:BO DCB, and consists of the following:

1. A small 88-byte bootstrap that loads the large bootstrap when booted from the console.
2. A large bootstrap that restores the RAD or disk pack from magnetic or paper tape.
3. An 88-byte RBM bootstrap used for booting the RAD or disk pack.
4. Records containing data to be restored.

Each record to be restored is preceded by a five-word header with the format

| | | | | | | |
|----------------------------|-----------------------------|-------------|-------|--------|-------------|----|
| No. words per sector | L R A | L R T | 0 — 0 | D P | Area ident. | |
| No. sectors in record | Device number | | | | | |
| Area FWA | | | | | | |
| No. sectors per track | No. sectors (zero) to write | | | | | |
| CKSM (2's complement form) | | | | | | |
| 0 | 14 | 15 | 16 | 17 | 23 24 | 31 |

where

No. words per sector is the size of the sector.

LRA is a flag to indicate the last record of an area if LRA = 1, last record.

LRT is a flag to indicate the last record of the tape if LRT = 1, last record.

DP indicates that the device is a disk pack if DP = 1.

Area ident. is the area to which the record belongs.

No. sectors in record is the size of record (in sectors).

Device number is the physical device number of the RAD or disk pack.

Area FWA is the absolute address where the data records should begin being restored.

No. sectors per track/No. sectors (zero) to write is the number of sectors containing all zeros preceding nonzero data in the record.

CKSM is the checksum of this record in the 2's complement form.

The saving of an area for subsequent restoration is performed as follows:

1. A small and large bootstrap are written with their checksums.
2. A header for the RBM RAD bootstrap is written. The FWA and device number for the header is obtained from K:RDBOOT.
3. The image of the RBM RAD bootstrap is read from the file RADBOOT in the SP area, and written.
4. Data records are written with each record being preceded by a header and followed by a checksum. Leading and trailing zeros of a record are not written. Size of the data records depends upon the amount of available background space used as a buffer.
5. After all the specified areas are saved, the tape is verified by using the checksum word of each header and data record.

:RESTORE The area(s) to be restored is specified on the :RESTORE command. The data is read using the device assigned to the M:BI DCB. The small bootstrap, large bootstrap, and RBM RAD bootstrap are skipped. Data records are read and restored using the headers that precede them with all leading and trailing zeros of a record also being restored. Restoration has to be made to the same type of RAD as that from which the records were saved.

The overall flow of the RAD Editor is illustrated in Figures 68 through 72.

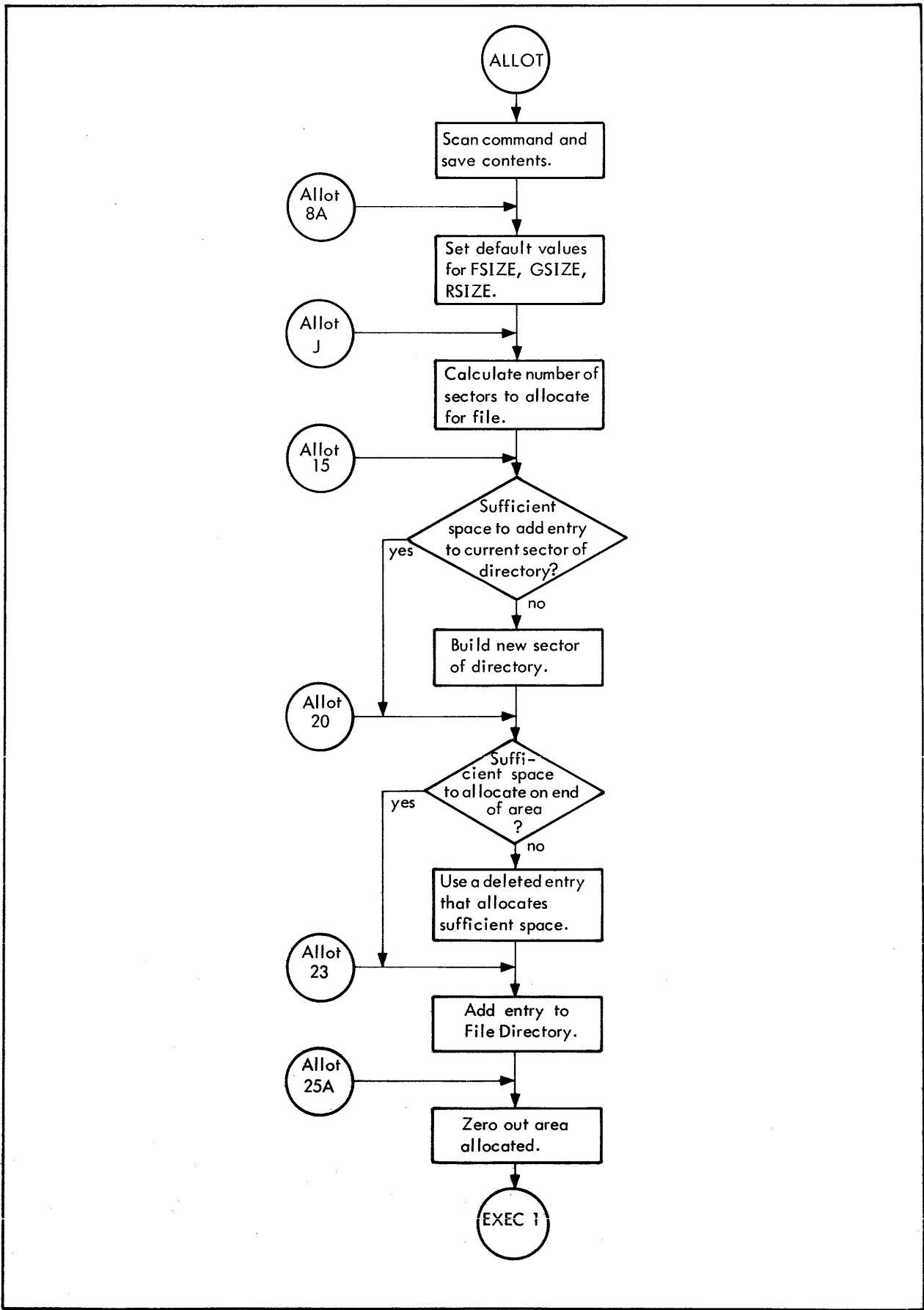


Figure 68. RAD Editor Flow, ALLOT

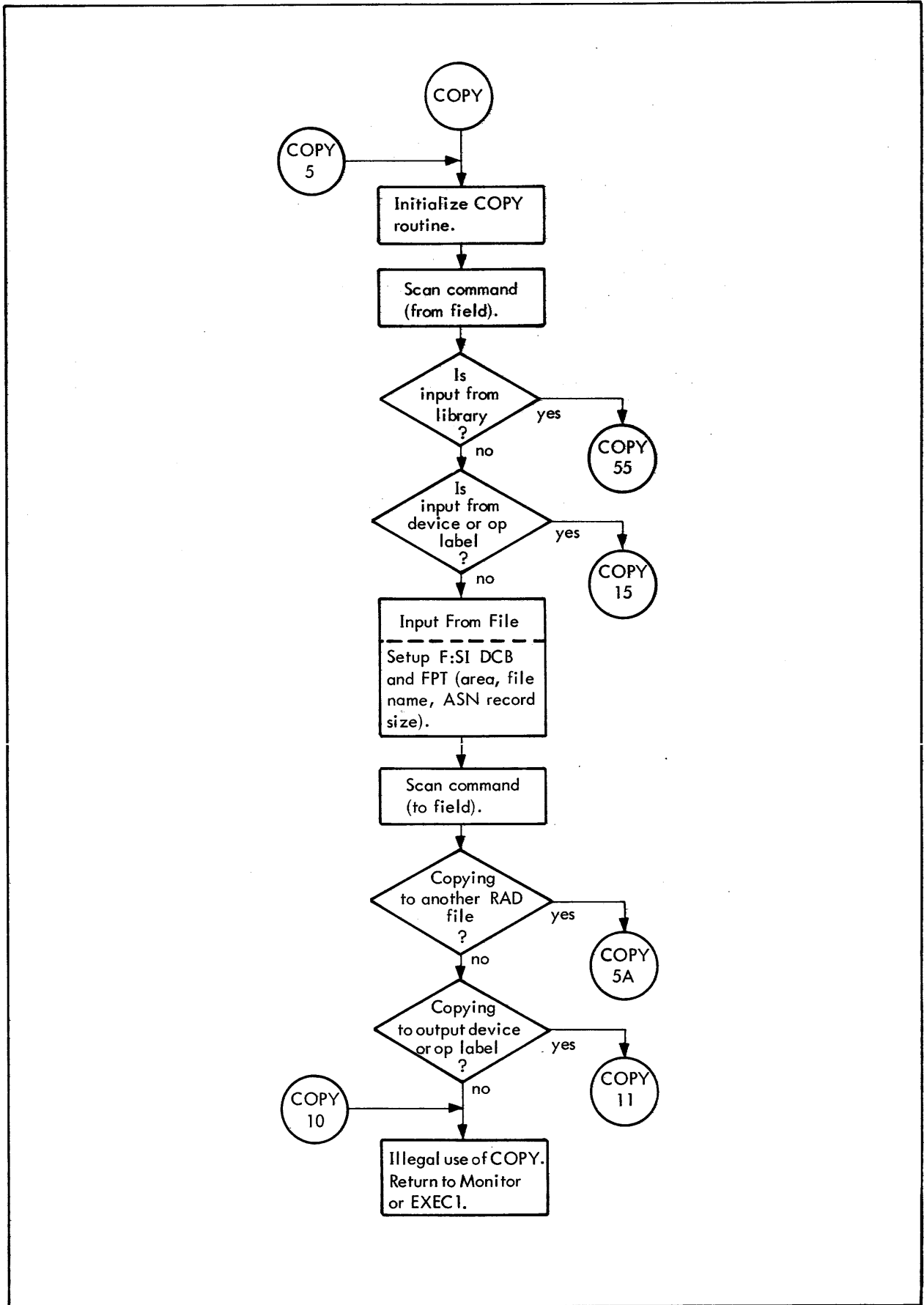


Figure 69. RAD Editor Flow, COPY

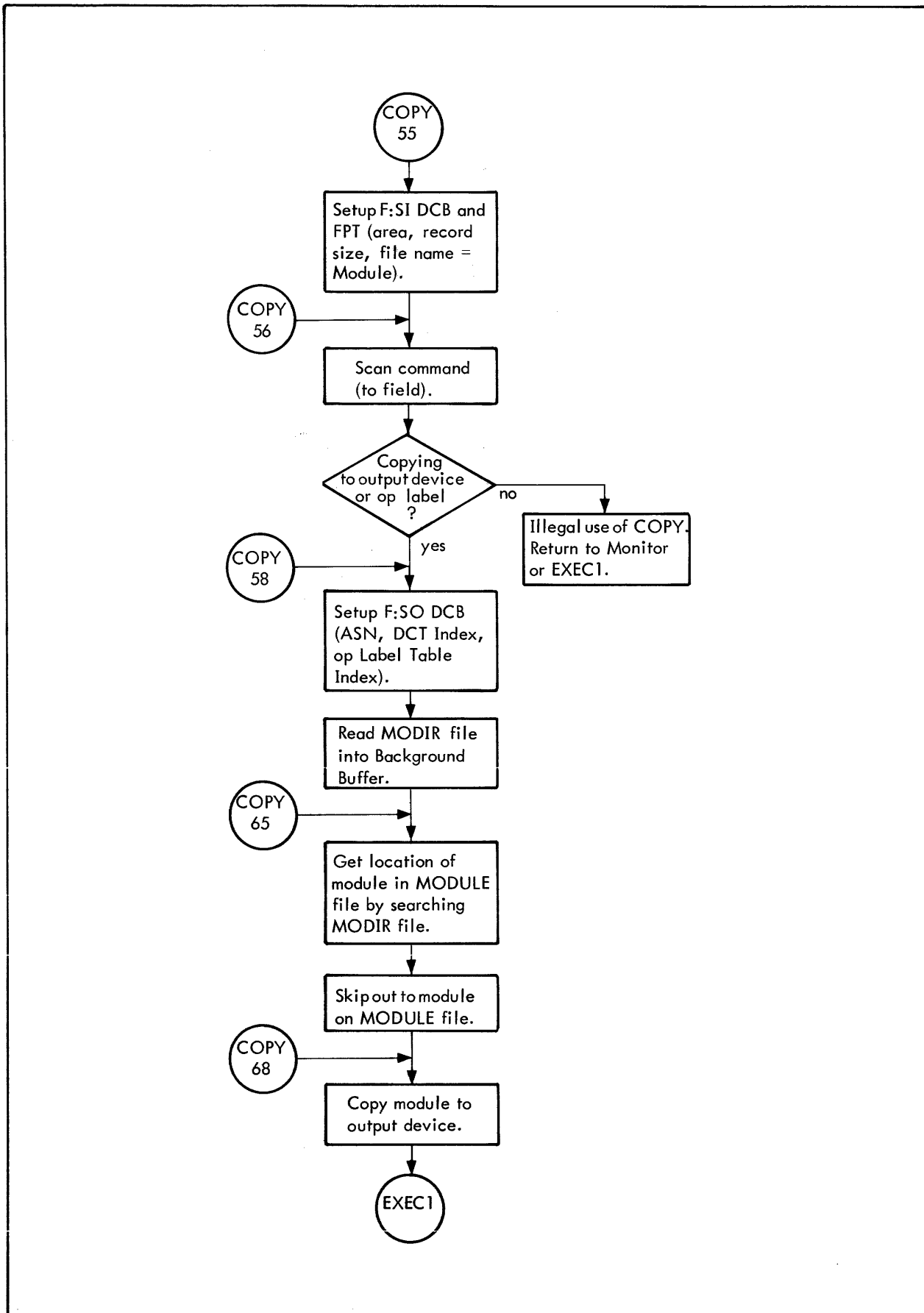


Figure 69. RAD Editor Flow, COPY (cont.)

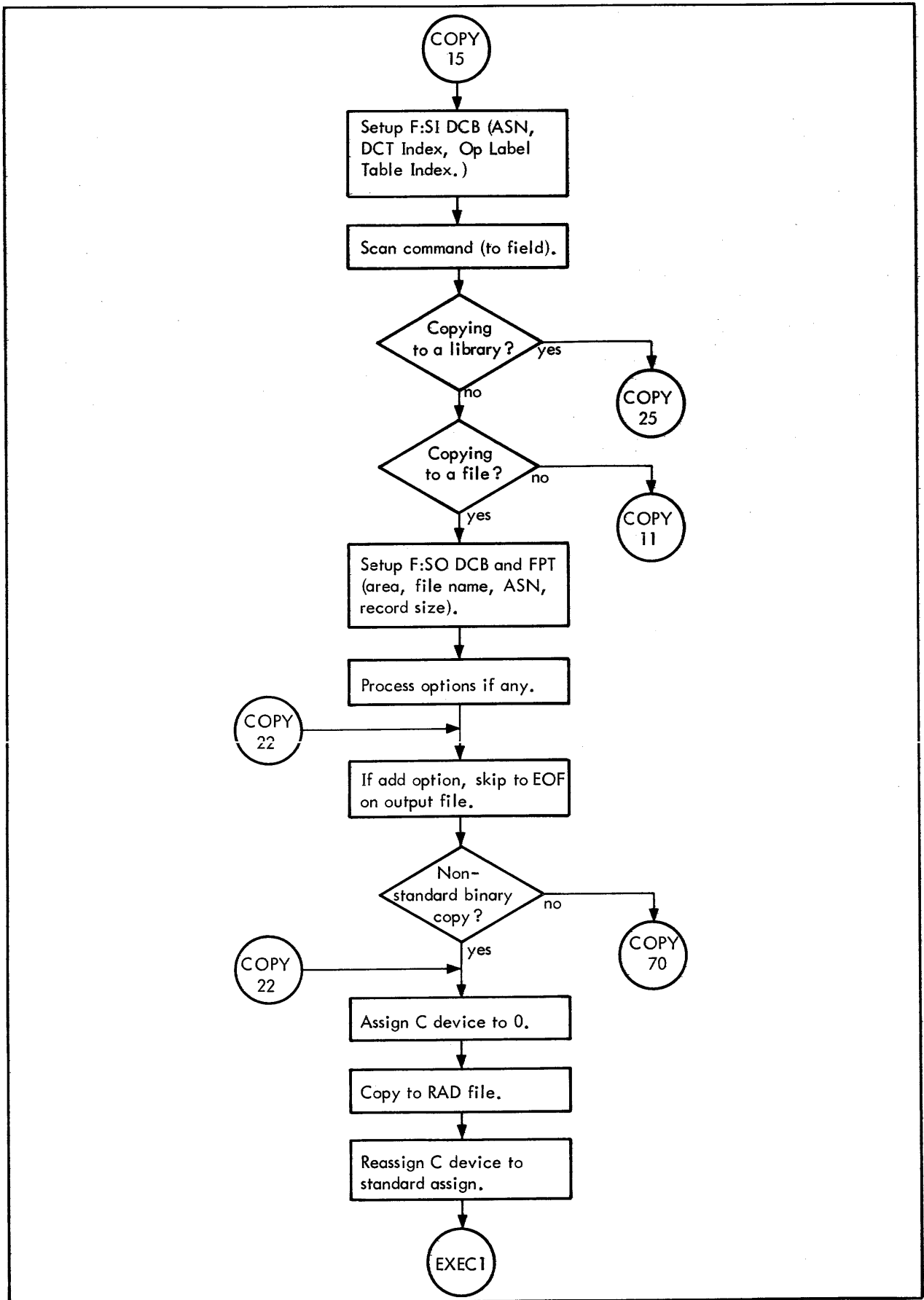


Figure 69. RAD Editor Flow, COPY (cont.)

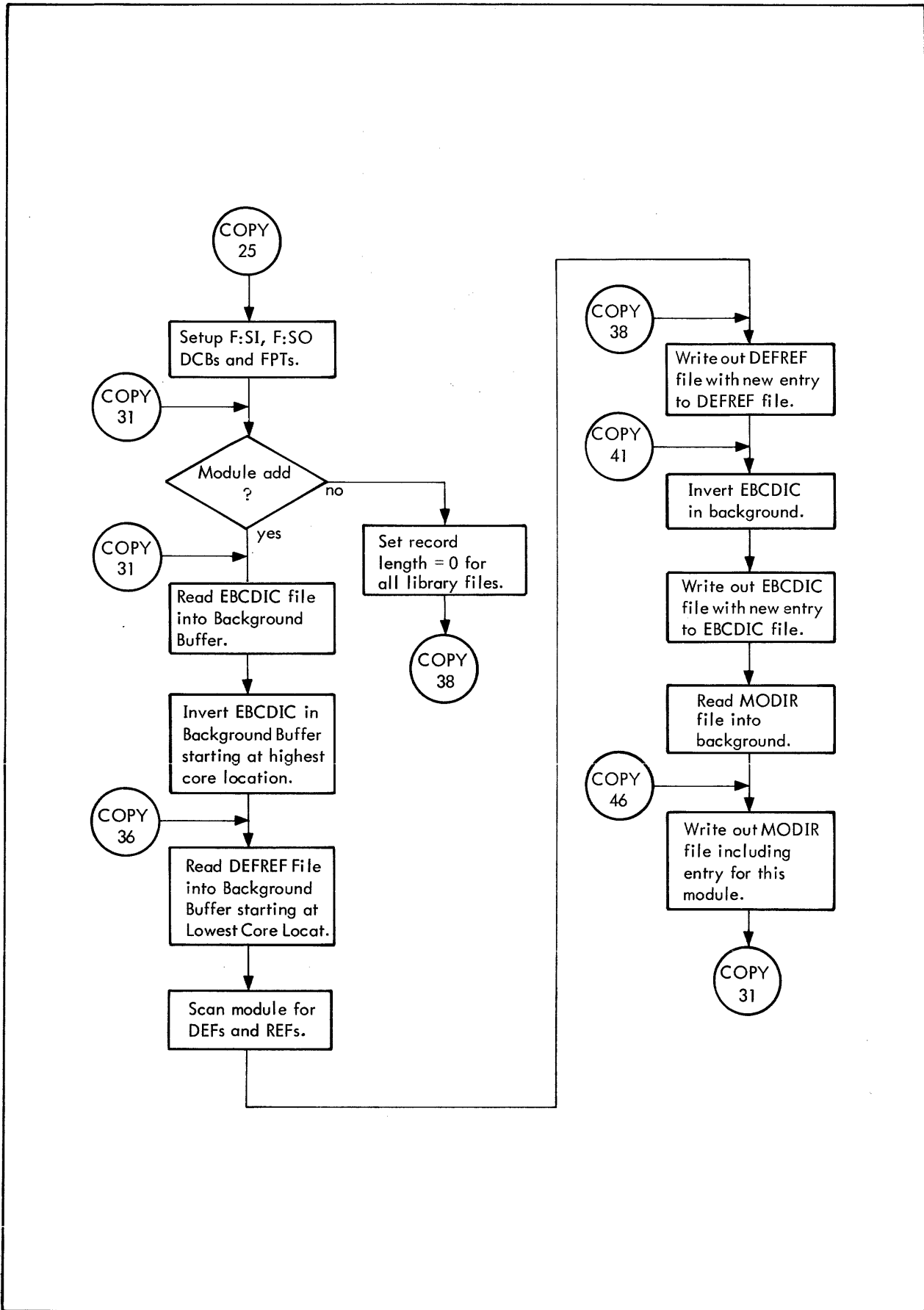


Figure 69. RAD Editor Flow, COPY (cont.)

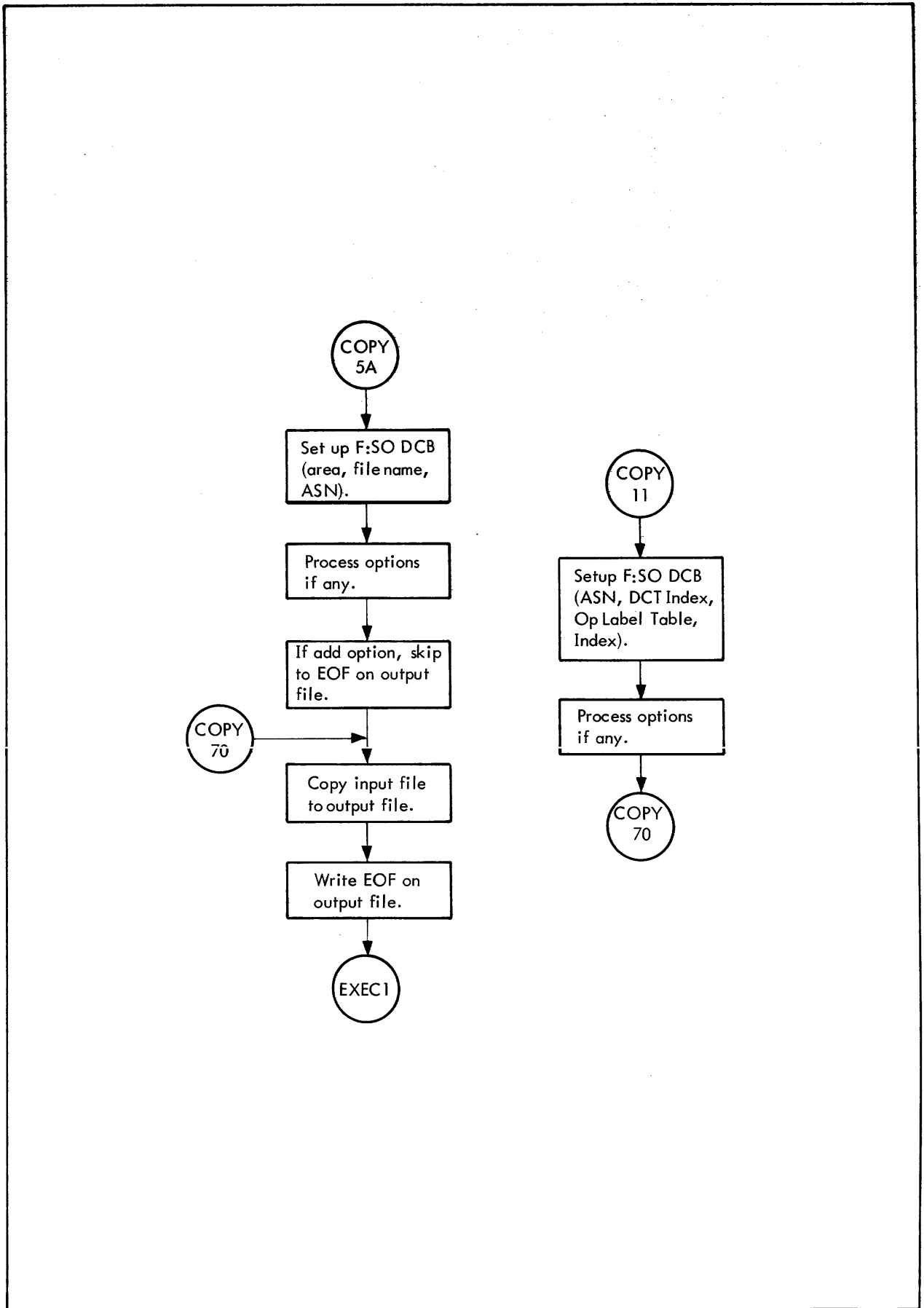


Figure 69. RAD Editor Flow, COPY (cont.)

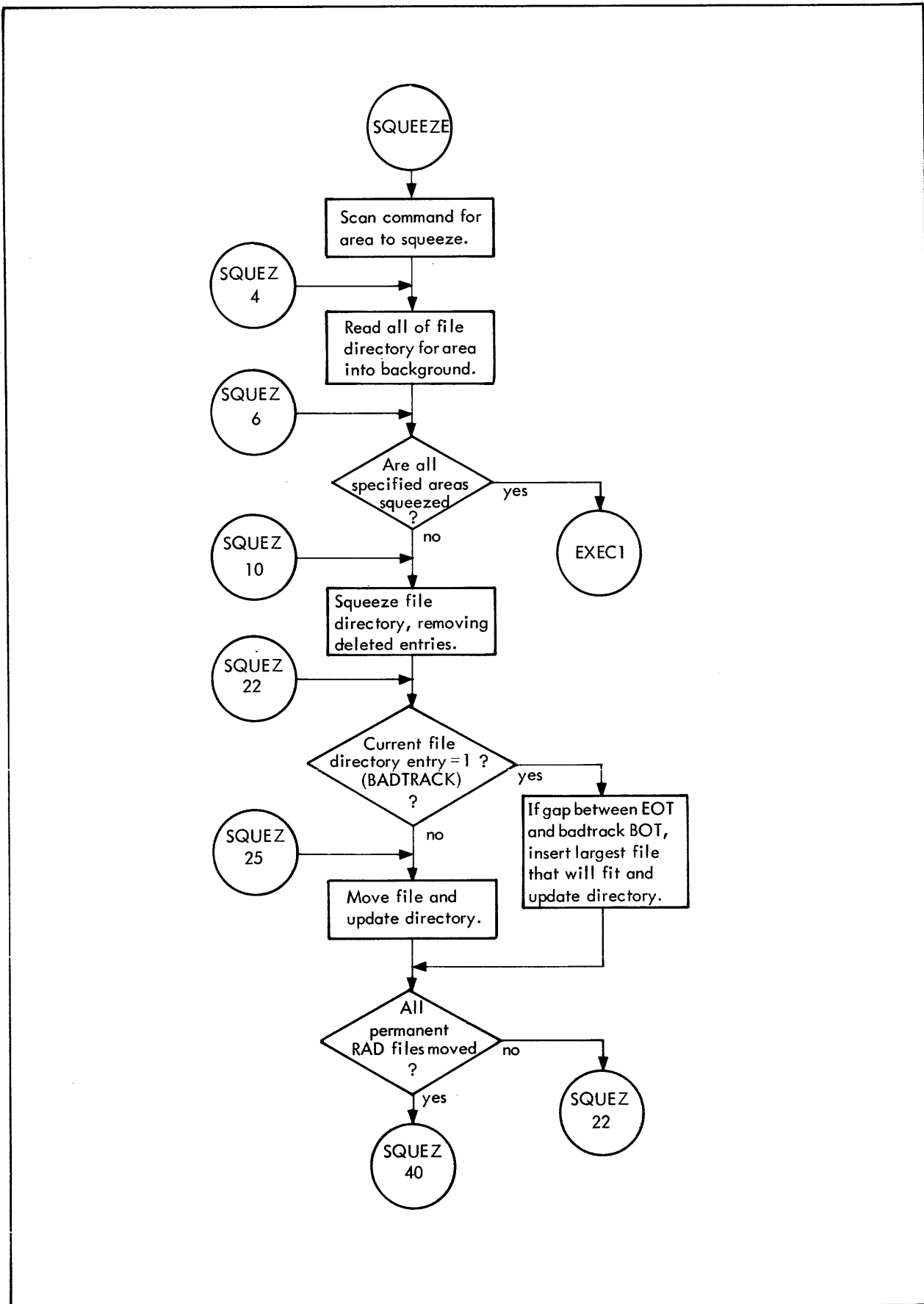


Figure 70. RAD Editor Flow, SQUEEZE

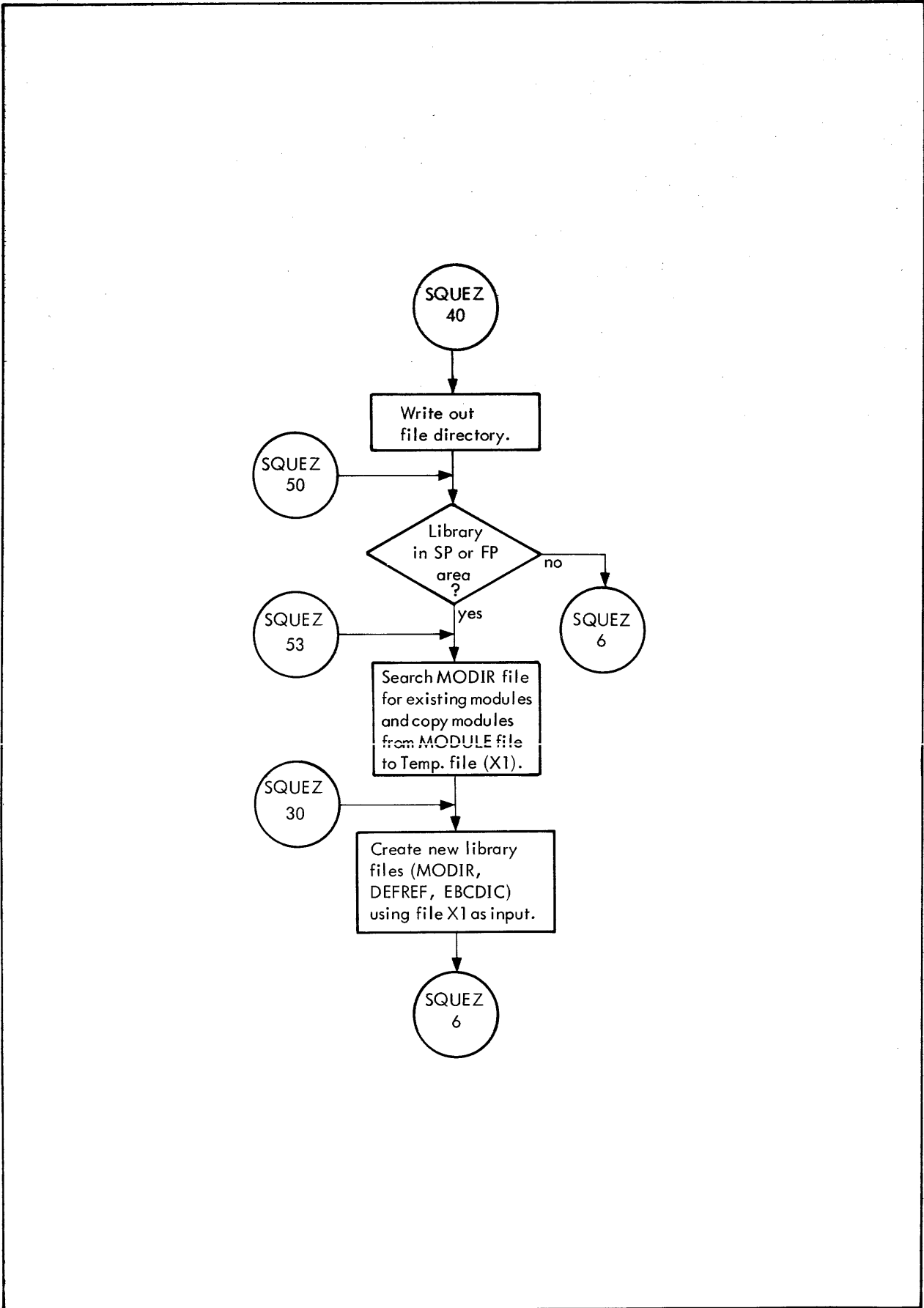


Figure 70. RAD Editor Flow, SQUEEZE (cont.)

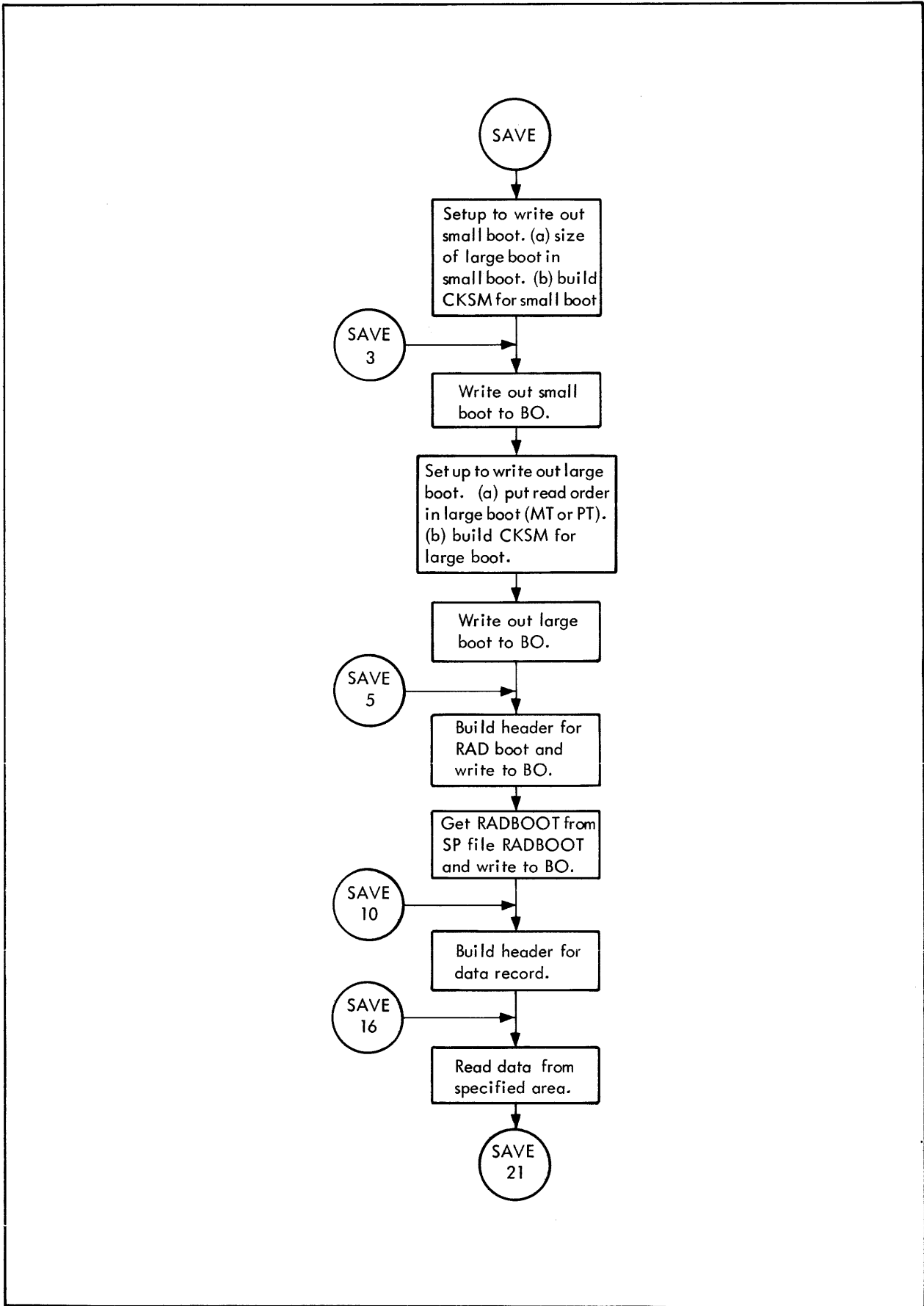


Figure 71. RAD Editor Flow, SAVE

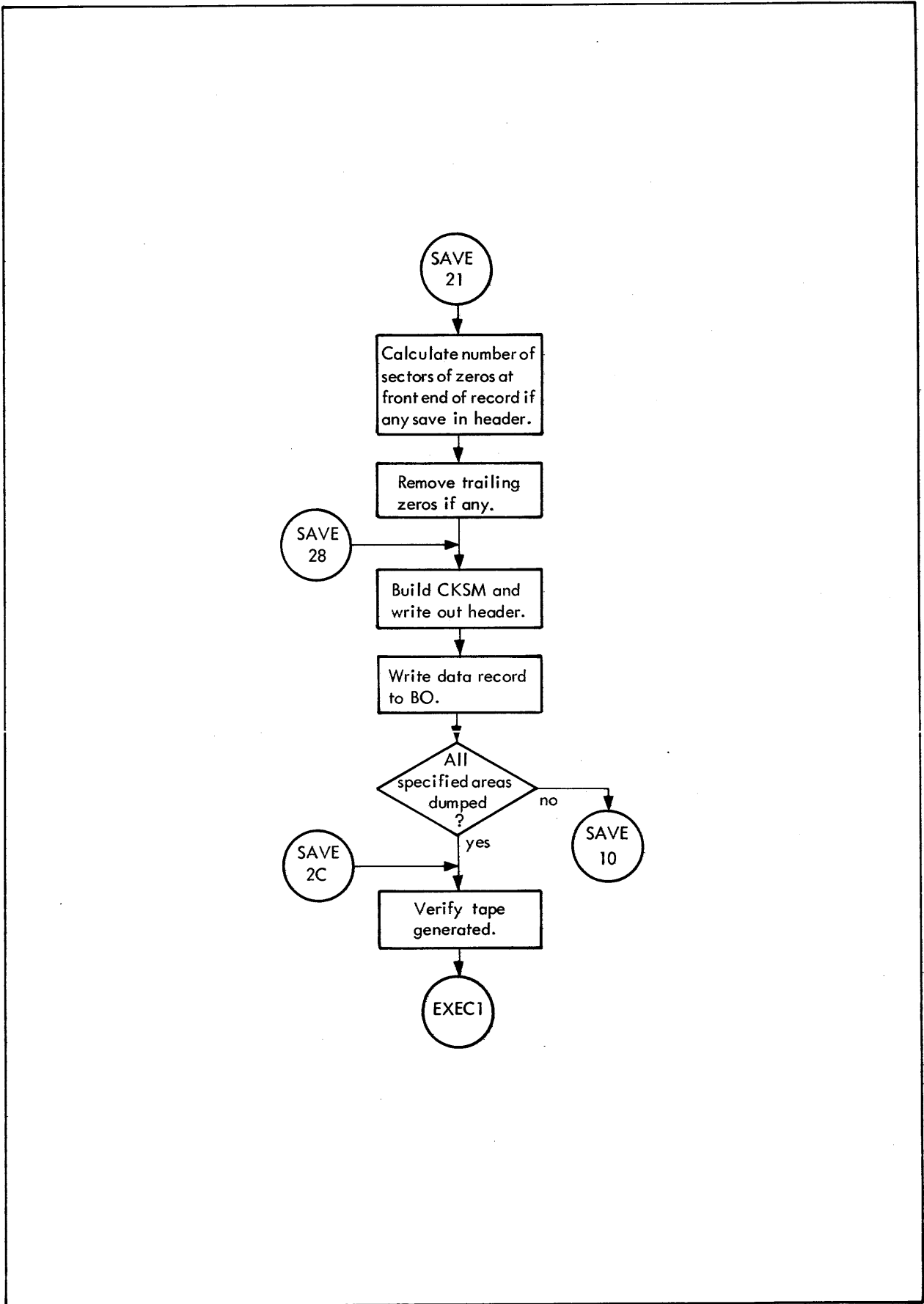


Figure 71. RAD Editor Flow, SAVE (cont.)

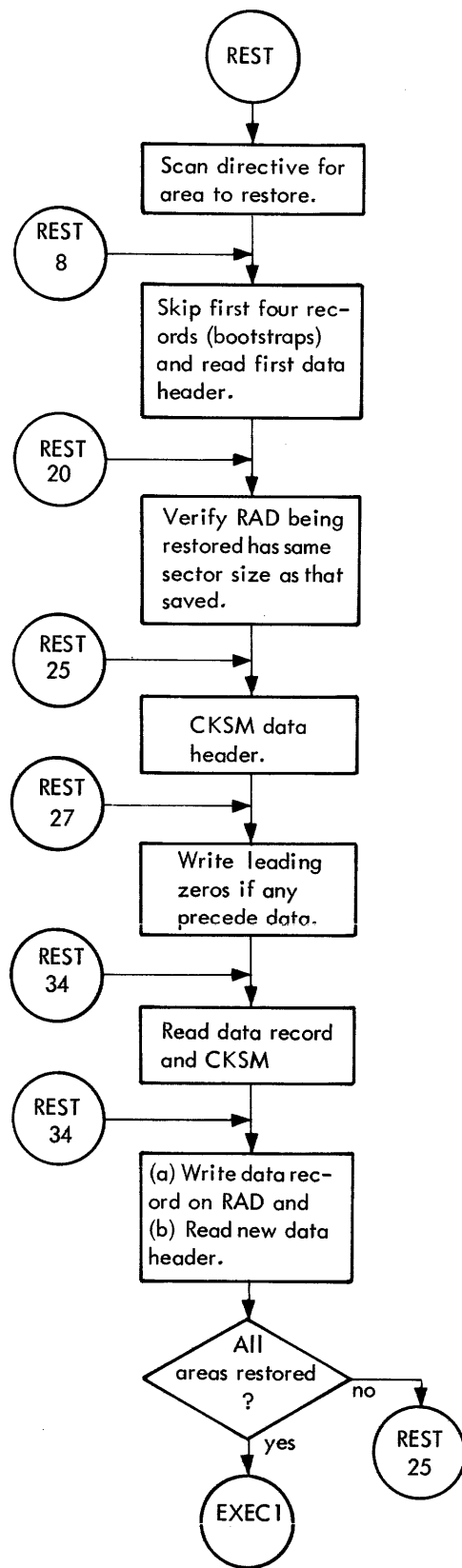


Figure 72. RAD Editor Flow, RESTORE

11. SYSTEM GENERATION

Overview

The System Generation program is assembled in absolute, using the ASECT directive, and is ORG'd (originated) at two locations:

1. The first ORG at location X'140' allocates and defines the system flags and pointers. It is the first location that cannot be used for an external interrupt. The system flags and pointers are a group of cells that provide communication between SYSGEN, all portions of the Monitor, and the system processors and service routines. Since these cells are in fixed, predetermined locations, they are defined via the EQU directive in all programs that reference them. Note that these cells must not be changed, deleted, or altered in any way in the SYSGEN listing unless the EQU directives are also changed in all programs that reference the cells. The system flags and pointers are followed by a skeleton of the Master Dictionary. The Master Dictionary is not necessarily fixed at its assembled location since it may be moved to the unused interrupt cells if sufficient space exists.
2. The next ORG at location X'28E0' fixes the start of the SYSGEN program. SYSGEN is ORG'd such that the program will occupy the highest address portion in a 16K memory. This provides the SYSGEN Loader with the maximum amount of room to load the Monitor and its overlays in the lower address portion of memory. If a user adds a significant amount of code to the Monitor, this ORG may have to be moved to a higher location to prevent the Monitor from overflowing SYSGEN during the load.

The System Generation program is divided into two sections designated as SYSGEN and SYSLOAD. SYSGEN processes all the SYSGEN control commands and allocates and initializes all the Monitor tables from the information on the control commands. It also builds a symbol table for SYSLOAD that contains the name and absolute address of all the Monitor tables. Optionally, SYSGEN will output on a rebootable deck containing the Monitor tables and SYSLOAD on cards, paper tape, or magnetic tape. The SYSGEN phase can be overwritten during the loading of the Monitor, and terminates by exiting to SYSLOAD.

SYSLOAD loads the Monitor, all optional resident routines, the RBM overlays, the Job Control Processor, and then writes these in to the RBM file in the SP area. A map containing the RBM table allocation and RAD allocation is output upon request. SYSLOAD terminates by reading in the RAD Bootstrap and exiting to it, simulating a booting of the system from the RAD.

Figure 73 illustrates the core layout of SYSGEN and SYSLOAD after the absolute object module is loaded by the Stand-Alone SYSGEN Loader.

| | |
|--|---------|
| Unchanged | X'140' |
| System Flags and Pointers | X'208' |
| Skeleton of Master Dictionary | X'236' |
| Unchanged | X'1800' |
| Stand-Alone SYSGEN Loader | X'1C00' |
| Unchanged | X'28E0' |
| SYSGEN Processing Routines | |
| Subroutines Unique to SYSGEN | X'3220' |
| SYSLOAD | |
| Subroutines Used by SYSGEN and SYSLOAD | X'4000' |

Figure 73. SYSGEN and SYSLOAD Layout Before Execution

Figure 74 depicts a typical core layout after SYSGEN and SYSLOAD have executed.

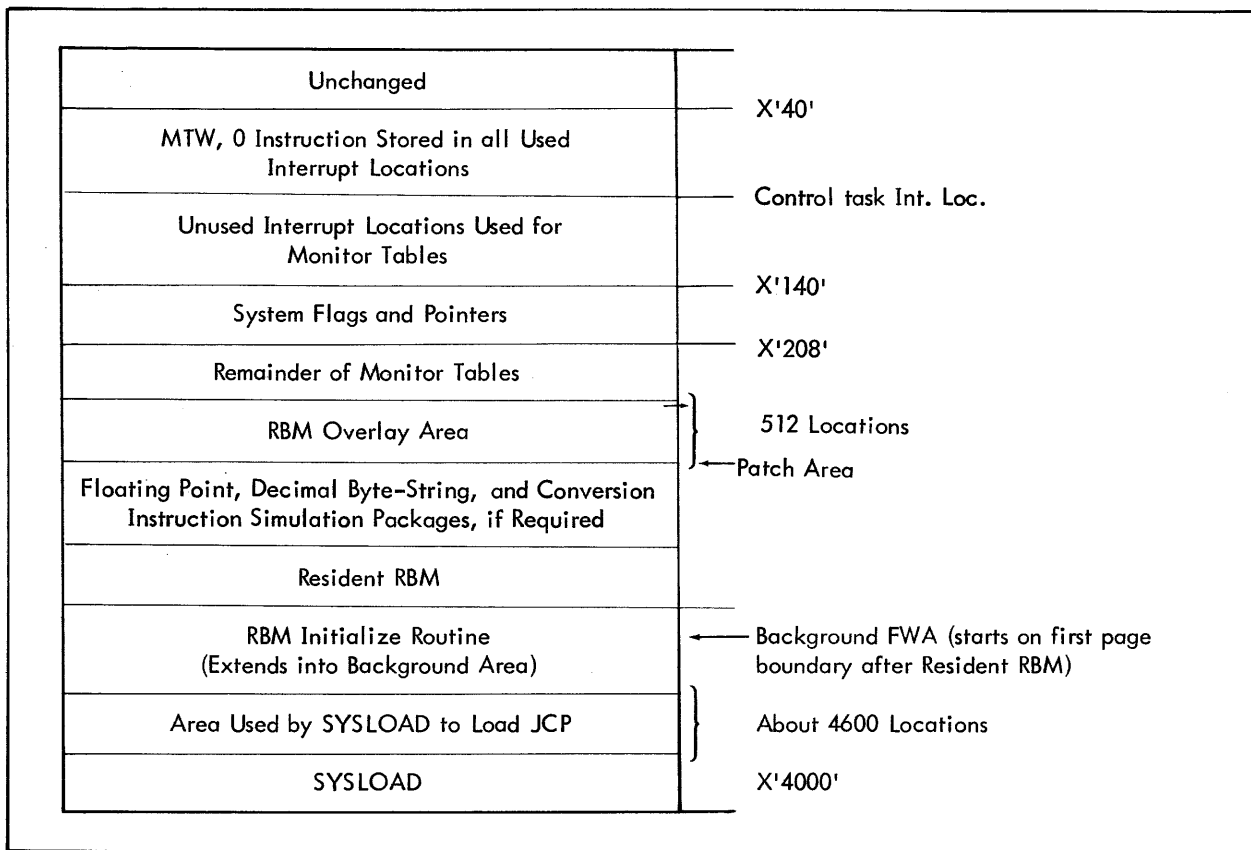


Figure 74. SYSGEN and SYSLOAD Layout After Execution

SYSGEN/SYSLOAD Flow

The flowcharts in Figure 75 depict the overall flow of SYSGEN and SYSLOAD. The labels used correspond to the labels in the program listing.

Loading Simulation Routines, RBM, and RBM Overlays

The S region of the SYSLOAD listing contains a loader that loads the instruction simulation packages, RBM, the RBM overlays, and the Job Control Processor (JCP). Each object module loaded must have one, and only one, DEF directive that identifies the object module to the loader.[†] The DEFs listed in Table 9 are recognized by the Loader.

Any DEF encountered that is not included in Table 1 results in an alarm

ILL. DEF.

Any object module loaded that is devoid of DEFs result in an alarm

OBJ. MOD. NOT RECOG.

[†]Except RBM object module.

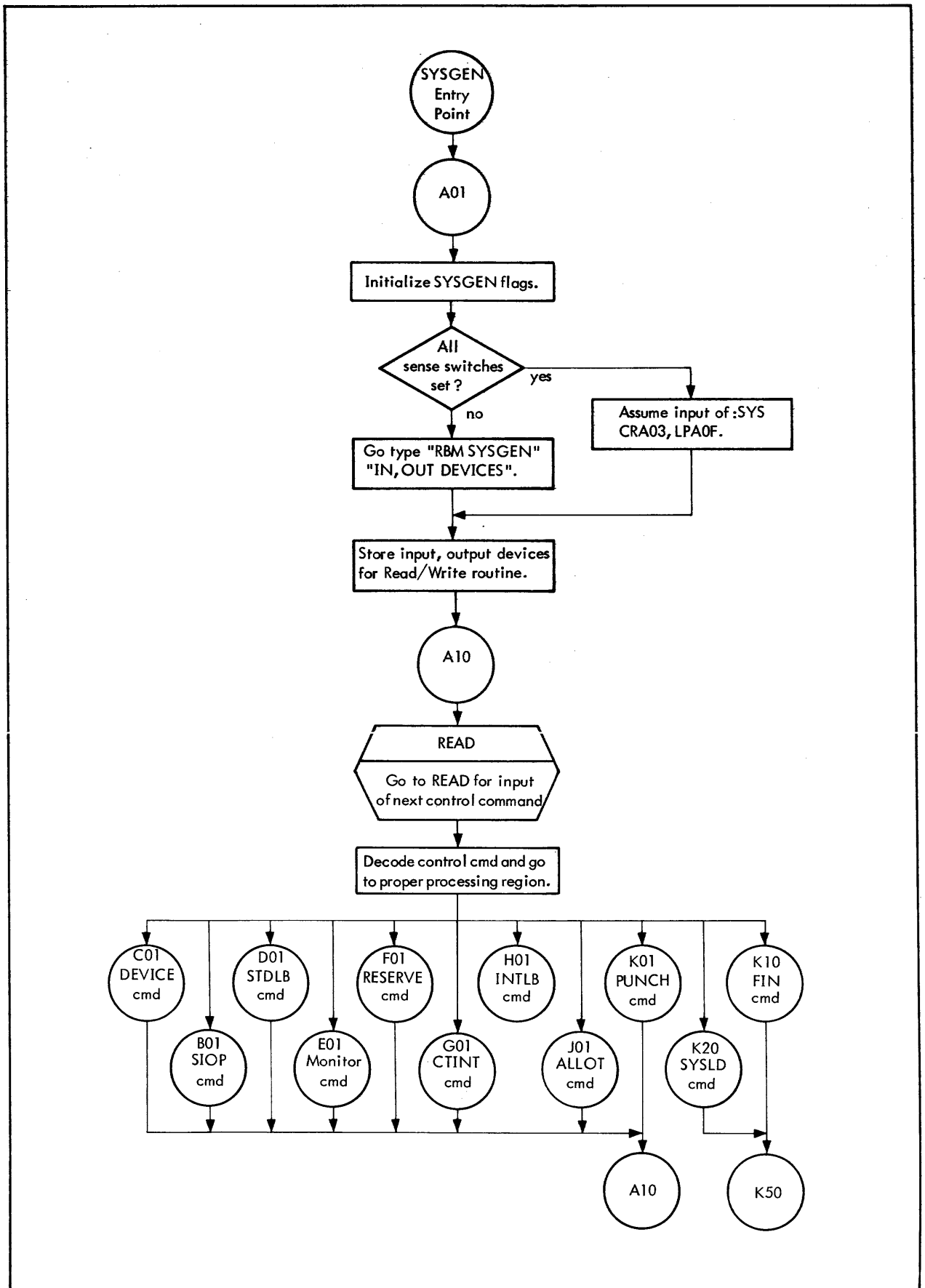


Figure 75. SYSGEN/SYSLOAD Flow

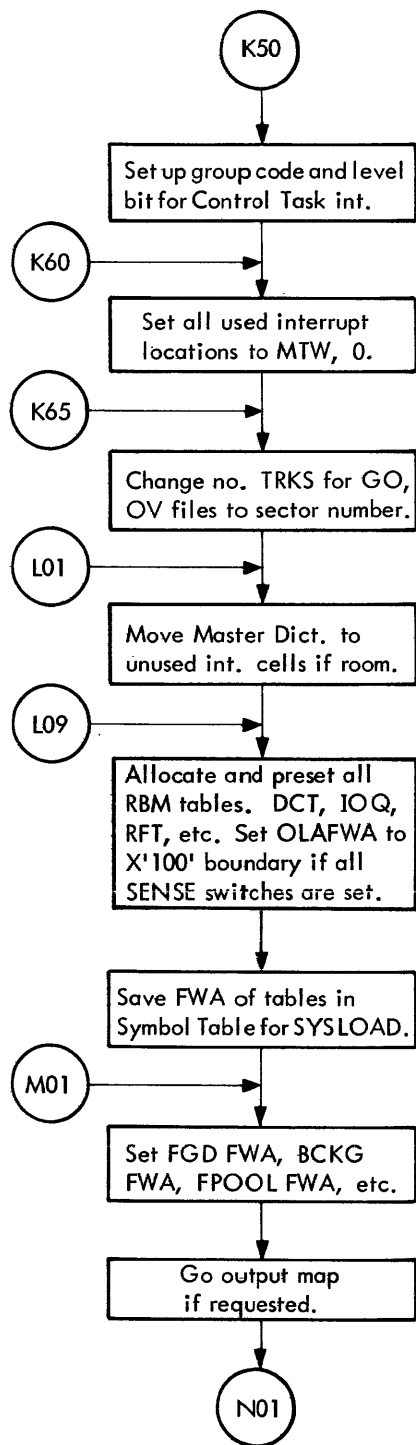


Figure 75. SYSGEN/SYSLOAD Flow (cont.)

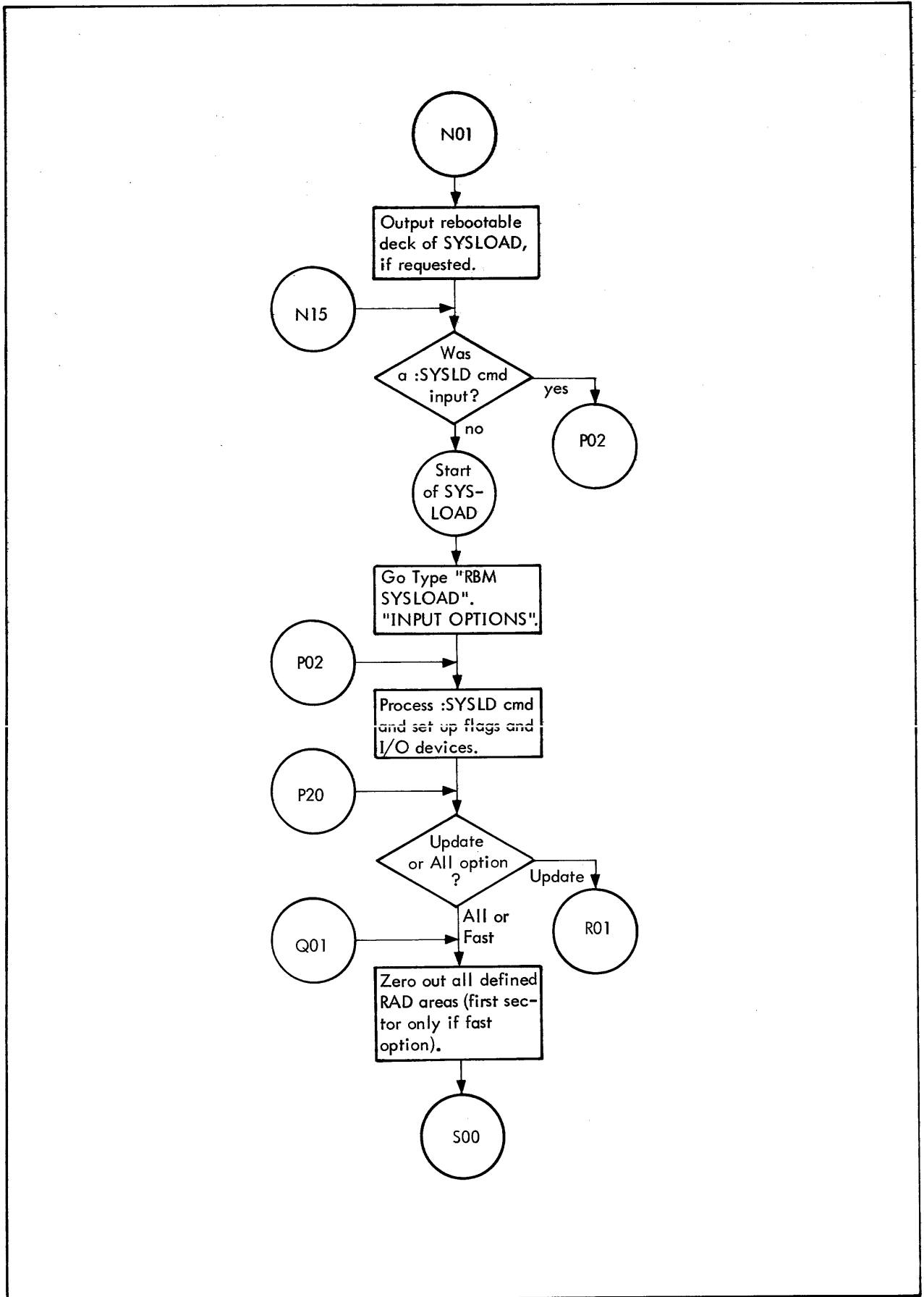


Figure 75. SYSGEN/SYSLOAD Flow (cont.)

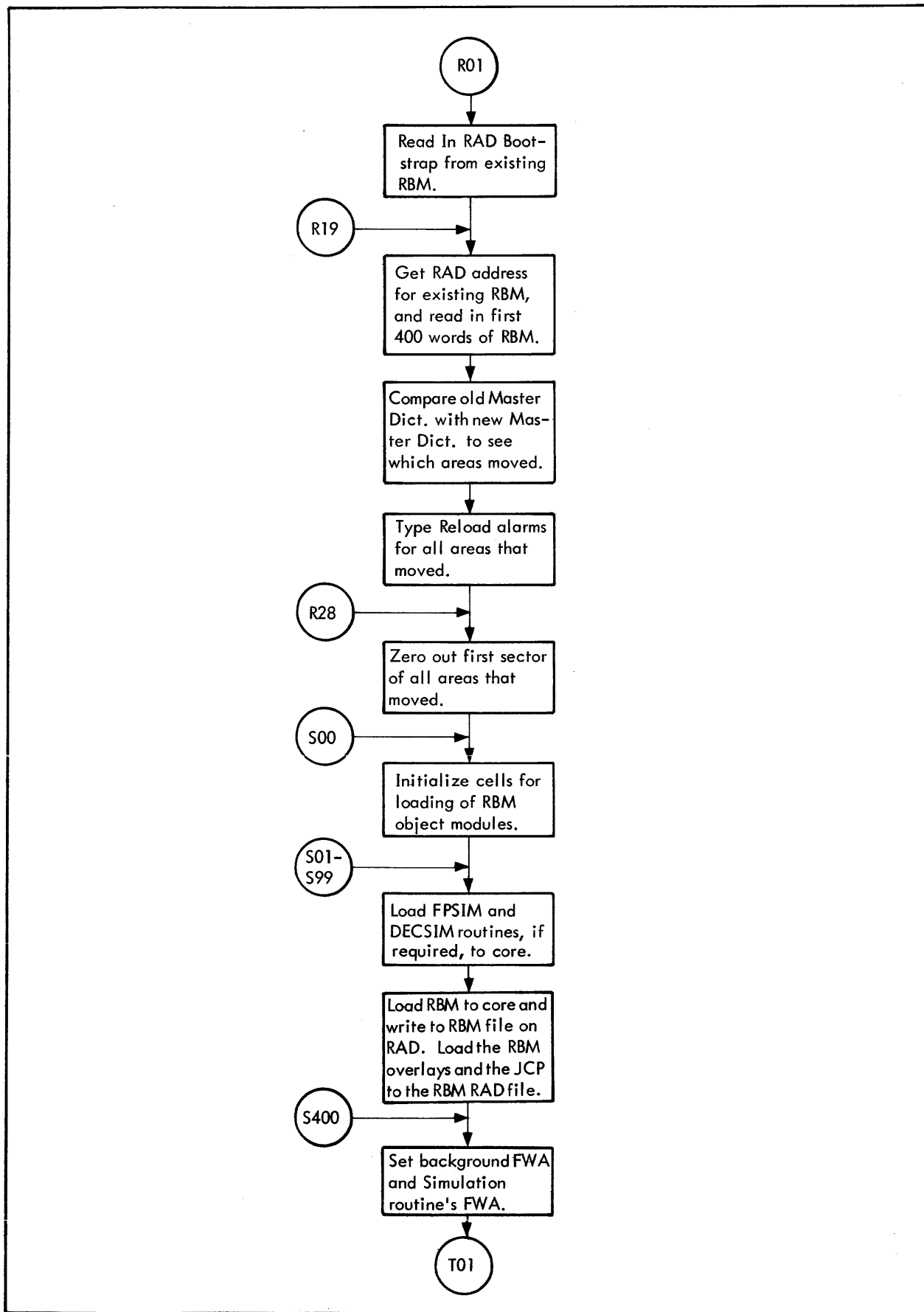


Figure 75. SYSGEN/SYSLOAD Flow (cont.)

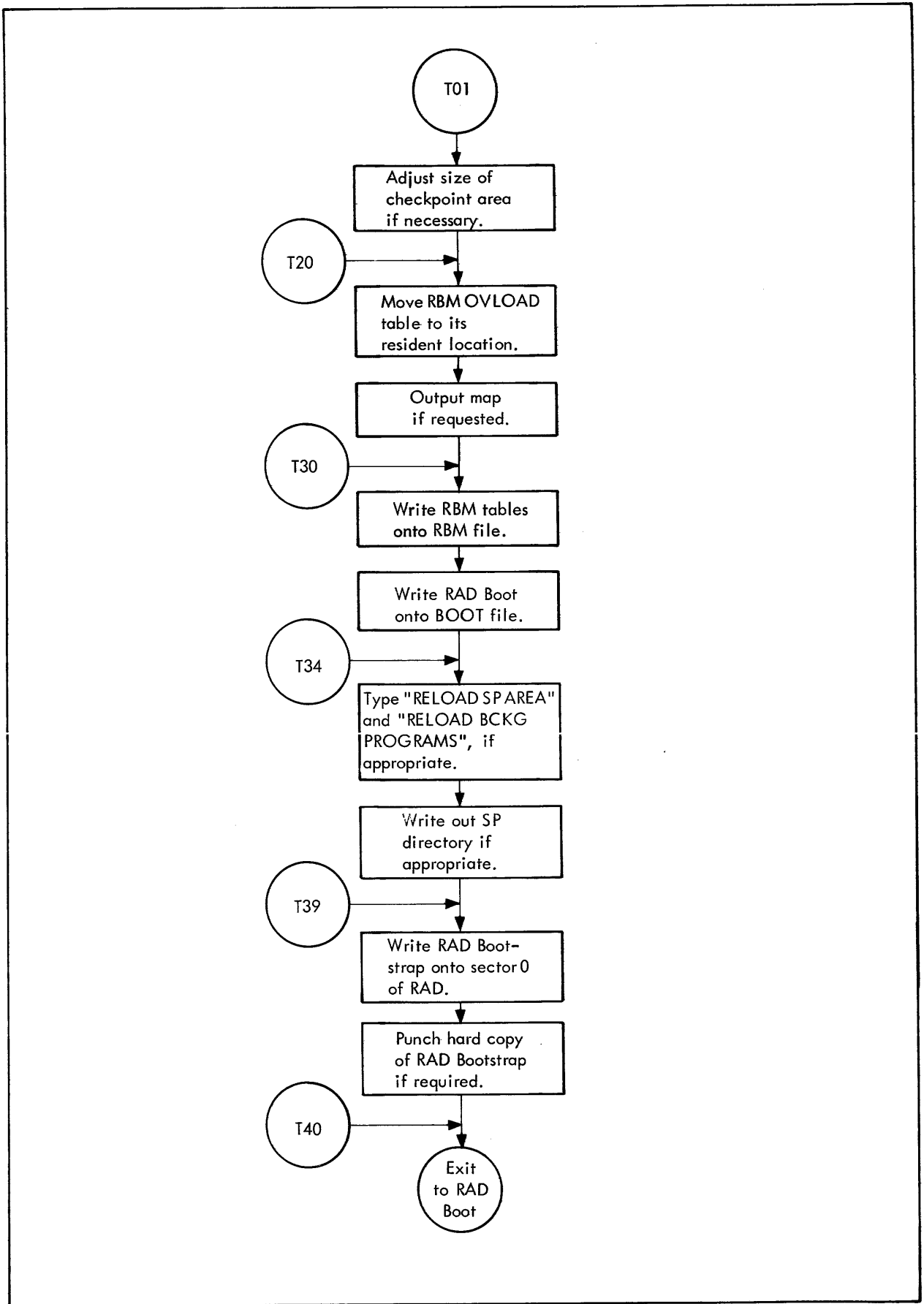


Figure 75. SYSGEN/SYSLOAD Flow (cont.)

Table 9. Standard SYSLOAD DEFs

| DEF Name | Program |
|-------------------|---|
| FPSIM | Floating Point Simulation Routine. |
| DECSIM | Decimal Instruction Simulation Routine. |
| BYTSIM | Byte String Instruction Simulation Routine. |
| CVSIM | Convert Instruction Simulation Routine. |
| DELTA | Debug Package. |
| RBM | Resident portion of RBM. |
| RBMEND | The end of resident RBM. The RBM initialize routine, which is non-resident, follows the RBMEND label. |
| JCP | Job Control Processor. |
| CKPT | Checkpoint/Restart overlay. |
| FGL1 | Overlay to release foreground programs. |
| FGL2 | Overlay to load foreground programs for execution. |
| ABEX | Abort/Exit Overlay. |
| KEY1 | Part 1 of Key-in overlay. |
| KEY2 | Part 2 of Key-in overlay. |
| PMD | Postmortem Dump overlay. |
| BKL1 | Part 1 of overlay to load JCP and background programs for execution. |
| BKL2 ¹ | Part 2 of overlay to load JCP and background programs for execution. |

The Loader satisfies references to any of the RBM tables in the object modules it loads. References that can be satisfied are contained in the RBM Symbol Table. The address of each RBM table is stored in the Symbol Table by SYSGEN when the tables are allocated. Labels that can be defined as an external reference in RBM or the RBM Overlays are

RBM Symbol Table Definitions

| | | |
|---------|---------|--------------------------------------|
| DCT1 | through | DCT19 (i.e., DCT1, DCT2, ..., DCT19) |
| CIT1 | through | CIT3 |
| IOQ1 | through | IOQ14 |
| RFT1 | through | RFT17 |
| FPI | through | FP5 |
| OPLBS1 | through | OPLBS3 |
| INTLB1 | through | INTLB2 |
| OVLOAD1 | through | OVLOAD3 |
| WLOCK | | |
| OLAYFWA | | |

Any external reference not in the above list will result in an alarm

| |
|--------------|
| "ILL. REF. " |
|--------------|

Note that this Loader will not satisfy any DEF/REF linkages between object modules. Only references to the RBM tables contained in the above list will be satisfied.

SYSGEN I/O

SYSGEN and SYSLOAD perform all of their own I/O via the READ/WRITE routine except for the typing of alarms performed by TYPE. The READ/WRITE routine will handle the peripherals itemized below:

| <u>Device</u> | <u>XDS Model Numbers</u> |
|-------------------------|--------------------------|
| Keyboard Printer | 7012, 7020 |
| Card Reader | 7121, 7122, 7140, 7120 |
| Paper Tape Reader/Punch | 7060 |
| Line Printer | 7440, 7445, 7450 |
| 9-Track Magnetic Tape | 7322, 7323 |
| 7-Track Magnetic Tape | 7362, 7372 |
| Card Punch | 7160, 7165 |
| RAD | 7204, 7232, 7212 |
| Disk Pack | 7242, 7246 |

The READ/WRITE routine makes extensive use of tables (called IOT0 through IOT18) that fully describe the characteristics of each peripheral device. (See the comments in the program listing for descriptions of the READ/WRITE routine and the tables.) The paper tape format used by SYSGEN on read operations is identical to the format used by RBM described in Appendix A.

Rebootable Deck Format

If a :PUNCH control command is read by SYSGEN, a rebootable deck is output that includes the RBM tables with their initialized values, SYSLOAD, and the RBM Symbol Table.[†] This deck can be used to load a new version of RBM without re-inputting all the SYSGEN control commands.

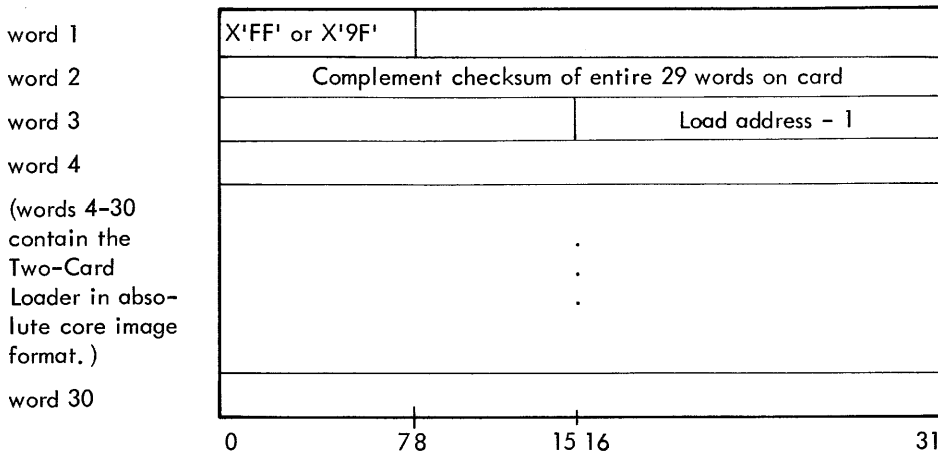
The first card in the rebootable deck consists of a one-card bootstrap program that loads the next two cards in the deck. These next two cards consist of a program that loads the remainder of the deck, consisting essentially of the RBM Table, SYSLOAD, and the RBM Symbol Table in core image format.

The two cards containing the Core Image Loader have the following format:

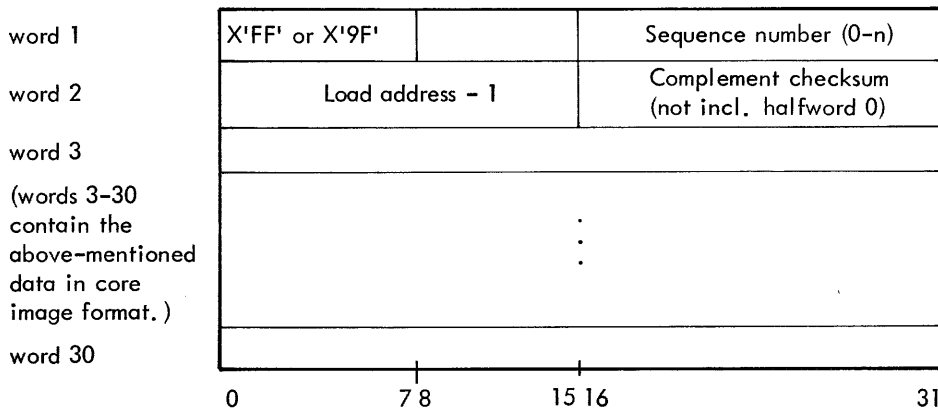
| <u>Byte No.</u> | <u>Contents</u> |
|-----------------|--|
| 0 | X'FF' (for card 1) X'9F' (for card 2) |
| 1, 2, 3 | Unused (all zeros) |
| 4, 5, 6, 7 | Complement checksum of entire card (carry out of bit 0 is ignored in computing checksum) |
| 8, 9 | Unused (all zeros) |
| 10, 11 | Load address, minus one, for following data |
| 12-119 | Loader in absolute core image format |

[†]If the rebootable deck is output to paper tape, there are no special additional characters. That is, the paper tape contains an exact card image.

The core image format of the Two-Card Loader is



The RBM Tables, SYSLOAD, and the RBM Symbol Table are output in the core image format



All cards contain an X'FF' in byte 0 except the last card. The last card contains an X'9F' in byte 0 and the SYSLOAD entry address in place of the load address in word 1. The last card contains no data other than the SYSLOAD entry address, the sequence number, and checksum.

Stand-Alone SYSGEN Loader

The Stand-Alone SYSGEN Loader is a small loader specifically created to load the SYSGEN absolute object module. Since SYSGEN is assembled in absolute, the SYSGEN Loader will only load absolute load items and handles only the small subset of the Sigma Object Language required to load SYSGEN.

The SYSGEN Loader I/O routine is a copy of the SYSGEN I/O, with the code performing the actual loading being similar to the code in the SYSGEN Loader.

The rebootable form of the SYSGEN LOADER is produced by loading its ROM with the Stand-Alone Absolute Dumping Loader (see Stand-Alone Systems Operations Manual, 90 10 53). The absolute binary deck produced can then be booted with the three-card Absolute Bootstrap Loader (can be obtained by ordering CN704145).

APPENDIX A. RBM SYSTEM FLAGS AND POINTERS

Table A-1. RBM System Flags and Pointers

| Name | Location | Description |
|----------|----------|--|
| K:BACKBG | X'140' | Beginning address of background. |
| K:BCKEND | X'141' | Ending address of background. |
| K:FGDBG1 | X'142' | Current beginning address of FGD. |
| K:FGDEND | X'143' | Ending address of FGD. |
| K:CCBUF | X'144' | Address of Control Card Buffer. |
| K:BPOOL | X'145' | Start address of BCKG Blocking Buffer Pool. |
| K:FGDBG2 | X'146' | Beginning address of FGD set at SYSGEN. |
| K:FMBOX | X'147' | Start address of FGD Mailboxes. |
| K:FPOOL | X'148' | Start address of FGD Blocking Buffer Pool. |
| K:UNAVBG | X'149' | Start address of unavailable memory. |
| K:MASTD | X'14A' | Start address of Master Dictionary. |
| K:NUMDA | X'14B' | Highest valid DW index for MASTD. |
| K:VRSION | X'14C' | RBM version. |
| K:ACCNT | X'14D' | Job Accounting flag. |
| K:OV | X'14E' | Permanent and current sizes of OV. |
| K:KEYST | X'14F' | Post status of key-in here. |
| K:JCP1 | X'150' | JCP and Control Task. Bits have the following meaning: Bit 0 = 1, JCP is executing. Bit 1 = 1, Background is active. Bit 2 = 1, Background is checkpointed on the RAD. Bit 3 = 1, Background is being used by Foreground but was not checkpointed. Bit 4 = 1, Waiting for key-in response. Bit 5 = 1, Skip to next JOB card. Bit 6 = 1, Set by ABORT for CALEXIT. Bit 7 = 1, Set by CALEXIT for ABORT. Bits 8 - 15, Previous assign. of C device (for TY key-in). Bits 16 - 21, Unused. Bit 22 = 1, System processor executing. Bit 23 = 1, Execute Delta for BKGD Program. Bits 24 - 25, 0 means no PMD requested. 1 means conditional PMD. 2 means unconditional PMD. Bit 26, Flag for CKPT that alarm typed. |

Table A-1. RBM System Flags and Pointers (cont.)

| Name | Location | Description |
|----------------|----------------|---|
| K:JCPI (cont.) | X'150' (cont.) | Bit 27 = 1, RBM Initialize routine is running. Bit 28 = 1, FG key-in active. Bit 29 = 1, TY key-in active. Bit 30 = 1, Attend command was input. Bit 31 = 1, JOB command was input. |
| K:CTST | X'151' | Flags to execute Control Task subtask. Bits have the following meaning: Bit 0 = 1, Execute CHECKPOINT. Bit 1 = 1, Execute FGD Loader/Releaser. Bit 2 = 1, Execute Restart. Bit 3 = 1, Time to check for I/O rundown. Bit 4 = 1, Execute ABORT/EXIT. Bit 5 = 1, Execute key-in. Bit 6 = 1, Execute PMD. Bit 7 = 1, Execute IDLE. Bit 8 = 1, Execute BCKG load. Bit 9 = 1, Load JCP. Bit 10 = 1, Load BCKG (Program not JCP). Bit 11 = 1, Key-in required by a higher priority subtask. Bit 12 = 1, Reload FGL1 for possible RLS. Bit 26 = 1, KEY2 doing STDLEBRAD file OPEN/CLOSE. Bit 27 = 1, FGL1 called from FGL2. Bit 28 = 1, Control Task is operating. Bit 29 = 0, Execute ABORT part of ABORT/EXIT. Bit 29 = 1, Execute EXIT part of ABORT/EXIT. Bit 30 = 1, PMD from key-in request. Bit 31 = 1, PMD from PMD command. |
| K:SY | X'152' | Nonzero if SY key-in active. |
| K:BPEND | X'153' | End of load area for BCKG program |
| K:CTWD | X'154' | WD code for Control Task. Byte 0 = Nonzero means CT was triggered. |
| K:CTGL | X'155' | Group level for Control Task. |
| K:BLOAD | X'156' | Name in BCD of BCK program to load two words. |
| K:BAREA | X'158' | Index of area to load BCK program from. |
| K:ASSIGN | X'159' | Address of ASSIGN table. |
| K:RUNF | X'15A' | Post run status here for FGD load. |
| K:HIINT | X'15B' | Highest address used for interrupt. |
| K:FGDBG3 | X'15C' | Begin address of FGD from FMEM key-in. |
| K:PMD | X'15D' | Cells to dump for PMD as DW address (5 words). |
| K:DCB | X'162' | DCB for Control Task to load in overlays (7 words). Always assigned to RBM File. |
| K:KEYIN | X'169' | Key-in Response Buffer (6 words). |
| K:FGDBG4 | X'16F' | Byte 0 = FWA of FGD prior to CKPT (Page Bits 15-31 = K:BCKEND prior to CKPT). |

Table A-1. RBM System Flags and Pointers (cont.)

| Name | Location | Description |
|----------|----------|---|
| K:DELTA | X'170' | Entry point for Delta. |
| K:QUEUE | X'171' | Address of Queue routine. Byte 0 = Nonzero, Stop I/O on BCKG. |
| K:BTFILE | X'172' | Status of BT Files Bits 0 - 8, 1 bit for each X1 file. Bit set to 1 means SAVE file. Bits 16 - 31, LWA to use for non-SAVE files. |
| K:GO | X'173' | Permanent and current sizes of GO. |
| K:PAGE | X'174' | Byte 0 = Number of lines per page. |
| K:RDBOOT | X'175' | FWA and device Number of RADBOOT. |
| K:DCT1 | X'176' | Addresses of tables. |
| K:DCT16 | X'177' | |
| K:OPLBS1 | X'178' | |
| K:OPLBS3 | X'179' | |
| K:RFT4 | X'17A' | |
| K:RFT5 | X'17B' | |
| K:SERDEV | X'17C' | Address of SERDEV. |
| K:REQCOM | X'17D' | Address of REQCOM. |
| K:INITX | X'17E' | Address to return to after INIT runs. |
| K:FGLD | X'17F' | Byte 0 = Nonzero, XEQ FGD Load/RLS. |
| K:PMD1 | X'180' | Format flag, FWA, number cells. |
| K:PMD2 | X'181' | Flags used in PMD Overlay (see listing of PMD Overlay for details). |
| K:PMD3 | X'182' | FPT for PMD to write on DO (3 words); status is posted in word 3. |
| K:RUNBPL | X'185' | Cells to post status in for BCKG Public Library load (3 cells). |
| K:CLK1 | X'188' | Clock cells must start on a DW boundary: there are counters for 4 clocks - 2 words/clock. [†] |
| K:CLK2 | X'18A' | Word 2 gets stored into word 1 when Counter = 0. |
| K:CLK3 | X'18C' | |

[†]The user never needs to access Clock 4.

Table A-1. RBM System Flags and Pointers (cont.)

| Name | Location | Description |
|----------|----------|---|
| K:ABTLOC | X'18E' | Abort location. |
| K:DCT4 | X'18F' | Address of DCT4. |
| K:MSG1 | X'190' | KEY-IN. |
| K:MSG2 | X'193' | KEY ERR. |
| K:MSG3 | X'196' | RLS Name NA. |
| K:MSG4 | X'19A' | File Name. |
| K:MSG5 | X'19E' | FGD AREA ACTIVE. |
| K:MSG6 | X'1A3' | NOT ENUF BCKG SPACE. |
| K:MSG7 | X'1A9' | UNABLE TO DO ASSIGN. |
| K:MSG8 | X'1AF' | BCKG CKPT. |
| K:MSG9 | X'1B2' | BKG IN USE BY FGD. |
| K:MSG10 | X'1B7' | BCKG RESTART. |
| K:MSG11 | X'1BB' | CK AREA TOO SMALL. |
| K:MSG12 | X'1C0' | I/O ERR ON CKPT. |
| K:MSG13 | X'1C5' | JOB ABORTED AT XXXXX. |
| K:MSG14 | X'1CB' | LOADED PROG. XXXXXXXX, etc. |
| K:MSG15 | X'1D5' | UNABLE TO LOAD BCKG PUB LIB. |
| K:MSG16 | X'1DD' | CKPT WAITING for BCKG I/O RUNDOWN. |
| K:FPSIM | X'1E6' | Address of Simulation Routines. |
| K:DECSIM | X'1E7' | |
| K:BYTSIM | X'1E8' | |
| K:CVSIM | X'1E9' | |
| K:MONTH | X'1EA' | Table of days/month and DCB names. |
| K:DATE1 | X'1F6' | Maximum number days in year, year - 1970. |
| K:DATE2 | X'1F7' | Day of year. |
| K:TIME | X'1F8' | Time of day in seconds. |
| K:ELTIM1 | X'1F9' | FGD saves BCKG elapsed time here. |
| K:LIMIT | X'1FA' | Maximum execution time for BCKG. |
| K:ASSNAM | X'1FB' | Account entry for AL file (8 words). |

Table A-1. RBM System Flags and Pointers (cont.)

| Name | Location | Description |
|----------|----------|----------------------------------|
| K:ELTIM2 | X'202' | Last WD of entry – elapsed time. |
| K:RFT12 | X'203' | Current record number. |
| K:RFT11 | X'204' | Current file number. |
| K:RFT1 | X'205' | RAD file name. |
| K:RFT8 | X'206' | Master Dictionary Index. |
| K:PTCH | X'207' | Beginning address of patch area. |
| K:PTCHND | X'208' | Ending address of patch area. |

APPENDIX B. PAPER TAPE STANDARD FORMAT

A binary record is signaled by an X'11' as the first character, and the two bytes following are the record sizes. The specified number of data bytes follow the count.

An EBCDIC record is one whose first character is not an X'11'. An EBCDIC record is terminated by an NL code (15₁₆), or a blank frame (00).

STAPLE

STAPLE

FOLD

FIRST CLASS
PERMIT NO. 229
EL SEGUNDO, CALIF.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY

Xerox Data Systems

701 South Aviation Boulevard
El Segundo, California 90245

ATTN: PROGRAMMING PUBLICATIONS



CUT ALONG LINE

FOLD

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

XEROX

XEROX® is a trademark of XEROX CORPORATION.