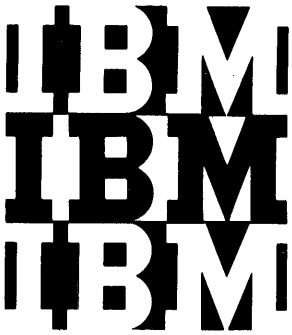


File No. S360-36 (OS)
GY28-6613-5

Program Logic

IBM System/360 Operating System:

Job Management,

Program Logic Manual,

Program Number 360S-CI-505

This publication describes the internal logic within the job management portion of the IBM System/360 Operating System Primary Control Program. Job management prepares jobs for execution, and directs the disposition of data sets created during job execution. It also handles all communication between the operator and the primary control program. Included in the publication are descriptions of tables and work areas used by the job management routines and a directory of names and purposes of control sections, assembly modules, and load modules.

The information contained in this publication applies only to the primary control program.

This manual is intended for persons involved in program maintenance, and system programmers who are altering the program design. Program logic information is not necessary for use and operation of the program.

Sixth Edition (June, 1970)

This is a major revision of, and obsoletes, GY28-6613-4. In addition to incorporating information previously released, this edition also describes the changes made to primary control program job management in Release 19 of the operating system. These changes include:

- The Write-To-Programmer (WTP) facility of the Write-To-Operator and Write-To-Operator-With-Reply macro instructions
- Description of and figure for the Write-To-Programmer Control Block (WTPCB)
- The In-Stream Procedure job control feature
- Description of and figure for the In-Stream Work Area
- Explanation of the loading and deletion of the Device Name and Device Mask Tables
- New Key Values for the JCL Scan Routine
- Changes to Job Management Tables
- Additional assembly modules in the SYS1.NUCLEUS, SYS1.SVCLIB, and SYS1.LINKLIB data sets

Other changes to text, and small changes to illustration, are indicated by a vertical line to the left of the change; changed or added illustrations are denoted by the symbol • to the left of the caption.

This edition applies to Release 19 of IBM System/360 Operating System, and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters. Changes are continually made to the information herein; before using this publication in connection with the operation of IBM Systems, consult the latest IBM System/360 SRL Newsletter, Order No. GN20-0360, for the editions that are applicable and current.

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

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Programming Systems Publications, Department D58, PO Box 390, Poughkeepsie, N. Y. 12602

Preface

This publication describes the structure of the sequential scheduler configuration of job management, its functions, and the control flow between its major routines. It is divided into an introduction in which job management is briefly described and three major sections, master scheduler, interpreter, and initiator/terminator, in which the corresponding components are described in greater detail. Included are four appendixes. Appendix A describes two subroutines used frequently by job management routines. Appendix B shows job management tables and work areas that are not described in the body of the publication. Appendix C lists job management load modules and the assembly modules that each contains. Appendix D lists the acronyms used in this publication and the meaning of each. Further information on job management may be obtained from the program listings.

Readers should have a thorough understanding of IBM System/360 programming and should be familiar with the following publications:

IBM System/360 Operating System:

- Introduction, GC28-6534
- Concepts and Facilities, GC28-6535
- Operator's Reference, GC28-6691
- Job Control Language Reference, GC28-6539
- Introduction to Control Program Logic, Program Logic Manual, GY28-6605
- System Control Blocks, GC28-6628

Contents

Figures

Charts

Summary of Major Changes--Release 19

Item	Description	Areas Affected
Write-to-Programmer (WTP)	A facility added to WTO and WTOR macro instruction processing, allowing programmer messages to be written to SYSPRINT	Charts, pages 114-120 Initiator-terminator, pages 38, 58 Job processing, page 12 Load and assembly modules, pages 83-84 Master scheduler, pages 14, 17-18 Tables and work areas, page 82
In-Stream Procedure	An added job control feature that allows procedures to be placed in the job stream rather than in a procedure library.	Assembly modules and control sections, pages 100-102, 105 Charts, page 127 Interpreter, pages 19-22 Load and assembly modules. pages 86, 89-90, 92, 94-95, 98 Tables and work areas, page 68
Device name table, Device mask table	Discussion of the separate loading and deletion of these tables has been added	Assembly modules and control sections, pages 99-100, 103 Initiator/terminator, pages 41, 50 Load and assembly modules, pages 84-85, 88, 90-91, 94, 96-98 Tables and work areas, page 66
JCL Processing	New key values have been assigned in scan processing; and the DD statement parameter disposition list has been updated.	Interpreter, pages 25-27, 30-33

Job management (Chart 1) is the first and last portion of the control program that a job encounters. Its primary function is to prepare job steps for execution and, when they have been executed, to direct the disposition of data sets used during execution. Prior to step execution, job management:

- Reads control statements from the input job stream.
- Places information contained in the statements into a series of tables.
- Analyzes input/output (I/O) requirements.
- Assigns I/O devices.
- Passes control to the job step.

Following step execution, job management:

- Releases main storage space occupied by the tables.
- Frees I/O devices assigned to the step.
- Disposes of data sets referred to or created during execution.

Job management also performs all processing required for communication between the operator and the control program. Major components of job management are the job scheduler, which introduces each job step to System/360, and the master scheduler, which handles all operator-system-operator communication.

Job Scheduler Functions

The job scheduler includes two programs: the reader/interpreter and the initiator/terminator. The interpreter is given control whenever a job step is to be obtained from the input job stream and processed. It directs the reading of control statements and from them constructs:

- A job control table (JCT) to describe the job.
- A step control table (SCT) to describe the job step.
- An account control table (ACT) to describe accounting information related to the job.
- Job file control blocks (JFCB) (one for each DD statement) to describe the data sets to be used by the job.
- Step input/output tables (SIOT) (one for each DD statement) to describe the I/O requirements of the job step.
- Volume tables (VOLT) (one for each step) with an entry for each DD statement containing serial numbers of volumes to be used by the job step.
- Data set name (DSNAME) tables (one for each step, with an entry for each DD statement) containing names of previously defined data sets to be used by the job step.

In addition to the above, the interpreter creates system message blocks, in which diagnostic messages to the programmer are stored before they are written onto the system output data set.

After all control statements for a job have been processed, or when data is encountered in the input job stream, the interpreter gives control to the initiator/terminator. The latter analyzes the I/O requirements of the job step and, upon considering such factors as requests for specific units, volumes, and channels and their current employment, it assigns devices in such a way as to achieve maximum overlap of I/O activity during step execution.

When all devices requested for the step have been assigned, the initiator/terminator issues mounting messages (if any are required) and verifies for direct access requests that the operator has mounted volumes on the correct units. Control is then passed to the job step. When the step has been executed, control is given to the initiator/terminator, which performs data set dispositions and releases I/O resources.

Master Scheduler Functions

The routines of the master scheduler process any communication between the operator and the system. The master scheduler processes:

- Operator commands, whether they are issued through the console or through the input job stream.

- Write-to-operator (WTO) and write-to-operator with reply (WTOR) macro instructions, either of which may involve write-to-programmer (WTP).

- Interruptions caused when the INTERRUPT key is pressed.

Job Processing

Figure 1 shows the major components of job management and illustrates the general flow of control.

Control is passed to job management whenever the supervisor finds that there are no program request blocks in the request block queue. This can occur for two reasons: either the initial program loading (IPL) procedure has just been completed or a job step has just been executed.

ENTRY TO JOB MANAGEMENT FOLLOWING INITIAL PROGRAM LOADING

Following IPL, certain actions must be taken by the operator before job processing can begin. Therefore, control passes to the master scheduler, which issues a message to the operator instructing him to enter commands. These "initialization" commands include a SET command, a start writer (START WTR) command, and a start reader (START RDR) command. The last initialization command to be issued is a START command with no parameters; when this command is issued, control passes to the interpreter for control statement processing.

ENTRY TO JOB MANAGEMENT FOLLOWING STEP EXECUTION

Following step execution, control is routed to the step termination routine of the initiator/terminator. If the job had been completed, control is also passed to the job termination routine of the initiator/terminator. Both routines are described under "Job and Step Termination."

CONTROL STATEMENT PROCESSING

After completion of the processing that immediately follows IPL, or after termination of a job or of a step containing data in the input job stream, control is passed to the interpreter. The interpreter reads and processes control statements until one of the following conditions is encountered:

- A DD * or DD DATA statement.
- Another JOB statement.
- A null statement.
- An end-of-data set (EOF) on the system input device.

Meanwhile, if the operator has pressed the REQUEST key and has entered a request (REQ) command during execution of the job step or any of the above processing, the master scheduler sets a command-pending indicator in the nucleus during the ensuing interruption. The indicator is now checked and, if found to be on, control is passed to the master scheduler, which issues a message instructing the operator to enter commands, and then processes the commands.

STEP INITIATION

Control next passes to the initiator/terminator, which examines I/O device requirements, assigns (allocates) I/O devices to the job step, issues mounting instructions, and verifies that direct access volumes have been mounted on the correct units. Finally, the initiator/terminator passes control to the job step.

JOB AND STEP TERMINATION

When processing program execution is completed, the supervisor, finding no program request blocks in its request block queue, passes control to the job management routines. Entry is first made to the step termination routine.

The step termination routine performs end-of-step housekeeping and passes control to the user's accounting routine, if one was provided. When the accounting routine has been executed, the supervisor returns control to the step termination routine. Control is then passed to the job termination routine if there are no more steps in the job; to the interpreter if the next step of this job has not been read yet (i.e., the step just terminated had data in the input stream); or to the step initiation routine if the next step of this job has been read.

The job termination routine performs end-of-job housekeeping. It exits to the user's accounting routine, if one was provided. After the accounting routine is executed, the supervisor returns control to the job termination routine, which passes control to the interpreter.

OPERATOR-SYSTEM COMMUNICATION PROCESSING

The routines that handle operator-system communication are contained in the master scheduler. Communication may take one of two forms: commands, which allow the

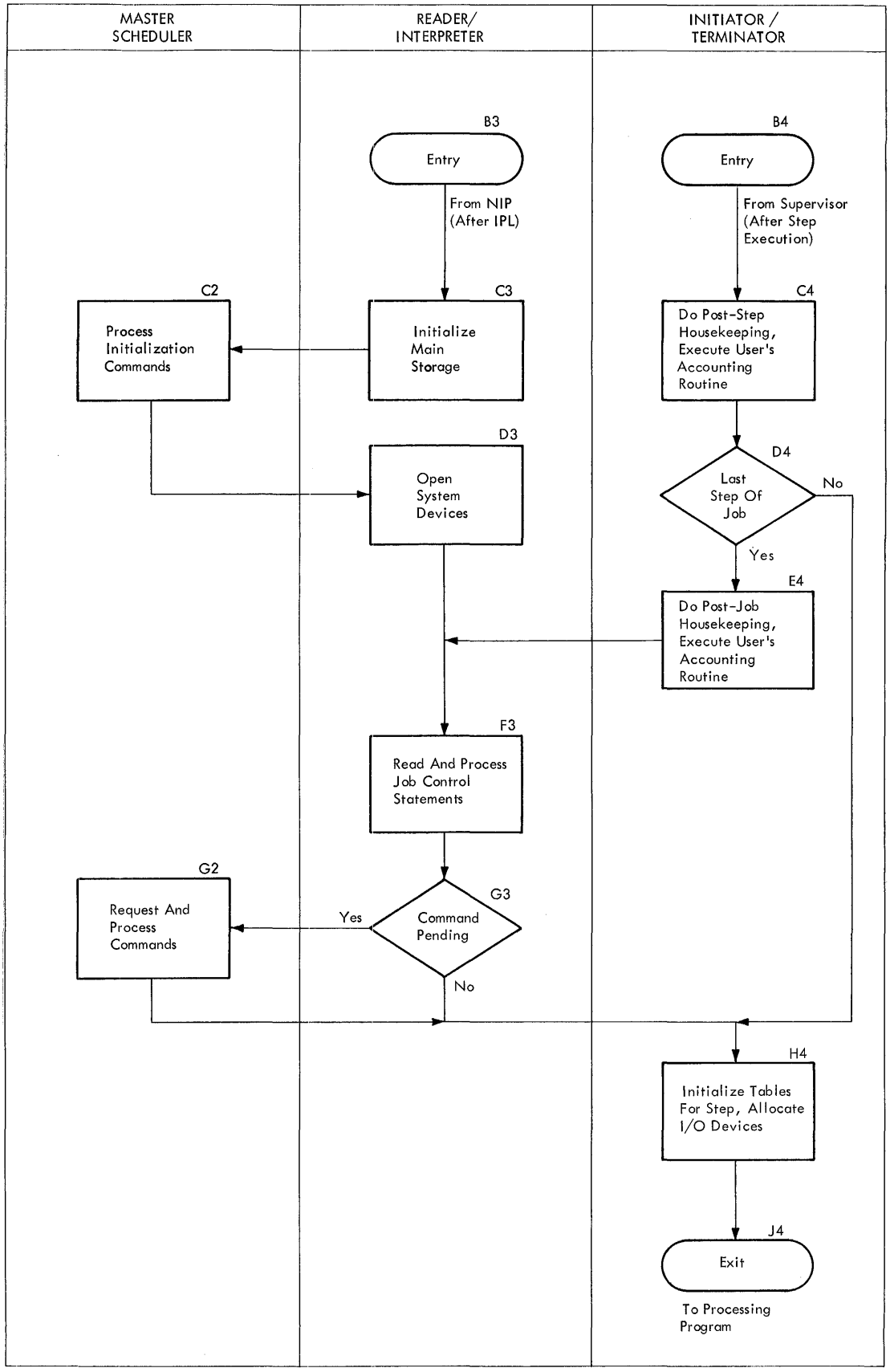


Figure 1. Job Management Control Flow

operator to change the status of the system or of a job or job step; and the WTO or WTOR macro instructions, which allow processing programs or system components to issue messages to the operator through the console output device, or to the programmer through the system message class output device when the write-to-programmer facility is invoked. The master scheduler also switches functions from the primary console device to an alternate console device when the INTERRUPT key is depressed.

Command Processing

Commands may be issued by the operator in two ways: he may insert command statements between job steps in the input job stream, or he may issue commands through the console input device. Commands encountered in the input job stream cause control to be passed to the master scheduler, which processes them. Before entering commands through the console, however, the operator must press the REQUEST key to cause an attention interruption. Figure 2 shows the actions taken after the key is pressed.

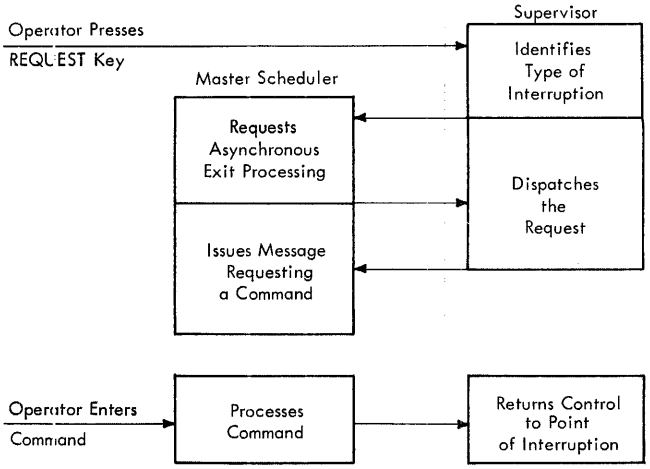


Figure 2. Attention Interruption Processing Flow

WTO/WTOR Macro Instruction Processing

Whenever the WTO or WTOR macro instruction is issued, an SVC interruption occurs. (See Figure 3.)

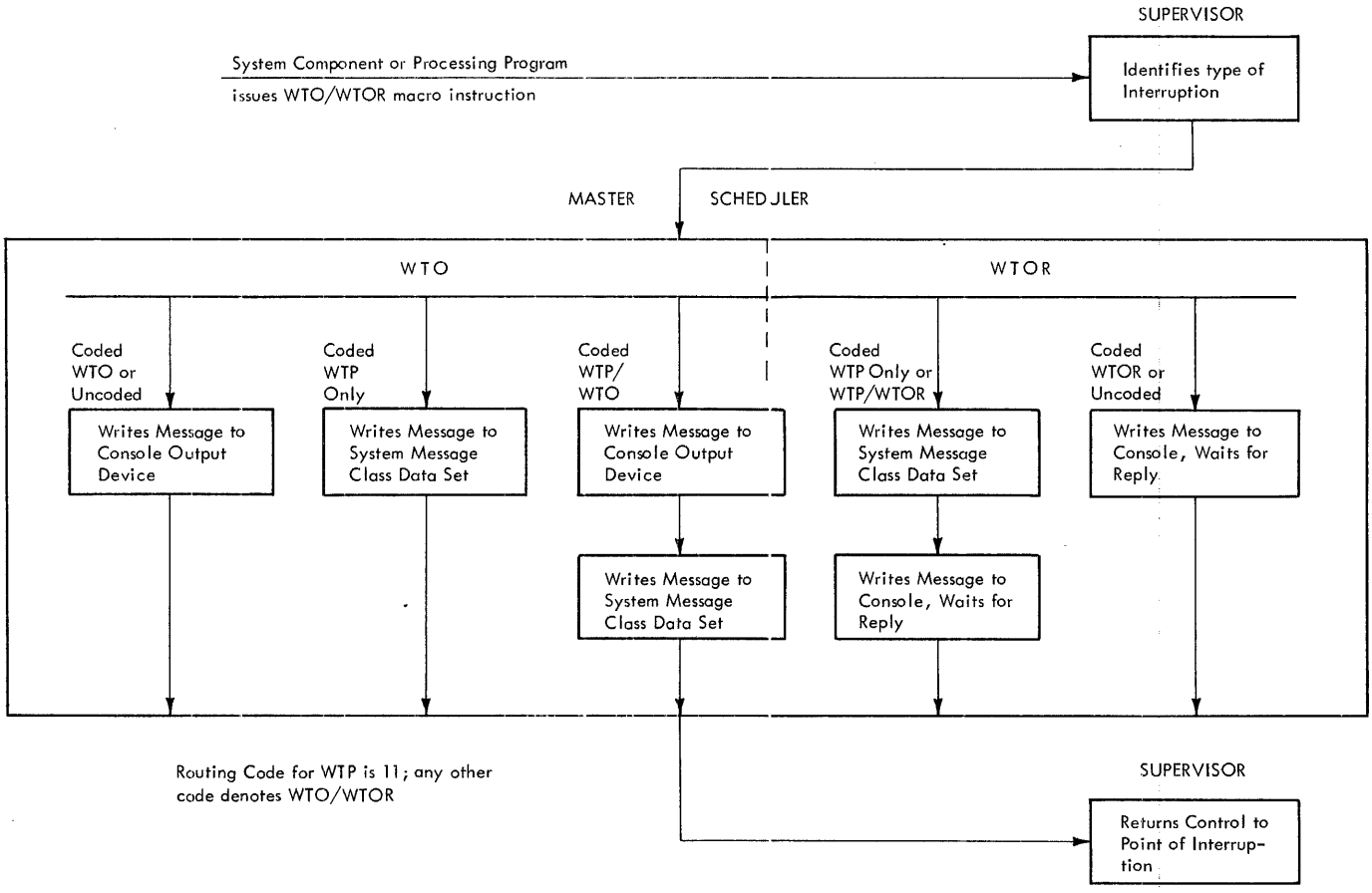


Figure 3. WTO/WTOR Macro Instruction Processing Flow

External Interruption Processing

When the operator presses the INTERRUPT key, an external interruption occurs, following which the master scheduler switches functions from the primary to the alternate console I/O device. (See Figure 4.)

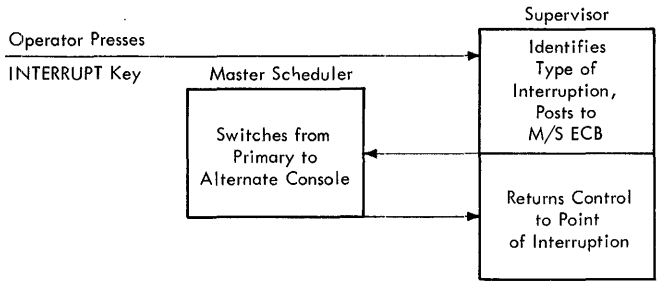


Figure 4. External Interruption Processing Flow

LOAD MODULES

Most job management routines exist as a series of load modules that reside in the link library (SYS1.LINKLIB). The exceptions are the interruption-handling routines of the master scheduler, which reside in the nucleus, and the master command EXCP routine which is in the SVC library (SYS1.SVCLIB). Appendix C contains a list of the routines that make up each job management load module.

Master Scheduler

The master scheduler (Chart 2) processes all operator commands and messages directed to the operator through use of the WTO and WTOR macro instructions. It also performs console switching when the secondary console is to be used in place of the primary console.

The five major routines of the master scheduler are:

- Console interrupt routine, which provides the supervisor with the information necessary to queue a request for processing an attention interruption.
- Master command EXCP routine, which reads commands from the console input device and processes all commands except SET, START RDR, and START WTR.
- Master command routine, which analyzes command verbs and routes control to appropriate command execution routines.
- Write-to-operator routine, which processes messages to the operator and/or the programmer, and all operator replies to these messages.
- External interrupt routine, which switches to the alternate console device when an external interruption occurs.

Master Scheduler Control Flow

Commands are issued through either the console I/O device or the input reader. (See Figure 5.) Before entering commands through the console I/O device, the operator must cause an I/O interruption by pressing the REQUEST key. When he does, control is given to the supervisor. The supervisor determines that an I/O interruption has occurred and passes control to the I/O supervisor. The I/O supervisor determines that an attention interruption has occurred and passes control to the master scheduler console interrupt routine.

The console interrupt routine resides in the nucleus. It passes to the supervisor the address of an interruption queue element to be added to an asynchronous exit queue. The interruption queue element contains the address of an interruption request block that points to the master scheduler interrupt request block routine. Control is passed to the interrupt request block routine when the request is honored by the supervisor. A description of the asynchronous exit queue and the manner in which it is used is contained in the publication IBM System/360 Operating System: Fixed-Task Supervisor, Program Logic Manual, GY28-6612. The format of the master scheduler interruption queue element is given in the section entitled "Console Interrupt Routine."

The interrupt request block routine causes the master command EXCP routine to be brought into the supervisor call (SVC) transient area of the nucleus, where control is passed to it.

The master command EXCP routine uses an EXCP macro instruction to read the command. (The PROCEED light on the 1052 Printer-Keyboard is turned on at this time.) Eight commands, the REQ, START (blank), CANCEL, DISPLAY, MOUNT, STOP, UNLOAD, and VARY commands, are always accepted and processed. All other commands are ignored (control is returned to the supervisor) if issued at any time other than in response to a message issued by the master command routine. If the command is acceptable, it is moved from the buffer into which it was read to a local buffer, and control is passed to the master command routine.

The master command routine analyzes commands and routes control to appropriate command execution routines. If a command is issued through the input job stream, control is passed directly to the master command routine by the interpreter. When all commands have been entered and processed, control returns to the interpreter.

The write-to-operator routine is entered from the SVC handler when a WTO or WTOR macro instruction is issued. When either macro instruction is issued, an SVC interruption occurs and the write-to-operator routine is brought into the SVC transient area of the nucleus. Basically, the write-to-operator routine uses an EXCP macro instruction to write the message on the console output device and, if a reply is expected, to read the reply, which is placed into an area designated by the requester. Either WTO or WTOR may contain parameters which will result in the message being written to the programmer on the system message class data set, with or without a write to the console, depending upon the coding. (See Figure 3.) Control is returned to the supervisor.

The external interrupt routine assigns the functions performed by the primary console device to the alternate console device. When the operator presses the INTERRUPT key on the console, an external interruption occurs and control is given to the supervisor, which identifies the interruption and passes control to the external interrupt routine. The external interrupt routine then switches consoles and returns control to the supervisor. Console functions may later be reassigned to the primary console device if the operator causes another external interruption (the external interrupt routine will again switch functions).

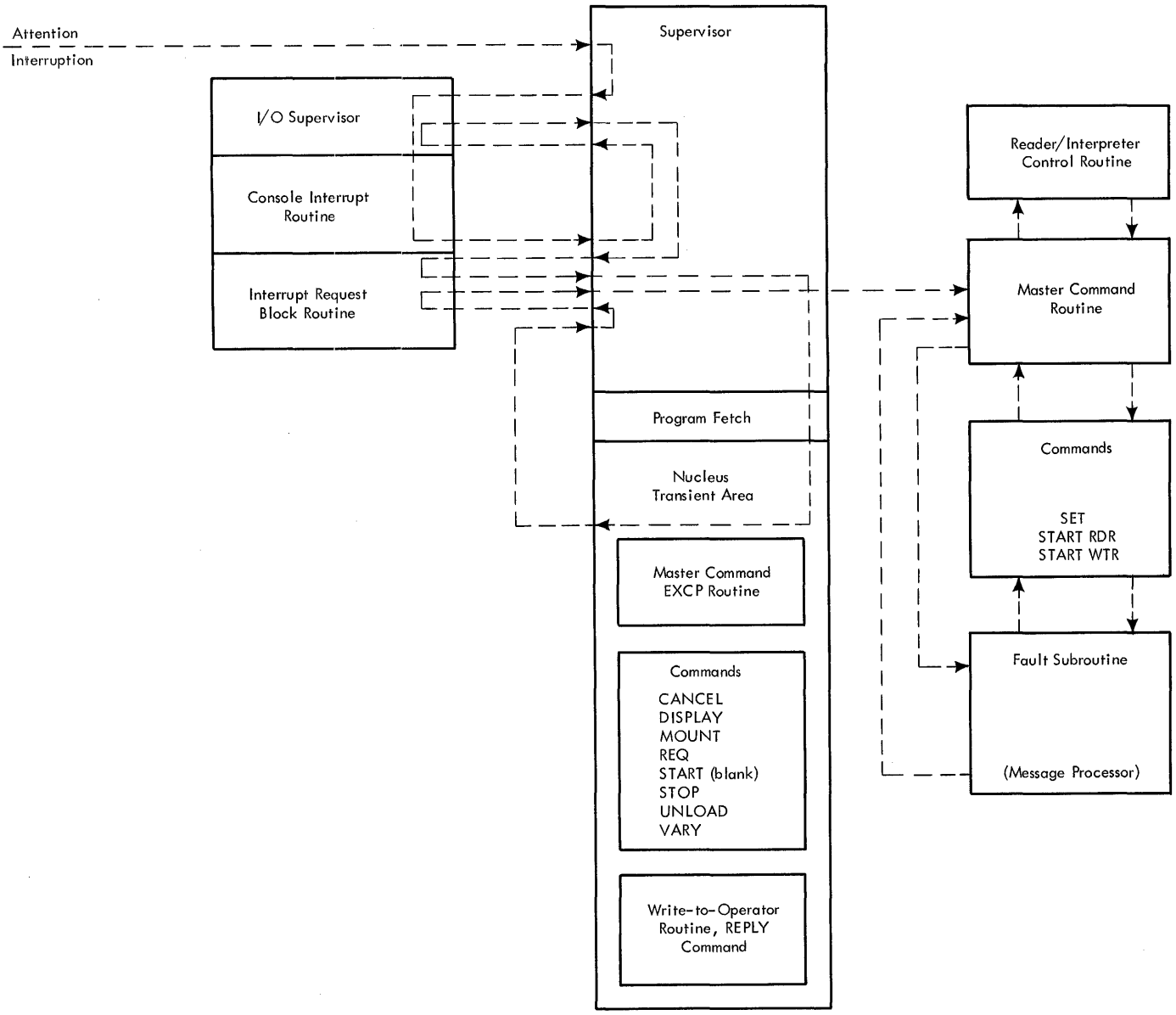


Figure 5. Master Scheduler - Command Processing Network

Console Interrupt Routine

The console interrupt routine (Chart 3) provides the supervisor with the address of the routine to be given control when the supervisor processes an attention interruption. The console interrupt routine is part of the nucleus and is entered from the I/O supervisor each time an attention interruption occurs.

Upon entry to the console interrupt routine, the console flag switch is checked. If this switch is on, either the master command routine or the console interrupt routine is processing a prior request, and a RETURN is made to the I/O supervisor.

When an interruption is not being processed by either routine, the console flag switch is turned on, the address of the master scheduler interruption queue element is placed into general register 1, and control is passed to the supervisor. The interruption queue element is shown in Figure 6.

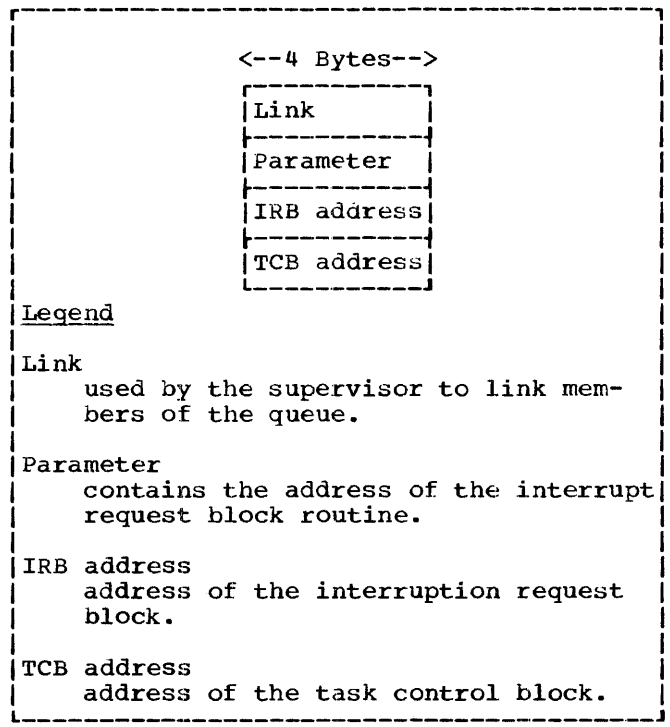


Figure 6. Master Scheduler Interruption Queue Element

The interruption request block contains the address of the interrupt request block (IRB) routine to which control is passed by the supervisor when it dispatches the request. The IRB routine uses an SVC 34 instruction to cause the master command EXCP routine to be brought into the transient area of the nucleus.

Master Command EXCP Routine

The master command EXCP routine (Chart 4) processes the CANCEL, DISPLAY, MOUNT, REQ, START (blank), STOP, UNLOAD, and VARY commands. It resides in SYS1.SVCLIB, and is brought into the transient area of the nucleus by the supervisor when an SVC 34 instruction is issued by the master scheduler interrupt request block routine or the master command routine.

If entry to this routine was from the interrupt request block routine, an EXCP macro instruction is used to read the command from the console and place it into the command buffer. If the command is one of the eight previously mentioned commands, it is processed.

SET, START RDR, START WTR, and STOP WTR commands are ignored unless they were issued in response to a message from the master command routine. If so, control is passed to the master command routine, which processes them.

Following return from the master command routine, or after execution of the REQ or START (blank) commands, the console flag switch is turned off to indicate to the console interrupt routine that another attention interruption can be processed.

If entry to the master command EXCP routine was from the master command routine, the command is available in a buffer (placed there by the master command routine). The command is processed.

The master command EXCP routine returns control to the supervisor.

Master Command Routine

The master command routine (Chart 5) analyzes command verbs and routes control to appropriate command execution routines. It also issues a message to the operator, informing him that commands will be accepted from the console. The master command routine is brought into main storage and entered when:

- The interpreter encounters a command in the input job stream.
- The interpreter is performing the initialization procedures that follow IPL.
- The interpreter finds the command pending switch on. (The command pending switch is turned on by the routine that processes the REQ command.)

- The interpreter encounters an end-of-data set condition in the input job stream, indicating the end of a job step or job. Control is passed to the master command routine after the job step has been processed.

Upon entry, general register 0 is examined. If it contains zeros, entry was made because the interpreter encountered a command in the input job stream. The command is moved to the master command routine buffer and is written out on the console output device for the operator's records. The command verb is then analyzed, and if it is a SET, START RDR, START WTR, or STOP WTR command, control is passed to an appropriate command execution routine. Otherwise, an SVC 34 instruction is used to pass control to the master command EXCP routine.

If general register 0 does not contain zeros upon entry to the master command routine, the IPL pending, new reader pending, and new writer pending switches are checked. If any of these switches are on, the command pending switch is turned on and a message is issued requesting the operator to enter commands. Control is then passed to the initialization command routine, which provides certain commands, specified by the installation during system generation (SYSGEN), to relieve the operator of entering initialization commands. Each of these commands, if there are any, is moved to the master command routine buffer, written on the console output device for the operator's records, and executed.

If general register 0 does not contain zeros and none of the previously mentioned pending switches are on, entry to this routine was made because the interpreter found the command pending switch on, or encountered an end-of-data set condition in the input job stream. A message is issued requesting commands from the operator. After the operator has issued commands and they have been processed, control is returned to the interpreter.

Write-To-Operator Routine

When a WTO or WTOR macro instruction is issued, the write-to-operator routine (Chart 6) gains control by means of an SVC 35 interruption. The routine searches for routing codes specified as values of the ROUTCDE= parameter of WTO/WTOR. If routing code 11, assigned to write-to-programmer messages, is present, the message will be written on the system message class data set in SYS1.SYSJOBQE. If, for a WTO macro instruction, it is desired that the message also be written to the console output device, an additional routing code (any other than 11) must be present. For a WTOR macro instruction, the message goes to the console whether or not an additional routing code is found.

If the message does not carry the assigned WTP routing code, either form of the macro instruction writes the message to the console device and immediately returns control to the supervisor. Processing is resumed at the point of interruption (with a WAIT macro instruction if an operator's reply is to be entered).

If the WTP code is present in the message, however, control is transferred to write-to-programmer processing under the following conditions:

- A WTO macro instruction containing both WTO and WTP routing codes will write the message to the console before transferring control to WTP.
- A WTO macro instruction containing only the WTP routing code will transfer control to WTP but will not write the message to the console.
- A WTOR macro instruction, regardless of routing code content, transfers control to WTP before writing the message to the console.

WTP messages may originate both in system components and in processing programs. The operating system uses WTP to provide the programmer with dynamic descriptions in the event of abnormal occurrences during execution of a processing program. The facility is used by processing programs to write a limited number of messages to the system message class output device when SYSOUT has not been specified in the JCL statements.

The limit to the number of WTP messages to be written to the message queue in a given job (defined in the JOBQWTP= parameter of the SCHEDULR macro instruction at SYSGEN time) is resident in the nucleus. The maximum is twenty messages; the default value assigned when the parameter is omitted is two. Each time WTP prepares to process a new message, a check is made to ensure that the limit has not been exceeded. If it has not, WTP utilizes the transient queue manager (SVC 90) to read and write messages and to assign system message blocks (SMBs).

The actual number of messages passed to the queue may well be greater than the SMB count maintained by WTP. Message queue record size is 176 bytes, of which 15 bytes are reserved for control information, leaving 161 bytes to contain message text.

However, the maximum allowable text length is 126 bytes. A full-length message would therefore leave a number of unused bytes in each record. To make maximum use of the available storage, WTP will, within a given job step, fit more than one message into a queue record when possible. For example, two 80-byte or three 53-byte messages can be placed into a single message queue record.

WTP Error Handling

During initiation of the first job step, WTP reserves two independent SMBs for its use in the event of error. Possible errors and the manner in which they are resolved are shown here:

Messages Exceed Limit: WTP uses one of its reserve SMBs to write an explanatory message on the system message class data set. All processing program WTP messages for the remainder of the job (including the one which initiated the condition, if such was the case) are ignored. From one to three additional system messages, including the possible error-initiating record, can be sucessfully processed (depending on length) through use of the reserved SMBs. Any in excess of the reserve storage capacity are lost.

Input/Output Error: When the transient queue manager is unsuccessful in a read or write operation when attempting to place a programmer message on the queue, WTP writes an explanatory message, followed by the unprocessed programmer message, on the console output device. Once this error has occurred, subsequent WTP messages within the job step are suppressed.

No Available SMBs: When the volume of normal (as opposed to WTP) system messages is so great that no more system message blocks are available, even though WTP has not used the full number assigned to its programmer messages, an explanation is written both to the message class data set (through use of the reserve SMBs) and to the console. Again, from one to three system messages can be processed after this condition occurs. Problem program messages encountered thereafter are bypassed.

WTP Control Transfer

Control is transferred by WTP in the following manner:

WTP entered from	WTP completed	Control passed to
WTO	Yes	Supervisor for return to point of interruption
WTO	No	WTO
WTOR	Yes or No	WTOR

If return from WTP is made to WTO because of unsuccessful handling of the WTP message, an EXCP macro instruction is used to write the message on the console output device, and control is passed to the supervisor for return to the processing point where the interruption occurred.

When it is WTOR which invoked WTP, return is made to WTOR regardless of WTP message completion. The message is written to the console output device. The supervisor then resumes control of processing. If a WAIT macro instruction is now encountered, the system waits for the operator's reply, places it in the storage area designated in the WTOR parameters, and posts the event control block (ECB).

External Interrupt Routine

The external interrupt routine (Chart 13) switches to an alternate console device when the operator presses the INTERRUPT key on the console. This routine resides in the nucleus.

The primary function of the interpreter (Chart 14) is to read job control statements, analyze their contents, and build tables that are used during initiation and execution of job steps.

Control is passed to the interpreter following:

- The IPL procedure.
- Execution and termination of a job step that was followed by data in the input job stream.
- Execution and termination of the last step of a job.

In each case, the interpreter begins reading and processing control statements.

The interpreter is a processing program that operates in the problem program mode with a protection key of zero. It is capable of taking information from an input stream and the procedure library, processing it, and storing it for convenient retrieval by other programs. It is used by the operating system to translate job processing information into convenient form for processing by the initiator/terminator.

The private procedure library (SYS1.PROCLIB) is a partitioned data set. Each member (called a cataloged procedure) is a series of job control language (JCL) statements describing frequently executed series of job steps.

An input stream is a sequential data set composed of JCL statements, operator command statements, system input data, and, if desired, in-stream procedures (a series of non-cataloged JCL statements that describe frequently executed job steps). PROC and PEND statements mark the beginning and end, respectively, of an in-stream procedure. For detailed information on preparing in-stream procedure statements, see IBM System/360 Operating System: Job Control User's Guide, GC28-6703.

Figure 7 shows the data flow in the interpreter. The interpreter is entered at the initialization routine, as a result of a START RDR command; the initialization routine stores the initializing parameters and opens the input stream and procedure library data sets, then passes control to the control routine.

The control routine reads the input stream and procedure library records. It passes JCL statements to the JCL scan routine.

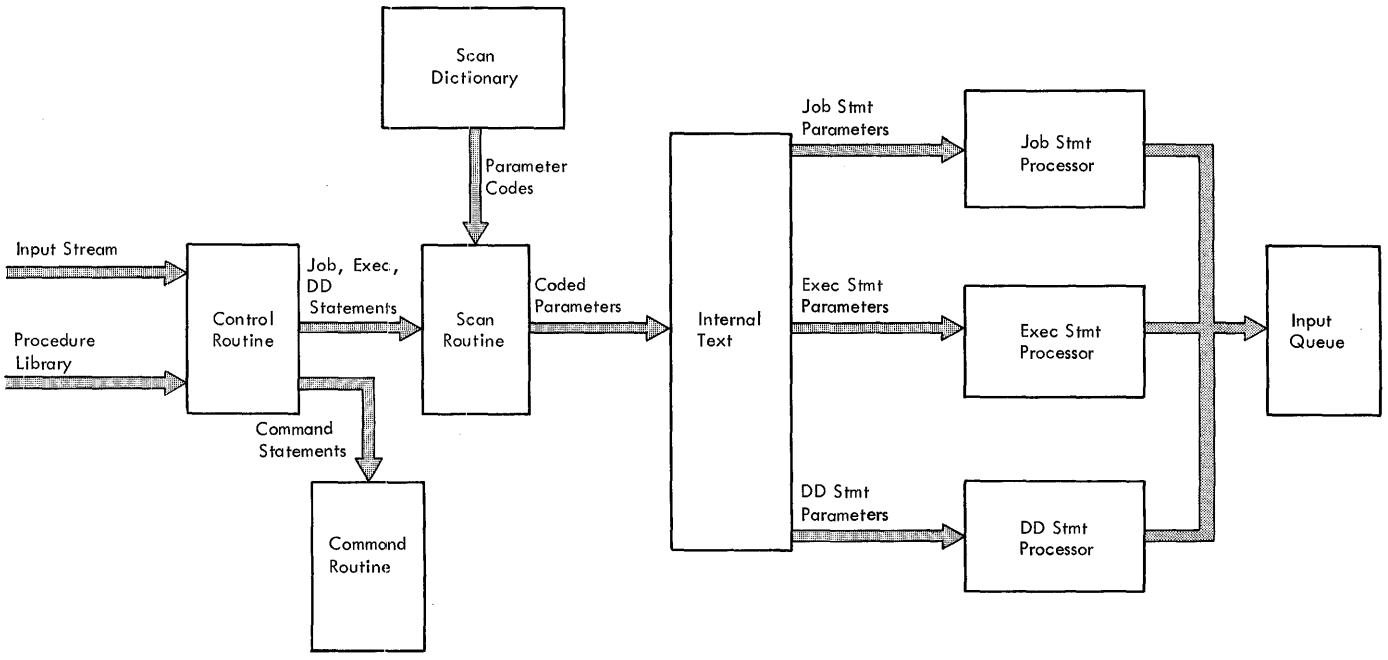


Figure 7. Interpreter Data Flow

The scan routine converts JCL statements into an internal text format. Since a JCL statement in the input stream may invoke and modify cataloged or in-stream procedures, the scan routine accumulates a complete logical statement (which may include several records from the input stream and the procedure library) before further processing is performed. When it has converted the complete logical statement into internal text, it passes the text to the appropriate JCL statement processor routine.

The processor routines build the tables for the job and write them into the queue data set. In addition, they create system message blocks as required and write them into the queue data set.

Initializing the Interpreter

The interpreter is entered at the initialization routine, which consists of two modules: the initialization module (IEFVH1) and the open module (IEFVH2). At entry, control passes to the initialization module, which obtains main storage for the interpreter work area (IWA), and for the local work area (LWA). The IWA contains information that is shared by two or more of the interpreter routines, while the LWA is used individually by each major routine and contains only information that need not be preserved outside the routine. Throughout the use of the interpreter, register 12 contains a pointer to the IWA; the IWA contains a pointer to the LWA.

When the main storage for the work areas has been obtained, the initialization routine obtains main storage for the input stream DCB and the procedure library DCB, then stores pointers to these areas in the IWA.

The routine then issues a TTIMER macro instruction and combines the time with the reader number in the interpreter option list to create the base for any unique set names to be generated for this input stream. Next, it examines the PARM field of the code list. It extracts the option fields, and sets the corresponding switches and values in the IWA.

The initialization module finally passes control to the open module, which opens the input stream and procedure library data sets. The input stream data set is opened for QSAM; the procedure library is opened for BPAM.

Input and Control Operations

When the initialization is complete, control is passed to the interpreter control routine (Chart 16), which reads records from the input stream and procedure library, determines the record type and processing required, and either performs the processing or passes control to the appropriate processing routine.

READING CONTROL STATEMENTS

The interpreter control routine is entered at the interpreter get routine (module IEFVHA). This routine uses the GET and READ macro instructions to obtain records from the input stream and procedure library, or the in-stream procedure buffers, respectively. Only one input source is read upon each entry to the routine, except when a blocked procedure library is specified. In this case, a block is read and a pointer is passed to the input source statement. Switches set in the verb identification routine (module IEFVHCB) determine which data set is read.

When the record is in main storage, the get routine determines if it is a control record (// in positions 1 and 2). If a non-control record is encountered, control is passed to module IEFVHB. This module will cancel the job and print the message INPUT STREAM DATA FLUSHED. Then return is made to the get routine.

END-OF-DATA AND NULL STATEMENTS

The physical end of an input stream is signalled by an end-of-data indication from the computing system. A null statement is the last statement in an input stream (or a job description), and is also the last statement in a cataloged procedure.

An end-of-data condition causes control to be passed to the interpreter EODAD exit routine (module IEFVHAA), which determines whether there is a job to be enqueued. If not, it passes control to the interpreter termination routine; if so, it constructs a null statement, and passes control to the continuation statement routine.

The continuation check routine passes control to the verb identification routine, which determines that the statement is a null statement, and passes control to the null statement routine.

The null statement routine (module IEFVHL) is given control by the verb identification routine whenever it encounters a null statement. The null statement routine

examines the conditions under which it was entered, and passes control as described below:

- If the statement is continued, control is passed to the interpreter get routine, so that the condition may be read.
- If the null statement represents the end of a procedure, but there are additional input stream records to process, control is passed to the verb identification routine, to process the current record from the input stream.
- If there are no more records to be processed in either the input stream or the procedure, control is passed to the job validity check routine, so that the last job can be enqueued.
- If there are no more input stream records to process, but there are additional records in the procedure, control is passed to the router routine.

When the last job has been enqueued, control is passed to the interpreter termination routine.

PROCESSING CONTROL STATEMENTS

When a record containing the characters "//" in the first two positions is read, control is passed to the continuation check routine (module IEFVHC). If the preceding record from the same input source contained a comma in its last non-blank position, the current record is expected to be a continuation of the preceding statement. The continuation routine inspects the current record to determine whether it is blank in position 3, and not blank starting any place from position 4 through position 16, inclusive. If so, control is passed to the pre-scan preparation routine; if not, or if no continuation was expected, control is passed to the verb identification routine.

The verb identification routine (module IEFVHCB) identifies the type of control statement that has been encountered, and processes it as follows:

- If a PROC statement from the input stream is encountered, indicating the beginning of a set of in-stream procedure statements, the verb identification routine passes control to the in-stream procedure routines (see Chart 19). Upon initial entry, the syntax of the PROC verb is checked and a 352-byte work area is obtained. Of this, 176 bytes are used for compression and expansion of the statements within procedures, and the remaining 176 bytes for a procedure directory (see Figure 38). A directory entry, containing the procedure's name and auxiliary storage address, is created for each in-stream procedure within a given job to a maximum of fifteen. Any in excess of this limit causes the job to fail.

  The next JCL statement is read and, unless it is a JOB, PEND, DD *, or DD DATA statement, it is compressed (blanks are removed and a count field added) and placed in a buffer. If the job's message level parameter stipulates the printing of statements, a job queue SMB is built. Printed listings of statements from in-stream procedures and those from cataloged procedures are differentiated by identifications of "++" rather than "XX" for JCL output statements and "+/" rather than "X/" for overridden parameters.

  Another statement is then read and the processing repeated. When the PEND statement, signaling the end of a given in-stream procedure, is read, its syntax is checked and control is transferred to the get routine.

  If a DD * or DD DATA statement is read in the in-stream procedure routine, a bit is set to flush the job, as data is not allowed in such procedures. Control is passed to the get routine.

  When a JOB statement is read within the in-stream procedure routine, control is immediately returned to the verb identification routine.

- If the statement identified by the verb identification routine is EXEC PROC, the in-stream procedure directory is searched. If an entry for the named procedure is found, the address of SYS1.PROCLIB's access method is saved in the IWA while a pseudo access method is used to read the procedure from the job queue and to expand it to its original form. Once expanded, the procedure is processed by the reader/interpreter as if it had originated in SYS1.PROCLIB. Switches are set enabling the interpreter get routine to read a statement from the procedure library or the job queue, and control is passed to the router routine.
- If the statement is a JOB, EXEC, or DD statement, control is passed to the router routine.
- If the statement is a null statement, control is passed to the null statement routine.

- If the statement appears to have a valid format, yet does not have one of the five valid JCL statement operators (JOB, EXEC, PEND, PROC, and DD), and is not a null statement, control is passed to the command routine. The command routine verifies the verb and calls the master command routine.

PROCESSING JOB, EXEC, AND DD STATEMENTS

When the verb identification routine determines that the statement is a JOB, EXEC, or DD statement, it passes control to the router routine (module IEFVHE), which determines whether there are tables from a previous step to be placed in the queue before the current statement can be processed.

If the router is entered with an EXEC statement in the buffer, the tables describing the previous step must be placed in the job's queue entry; control is passed to the job and step enqueue routine.

If the statement in the buffer is a JOB statement, the previous step is the last step of a job. The storage space used for in-stream procedure work and job queue record areas is freed. Control is passed to the validity check routine.

If the statement in the buffer is a DD statement, or if it is an EXEC statement representing the first step in a job, or if it is a JOB statement representing the first job in the input stream, there are no tables to be written into the queue, and control is passed to the pre-scan preparation routine.

The pre-scan preparation routine (module IEFVHEB) is entered when the statement to be processed is a JOB, EXEC, or DD statement. If the statement is a JOB statement, it passes control to the queue manager interface routine, which uses the queue management assign and start routines to start an input queue entry with an assignment of five records.

On the return, the pre-scan preparation routine starts the construction, in main storage, of the JCT and the SCT for the first step of the job, by inserting the queue addresses of the first two records assigned to the job's entry.

The routine then uses the message writing routine to copy the JOB statement into an SMB. If the JOB statement specifies MSGLEVEL=1, the other JCL statements in the job are also placed in SMBs. If the JOB statement does not specify any of its optional parameters, the sysgen default options, placed in the IWA when the interpreter is initialized, are used. The pre-scan preparation routine finally passes control to the JCL scan routine (module IEFVFA), which converts statements to internal text, and passes them to the appropriate processor, so that the tables can be constructed.

QUEUE ENTRY PROCESSING

When the presence of a JOB or null statement in the input stream indicates that the input queue entry describing the previous job is to be enqueued, the job validity check routine is entered. The routine determines whether the job to be enqueued has any steps; if so, control is passed to the job and step enqueue routine. If not, the validity check routine constructs a dummy SCT and sets the job-failed bit on before passing control to the job and step enqueue routine.

The job and step enqueue routine (module IEFVHH) is entered from the router when the presence of an EXEC statement indicates that the tables representing the previous step are to be placed in the input queue, and from the job validity check routine (module IEFVHEC) when the presence of a JOB or null statement indicates that the step was the last step of a job. The job and step enqueue routine inspects switches in the IWA to determine which tables are to be placed in the queue, then passes control to the queue manager interface routine to have each table written to the queue by queue management.

If the step whose tables are to be placed in the queue is the last step in a job, a switch in the IWA indicates that the JCT is to be written. When the tables describing the step have been placed in the queue, the job and step enqueue routine instructs the queue manager interface routine to have the JCT written by the queue manager. An exit is then taken to the interpreter-initiator interface module.

If the step whose tables are to be placed in the queue has a DD * statement, the same exit is taken to the interpreter-initiator interface module.

Otherwise, control is passed to the pre-scan preparation routine, and the statement currently in the buffer is processed.

POST-PROCESSING ENTRY

The control routine is reentered at the post-scan routine (module IEFVHF) from the JCL scan routine if a continuation statement is expected, if the statement scanned was an overriding statement, or if a JCL error was detected. It is entered from a statement processor routine when the processing of a statement is completed. The post-scan routine determines the conditions under which it was entered, then passes control to the appropriate control routine module:

- If a continuation statement is expected, control is passed to the interpreter get routine to read the statement.
- If an overriding statement has been processed by the JCL scan routine, the overridden statement must be scanned before the statement processor routine is entered. The overridden statement is in the buffer, and control is passed to the pre-scan preparation routine.
- If a JCL error was encountered, the job-failed bit has been set on. The remaining statements in the job (except for procedure library statements) will be processed by the interpreter, so that any other errors may be found; but the job will not be run. Control is passed to the interpreter get routine, and processing continues.
- If the statement has been successfully processed, control is passed to the interpreter get routine.
- If the statement processed was a DD * or DD DATA statement, control is passed to the job and step enqueue routine.

Scanning the JCL Statement

The job control language scanning routine (module IEFVFA) converts a JCL record into a coded internal list (see Figure 8). When it has accumulated a complete JCL statement, (including continuations and overrides) it then passes the list to a statement processing routine. An example showing the scanning and encoding of a DD statement follows this section.

Each statement is scanned from left to right. The scan routine recognizes keywords and positional parameters, and is able to identify the existence of a name field and one level of subparameters following a keyword.

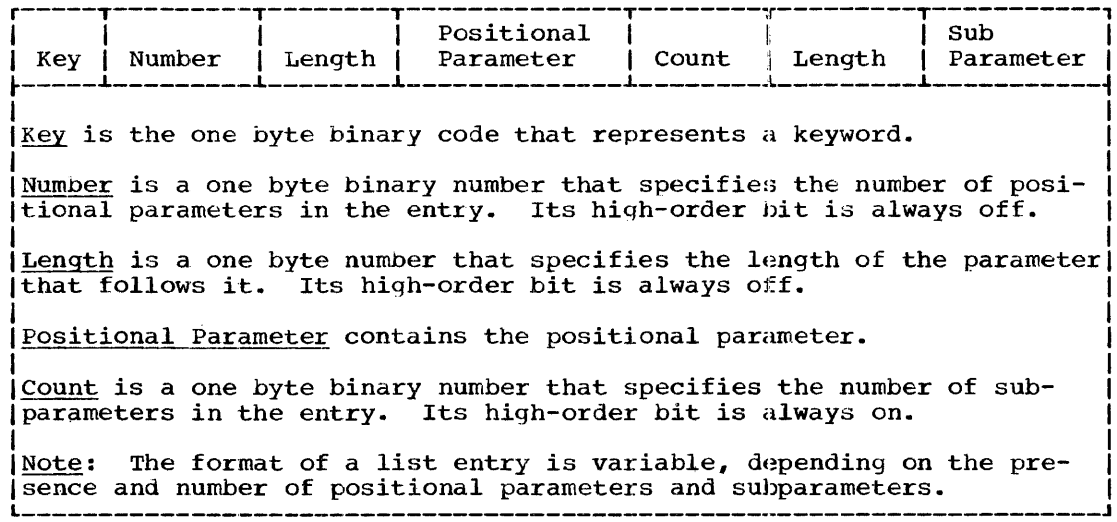


Figure 8. Internal List Entry Format

As the statement is examined, the name field, keywords, and positional parameters are identified and looked up in the scan dictionary (see Figure 9). For each keyword the scan dictionary entry contains the corresponding one-byte binary "key", and lists the keys of any mutually exclusive parameters (the DDNAME and DCB parameters, for example, are mutually exclusive). The entry also lists the keys of any minor keywords associated with the keyword that the entry represents; SEP, for example, is a minor keyword of the UNIT parameter, and is listed as a minor keyword in the UNIT entry of the scan dictionary. The list of mutually exclusive keys is used for error checking and the list of minor keys for overriding major keywords in a cataloged procedure.

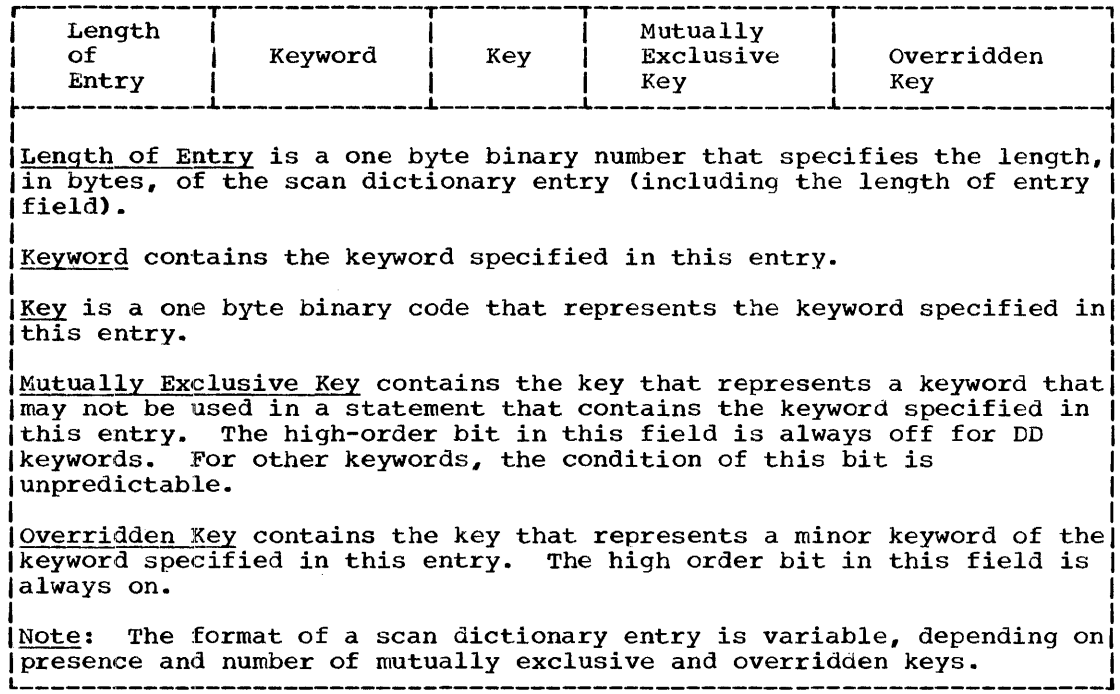


Figure 9. Scan Dictionary Entry Format

When the correct scan dictionary entry has been found, the scan routine determines whether the parameter has been encountered previously, or whether a mutually exclusive parameter has been encountered, by testing the appropriate bits in the duplicate table.

The duplicate table is a 16-byte table that contains a bit for each key. The position of the bit in the table corresponds to the key; the eighth bit in the second byte corresponds to the key X'0F' (the DCB keyword in a DD statement), the first bit in the fourth byte corresponds to the key X'18" (the DDNAME keyword in a DD statement), etc.

When it makes an entry in the internal list, the scan routine turns on the bit that corresponds to the key it is processing. It also turns on the bits that correspond to any mutually exclusive keys, as defined in the scan dictionary entry. Thus, if a bit in the table is on, it means that the key, or a mutually exclusive key, has been encountered previously.

This condition is an error (and the scan routine turns on the job-failure bit and exits) unless the scan routine is processing the procedure library statement. During a procedure merge, the condition means that the field being processed was overridden, and the scan routine proceeds to the next field.

When the scan is complete, control is passed to the appropriate JCL statement processor routine.

Example:

The JCL scan routine encounters the following source statement.

```
//SYSUT1 DD DSNAME=LINKEDIT.WORK,UNIT=190
                SPACE=(TRK,(30,10)),VOLUME=SER=111111
```

1. The name field (SYSUT1) is identified as such because of its position, and encoded as follows:

Key	Number	Length	Parameter
6E	01	06	E2 E8 E2 E4 E3 F1

2. The DSNAME= field is found in the scan dictionary entry shown below:

Length	Keyword	Key	Mutually Exclusive keys	
0B	C4 E2 D5 C1 D4 C5 7E	4A	49	4B

3. It is encoded and placed in the list as shown below:

Key	Number	Length	Parameter
4A	01	0D	D3 C9 D5 D2 C5 C4 C9 E3 4B E6 D6 D9 D2

4. The UNIT= field is found in the scan dictionary entry shown below:

Length	Keyword	Key	Mutually Exclusive Key	Overridden Key	Overridden Key
0A	E4 D5 C9 E3 7E	41	49	CD	CE

5. It is encoded and placed in the list as shown below:

Key	Number	Length	Parameter
41	01	03	F1 F9 F0

6. The SPACE= field is found in the dictionary entry shown below:

Length	Keyword	Key	Mutually Exclusive Keys		
0B	E2 D7 C1 C3 C5 7E	47	48	4C	49

7. It is encoded and placed in the list as shown below:

Key	Number	Length	Parameter	Count	Length	Parameter	Length	Parameter
47	02	03	E3 D9 D2	82	02	F3 F0	02	F1 F0

8. The VOLUME= field is found in the dictionary entry shown below:

Length	Keyword	Key	Mutually Exclusive Keys		Overriden Key	Overriden Key
0D	E5 D6 D3 E4 D4 C5 7E	43	49	4B	CF	D0

9. It is encoded and placed in the list as shown below:

Key	Number
43	00

Since there are no positional parameters associated with the VOLUME keyword, the number field is 00, and it terminates the entry.

10. The serial number field (SER=) is found in the scan dictionary entry shown below:

Length	Keyword	Key	Mututally Exclusive Key
07	E2 C5 D9 7E	4F	50

11. It is encoded and placed in the list as shown below:

Key	Number	Length	Parameter
4F	01	06	F1 F1 F1 F1 F1 F1

12. Since the serial number field is the last field in the statement, the list is closed with the entry:

Key
FE

The scan routine then passes control to DD statement processor routine (IEFVDA).

Processing JCL Statements

When a statement has been scanned, and its contents placed in an internal text buffer, tables must be built from the internal text. This function is performed by the JOB statement processor routine (module IEFVJA), the EXEC statement processor routine (module IEFVEA), and the DD statement processor routine (module IEFVDA). These three routines are similar in construction (see chart 18); each processor consists of a single control section containing a header routine, a keyword routine for each keyword in the statement, and a cleanup routine.

When a statement processor routine is first entered, the header routine performs initializing functions, which include clearing the storage area occupied by the tables to be created by the routine (except for fields filled in by previously executed routines), and initializing the local work area (LWA). It then uses a BALR instruction to pass control to the get parameter routine, which performs basic error checking of a parameter, then passes control to the appropriate keyword routine.

Each keyword routine controls the processing of the positional parameters and subparameters associated with a given keyword. The routine is entered initially when the get parameter routine encounters its keyword, and again as each positional parameter and subparameter is found. In some cases, the required processing is done directly by the keyword routine; in most cases, however, the keyword routine passes control to the test and store routine, which processes the parameter in accordance with the description in the parameter descriptor table (PDT) and returns control to the keyword routine. Control is then passed to the get parameter routine for the next parameter.

When the last parameter in the statement has been processed, or when the test and store routine or get parameter routine finds an error, control is passed to the cleanup portion of the JCL statement processor.

Each cleanup routine uses the message routine to write any error messages to the programmer. In addition, the cleanup routines perform the processing described below:

- The JOB statement processor cleanup routine checks for the presence of programmer name and account number, and uses the queue manager interface routine to write out the job account control table (ACT).

  If the EXEC statement specifies "PROC=", the execute statement processor cleanup routine uses the queue management interface routine to write out the last override table; if the statement was in a procedure, the routine reads the appropriate override table into main storage, and stores overriding information in the SCT.

- The DD statement processor cleanup routine sets initializing values in the JFCB, where no value has previously been set. It marks the disposition fields for implied dispositions, and sets bits to indicate whether the data set is public, private, temporary, or shareable. If the DSNAME keyword was omitted, or if its parameter is "&", the routine generates a data set name. It uses the queue manager interface routine to assign records in the queue for the SIOT and JFCB (unless the DDNAME or SYSOUT keyword was used in the statement), then writes the SIOT and JFCB into the assigned records. If the DDNAME keyword was used, the records have previously been assigned, and the JFCB and SIOT need only to be written out. If the SYSOUT keyword was used, the routine passes control to the interpreter system output routine.

JOB, EXEC and DD statement parameter dispositions are shown in Figures 10, 11, and 12.

If the system includes Main Storage Hierarchy Support, selective access is permitted either to hierarchy 0 or to hierarchy 1 portions of main storage. The interpreter processes the HIARCHY subparameter of the DD statement DCB parameter. If Main Storage Hierarchy Support is not included in the system, requests for storage within hierarchy 1 are treated exactly the same as normal requests for main storage.

When a cleanup routine has completed its processing, it passes control to the interpreter routine, at the post-scan routine.

Recognizing Checkpoint Restart

When a restart is to occur, the JOB statement processor routine (IEFVJA) recognizes the RESTART keyword. If the CHKID subparameter is present, the restart is a checkpoint restart, and CHKID is saved in the JCT. If the CHKID subparameter is not present, the restart is a step restart.

During control statement processing, module IEFVHCB tests for the CHKID parameter in the JCT. When the parameter is present (checkpoint restart), the pre-scan routine (IEFVHEB) initializes job step IEFD3DRP.

The execute card scan routine (IEFVEA) indicates which step will be the first to be executed in a restarted job. In the case of a checkpoint restart, IEFDSDRP will be the first step to be executed. In the case of a step restart, the step to be restarted will be the first to be executed.

JOB Statement Parameter	Table	Table Item
jobname	JCT	Jobname
account number	ACT	Account number, length of account number
programmer's name	ACT	Programmer's name
TYPRUN	Ignored in primary control program	
PRTY	Ignored in primary control program	
COND	JCT	Code, operator
MSGLEVEL	JCT	Message level
MSGCLASS	Ignored in primary control program	
REGION	Ignored in primary control program	
CLASS	Unused	
ROLL	Unused	

Figure 10. JOB Statement Parameter Dispositions

EXEC Statement Parameter	Table	Table Item
stepname	SCT	Stepname
PGM	SCT	Programname
PROC	Cataloged control statements are interpreted and merged with input statements.	
TIME	Ignored in the primary control program	
COND	SCT	Code, operator, auxiliary storage address of referenced SCT
PARM	SCT	Initializing parameter values
ACCT	ACT	Step accounting fields
REGION	Ignored in primary control program	
DPRTY	Unused	
ROLL	Unused	

Figure 11. EXEC Statement Parameter Dispositions

DD Statement Parameter	Table	Table Item	Bit(s)
AFF=	SIOT	SCTCSADD	
	SIOT	SCTSBYT2	0
*, DATA	SIOT	SCTUTYPE	
	SIOT	SCTSBYT1	1
	SIOT	SCTSDISP	5
	SIOT	SCTSBYT3	7
	SCT	SCTSTYPE	
	JFCB	JFCBTSDM	2
	JFCB	JFCBIND2	1
	JFCB	JFCBDSNM	
COPIES=	SIOT	SIOTOUTC	
DCB=			
dsname	SIOT	SIOTDCBR	
BFALN=D	JFCB	JFCBFALN	6
BFALN=F	JFCB	JFCBFALN	7
BFTEK=A	JFCB	JFCBFTEK	1,2
BFTEK=B	JFCB	JFCBFTEK	0
BFTEK=D	JFCB	JFCBFTEK	4
BFTEK=E	JFCB	JFCBFTEK	3
BFTEK=R	JFCB	JFCBFTEK	2
BFTEK=S	JFCB	JFCBFTEK	1
BLKSIZE	JFCB	JFCBLKSI	
BUFL	JFCB	JFCBUFL	
BUFNO	JFCB	JFCBUFNO	
BUFOFF	JFCB	JFCBUFOF	
BUFRQ	JFCB	JFCBUFRQ	
CODE=A	JFCB	JFCCODE	5
CODE=B	JFCB	JFCCODE	3
CODE=C	JFCB	JFCCODE	4
CODE=F	JFCB	JFCCODE	2
CODE=I	JFCB	JFCCODE	1
CODE=N	JFCB	JFCCODE	0
CODE=T	JFCB	JFCCODE	6
CPRI=E	JFCB	JFCCPRI	6
CPRI=R	JFCB	JFCCPRI	5
CPRI=S	JFCB	JFCCPRI	7
CYLOFL	JFCB	JFCCYLOF	
DBUFNO	JFCB	JFCDBUFN	
DEN=0	JFCB	JFCDEN	6,7
DEN=1	JFCB	JFCDEN	1,6,7
DEN=2	JFCB	JFCDEN	1,3,6,7
DEN=3	JFCB	JFCDEN	0,1,6,7
DSORG=CQ	JFCB	JFCDSORG	4
DSORG=CX	JFCB	JFCDSORG	3
DSORG=DA	JFCB	JFCDSORG	2
DSORG=DAU	JFCB	JFCDSORG	2,7
DSORG=IS	JFCB	JFCDSORG	0
DSORG=ISU	JFCB	JFCDSORG	0,7
DSORG=MQ	JFCB	JFCDSORG	5
DSORG=PO	JFCB	JFCDSORG	6
DSORG=POU	JFCB	JFCDSORG	6,7
DSORG=PS	JFCB	JFCDSORG	1
DSORG=PSU	JFCB	JFCDSORG	1,7
EROPT=ABE	JFCB	JFCEROPT	2
EROPT=ACC	JFCB	JFCEROPT	0
EROPT=CLE	JFCB	JFCEROPT	4
EROPT=SKP	JFCB	JFCEROPT	1
GDSORG	JFCB	JGDSORG1	0
GNCP	JCFB	JFCBFTEK	
HIARCHY=0	JFCB	JFCBFTEK	none
HIARCHY=1	JFCB	JFCBFTEK	5
INTVL	JFCB	JFCINTVL	

Figure 12. DD Statement Parameter Dispositions (Part 1 of 4)

DD Statement Parameter	Table	Table Item	Bit(s)
KEYLEN	JFCB	JFCKEYLE	
LIMCT	JFCB	JFCLIMCT	
LRECL	JFCB	JFCLRECL	
MODE=C	JFCB	JFCMODE	0
MODE=E	JFCB	JFCMODE	1
NCP	JFCB	JFCNCP	
NTM	JFCB	JFCNTM	
OPTCD=A	JFCB	JFCOPTCD	4
OPTCD=B	JFCB	JFCOPTCD	1
OPTCD=C	JFCB	JFCOPTCD	2
OPTCD=E	JFCB	JFCOPTCD	2
OPTCD=F	JFCB	JFCOPTCD	3
OPTCD=H	JFCB	JFCOPTCD	3
OPTCD=I	JFCB	JFCOPTCD	3
OPTCD=L	JFCB	JFCOPTCD	6
OPTCD=M	JFCB	JFCOPTCD	2
OPTCD=O	JFCB	JFCOPTCD	3
OPTCD=P	JFCB	JFCOPTCD	2
OPTCD=Q	JFCB	JFCOPTCD	4
OPTCD=R	JFCB	JFCOPTCD	7
OPTCD=T	JFCB	JFCOPTCD	6
OPTCD=U	JFCB	JFCOPTCD	1
OPTCD=W	JFCB	JFCOPTCD	0
OPTCD=Y	JFCB	JFCOPTCD	4
OPTCD=Z	JFCB	JFCOPTCD	5
PRTSP=0	JFCB	JFCPRTSP	7
PRTSP=1	JFCB	JFCPRTSP	4,7
PRTSP=2	JFCB	JFCPRTSP	3,7
PRTSP=3	JFCB	JFCPRTSP	3,4,7
RECFM=A	JFCB	JFCRECFM	5
RECFM=B	JFCB	JFCRECFM	3
RECFM=D	JFCB	JFCRECFM	2
RECFM=F	JFCB	JFCRECFM	0
RECFM=G	JFCB	JFCRECFM	5
RECFM=K	JFCB	JFCRECFM	7
RECFM=M	JFCB	JFCRECFM	6
RECFM=R	JFCB	JFCRECFM	6
RECFM=S	JFCB	JFCRECFM	4
RECFM=T	JFCB	JFCRECFM	2
RECFM=U	JFCB	JFCRECFM	0,1
RECFM=V	JFCB	JFCRECFM	1
RETPD	DDWA	DDETPD	
RKP	JFCB	JFCRKP	
SOWA	JFCB	JFCSOWA	
STACK	JFCB	JFCSTACK	6,7
TRTCH=C	JFCB	JFCTRTCH	3,6,7
TRTCH=E	JFCB	JFCTRTCH	2,6,7
TRTCH=ET	JFCB	JFCTRTCH	2,4,6,7
TRTCH=T	JFCB	JFCTRTCH	2,3,4,6,7
TRTCH=TE	JFCB	JFCTRTCH	2,4,6,7
DISP=	JFCB	JFCBIND2	
CATLG	SIOT	SCTSDISP	6
DELETE	SIOT	SCTSDISP	5
KEEP	SIOT	SCTSDISP	4
MOD	JFCB	JFCBIND2	0
	SIOT	SCTSBYT3	6
NEW	JFCB	JFCBIND2	0,1
	SIOT	SCTSBYT3	5
OLD	JFCB	JFCBIND2	1
	SIOT	SCTSBYT3	7
PASS	SIOT	SCTSDISP	3
SHR,SHARE	JFCB	JFCBIND2	1,4

Figure 12. DD Statement Parameter Dispositions (Part 2 of 4)

DD Statement Parameter	Table	Table Item	Bit(s)
	SIOT	SCTSBYT3	7
UNCATLG (conditional disposition)	SIOT	SCTSDISP	7
CATLG	SIOT	SIOTALTD	6
DELETE	SIOT	SIOTALTD	5
KEEP	SIOT	SIOTALTD	4
UNCATLG	SIOT	SIOTALTD	7
DSNAME, DSN=	SIOT	SCTSBYT4	0
	JFCB	JFCBDSNM	
	JFCB	JFCBELNM	
	JFCB	JFCBIND1	6,7
	JFCB	JFCBIND2	7
DUMMY, DDNAME=	SIOT	SCTSBYT1	0
	JFCB	JFCBDSNM	
LABEL=			
AL	JFCB	JFCBLTYP	6
	SIOT	SCTSBYT4	3
AUL	JFCB	JFCBLTYP	6
	SIOT	SCTSBYT4	3
BLP	JFCB	JFCBLTYP	3
	SIOT	SCTSBYT2	4
data set sequence no.	JFCB	JFCBFLSQ	
EXPDT	JFCB	JFCBCRDT	
	JFCB	JFCBXPDT	
IN	JFCB	JFCBMASK (byte 6)	0
NL	JFCB	JFCBLTYP	7
	SIOT	SCTSBYT2	4
NSL	JFCB	JFCBLTYP	5
	SIOT	SCTSBYT2	5
OUT	JFCB	JFCBMASK (byte 6)	1
PASSWORD	JFCB	JFCBIND2	2,3
RETPD	JFCB	JFCBCRDT	
	JFCB	JFCBXPDT	
	DDWA	DDETPD	
SL	JFCB	JFCBLTYP	6
SUL	JFCB	JFCBLTYP	4
OUTLIM=	JFCB	JFCOUTLI	
PATTERN=	SIOT	SIOTOUTR	
SEP=	SIOT	SCTCSADD	
	SIOT	SCTSBYT2	1
SPACE=			
ABSTR	DDWA	ABSTRZ	7
ALX	JFCB	JFCBCTRI	6
average record length	JFCB	JFCBCTRI	1
	JFCB	JFCBDRLH	
beginning address	JFCB	JFCBABST	
CONTIG	JFCB	JFCBCTRI	4
CYL	JFCB	JFCBCTRI	0,1
directory quantity	JFCB	JFCBDQTY	
MXIG	JFCB	JFCBCTRI	5
primary quantity	JFCB	JFCBPQTY	
RLSE	JFCB	JFCBIND1	0,1
ROUND	JFCB	JFCBCTRI	7
secondary quantity	JFCB	JFCBSQTY	
TRK	JFCB	JFCBCTRI	0
SPLIT=	SIOT	SCTSBYT1	2,3
average record	JFCB	JFCBCTRI	1

Figure 12. DD Statement Parameter Dispositions (Part 3 of 4)

DD Statement Parameter	Table	Table Item	Bit(s)
length			
CYL	JFCB	JFCBCTRI	0,1
directory quantity	JFCB	JFCBDQTY	
n	JFCB	JFCBSPTN	
primary quantity	JFCB	JFCBPQTY	
secondary quantity	JFCB	JFCBSQTY	
SUBALLOC=	SIOT	SCTSBYT1	4
average record length	JFCB	JFCBCTRI	1
CYL	JFCB	JFCBCTRI	0,1
ddname	SIOT	SIOTVRSB	
	SIOT	SCTSBYT3	3
directory quantity	JFCB	JFCBDQTY	
primary quantity	JFCB	JFCBPQTY	
secondary quantity	JFCB	JFCBSQTY	
stepname.ddname	SIOT	SIOTVRSB	
TRK	JFCB	JFCBCTRI	0
SYSOUT=	JFCB	JFCBDSNM	
	JFCB	JFCBTSDM	2
	JFCB	JFCBLTYP	6
	JFCB	JFCBVLCT	7
	SIOT	SCTSBYT3	4
	SIOT	SCTSBYT1	0
classname	SIOT	SCTOUTPN	
form number	SIOT	SCTOUTNO	
progname	SIOT	SCTOUTNM	
UCS=			
FOLD	JFCB	JFCINTVL	1
VERIFY	JFCB	JFCINTVL	3
UNIT=			
AFF=(minor)	SIOT	SCTUSADD	
	SIOT	SCTSBYT1	6
DEFER	SIOT	SCTSBYT2	6
n	SIOT	SCTNMBUT	
name	SIOT	SCTUTYPE	
P	SIOT	SCTSBYT1	5
POOL	none		
poolname	SIOT	SCTSPOOL	
SEP=(minor)	SIOT	SCTUSADD	
	SIOT	SCTSBYT1	7
0	SIOT	SCTNMBUT	
1	SIOT	SCTNMBUT	
VOLUME=			
PRIVATE	SIOT	SCTSDISP	2
RETAIN	SIOT	SCTSDISP	1
SER=	JFCB	JFCBNVOL	
	JFCB	JFCBEXAD	
	JFCB	JFCBVOLS	
	SCT	SCTVOLTB	
	SCT	SCTVOLTL	
	SIOT	SCTVOLCT	
	SIOT	SCTVLTPR	
	VOLT	INDMVOLT	
volume count	JFCB	JFCBVLCT	
volume sequence no.	JFCB	JFCBVLSQ	
REF=	dsname	INDMDSNT	
	SCT	SCTADSTB	
	SCT	SCTLDSTB	
	SIOT	SCTVLPTR	
	SIOT	SCTVOLCT	
	SIOT	SIOTVRSB	
	SIOT	SCTSBYT2	2
	SIOT	SCTSBYT3	0

Figure 12. DD Statement Parameter Dispositions (Part 4 of 4)

Auxiliary Routines

During the performance of the reading task, the interpreter routines must frequently perform functions common to several routines. These common functions are performed by a set of auxiliary routines, which are described below:

- The get parameter routine (module IEFVGK) is used by the statement processor routines. It searches for the next parameter in a statement, performs basic error checking, and passes control to the proper keyword routine, with a pointer to the parameter.
- The test and store routine (module IEFVGT) is used by the statement processor routines. It processes the parameter as described in the parameter descriptor table (PDT) and passes control back to the keyword routine.
- The dictionary entrance routine (module IEFVGI) is used by the statement processor routines. It makes entries for the dictionary used in refer-back processing.
- The dictionary search routine is used by the statement processor routines. It searches the refer-back dictionary during refer-back processing.
- The message routine (module IEFVGM) stores messages in system message blocks (SMBs) for transmittal to the programmer.
- The queue manager interface routine (module IEFVHQ) is used by those interpreter routines that reserve space, write records in, or read records from the queues.

THE GET PARAMETER ROUTINE

The get parameter routine (module IEFVGK) is an auxiliary routine used by the JCL statement processor routines to find the next parameter in a statement, perform basic error checking of that parameter, and find and pass control to the appropriate keyword routine with pointers to the parameter and to the appropriate parameter descriptor table (PDT) entry.

When the get parameter routine is initially entered, the only non-zero portion of the auxiliary work area (AWA) is the address of the keyword branch table (KBT). The KBT (Figure 13) is a table of offsets that allows the get parameter routine to determine the actual main storage address of the appropriate keyword routine and PDT entry. Additional fields in the table allow basic error checking to be done.

When the get parameter routine is entered to find the first parameter in a new statement, it extracts the base key (the key number that represents JOB, EXEC, or DD) from the text buffer and stores it. The base key is the offset of the last entry in the table from the first entry. Whenever the routine is entered, it subtracts the current key from the base key, multiplies the result by 6 (the size of an entry), and adds the product to the machine address of the first entry in the table. The result is the machine address of the KBT entry corresponding to the current keyword.

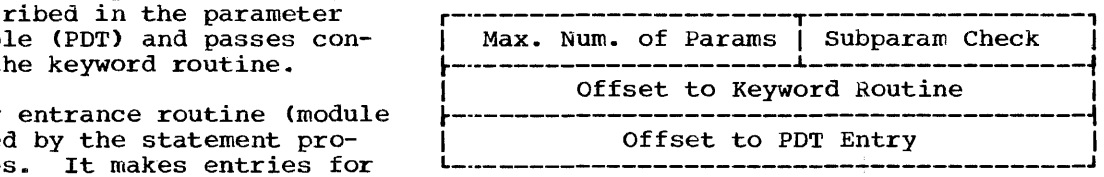


Figure 13. Keyword Branch Table Entry

The get parameter routine first finds the proper KBT entry, then determines whether the maximum number of parameters for the keyword has been exceeded, and stores the subparameter check byte in the AWA. Each bit in the subparameter check byte corresponds to a positional parameter; if the bit is on, it means that the corresponding parameter may have subparameters associated with it. For example, if the first positional parameter associated with a keyword were the only one that could consist of a subparameter list, the high-order bit in the field would be on. If the seventh and eighth positional parameters could have subparameters, the two low-order bits would be on.

The two offset fields are used to compute the actual main storage address of the appropriate keyword routine and of the appropriate PDT entry; the positional parameter length, the parameter length byte address (in the internal text buffer), and the PDT entry address are placed in general registers, and control is passed to the keyword routine.

On subsequent entries to the routine, the pointers are updated so that they point to the next operand (positional parameter or subparameter), and control is returned to the keyword routine at the instruction after the branch to the get parameter routine. When the next keyword is encountered, however, the branch table is again used, and control is passed to a new keyword routine.

THE TEST AND STORE ROUTINE

The test and store routine (module IEFVGT) is an auxiliary routine used by the JCL routines to determine the processing required for a parameter (as described in the PDT), and to perform that processing. When processing of a keyword is complete, control is returned to the appropriate keyword routine.

The parameter descriptor table (Figure 14) included in each JCL processor describes the processing to be done for each parameter that may be found in the statement. There is an entry for each keyword, which begins with a field containing the length of the keyword entry. The keyword entry is made up of positional parameter entries describing the processing to be done on the positional parameters associated with the keyword.

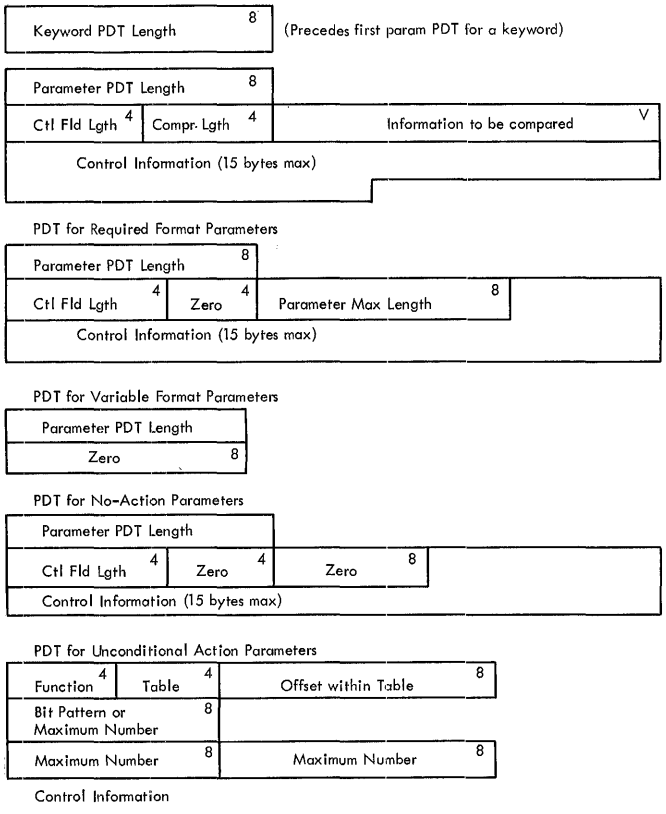


Figure 14. Parameter Descriptor Table (PDT)

Each parameter entry contains two kinds of information. Length and error checking information is followed by control information, which describes the functions to be performed on the parameter, and the location in which the result is to be stored.

The first byte in each parameter entry (the parameter PDT length field) contains the length of the entry; the first half of the second byte (the control field length field) contains the length of the control information. The format of the remainder of the entry depends on the type of parameter and on the functions to be performed. There are four types of parameters:

- A required-format parameter is a known string of characters. The first positional parameter following the DISP= keyword, for example, must be either "OLD", "NEW", "MOD", or "SHARE". In this case, since there are four possibilities, there are four parts to the entry; the test and store routine compares the parameter to the constant in each of the four parts, and performs the function specified in the control information field of the part in which it obtained an equal compare.

- A variable-format parameter may be any string of characters up to a known maximum length. The classname parameter of the SYSOUT keyword is an example; since there are 36 system output class names permitted in the system, a series of comparisons would be unwieldy. The compare length byte in such an entry is zero; the third byte in the parameter entry specifies the maximum number of digits allowed.

- A no-action parameter is one that refers the system to bit configurations established when the system is generated. These bits specify a default option that the system may use without taking action to reset any bits. For example, the applications programmer may omit the COND keyword, in which case the system uses the default option and makes no return code tests.

- An unconditional-action parameter indicates that the presence of the parameter requires that the same functions be performed regardless of the form or contents of the parameter. When the SPACE keyword is encountered, for example, certain switches must be set, regardless of how much or what kind of space has been requested.

The control information portion of a parameter PDT entry defines the operations to be performed when the parameter is processed, specifies the location in which the results are to be stored, and may contain data to be used in the operation. The control information portion may be up to 15 bytes in length; it consists of the following fields:

- Function: The first four bits of a control information field contain a number from 0 to 7, which specifies one of the following operations:

- OR (Code 0): A logical OR operation is performed, using the bit pattern field in the control information portion of the entry, against the bit pattern at the location specified by the table and offset fields.

- CVB1 (Code 1): A convert to binary operation is performed and a maximum value check is made. The converted information is stored (right justified) in the one-byte field specified by the table and offset fields, and compared against the maximum value, which is right-justified in the third byte of the control information part of this entry.

- CVB2 (Code 2): This operation is similar to CVB1, except that the result is right-justified in a two-byte field, and the maximum value is found right-justified in the fourth byte of the control information portion of the entry.

- CVB3 (Code 3): This operation is similar to the CVB1 and CVB2 operations, except that the result is right-justified in a three-byte field, and the maximum value is found in the fifth byte of the control information portion of the entry.

- AND (Code 4): A logical AND operation is performed, using the bit pattern field in the control information portion of the entry against the bit pattern at the location specified by the table and offset fields.

- MVC (Code 5): A move characters operation is performed, using the parameter length specification in the internal text buffer. The parameter is moved to the location specified in the table and offset fields in the entry.

- First Character Alpha Check and MVC (Code 6): This function is similar to the MVC function, except that the first character of the parameter is inspected to determine that it is alphabetic.

- Alpha/Numeric Check (Code 7): A character (usually a one character parameter) in the text buffer is inspected to determine that it is alphabetic.

- Table: The second four bits of the control information portion of a parameter PDT entry contain a number between 0 and 15 that specifies the table in which the result of the operation is to be stored.

- Offset: The second byte of the control information of an entry contains the offset, from the beginning of the table, of the field in which the results of the operation are to be stored.

- Bit-pattern/Maximum Number: The third through fifth bytes of the control information portion of the entry are used for those operations that require data for logical or comparison functions. If the operation is AND or OR, the third byte contains the bit pattern. If the operation is a CVB operation, the third, fourth and fifth bytes contain the binary representation of the maximum value allowed for that parameter.

THE DICTIONARY ENTRY ROUTINE

The dictionary entry routine (module IEFVGI) is used by the EXEC statement processor routine and the DD statement processor routines to place an entry in the refer-back dictionary. The dictionary is maintained in the IWA; if the number of entries exceeds five, a copy of the dictionary is written out to the queue, a new dictionary is initialized in the IWA, and the new dictionary is chained to the previous copy in the queue.

THE DICTIONARY SEARCH ROUTINE

The dictionary search routine (module IEFVGS) is used by the EXEC and DD statement processor routines to search the refer-back dictionary for the address of a previously defined SCT, SIOT, or JFCB. It returns control to the calling routine with a pointer to the required table.

THE INTERPRETER MESSAGE ROUTINE

The interpreter message routine (module IEFVGM) is used by the interpreter control routine and JCL statement processor routines when a JCL statement or diagnostic message must be placed in an SMB, and to enqueue SMBs for each job.

THE QUEUE MANAGER INTERFACE ROUTINE

The queue manager interface routine (module IEFVHQ) is used by those interpreter routines that need to assign space, and to read and write records in the queue. It provides a queue manager parameter area, and passes control to the queue manager to

perform the function specified by the calling routine. On the return from the queue manager, it resets the parameter area so that it specifies an assign and write 1 record operation, and returns control to the caller.

Interpreter Termination

At end-of-data in the input stream, or when the interpreter determines that a START RDR command has been issued, control is passed to the interpreter termination routine (module IEFVHN). This routine obtains main storage for the interpreter entrance list (NEL), stores a pointer to the command scheduling control block (CSCB), and if the input stream is an internal input stream it also stores a pointer to the queue manager parameter area and the JCT. If the input stream is on external storage, it closes the input stream data set. In either case, it closes the procedure library PDS, and releases the main storage obtained for the two DCBs, the IWA and the LWA. When processing is complete, it returns control to its caller.

Initiator/Terminator

The initiator/terminator (Chart 14) ensures that all I/O resources needed by a job step are available before control is passed to the step. The initiator/ terminator analyzes the I/O device requirements of job steps and allocates devices to them. If necessary, it issues mounting instructions and verifies that volumes were mounted on the correct units.

Control is passed to the initiator/ terminator from:

- The interpreter, when the interpreter encounters a second JOB statement, a DD *, DD DATA, or null statement, or an EOF in the input job stream.
- The supervisor, following step execution.

The initiator/terminator passes control to:

- The job step, when all I/O devices needed by the step have been assigned and the step is ready for execution.
- The interpreter, when termination procedures have been completed for a step or job.

Initiator/terminator routines are arranged into four groupings:

- Initiator control
- Allocation and setup
- Step initiation
- Termination

Initiator control routines perform housekeeping functions, analyze condition codes specified by the programmer in the EXEC statement, and update JFCBs and other tables associated with the step.

Allocation and setup routines analyze a step's I/O requirements (taking into consideration, for example, requests for absolute assignments and unit and volume affinity). They then allocate devices and issue messages instructing the operator to mount required volumes.

Step initiation routines open the job library or step library data set if the JOBLIB or STEPLIB DD statements are present. Also, if the step being initiated consists of a program that was created by a previous step (commonly known as "compile, load, and go"), a step initiation routine opens the data set containing the program. Before passing control to the job step, a step initiation routine takes several preparatory steps. It loads control information that followed the PARM keyword of the EXEC statement into main storage. It also uses the table store subroutine to store all tables associated with the job step, thereby protecting them for use by the termination routines. It initializes the write-to-programmer control block (WTPCB) (see Figure 46) for the processing program. If the initiation is for the first step of the job, two SYS1.SYSJOBQE SMBs are reserved for use in processing WTP error conditions. If an automatic checkpoint restart is in progress, the WTP messages previously written to the message queue are retrieved, using information from the step control table (see Figure 43) to rebuild the WTPCB. The information, gleaned from the WTPCRSMB and WTPCRCNT fields of the WTPCB, is stored in the SCT prior to termination of the original job step.

Termination routines are entered after each job step is executed. They supervise entry to the user's accounting routine (if one exists) and, upon return, dispose of data sets referenced by the step during execution and release devices allocated to the step.

Information is passed between initiator/ terminator routines by means of the linkage control table (LCT) (see Figure 15). The LCT is built and initialized during IPL. It is stored before processing program execution and, following execution, is retrieved by initiator/terminator termination routines. The beginning address of the LCT is maintained in general register 12 during execution of the initiator/ terminator.

Offset Hex	Offset Dec		
0	0	Length of LCT	4
4	4	Address of I/O supervisor UCB lookup table	4
8	8	Address of TCB	4
C	12	Not used in the primary control program	4
10	16	Main Storage Address of JCT	4
14	20	Main Storage Address of SCT	4
18	24	Auxiliary Storage Address of SCT	4
1C	28	Not used in the primary control program	4
20	32	Error code	4
24	36	Parameter 1	4
28	40	Parameter 2	4
2C	44	Parameter 3	4
30	48	Parameter 4	4
34	52	Address of register save area	4
38	56	Reserved (1) / JFCB hsk indicators (1) / Current step no. (1) / Action code (1)	
3C	60	Address of current system message block (SMB)	4

• Figure 15. Linkage Control Table

Initiator Control

Initiator control (Chart 22) performs certain housekeeping functions for the initiator/terminator, and also checks EXEC statement condition codes (if any). Condition codes appearing in EXEC statements determine whether or not a job step is to be executed.

Routines that comprise initiator control are:

- System control routine, which is the entry point for the initiator/terminator. Control is passed to the initiator/terminator when a step is ready for initiation and also after one has been executed and terminated, if another step is to be initiated. Housekeeping is performed and control is passed to the execute statement conditional execution routine.
- Execute statement conditional execution routine, which checks any dependencies encountered in EXEC statements.
- JFCB housekeeping routines, which complete portions of JFCBs and SIOTs that describe the volumes to be used during step execution. These routines also construct a passed data set queue (PDQ) to describe data sets being passed and update the PDQ for data sets being received by the step being processed.

SYSTEM CONTROL ROUTINE

The system control routine (Chart 23) is entered from the interpreter when it completes the processing of a step that was followed by data in the input job stream, or when it reads the last step of a job. It is also entered from the step termination routine if additional steps remain to be initiated.

Upon entry, the system control routine updates the step number in the LCT. Then, if the step is the first step of the job, its job name is placed into the selected job queue. (See Figure 16.)

Jobname	Cancel ECB

Figure 16. Selected Job Queue

If the step being processed is the first step of the job, and if a DISPLAY JOBNAMES command has been issued, the WTO macro instruction is used to write the message:

IEF401I jobname STARTED

on the control output device. If the job being processed is restarting, the system control routine restores saved data from the CVT and sets the restart switches.

In the case of a JCL error or an allocate error in the first step, the WTO macro instruction is used to write the message:

IEF452I jobname JOB NOT RUN - JCL ERROR

on the console output device. Control is then passed to the executed statement condition code routine.

EXECUTE STATEMENT CONDITIONAL EXECUTION ROUTINE

The execute statement conditional execution routine (module IEFVKIMP) checks the keyword COND parameter of the execute statement to determine whether or not the current step should be executed.

To determine when and if it should pass control to the JFCB housekeeping routines for step execution, the execute statement conditional execution routine (Chart 24) first determines that no abnormal terminations have occurred in the previous steps, then sees if either of the following conditions is also true:

- The step is the first step of the job and the programmer did not specify the COND=ONLY parameter.
- The programmer did not specify either any return code tests or the COND=ONLY parameter.

Otherwise the routine then tests for abnormal terminations and for up to eight return codes from previous steps before it determines the proper disposition of the step from the coding of its execute statement and all the pertinent environmental factors. Figure 17 summarizes the conditions that can exist and, corresponding to each condition, whether or not the step will be executed.

	1	2	3	4	5	6	7	8
COND parameter omitted	X							X
COND = ONLY specified		X					X	
COND = EVEN specified			X		X			
COND parameter satisfied by return codes						X		
COND parameter NOT satisfied by return codes				X				
Prior step abnormally terminated *		X	X					X
No previous abnormal terminations	X				X		X	
Step will execute	X	X	X	X	X			
Step will NOT execute						X	X	X

* A special case of a previous abnormal termination is that indicated by a System 806 message code (problem program not found).

Figure 17. Execute Statement COND Parameter Options

* A special case of a previous abnormal termination is that indicated by a System 806 message code (problem program not found).

The execute statement conditional execution routine first tests the job control table's abnormal termination indicator (JCTABEND bit in the JCT) to see whether one or more prior steps have terminated abnormally during execution of the problem program. If none have, the routine tests the SCTONLY bit in the SCTABCND field of the current SCT to see whether the programmer specified the COND=ONLY parameter. If he did, the routine writes a message to the system output device, and the job scheduler bypasses the step. (See column 7 in the table.) However, if one or more abnormal terminations have occurred, the routine tests the SCTONLY and SCTEVEN bits of the SCTABCND field to see whether the programmer specified either the COND=ONLY or the COND=EVEN parameters. If neither bit is on, the job scheduler bypasses the step. (See column 8. These circumstances produce the default situation wherein a step whose execute statement does not specify either the COND=ONLY or the COND=EVEN parameter is failed after one or more abnormal terminations in the job.) If either bit is on, however, the routine makes any return code tests specified in the COND parameter. The routine passes control directly to the JFCB housekeeping routines when the COND parameter has been omitted and no previous abnormal terminations have occurred. (See column 1.)

When the programmer has specified return code tests, the execute statement conditional execution routine uses the queue management read routine to read in the SCTs of the specified steps. (For this read operation the queue manager uses TTRs saved in the current step at interpretation time.) The first return code that satisfies a set of test conditions delineated by the COND parameter causes: 1) the routine to send a message to the system output device; and 2) the job scheduler to bypass the current step. (See column 6.)

To cause the job scheduler to bypass this step (but not necessarily the succeeding steps of the job), the execute statement conditional execution routine places a special error code into the LCTERROR field of the LCT and passes control to the step termination control routine.

JFCB HOUSEKEEPING ROUTINES

The JFCB housekeeping routines (Chart 25) complete volume information within certain tables, in preparation for their use by allocation routines. This information is generally the type that requires reference to the catalog (use of the LOCATE and OBTAIN macro instructions) or to passed data sets. Tables in which entries are made include:

- Job file control block.
- Step input/output table.
- Step control table.
- Volume table.

If it is discovered as a result of a reference to the system catalog through the LOCATE macro instruction that the required control volume is not mounted, a new load module, IEFMCVOL, will be brought into main storage. This load module creates the tables required by the allocation routines to allocate a device for the required control volume, and requests the operator to mount the volume before other requests for the step are satisfied. If the allocation for the control volume is successful, control returns to the JFCB housekeeping routine, where the LOCATE macro instruction is reissued with the control volume mounted.

For passed data sets, a PDQ is constructed and entries are made for the first occurrence of each data set being passed to a subsequent step. The existing data set queue entries are then updated when a data set is received from a previous step.

The JFCB housekeeping routines include the following:

- JFCB housekeeping control routine.
- Allocate processing routine.
- Fetch DCB processing routine.
- GDG single processing routine.
- GDG all processing routine.
- Patterning DSCB processing routine.
- Error message processing routine.

JFCB Housekeeping Control Routine

The JFCB housekeeping control routine (Chart 26) determines what processing (if any) is required, and directs control to the first appropriate processing routine. Upon return of control, it redirects control to the next required processing routine. This routine places each SIOT for a job step into a main storage work area, examines it, and, depending on the type of information required, passes control to the processing routine which performs the actions necessary to retrieve the required information.

When all SIOTs for a job step have been examined, the JFCB housekeeping control routine passes control to the allocation and setup function of the initiator/terminator.

Allocate Processing Routine

The allocate processing routine (Chart 28) completes information about data sets which reference another data set by data set name (indicating a passed or cataloged data set) or by ddname or stepname.ddname (indicating a data set described in a previously processed DD statement).

When the data set reference is a data set name, the passed data set queue is examined and, if it contains an entry for the referenced data set, the SIOT and JFCB for that data set are placed into a main storage work area and are used to complete device and volume information for the subject data set.

If there is no entry for the referenced data set in the PDQ, a LOCATE macro instruction is issued to find that data set in the catalog. Its volume control block or data set pointer entry is then used to complete the volume and device information for the subject data set.

When the data set reference is by ddname or stepname.ddname, a check is made to determine if the DD statement appeared in the step being processed. If so, the SIOT and JFCB associated with the referenced DD statement are placed into a main storage work area. These are used to complete the device and volume information of the subject data set.

If the DD statement appeared in a previous step of the job being processed, the SIOT and JFCBs constructed by the last step to reference the data set are placed into a main storage work area and are used to complete the volume and device information of the subject data set.

When a unit name is specified in the DD statement, the unit name is converted to unit type, through use of the device name table. This table is loaded from SYS1. LINKLIB and is deleted when unit name conversion is complete.

Fetch DCB Processing Routine

The fetch DCB processing routine (Chart 29) completes volume and device information when the data set referred to contains a program that was created in a previous step and is to be executed as the current step.

GDG Single Processing Routine

The GDG single processing routine(Chart 30) obtains the data set name of a generation data group (GDG) member and completes volume and device information entries for that member.

GDG All Processing Routine

The GDG all processing routine (Chart 31) builds an SIOT, JFCB, volume table entry, and PDQ entry for each GDG member when the entire generation data group is specified by the programmer.

Patterning DSCB Processing Routine

The patterning DSCB processing routine (Chart 32) completes control information in a JFCB when a new data set is to be patterned after a previously cataloged data set. The volume control block or data set pointer entry, which contains the volume serial number of the volume that contains the data set, is placed into a main storage work area. Fields in the JFCB are checked for zeros. If a field contains zeros, the corresponding field from the DSCB is moved into the JFCB.

Error Message Processing Routine

The error message processing routine (Chart 33) is entered and issues error messages whenever an error condition is encountered within a JFCB housekeeping routine.

Allocation and Setup

The allocation and setup function of the initiator/terminator (Chart 34) allocates I/O devices, issues any necessary mounting instructions to the operator, and ensures that enough I/O requirements have been satisfied to begin execution of a job step. The routines in the allocation and setup function are:

- Allocation control routine, which performs housekeeping for the allocation and setup function by obtaining space for tables used during allocation.
- Demand allocation routine, which constructs the allocate tables and begins actual allocation by assigning devices to any data sets for which the programmer requested specific devices.
- Automatic volume recognition routine, (optional) which can determine that named volumes have been mounted on certain devices and which allocates those devices to satisfy requests for the volumes.
- Decision allocation routine, which performs allocation when a choice of devices is to be made.
- TIOT construction routine, which builds a task input/output table (TIOT) that will be used by data management routines during step execution.
- External action routine, which issues mounting instructions, verifies that volumes are mounted on the correct units, and unloads incorrectly mounted volumes.
- Space request routine, which obtains, from the direct access device space management (DADSM) routines, space on direct access devices, and which satisfies requests for data set space.
- TIOT compression routine, which compresses the TIOT to its final size, updates JFCBs with scratch information whenever necessary, places the allocation messages in SMBs, and exits to the step initiation routine.
- DADSM error recovery routine (module IEFXT003), which determines what action should be taken when the request for space on a particular volume cannot be satisfied.
- Allocation error routines, which process error conditions encountered during allocation.

ALLOCATION CONTROL ROUTINE

The allocation control routine (Chart 35) performs housekeeping operations for the allocation and setup function of the initiator/terminator. It determines the size of certain tables to be constructed by subsequent allocation routines, obtains main storage space for the tables, and places the addresses of the portions of storage reserved for each table onto a directory of tables called the allocate control block.

Entry to the allocation control routine is made from the JFCB housekeeping control routine. Exit is to the demand allocation routine.

Upon entry, the storage requirements of the tables needed by allocation routines are calculated (see Figure 18). First, all requirements except those for the allocate volume table and TIOT are determined. The required amount of main storage space is requested and the addresses of the areas assigned to each table are calculated. (The first table is assigned the first available byte. Other addresses are determined by incrementing the last assigned address by length of the the respective table.) The relative position of each table except the device mask table is shown in Figure 19. The device mask table is included with the coding and is not positioned relative to the tables shown. As each address is determined, it is placed into the allocate control block.

When storage areas have been assigned for all but the allocate volume table (AVT) and task input/output table (TIOT), all step input/output tables (SIOTs) are placed into the area assigned to them. The size of the allocate volume table may then be determined. The number of volumes required by each data set (DD statement) is obtained from each SIOT and is used to calculate the number of AVT entries (one per volume) required. A second request for main storage space is issued and the address of the assigned area is placed into the allocate control block.

The storage requirements for the TIOT are calculated by the TIOT construction routine.

DD number table*	4 |A| 4 |4|
Buffer	176
Allocate control block	44
Channel load table	4 x the number of channels
Allocate work table	(20 + 8 |D| |32|) A
Potential user on device table	|D| 4 |4|
Separation strikeout pattern	|D| |32|
Each SIOT	68
Volume table	6S
TIOT	Determined by the TIOT construction routine
Allocate volume table	8B
Device mask table	4 + (8 + |D| |32|) F

Legend:
- * = Not used.
- || = Next higher integer if a fraction.
- A = Number of DD statements.
- B = Number of volumes or devices (whichever is greater).
- D = Number of entries in the I/O supervisor UCB lookup table.
- F = Number of entries in device mask table.
- S = Number of volume serial numbers.

Figure 18. Formulas for Determining Allocation Table Sizes

If, after a request for space, the required amount of main storage space is not available, the job is canceled.

Figure 20 shows the completed allocate control block. In addition to table addresses, the allocate control block contains other entries initialized by the allocate control routine.

All allocation tables are described in the descriptions of routines in which they are completed. When the allocate control block has been completed, control is passed to the demand allocation routine.

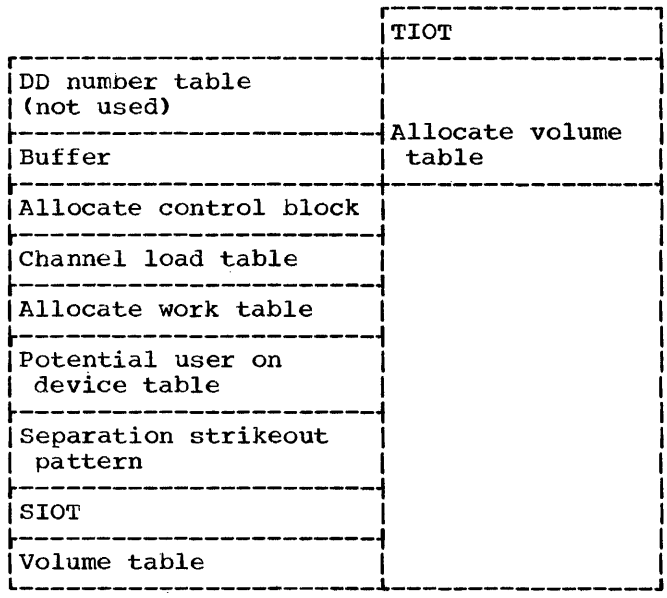


Figure 19. Relative Positions of Tables Used for Allocation

Hex	Dec			
0	0	Channel Load Table Address		4
4	4	Address of First Empty Slot in Allocate Volume Table		4
8	8	Potential-User-On-Device Table Address		4
C	12	Allocate Work Table (AWT) Address		4
10	16	Allocate Volume Table Address		4
14	20	Volume Table Address		4
18	24	Separation Strikeout Pattern Address		4
1C	28	Number of Satisfied Requests[1]	2	Number of Requests Not Satisfied[2] 2
20	32	Number of Bytes Per AWT Entry	2	Number of Work Table Entries with Separation 2
24	36	Length of Bit Pattern[3]	2	Number of DD Statements in Job Step[4] 2
28	40	Not Used	2	Number of Devices in Configuration[5] 2

Notes: (Entry length is shown in upper right corner of field.)

[1] Set to zero initially and incremented by one each time a request is satisfied.

[2] Initially set to the number of data sets to be allocated (the number of DD statements in the step). This number is decremented by one each time a request is satisfied.

[3] The length (in words) of the primary bit pattern.

[4] The number of DD statements to be processed.

[5] The number of UCB addresses in the I/O supervisor UCB lookup table.

Figure 20. Allocate Control Block

DEMAND ALLOCATION ROUTINE

The demand allocation routine (Chart 36) constructs the allocate work table and the allocate volume table. It also begins the allocation process by assigning devices to data sets that require specific devices. A specific device may be required because (1) the programmer specified it in a DD statement, or (2) all device requirements for a step could be met with only one combination of devices. The demand allocation routine performs the following eight functions:

- Allocate work table construction.
- Volume affinity resolution.
- Data set device requirement calculation.
- Channel load table construction.
- Allocation of resident devices.
- Device range reduction.
- System input device (SYSIN) allocation.
- Specific device allocation.

Allocate Work Table Construction

Two tables, the allocate volume table (see Figure 21) and the allocate work table (see Figure 22), are constructed by this function. The allocate work table contains information that describes a data set and certain other information that is used in allocating a device (or devices) to it. One entry, as shown in Figure 22, is built for each DD statement. The allocate volume table describes the volume on which the data set resides or will reside. One entry is made in the allocate volume table for each volume required by a data set.

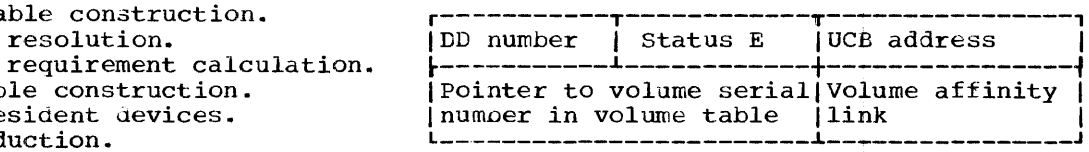


Figure 21. Allocate Volume Table Entry

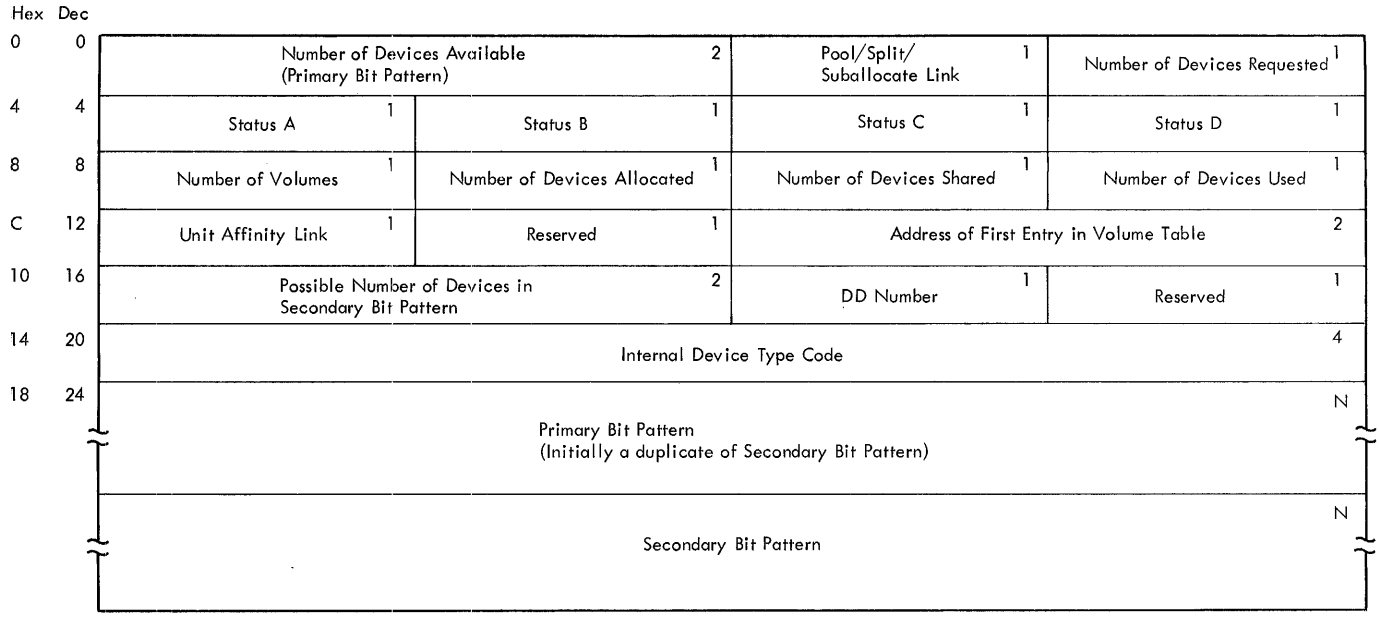


Figure 22. Allocate Work Table Entry

Most entries made in the allocate work table are obtained directly from other tables. The source of each such entry is shown in Figure 23. The device type is obtained from the SIOT and placed into the device type field of the allocate work table. It is then used as a search argument and a search of the device mask table, loaded from SYS1.LINKLIB, is made. The device mask table contains bit patterns that correspond to each group of units described either by a generic name or by a user's esoteric name. For devices that fall into either of these categories, a matching device type found by the search causes the corresponding bit pattern from the device mask table to be placed into the primary and secondary bit pattern fields of the allocate work table. These bit patterns indicate devices that are eligible for allocation to a data set.

The demand allocation routine builds its own bit pattern for devices described by specific unit names. To build the bit pattern, the demand allocation routine secures the device type from the SIOT and uses it as a search argument for a search of the UCB lookup table, from which the bit pattern can be extracted.

The demand allocation routine moves the private and nonshareable flag bits from the step input/output table (SIOT) to the allocate work table (AWT). The demand allocation routine also sets the nonshareable bit in the allocate work table entry for a request if the request does not specify a direct access device, and sets the private bit if the request is specifically for a nondirect access device (unless request applies to passed data sets).

Data sets that have similar I/O device requirements are then linked together. Similar requirements are implied when the programmer specifies the following in a DD statement:

- SPLIT=, which indicates that two or more data sets in the same job step are to share a cylinder of a direct access device.
- SUBALLOC=stepname.ddname or ddname, which indicates that space for the data set will be suballocated from the space allocated to the data set described in the DD statement named ddname.

Pointers are placed into the SPLIT/SUBALLOC link field and unit affinity link field of the allocate work table to link all such groups together.

Volume Affinity Resolution

Volume affinity means that a certain volume is requested for more than one data set. Volume affinity may be requested explicitly by use of the REF parameter of the VOLUME field of the DD statement, or implicitly by specifying the same volume serials in one or more other DD statements. In either case, the subject volumes are linked with pointers placed into the volume link field of the allocate volume table by the demand allocation routine. All requests for the same volume that appear in the volume

affinity chain subsequently will be satisfied with allocation of the device that bears the named volume.

Entry	Source
Number of devices available	Device mask table
POOL/SPLIT/SUBALLOC link	SIOT
Number of devices requested	SIOT
Number of volumes	SIOT
Status A	SIOT
Status B	SIOT
Status C	SIOT
Status D	SIOT
Number of devices allocated	Inserted as devices are allocated
Number of devices shared	Calculated
Number of devices required	Calculated
Unit affinity link	SIOT
Address of first entry in volume table	Calculated
Possible number of devices in secondary bit pattern	Device mask table
DD number	SIOT
Device type	SIOT
Primary bit pattern	Device mask table
Secondary bit pattern	Device mask table

Figure 23. Allocate Work Table Entry Sources

Data Set Device Requirement Calculation

Information obtained from the allocate work table is used to determine the number of devices required by each data set. The following calculations are used:

1. For a data set marked parallel mount (the P subparameter of the UNIT keyword was specified in the DD statement):

   $D_1 = V_2$

2. For data sets not marked parallel mount:
   a. If $V_1 = V_2$ then $D_1 = V_2$

   b. If $V_1 < V_2$ and
      if $V_1 < D_2$ then $D_1 = D_2$ or
      if $V_1 \geq D_2$ then $D_1 = V_1 + 1$

where:
D_1 = Number of devices actually to be used for the data set.
D_2 = Number of devices requested for the data set.
V_1 = Number of volumes to be shared by two or more data sets.
V_2 = Number of volumes on which the data set exists.

The number of devices to be used (D_1) is placed into the number of devices required field of the allocate work table.

Channel Load Assignments

For the purposes of allocation, a channel is a discrete path from a device to the CPU (or main storage). The load on a channel is the number of data sets accessible through it. The channel load table (CLT) furnishes a place to record these channel loads. After the allocation control routine (IEFXCSSS) builds the CLT in the scheduler work area, the various allocation routines use its information about channels and their loads to manage the channel and device resources efficiently.

Device allocation does not depend on physical channel addresses. Instead, the CLT defines channels by means of pointers to a list of device UCB addresses in the scheduler lookup table (see Figure 24). Each pointer defines a single channel, but may point to a series, or block, of several UCB address entries in the table. Each entry, in turn, is the address of a single device, so that a single channel may provide access to a number of devices. This proliferation of the data paths that a channel provides is illustrated in Figure 24, which also shows how more than one channel pointer from the CLT can ultimately provide access to a single device. The flexibility in device allocation that this scheme provides is the flexibility, for example, that the Model 2870 multiplexor channel (with its subchannels) requires.

The scheduler lookup table makes this flexibility possible by interposing one level of addressing between the CLT and the device UCBs. The allocation control rou-

tine builds the table in the scheduler work area, constructing it in three sections of halfword entries. The first section is a copy of the device list portion of the I/O supervisor lookup table. The entries in this section contain the addresses of specific device UCBs. Addresses in the first halfword of each fullword entry in the CLT point to blocks of entries in the scheduler lookup table to define the discrete channels for the allocation routines.

For example, in Figures 24 and 25, P_1 points to the first scheduler lookup table entry for channel one. Channel one entries include all the succeeding entries to the point where the second pointer, P_2, designates the beginning of the block of entries representing devices accessed through channel two. In Figure 24, the pointers from the CLT illustrate that channel one provides a data path to the 2311, the 2314, and the 2400 devices, while channel two provides a data path to the 2321 device. Note, however, that channel two also provides a data path to the 2311 device, because the first table entry for that channel also points to the UCB for that device.

The allocation routines can keep track of all the data paths provided to a device by using an allocation channel mask. This mask is a bit configuration that subroutine IEFXDPTH builds for use by the following allocation routines;

- The device strikeout routine -- IEFX300A;
- The separation strikeout routine -- IEFXH000;
- The decision allocation routine -- IEFX5000;
- The TIOT construction routine -- IEFWCIMP.

When an allocation routine calls subroutine IEFXDPTH, it passes to it a standard parameter list that includes a pointer to a UCB and a space for the channel bit pattern used as the mask. The subroutine searches the scheduler lookup table for UCB pointers identical to the one passed, notes the channel number associated with any such pointer entry, and turns on the bit corresponding to that channel in the mask space provided by the parameter list. The subroutine then returns control to the calling allocation routine, which now has channel information at its disposal.

The second halfword of each CLT entry is the number of data sets that constitutes the load on the channel to which the first halfword points. Hence, in Figure 25, L_1 and L_2 are the respective loads on channels one and two. In the same figure, P_n points to the last channel for which there is a set of one or more scheduler lookup table entries, and L_n is the load on that channel. P_X points to the first field of hexadecimal Fs in the scheduler lookup table. This field separates the first section, which contains the UCB addresses, from the second section. Allocation routines use this boundary and its CLT pointer to facilitate rapid searching of the table.

The second section in the table contains sets of ten pointers each for the sub-UCBs associated with every main UCB controlling a 2321 datacell drive. Such a set exists for every 2321 device that the operating system is using. A second entry of hexadecimal Fs follows the last sub-UCB entry in the section to delimit the entries from the different type that follows. The P_Y address in the CLT points out this quick-reference delimiter.

The third section in the table contains pointers to the first section. These pointers relate each set of ten sub-UCBs to its 2321 device main UCB. For example, in Figure 24, Q_n is a pointer associated with the set of sub-UCB pointers $P_{n,10}$, and it refers the set back to its proper main UCB via the pointer in the first section.

Allocation of Resident Devices

The resident device allocation routine allocates direct access devices containing reserved and permanently resident volumes to satisfy requests by serial number for these volumes. The devices that contain these volumes are known as resident devices.

A volume is placed into the reserved status either when the operator issues a MOUNT command specifying the device on which the volume is mounted or when the volume is so listed in the PRESRES member of the IPL/NIP parameter list data set (SYS1.PARMLIB). This type of volume cannot be dismounted unless its device is unloaded by means of an UNLOAD command.

A permanently resident volume has at least one of the following characteristics:

- The volume cannot be physically dismounted from its device.
- The volume is a system residence volume that contains the initial program loader (IPL) program.
- The volume contains the linkage library (SYS1.LINKLIB) data set, procedure library data set, or any part of the job queue (SYS1.SYSJOBQE) data set.

- The volume is listed as permanently resident in the PRESRES member of SYS1.PARMLIB.

For more information about the PRESRES data set member, refer to IBM System/360 Operating System: System Programmer's Guide, GC28-6550. For more information about reserved and permanently resident volumes, refer to IBM System/360 Operating System: Job Control Language Reference, GC28-6539.

The resident device routine determines which direct access devices are resident and then allocates them to satisfy any requests for the volumes they contain.

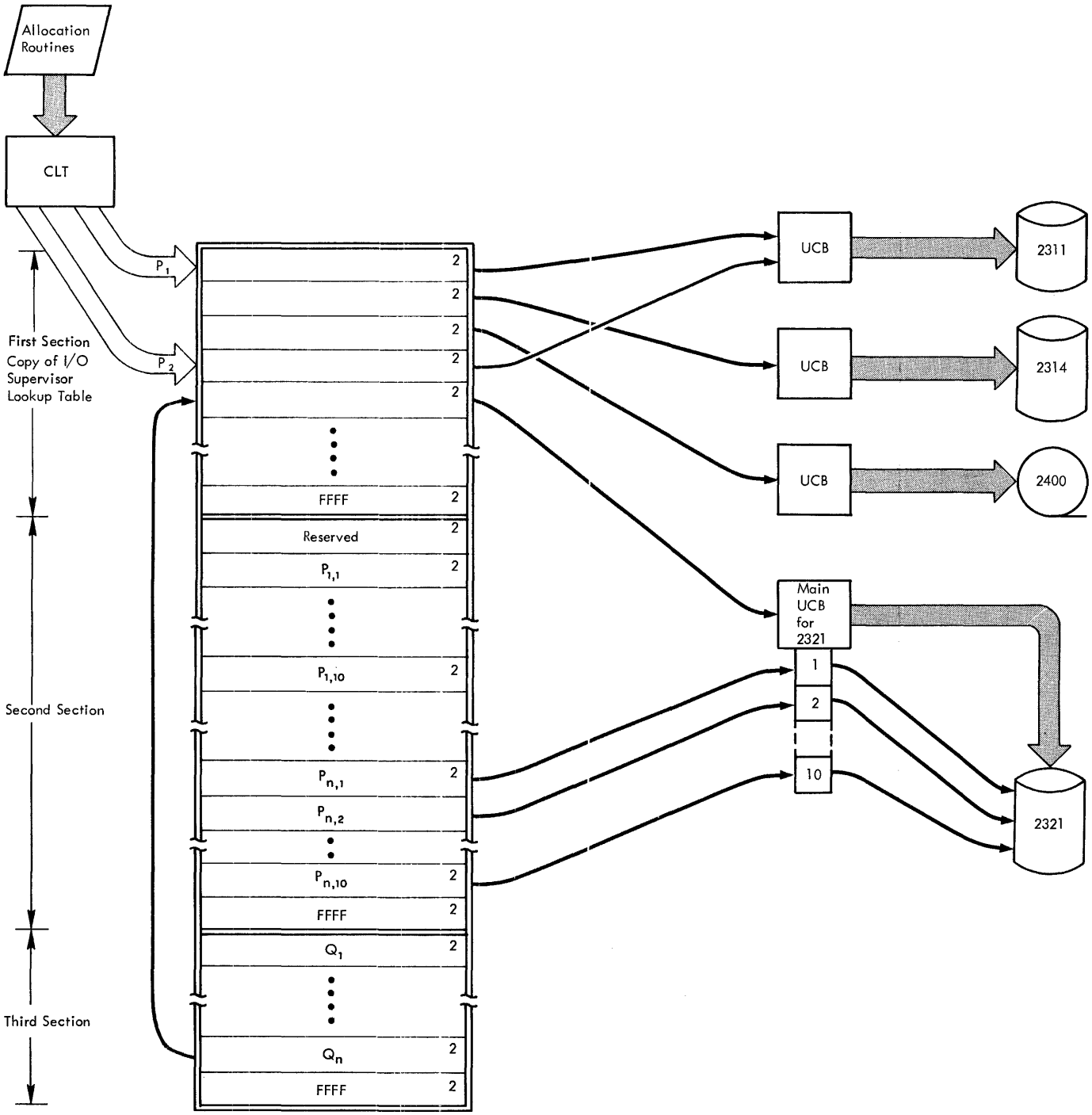


Figure 24. Scheduler Lookup Table

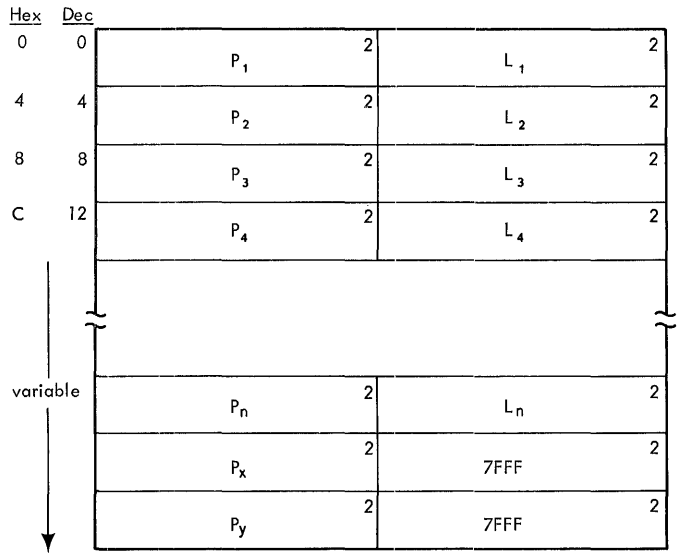


Figure 25. Channel Load Table

From the device mask table (DMT, Figure 35), the resident device routine first creates a special bit pattern that represents all direct access devices in the system. It sets a bit in the pattern to one for each direct access device. It then searches for unit control blocks representing direct access devices, using this bit pattern to identify the unit control blocks.

The routine compares the volume serial number in each request with the serial number in each unit control block in which the permanently resident bit or the reserved bit is one. If the serial numbers match, the routine passes control to the device strikeout routine to allocate the device. (That is, it places the address of the unit control block into the allocate volume table entry, Figure 21, and increases, by one, the count of allocated devices in the allocate work table entry that represents the data set, Figure 22.)

Device Range Reduction

The device range reduction routine reduces the number of devices that can be allocated to satisfy certain requests. In addition, this routine allocates devices containing reserved tape volumes.

The device range reduction routine prevents allocation of devices that are ineligible to satisfy certain requests. Devices are ineligible under the following conditions:

- The device is the primary console.
- The device is offline or is being changed to offline status.
- The device has either been allocated or is resident, and the request is for an unspecified private volume. (Each such request requires an unused volume.)
- The device has either been allocated or is resident; the device contains a private volume; and the request is for temporary data set space on a volume that is neither specific nor private.
- The device is a resident, direct access device, and the request is for a specific volume.
- The device is neither a direct access device nor a tape device (unit record or graphic equipment, for example) and is allocated, unless one of the two following conditions exists:
  - The device is the system output device, and the request is for a SYSOUT data set.
  - The device is the system input device, and the request is for a SYSIN data set.
- The device does not contain a storage volume, and the request has all of the following characteristics:
  - The request is not for temporary data set space.
  - The request is not for a specific volume.
  - The request is not for a private volume.

A storage volume is a permanently resident or reserved volume that may be used to keep any data set specified in a DD statement in which KEEP has been specified.

To prevent allocation of these ineligible devices, the device range reduction routine alters primary bit patterns representing devices that are available for allocation. In each bit pattern, ones represent devices that can be allocated, and zeros represent those that can not. A primary bit pattern forms part of each allocate work table (AWT) entry. (Each entry stands for one request.) The device range reduction routine eliminates each device that is ineligible to satisfy a particular request by changing the bit corresponding to the device from a one to a zero in the bit pattern corresponding to the request. The final bit pattern thus represents only devices that can satisfy the request.

As each ineligible device is disqualified, a count of eligible devices in each affected allocate work table entry is reduced by one. If this count becomes less than the number of devices needed to satisfy the request represented by the entry, the device range reduction routine passes control to the allocation error recovery routine. If recovery is possible, this routine provides a list of devices that can satisfy the request. The operator may either reply with a three-character device name or cancel the job. (If allocation error recovery is necessary, the entire allocation procedure is repeated.)

If, during this processing, the device range reduction routine finds a unit control block representing a tape unit with a reserved volume mounted on it, it allocates the device if the volume was requested.

SYSIN Allocation

If the device range reduction routine encounters a request for the device designated as the system input device, it allocates that device.

Specific Device Allocation

Allocation is next made to requests for specific devices or requests which, because of range reduction or previous allocation, can be satisfied only by a specific device.

Exits From Demand Allocation

When all processing is completed in the demand allocation routine, all requests within the step may have been satisfied. If so, exit is made to the TIOT construction routine. If, however, some requests remain outstanding, control is passed to the automatic volume recognition routine if it was specified during system generation. If additional requests remain, control is passed to the decision allocation routine. When allocation is complete, the "number of unallocated entries" field in the allocate control block (ACB) reaches zero. If the number of devices required exceeds the number of devices available, control is passed to an allocation error routine. Before any exit is taken, the device mask table is deleted.

AUTOMATIC VOLUME RECOGNITION

The automatic volume recognition (AVR) routine decreases the time required for job step initiation by enabling the operator to mount volumes needed for subsequent job steps as soon as devices become available. During subsequent job step initiation, the AVR routine recognizes that volumes needed for the current job step are mounted, thus saving the time that the system otherwise would spend waiting for the operator to find and mount them.

Before the next job step after a volume has been mounted, the AVR routine reads the volume label and associates the volume with the device containing it, using information from the label. When the volume is needed for a subsequent job step, the AVR routine can then identify and allocate the device on which it is mounted.

The AVR routine contains two modules, IEFXV001 and IEFXV002, as shown in Charts 37 and 38 respectively. Most of the AVR routine's function is performed by IEFXV001, the first module to receive control. The demand allocation routine passes control to module IEFXV001 of the AVR routine. Then IEFXV001 uses a BALR instruction to branch and link to the VCON type address of the second module, IEFXV002, whose main function is primarily one of reading volume serial numbers.

IEFXV002 reads the volume serial number and verifies it. If the volume serial number is valid, IEFXV002 then places it in the unit control block (UCB) and returns control to IEFXV001. However, if an I/O error occurs, IEFXV002 sets an error return code without altering the UCB. When it encounters nonstandard labels during the the reading process, it branches to IEFXVNSL, the nonstandard label (NSL) processing routine. If IEFXVNSL returns no error code, IEFXV002 places the volume serial number into the UCB as though the NSL routine had never received control for special processing, then returns control to IEFXV001. Errors detected upon return from the NSL routine, however, cause IEFXV002 to bypass alterations of the UCB and instead to return control directly to IEFXV001. IEFXVNSL returns an error code if no user written routine has replaced the IBM supplied one, or for whatever reason the user written routine specifies.

The AVR routine allocates devices to satisfy requests that specify 2311 and 2314 direct access volumes, 7-track tape volumes having a tape density specified during system generation, and 9-track tape volumes. These volumes must be specified by either a serial number or a data set name that implies a serial number. The AVR routine first allocates devices containing mounted volumes. If any of the volumes have been mounted after the start of the last job step, and have consequently not had their labels read, the AVR routine reads them at this time.

When all devices containing mounted volumes which are needed for the current job step have been allocated, the AVR routine attempts to satisfy any remaining requests for 2311 and 2314 direct access volumes and 9-track tape volumes. The AVR routine determines whether there are sufficient unused devices of each device type to satisfy the outstanding requests for that device type. If necessary, volumes not needed for the job step are unloaded. If the AVR routine can obtain enough devices, it prints a list of the requested volume serial numbers and allocates the devices as the operator mounts the volumes. If enough devices are not available or if all of the needed volumes cannot be mounted, however, the operator must cancel the job.

Processing Requests for Mounted Volumes

The AVR routine first satisfies requests for volumes which are already mounted. The AVR routine searches for such volumes by examining all unit control blocks that represent online, ready 2311 and 2314 direct access devices and 9-track and 7-tape devices. If the serial number in the unit control block is zero, it means that the volume has been mounted since the start of the last job step and has therefore never had its label read. The AVR routine, at the time it finds such a volume, reads the volume label into main storage, extracts the serial number from the label, and records it in the unit control block representing the device. (To extract the serial number from a nonstandard label, the AVR routine uses a volume serial number routine, IEFXVNSL, which must be supplied by the user. A routine with the same name is supplied by IBM to indicate an error if the user has provided a nonstandard label but has not substituted his own routine to read it.) If the volume had been mounted before the start of the last job step, the serial number has already been read.

The AVR routine next determines, for each mounted volume, whether it is needed for the current job step. To make this determination, it searches in the volume table (VOLT) for the serial number of the mounted volume. (Each entry in this table represents a volume that has been specifically requested.) If the AVR routine locates the serial number, the volume is needed for the job step. The AVR routine then uses the device strikeout routine to allocate the device to satisfy all requests for the volume. If the serial number is not in the volume table entries for this job step, however, the volume is not presently needed. The AVR routine subsequently ignores the device and looks for another previously mounted volume. If the device has already been allocated to a different volume, or if the volume has been allocated to a different device by the demand allocation procedure, the AVR routine notifies the operator and unloads the volume using the external action routine.

Processing Requests for Unmounted Volumes

The AVR routine finally attempts to satisfy all remaining specific volume requests. For these requests to be satisfied, enough devices for all of the requests either must be available or must be made available. If enough devices become available, the AVR routine provides the operator with a list of volumes to mount and allocates the devices as he mounts the volumes on them. If sufficient devices for the job step cannot be made available or if all of the required volumes cannot be mounted, the operator must cancel the job.

Obtaining Devices: Before the AVR routine requests that the operator mount any unmounted volumes, it determines whether enough devices to contain them are available. If there are not enough devices without mounted volumes to begin with, the AVR routine determines whether it can unload enough devices. The devices it considers for unloading contain mounted volumes not needed for the job step. If it can, it unloads these devices so that the operator can replace the mounted volumes with volumes needed for the job step. Otherwise, the AVR routine attempts to have enough offline devices placed into online status to satisfy the remaining specific requests.

To determine whether there are enough devices, the AVR routine compares, by device type, a count of available devices with a count of needed devices. Because the need for each device type is filled separately, a shortage of any one type means that not enough devices are available for the job step.

The available devices comprise all online 9-track tape units, 2311 disk units, and 2314 disk units that have not been allocated. Separate counts are made of devices not in the ready status (which normally do not contain mounted volumes) and devices that are ready (all of which have mounted volumes).

To eliminate any unnecessary unloading of devices, the AVR routine compares, first, the number of devices needed with the number of online devices not having mounted volumes (that is, those that are not in the ready status). If there are enough such devices, none need be unloaded, and the AVR routine can immediately print a list of volumes to be mounted.

If ready devices must be unloaded, the AVR routine determines the number of ready devices still needed and whether enough can be unloaded.

If the AVR routine has determined that enough ready devices can be unloaded, it stores the identities of a sufficient number of devices and then unloads them. To fill the quota, it first tries to obtain enough ready devices not containing retained volumes or volumes with data sets. If the AVR routine cannot find enough devices, it obtains the remainder needed from among devices containing these kinds of volumes. The AVR routine unloads the devices with the external action routine, which also prints a list of unit addresses so that the operator will know which devices have volumes to be dismounted. The AVR routine then provides the operator with a list of the serial numbers of volumes to mount.

In an attempt to make more devices available, if it is apparent that enough ready devices cannot be unloaded, the AVR routine uses the allocation error recovery routine (IEFXJIMP) to print a list of offline devices that can be made available. The operator either may reply with a three-character device name to place a device into online status or cancel the job. (If allocation error recovery is necessary, the entire allocation procedure is repeated.)

Allocating Devices on which Volumes have been Mounted: When the AVR routine has determined that the required number of devices is available for allocation, it provides the operator with a list of serial numbers of the needed volumes . As the operator mounts these volumes, the AVR routine allocates the corresponding devices to satisfy requests for these volumes.

After printing the list, the AVR routine waits for the operator to mount a volume. A device-end I/O interruption releases the AVR routine from its waiting status when the operator mounts the first volume and presses the START button on the device. The AVR routine extracts the new serial number from the volume label , removes the serial number from the list of required volumes, and allocates the device. Then the AVR routine waits for the operator either to mount the next volume or to cancel the job. It repeats the procedure until either all specific volume requests have been satisfied or the job is canceled.

When the devices have been allocated, the AVR routine passes control to the TIOT construction routine, unless there are more volume requests. If there are, the AVR routine passes control to the decision allocation routine, which satisfies the remaining requests.

DECISION ALLOCATION ROUTINE

The decision allocation routine (Chart 40) allocates devices to most data sets for which devices have not yet been allocated by either the demand allocation or the automatic volume recognition routine. This includes all remaining requests except requests for space on unspecified public or unspecified storage volumes. The latter requests are fulfilled by the space request routine.

Upon entry to the decision allocation routine, an attempt is made to reduce the number of devices that are candidates for allocation. A request for unit or channel separation from devices allocated by either the demand allocation or automatic volume recognition routines eliminates the units or additional devices on the selected channels from further consideration. If this is the case, the separation strikeout subroutine is entered. This subroutine, by changing corresponding bits in the primary bit pattern, eliminates these devices from consideration for allocation.

The number of data sets directed to each channel is then determined and added to the totals in the channel load table (see Figure 24). This table is later used to "spread the load" across the channels, thereby:

- Obtaining maximum overlap of I/O activity.
- Reducing the possibility of making a channel ineligible because all of its devices had been allocated too early. (Some channel separation requests would then be impossible to satisfy.)

The maximum number of data sets that could use each device is next determined and placed into the potential user on device table (see Figure 26). This table is later used to determine the order in which devices will be selected for data sets. (Devices first selected are those with the fewest potential users.)

Figure 26. Potential User on Device Table

The remainder of the decision allocation routine allocates devices. First, devices are allocated to data sets for which only one device is eligible. Then all other requests (except those for unspecified public or unspecified storage volumes) are processed in the following manner. A data set is selected and then a device for the data set is selected and allocated to it. Another data set is then processed.

Data Set Selection

Data sets are selected by considering the number of devices eligible for allocation to them. That is, the first data set selected is the one for which the smallest number of devices is eligible.

The decision allocation routine selects two kinds of requests, both of which must be satisfied with the allocation of devices containing nonshareable volumes:

- Requests for nonshareable volumes. (Each such request has a nonshareable flag in its allocate work table entry, shown in Figure 22.)
- Requests that may be satisfied with the allocation of either a direct- or sequential-access device, if sequential-access devices are available for them. (As each of these requests is satisfied, a nonshareable flag is placed into its allocate work table entry to mark the allocation of a device containing a nonshareable volume.)

Selection is performed by scanning the allocate work table. If two or more data sets have the same number of eligible devices, they are selected in the following order:

1. Data sets with separation requests.
2. Data sets with affinity requests.
3. Passed data sets.
4. All others.

Device Selection

When a data set has been selected, a device is selected and allocated for it. Devices are considered in the following order:

1. If the possible devices for a data set exist on more than one channel, the channel with the greatest number of free devices of the type requested is chosen.

2. If two channels have the same number of free devices of the requested type, the channel with the lightest load is chosen; the device which has the fewest possible users is chosen.

3. To satisfy requests for public nonspecific (scratch) tape volumes, devices with mounted tape volumes are given preference. To satisfy requests for direct access volumes and specific tape volumes (including private volumes and volumes which are used for multi-volume public data sets), devices without mounted volumes are given preference.

4. If two devices have the same number of possible users, the first one in the I/O supervisor UCB lookup table is chosen.

Device Allocation

As indicated previously, the decision allocation routine selects a data set and an eligible device, allocates the device, and then selects another data set. To allocate a device, the decision allocation routine places the address of the unit control block representing the device into the allocate volume table entry (Figure 21) representing the required volume and adds one to the "number of devices allocated" field of the allocate work table entry for the data set (Figure 22).

While a request is being satisfied, the same device is also allocated to satisfy any other requests that specify the same volume. Multiple allocations may be performed in this case, because all requests for the same volume appear in a volume affinity chain, which is a series of linked allocate volume table entries (Figure 21).

The decision allocation routine satisfies, in the same way, requests that specify unit affinity or that have a split or suballocate relationship (Figure 22).

When a device is allocated, the decision allocation routine alters bit patterns in the allocate work table entries for certain other requests. Each bit pattern specifies the devices that are eligible to contain the data set represented by the allocate work table entry.

If a private volume request was satisfied, the decision allocation routine changes the bit representing the allocated device to zero in all primary and secondary bit patterns so that the device cannot be selected to satisfy another request. Such devices are exempted from further allocation because each private volume may not contain other data sets and must be removed after use.

If the request was satisfied with a device containing a nonshareable volume, the decision allocation routine changes the bit representing the device to zero in the primary and secondary bit patterns of the allocate work table entries that represent all other data sets that require nonshareable volumes. A device allocated to satisfy a request for a nonshareable volume thus cannot satisfy additional requests of this kind.

If all eligible devices are allocated before all data sets for a step have been selected for allocation, the decision allocation routine passes control to an allocation error routine.

Upon successful completion of processing by the decision allocation routine, exit is made to the TIOT construction routine.

TIOT CONSTRUCTION ROUTINE

The task input/output table (TIOT) construction routine (Chart 41) obtains space for and builds the processing program's task input/output table. The primary function of the TIOT is to provide the data management open, close, and end-of-volume (EOV) routines with pointers to JFCBs and allocated devices.

Entry to the TIOT construction routine is made when all requests for I/O devices have been satisfied except requests for unspecified public or unspecified storage volumes. Therefore, entry may be from the demand allocation routine, the automatic volume recognition routine, or the decision allocation routine. Exit is to the external action routine.

Upon entry, main storage space required to build the TIOT is calculated using the first formula shown in Figure 27, and space is requested. The standard TIOT is shown in Figure 28. TIOT entries are constructed for each data set in a step. Entries are also constructed when use of the job library is requested or when a program, created in a previous step, is to be executed as the current step. Figure 29 shows the sources of entries in the TIOT.

The TIOT construction routine determines, for each request for an unspecified storage or unspecified public volume, which devices are eligible to be allocated by the space request routine. It obtains this information from the allocate work table entry (Figure 17) for the request, which contains a primary bit pattern representing the devices that are eligible to satisfy the request.

The TIOT construction routine places pointers to all unit control blocks representing eligible devices into the TIOT entry for each such request. If more than one device can satisfy a request, it selects, first, the channel with the lightest load, and, on this channel, the device that has been allocated to satisfy the smallest number of requests. When the first device has been selected, it places other devices in order, using the following criteria:

1. Devices on the same channel as the first device selected, but which do not contain passed data sets.
2. Devices that do not contain passed data sets and do not violate requests for separation.
3. Devices that contain passed data sets and do not violate separation requests.
4. Devices that do not contain passed data sets and violate separation requests.
5. All other devices eligible to receive public volumes.

Should more than one device have similar attributes, their pointers are arranged in the order in which the devices are represented in the primary bit pattern.

Space required to build TIOT = $28 + 16N_1 + 4N_2 + 4(N_3 \times N_4)$
Space occupied by completed TIOT = $28 + 16N_1 + 4N_2$
Where: N_1 = Number of DD statements. N_2 = Number of devices allocated to the step. N_3 = Number of requests for public volumes. N_4 = Number of devices available for public volumes.

Figure 27. Formulas for Determining Task Input/Output Table Space Requirements

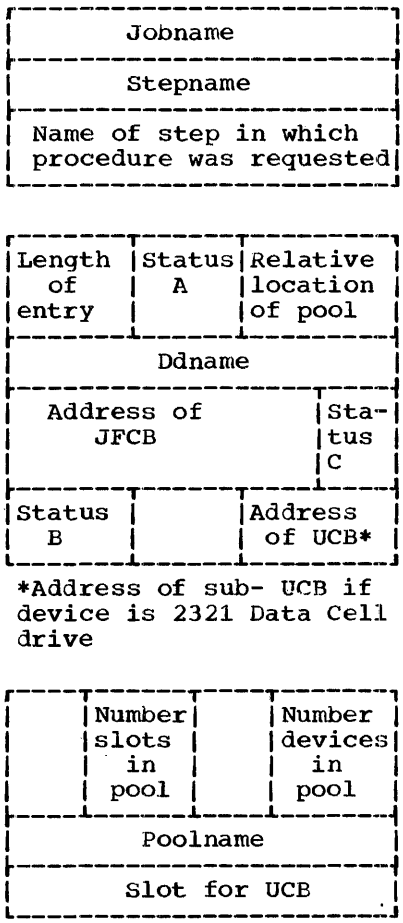


Figure 28. Task Input/Output Table

Entry	Source
Jobname	JCT
Stepname	SCT
Stepname of step in which procedure was requested	SCT
Length of entry	Calculated
Status A	Calculated
Relative location of pool	Calculated
Ddname	SIOT
Address of JFCB	SIOT
Status C	Calculated
Status B	Calculated
Address of UCB	I/O supervisor UCB Lookup Table
No. of slots in pool	Calculated
No. of devices in pool	SIOT
Poolname	SIOT
Slot for UCB	I/O supervisor UCB Lookup Table

Figure 29. Task Input/Output Table Entry Sources

EXTERNAL ACTION ROUTINE

The external action routine (Chart 42) issues mounting instructions, verifies that the correct volumes have been mounted, and unloads incorrectly mounted volumes.

Entry to the external action routine is made from the TIOT construction routine. Exit is made to the space request routine.

Upon entry, devices allocated to each data set are checked and any required dismounting is requested. (The operator is notified of volume dispositions.) Messages instructing the operator to mount the required volumes are then issued, and checks are made to ensure that volumes were mounted on the correct units.

SPACE REQUEST ROUTINE

The space request routine (Chart 43) processes requests for space on direct access volumes. It determines whether a volume has enough space for the data set specified in a particular request, and, if so, it obtains space on the volume for the data set. If space is not available initially, the space request routine attempts to locate another volume with sufficient space.

The space request routine, which receives control from the external action routine, searches among the task input/output table (TIOT) entries for requests for direct access volume space. It processes these requests in two different ways, depending on whether or not a device was previously allocated to satisfy the request.

Obtaining Space If a Device Was Allocated

If a device has been allocated to satisfy the request (because a specific device or volume was named), the space request routine attempts to obtain space on the volume that is mounted on the device. It passes control to the direct access device space management (DADSM) routines, which record the limits of an extent on the volume into a data set control block (DSCB) if space is available. If the mounted volume does not have space for the data set, and is not being used to contain another data set for the job step, the space request routine passes control to the external action routine, which directs the operator to mount another volume on the allocated device.

Obtaining Space If a Device Was Not Allocated

If a device has not been allocated to satisfy the request, the space request routine attempts to obtain space on an unspecified public or unspecified storage volume, depending on the type of request. (Either unspecified public or unspecified storage volumes can contain temporary data sets, but only storage volumes are eligible to contain data sets that are to be kept.) If the space request routine determines that a volume has space for a data set, it allocates the device containing the volume.

The space request routine attempts to obtain space for the data set on a volume that is mounted on an eligible device. (The devices that are eligible to satisfy a particular request are indicated in the task input/output table entry for the request. Each entry contains pointers to the unit control blocks representing eligible devices.) To determine whether space is available, the space request routine passes control to the direct access device space management (DADSM) routines. These routines attempt to specify an extent on the volume. If space is not available, control passes to the DADSM error recovery routine, to determine whether another volume can be mounted. If no volume can be mounted, exit is taken to the external action routine, which requests the operator to mount a volume on an eligible device that does not contain a volume.

When all requests for space have been satisfied through the above procedure, or when an unrecoverable error has been detected (that is, when space cannot be allocated), the space request routine exits to the TIOT compression routine.

TICT COMPRESSION ROUTINE

The TIOT compression routine is entered from the space request routine when all requests for space have been satisfied, or when an unrecoverable error has been detected.

In the case of a normal entry, the TIOT compression routine reduces the TIOT to its final size, adds scratch information to JFCBs where necessary, and adds allocation messages to SMBs when the allocation message level is one. This message level may be either the system generation default option or the result of a coded job control language JOB statement parameter, "MSGLEVEL=(x,1)". The routine exits to the step initiation routine of the initiator/terminator.

In the case of an error entry, the routine reduces the TIOT to its final size and exits to the allocation error routine (see below). The format of the TIOT is shown in Figure 28.

DADSM ERROR RECOVERY ROUTINE

The DADSM error recovery routine is entered from the space request routine when space is not available on a requested volume. The routine determines whether the requested volume is unused and removable (that is, not permanently resident and not reserved). If the volume can be removed, the DADSM error recovery routine returns to the space request routine, which exits to the external action routine to request that the operator mount another volume on the same device.

If the requested volume cannot be removed, the DADSM error recovery routine selects another device, then returns control to the space request routine; the space request routine then attempts to obtain space on another mounted volume.

(If no other device is available, the request for space cannot be fulfilled.) If the failing request was one of several nonspecific requests for space on the same volume for the job step, and all users on that volume are those assigned by the space request routine, the allocated data sets will be unallocated and the volume may be removed. When a new volume is mounted, the space request routine will again attempt to obtain space for the data sets.

ALLOCATION ERROR ROUTINES

Allocation error routines are entered when error conditions are encountered by allocation and setup routines. There are two error routines: the recovery routine and the nonrecovery routine.

The recovery routine is entered if an error condition is detected before a TIOT is built for the step. It may be entered from the demand allocation, automatic volume recognition, decision allocation, or TIOT construction routine. If allocation requirements can be satisfied by changing the status of a device from offline to online (determined by checking the secondary bit pattern), the recovery routine issues a message to the operator requesting him to place additional devices online. If he does, allocation for the step is begun anew by entry to the allocation control routine. If the operator does not or cannot add devices to the configuration, the recovery routine cancels the job.

The nonrecovery routine is entered when an error condition is detected after the TIOT has been built for the step. It passes control to the step termination portion of the initiator/terminator.

Step Initiation

The step initiation routine of the initiator/terminator (Chart 46) makes preparations for passing control to the processing program. If a STEPLIB DD statement is present in the step, the step library data set is opened. If not, and if a JOBLIB DD statement is included in the job, the job library data set is opened. If the program to be executed exists on a data set created in a previous step, a DCB is created for that data set and is opened. Also, several tables are stored, releasing to the processing program the space they occupied. Step initiation passes control to the processing program.

The step initiation routine is entered from the space request routine. Upon entry, control is passed to the pseudo-sysout subroutine, which writes the contents of system message blocks (SMBs) onto the system output data set.

When control returns from the pseudo-sysout subroutine, the step initiation routine scans the TIOT for entries indicating SYSOUT processing. The UCB address for these entries is zero. When such an entry is found, the corresponding JFCB is read into main storage. The device class (placed in the JFCB by the interpreter) is obtained, and the UCB address of the writer currently active for that class is placed in the TIOT entry. If a DSNAME parameter was specified in the START command, the step initiation routine places the DSNAME in the JFCB. The LCT and JCT are then stored and the space that they had occupied is released.

Main storage space to be used by the processing program is then obtained. A portion of this area is reserved for the following:

- One DCB for step or job library (if any).
- Fetch DCB (if any).
- Macro-parameter list.
- TIOT.
- Processing program register save area.

First, the TIOT is moved from the initiator/terminator work area to the area of processing program storage assigned to it. The TIOT is also stored, and the space it occupied is released. The macro parameter list (see Figure 30) is then built and the programname entry and initializing parameter values entry (PARM information) are inserted. The SCT is then stored, and the space it occupied is released. If a step or job library has been requested, the data set is opened, and the address of its DCB is placed into the TCB. If a fetch DCB is required (PGM=*.stepname.ddname was specified in the EXEC statement) a DCB is created and opened, and its address is placed into the macro parameter list.

The cancel ECB in the selected job queue[1] is then set up for the processing program: i.e., the low-order byte is changed to the number 255. If a CANCEL command was issued, the step initiation routine issues the ABEND macro instruction.

---

[1]Just prior to passing control to the job step, the low-order byte of the cancel ECB in the selected job queue is changed to all ones. This causes issuance of an ABEND or ABTERM rather than a POST by the master scheduler if the operator issues a CANCEL command for the job.

If a CANCEL command was not issued, an XCTL macro instruction is used to pass control to the processing program.

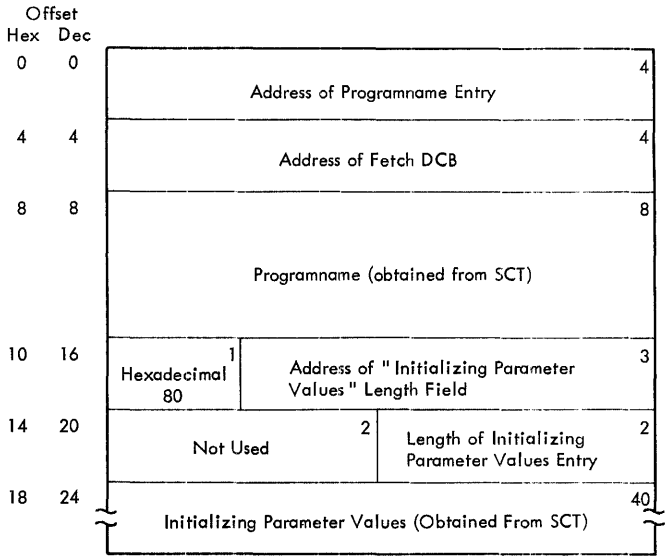


Figure 30. Macro Parameter List

Termination

The termination function of the initiator/terminator (Chart 47) performs post-step and post-job housekeeping. It is normally given control following step execution, but is also given control when a job management routine encounters an irrecoverable error while processing a job step. Termination routines:

- Release space occupied by tables.
- Free I/O devices.
- Dispose of data sets referred to or created during execution.

Major components of termination are:

- The step termination routine, which performs post-step housekeeping functions.
- The job termination routine, which performs post-job housekeeping functions.

The disposition and unallocation subroutine is used by both the step and job termination routines. Basically, this subroutine handles disposition of data sets and frees devices allocated to a step. The disposition and unallocation subroutine is described in Appendix A.

STEP TERMINATION

The step termination routines (Chart 48) perform cleanup operations for each job step. They are entered from the supervisor when a step has been terminated either normally due to successful completion of execution or abnormally due to an error condition. They are also entered from job management routines when an unrecoverable error condition has been detected.

When entry is from the supervisor, the step termination entrance routines (IEFSD011 and IEFW42SD) perform initialization functions. These functions include:

- Setting the cancel ECB in the selected job queue to zero.
- Placing the LCT, JCT, SCT and problem program TIOT into a main storage work area.
- Constructing a parameter list containing the address of the above tables.
- Initializing an SMB for use by step termination routines. If write-to-programmer messages were produced during execution of the step, SMBs containing WTP messages will precede those used to contain termination messages.

In the case of normal termination, the entrance routines reset the restart information in the JCT; in any case, the JCT is stored in the job queue.

If the job step has terminated abnormally, control is passed to the indicative dump routine (IEFIDUMP). After the dump has been performed, control passes from the indicative dump routine to the step termination control routine. If the job step has terminated normally, the indicative dump routine is bypassed.

The step termination control routine (IEFYNIMP) is entered from the step termination entrance routines, from the indicative dump routine, or from a job management routine as a result of an unrecoverable error. It uses these major subroutines:

- Restart preparation routine (IEFRPREP).
- Step termination data set driver routine (IEFYPJB3).
- Job statement condition code routine (IEFVJIMP).
- Disposition and unallocation subroutine (IEFZGST1, IEFZGST2).

- User's accounting routine (IEFACTLK), if included in the system.

The control routine places the problem program TIOT address into the TCB, and the task completion code into the SCT. In the case of abnormal termination, the WTO macro instruction is used to inform the operator that the step has failed, and control is then passed to the restart preparation routine.

The restart preparation routine (Chart 49) determines if a restart is possible. If it is not, it sets the "no restart" indicators in the JCT (bit JCTNORST in byte JCTRSW2 of the JCT). If a step restart is to occur, the restart preparation routine sets bit 5 in byte JCTRSW1 of the JCT; this indicates to the termination routines that all NEW data sets are to be deleted, and OLD data sets are to be kept. If a checkpoint restart is to occur, the routine sets bit 4 in JCTRSW1; this indicates that all data sets are to be kept. After the restart information has been placed in the JCT, the restart preparation routine requests special disposition of data sets. Control returns to the step termination control routine.

If no restart is possible, and if the step failed with either a user or a system abnormal termination (bit 0 of the TCBFLGS field is on), the step termination control routine sets the JCTABEND and the SCTABEND bits. Setting these bits causes the job scheduler to bypass all the following steps unless either the COND=ONLY or the COND=EVEN parameter specifies execution after abnormal termination. If any other failure has occurred, such as an allocation failure or the issuing of a CANCEL command, the step termination control routine sets the job failed bit (INCMSTS) in the JCT, and the job scheduler will not execute any other step of the job.

The step data set driver routine is then entered. Whenever the problem program has abnormally terminated, this routine tests for an allocation message level of zero. If the programmer did specify zero in the JOB statement, the routine reconstructs the allocation messages and places them in the current system message block (SMB). After this initial processing, the routine places the SIOT for each data set into a main storage work area and branches to the disposition and unallocation subroutine. The loop through the data set driver routine and the disposition and unallocation subroutine is then repeated for each SIOT. If the JOB statement specified an allocation message level of one, or if an abnormal termination occurred, the data set driver routine places, in the current SMB, termination data set disposition messages for each data set in the step.

When all data sets have been processed by the disposition and unallocation subroutine, the problem program TIOT is released. Control is then passed to the job statement condition code routine, unless the job is to restart; in this case, control is passed to the user's accounting routine.

The job statement condition code routine (Chart 50) processes condition codes specified in the JOB statement. If upon entry it is found that there were no condition codes specified, control is passed to the user's accounting routine. If there were condition codes specified, the job statement condition code routine compares each condition code in the JCT with the step completion code of the previous step, which appears in the SCT. Up to eight conditions for each step are checked; any additional condition codes are ignored. If any of the condition operators are satisfied by the codes, the job-failed indicator in the JCT is updated to indicate that the job failed; the message subroutine is used to issue a message to the programmer, and the WTO macro instruction is used to issue a message to the operator. Control is then passed to the user's accounting routine.

From the user's accounting routine control passes to the step termination exit routine (IEFW22SD). This routine stores the SCT in the job queue, updates the LCT, and writes the last terminate SMB to the job queue. It then exits to the interpreter/initiator interface module (IEFSD002) for return to the interpreter or the initiator.

JOB TERMINATION ROUTINE

The job termination routine (Chart 51) performs its functions when an entire job has been executed and step termination for its last step has been completed. It consists of four major routines:

- Job termination control routine.
- Release job queue routine.
- Disposition and unallocation subroutine.
- User's accounting routine (if included in the configuration).

Control is passed to the job termination control routine from the step termination routine.

The job termination control routine determines if a passed data set queue exists and, if so, places each block into main storage work area and tests for unreceived data sets. (An unreceived data set is a

passed data set to which no reference is made after PASS is specified.) When an unreceived data set is found, entry is made to the disposition and unallocation subroutine. When all unreceived data sets have been processed, or if no passed data set queue exists, the job termination control routine passes control to the accounting routine, if there is one. As in step termination, if the allocation message level is one (if the job statement parameter is "MSGLEVEL=(x,1)"), or if an abnormal termination has occurred, final disposition messages describing the data sets handled by job termination are placed in the current SMB.

When the accounting routine returns, or if there is none, the completed job's control tables are removed from the system by the release job queue routine. This routine releases the auxiliary storage space (or, if the resident job queue option was selected during system generation, the main storage space) occupied by all control tables for the job. If the job notification switch is on, the message

IEF404I jobname ENDED

is written on the console device. This message is not issued in any case where the job was terminated abnormally. If the job was terminated because of a JCL error in any but the first job step, the WTO macro instruction is used to issue the message:

IEF452I jobname JOB FAILED - JCL ERROR

on the console. Control is then passed to the interpreter control routine.

Appendix A: Major Subroutines

Table Store Subroutine

The table store subroutine stores records into and retrieves records from the SYS1.SYSJOBQE data set. This data set may be either completely on a resident direct access device, or partly in main storage and partly on such a device, depending on whether the resident job queue (RESJQ) option was specified during system generation. The table store subroutine provides the following services on request:

- Supplies the requester with an auxiliary storage address or addresses into which records may later be written.
- Writes a record (or records) onto SYS1.SYSJOBQE locations specified by the requester.
- Reads a record (or records) from SYS1.SYSJOBQE locations specified by the requester .

The table store subroutine is used by job management routines to temporarily store tables and work areas that need to be communicated from one routine to another.

As part of the preparation for system generation (initializing system data sets), a specified number of tracks is assigned to data set SYS1.SYSJOBQE. During IPL, this extent is formatted for 176-byte records. (All records handled by the table store subroutine are 176-byte records.)

If the resident job queue option was selected during system generation, a specified number of records, starting at the beginning of the data set, will occupy a main storage area, thus saving time when tables are to be stored or retrieved. If there is room within this area of main storage, the I/O supervisor causes the records to be moved in response to the table store subroutine's WRITE macro instruction; if desired records are stored in this main storage area, the I/O supervisor causes them to be moved in response to a READ macro instruction.

The calling routine may request one of five functions. These are:

- Assign and start. The requested number of track addresses are assigned, beginning with the first assignable address in the extent.
- Assign. The requested number of track addresses are assigned, beginning with the next available address in the extent.
- Write and assign. The requested number of records are written, and the requested number of addresses are assigned.
- Write. The requested number of records are written.
- Read. The requested number of records are read.

Before passing control to the table store subroutine, calling routines must construct a parameter area (see Figure 31) and place its address into general register 1. Calling routines must also provide a QMPCA-QMPEX list (see Figure 32). Figure 33 shows the parameters required when a function is requested. The parameters are:

- QMPOP. A function code that indicates the function to be performed.
- QMPCM. The number of records (maximum of 15) for which addresses are to be assigned.
- QMPNC. The number of records (maximum of 15) to be stored into or retrieved from SYS1.SYSJOBQE.
- QMPCL. The beginning address of the QMPCA-QMPEX list.
- QMPCA. The main storage address from which the record is to be read or into which the record is to be written.
- QMPEX. The record address (in SYSJOBQE) into which the record is to be written or from which the record is to be read.

An entry in the QMPCA-QMPEX list is required for each record when a read or write function is requested. For assign functions, the table store subroutine returns the assigned track addresses in these parameters. The first assigned record address is placed into QMPCA1, the second into QMPEX1, and the remaining record addresses into ...QMPCAn, QMPEXn.

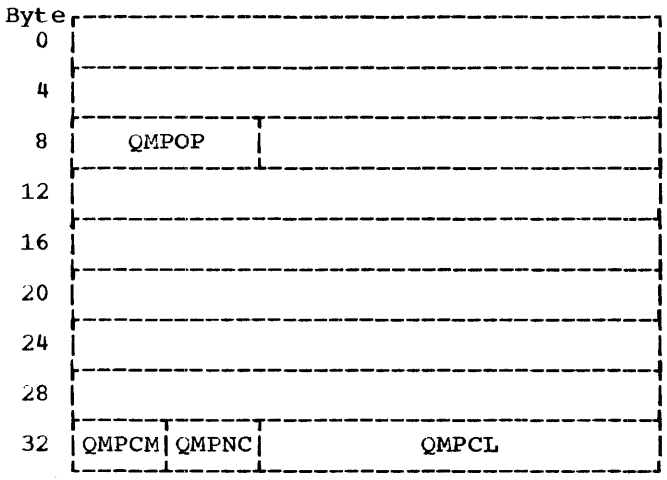


Figure 31. Table Store Subroutine Parameter Area

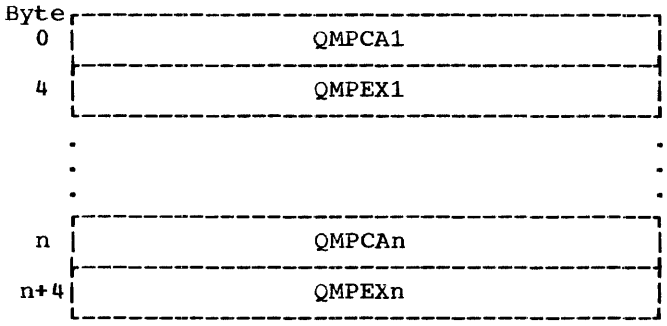


Figure 32. QMPCA-QMPEX List

	Input Parameters					
	QMPOP	QMPCM	QMPNC	QMPCL	QMPCA	QMPEX
Assign and start	00	X		X	X	X
Assign	01	X		X	X	X
Write and assign	02	X	X	X	X	X
Write	03		X	X	X	X
Read	04		X	X	X	X

Figure 33. Table Store Subroutine Parameter Requirements

Disposition and Unallocation Subroutine

The disposition and unallocation subroutine is divided into two sections: disposition processing, which performs data set dispositions specified in the DISP field of DD statements, and device availability processing, which makes the associated devices available for allocation to the next job step. Control enters the disposition and unallocation subroutine from the step termination routine and the job termination routine. In all cases, disposition processing is performed, followed by device availability processing. A message containing the data set name, its disposition, and the serial numbers of the volume (or volumes) in which it is contained, is always issued to the programmer.

ENTRY FROM THE STEP TERMINATION ROUTINE

When the step termination routine passes control to the disposition and unallocation subroutine (Chart 52), it provides pointers to the TIOT and SIOT of a data set. The disposition field of the SIOT indicates the disposition to be performed.

Disposition Processing

Dispositions that may have been specified in the DD statement are DELETE, KEEP, PASS, CATLG, and UNCATLG.

If the disposition is DELETE and the data set is cataloged, and if the JFCB housekeeping routine obtained volume information from the catalog, the UNCATALOG macro instruction is issued. If the devices containing the data set are not direct access devices, no SCRATCH macro instruction is issued. If the devices are direct access devices, a check is made to determine if the SCRATCH macro instruction can be issued. It can be issued if one of the following conditions exists:

- All volumes containing the data set are mounted.
- All volumes containing the data set are not mounted, but at least one dismountable volume is mounted.

If neither of these requirements is met, an error message is issued.

If the disposition specified in the DD statement is KEEP, the disposition subroutine issues a message to the operator and passes control directly to device availability processing.

If the disposition is PASS, no message is issued to the operator. Control is passed to device availability processing.

If the disposition is CATLG, the disposition subroutine determines if the data set is already cataloged. If not, the CATALOG macro-instruction is issued. If it is cataloged, a further check is made to determine whether its volume list was altered during execution of the job step. (The data management OPEN, CLOSE, or EOV routines may have altered the volume list.) If the volume list was altered, a RECATALOG macro instruction is issued. If the volume list was not altered, control passes directly to device availability processing.

An UNCATLG disposition causes an UNCATALOG macro instruction to be issued.

If a disposition is not specified in the DD statement, but if the SYSOUT keyword is specified, control returns directly to the step termination routine.

When neither a DISP nor a SYSOUT keyword is specified in the DD statement a check is made to determine if an entry for the data set exists in the passed data set queue (PDQ), and if so, the status indicator in that entry is checked. If the status is old (the data set was created by a previous step or job), a KEEP disposition is assumed. If the status is new, a DELETE disposition is assumed. If there is no entry for the data set in the PDQ, the status indicator in the step input/output table is examined, and as in the conditions for a PDQ entry, either a KEEP or DELETE disposition is assumed.

If the job step has been abnormally terminated, the conditional disposition (third parameter for DISP keyword) is honored instead of the normal disposition (second parameter). Possible conditional dispositions are: DELETE, KEEP, CATLG, and UNCATLG. If one of these specifications is present, it is resolved in the same manner as normal disposition. If there is no specification for the conditional disposition, the normal disposition will be honored (as above).

Device Availability Processing

After the disposition of a data set is determined and processed, the device availability portion of the disposition and unallocation subroutine is entered. First, a check is made to determine if the operator has issued a VARY or UNLOAD command. If so, the status of the device is changed, and a message indicating that the command was processed is issued to the operator.

When there are no pending VARY or UNLOAD commands or when these commands have been processed, tests are made to determine if any of the volumes containing the data set can be dismounted. Dismount messages are issued for any that can be dismounted. The following volumes are not dismountable:

- Public volumes.
- Volumes on system residence or RESERVED devices.
- Volumes on permanently resident devices.
- Volumes whose status is RETAINED.
- Volumes on system input or system output devices.
- Volumes containing data sets with PASS dispositions.

The addresses of appropriate UCBs are obtained from the TIOT, and the status of the devices used is changed to ALLOCATABLE. When device availability processing of a data set is completed, the disposition and unallocation subroutine returns control to the step termination routine.

ENTRY FROM THE JOB TERMINATION ROUTINE

When the job termination routine passes control to the disposition and unallocation subroutine (Chart 52), a test is made for special disposition processing. If the step is to be restarted, the disposition of OLD data sets is changed to KEEP; the disposition of NEW data sets is changed to KEEP for a checkpoint restart, to DELETE for a step restart.

Only two types of data sets remain to be processed:

- Data sets that were passed but were not received.
- Data sets contained on volumes that were retained but to which reference was never made.

Each time the job termination routine passes control to the disposition and unallocation subroutine, it passes a pointer to an entry in the PDQ describing a data set that was passed but not received.

If the job has been abnormally terminated (job failed bit is on), the conditional disposition stated for this data set must be honored. The SIOT for this data set is read into main storage, and the conditional disposition checked. The specified disposition is then processed in the same manner as when entry is from the step termination routine.

If no conditional disposition was specified, only two dispositions are possible: DELETE and KEEP. If the data set existed before the first time it was passed in this job, a KEEP disposition is assigned; otherwise, a DELETE disposition is assigned. These dispositions are processed in the same manner as when entry is from the step termination routine.

When the job termination routine has scanned all PDQ entries for a job, it enters the disposition and unallocation subroutine, but provides no pointer to a PDQ entry. The disposition and unallocation subroutine scans <u>all</u> UCBs and issues dismount messages for any dismountable volumes on devices whose UCB contains the current job identification. Control is then returned to the job termination routine.

Appendix B: Tables and Work Areas

This appendix contains descriptions and formats of major tables and work areas that are used by job management routines and that are not described in the body of this publication. Most table entries are self-explanatory. Those entries that require further explanation are described with each table. Tables are shown here four or eight bytes wide for convenience, but are not necessarily drawn to scale.

The length of each field of the tables is given in bytes in the upper right corner of the field, and each table is limited to a 176-byte length by convention. The tables are presented in the following alphabetical order:

Account control table
Device mask table
Dsname table
Generation data group (GDG) bias count table
In-Stream procedure work area
Job control table
Job file control block
New reader or writer table
Passed data set queue
Step control table
Step input/output table
System message block
Volume table

Auxiliary storage addresses appearing in the tables are relative track addresses (TTRs), in relation to the beginning of the SYS1.SYSJOBQE data set, whether the table is stored into main storage or into auxiliary storage by the table store subroutine and the I/O supervisor. All TTRs are three bytes long and begin on a fullword boundary. The format of all storage addresses appearing in the following tables is:

Relative track address (2)	Relative record address (1)	Table ID or Not used (1)

Account Control Table

The account control table (ACT), shown in Figure 34, contains accounting information obtained from JOB and EXEC statements. This information is made available to user accounting routines. One or more ACTs are created for each job. The job routine of the reader/interpreter creates one ACT for each JOB statement, and the execute routine creates an ACT whenever the accounting (ACCT) parameter with its subsequent information is specified on an EXEC statement. The "number of accounting fields" entry contains the number of elements of accounting information specified in the ACCT parameter of the EXEC statement, or in the first positional parameter of the JOB statement (see IBM System/360 Operating System: Job Control Language Reference). ACTs are stored by the table store subroutine.

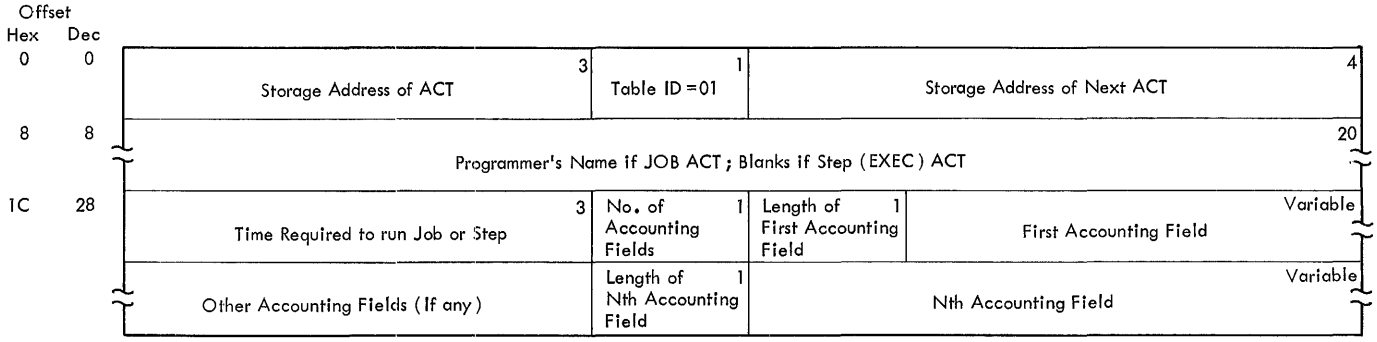


Figure 34. Account Control Table

Device Mask Table

The device mask table (DMT), shown in Figure 35, is built at SYSGEN time, and permits system access to the unique group of I/O devices represented by one unit name. This group may consist of any combination of device types or device numbers, and will be unique for any user's system. The user may determine specific device assignment bit patterns for his system from a symbolic listing taken after system generation. There is one table entry for each esoteric or generic name or for each direct access device. Within each entry, the bit pattern signifies the devices associated with a particular device name. The bit pattern within any entry is extended in fullword increments when the number of devices exceeds 32 or a multiple of 32. The entry status byte, bit 0, if 1, signifies that the group of devices is a homogeneous group.

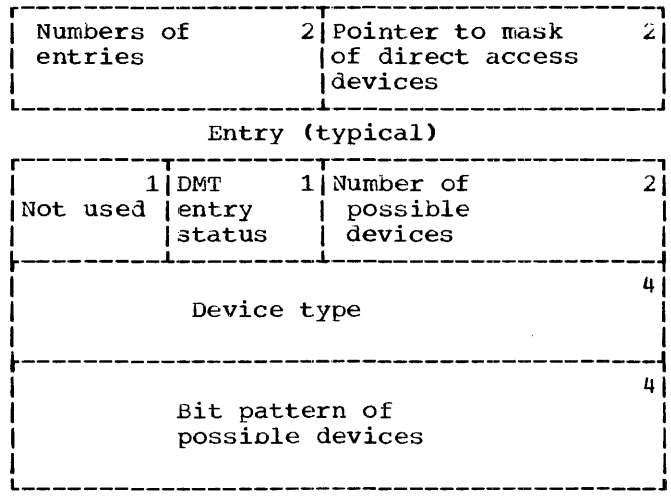


Figure 35. Device Mask Table

At SYSGEN time, device type codes are obtained from tables internal to the SYSGEN program, or are generated, and placed in the device mask table. The DMT is used as a source of device-type codes for the device name table (DNT) (see IBM System/360 Operating System: System Control Blocks). During device allocation, these codes are used as search keys to gain access to the DMT for device groups or single devices.

DSNAME Table

The dsname table, (see Figure 36), contains the volume reference data set names for one step as found in the DD statement. The table is created by the DD routine of the interpreter for each job step. One entry is made in the dsname table for each DD statement containing the VOLUME=REF=dsname parameter.

The step control table (SCT) points to the dsname table, and also contains a count of the total bytes occupied in the dsname table by dsnames for the current step. The SIOT for each data set also contains a pointer to the dsname table entry for this SIOT before volume resolution and a pointer to the volume table (VOLT) after volume information has been resolved.

The dsname table is used by the JFCB housekeeping routine of the initiator/terminator to retrieve volume information concerning data sets referred to by data set name in the DD statement VOLUME=REF parameter. The dsname table is fragmented into 176-byte blocks before being stored, prior to job step execution.

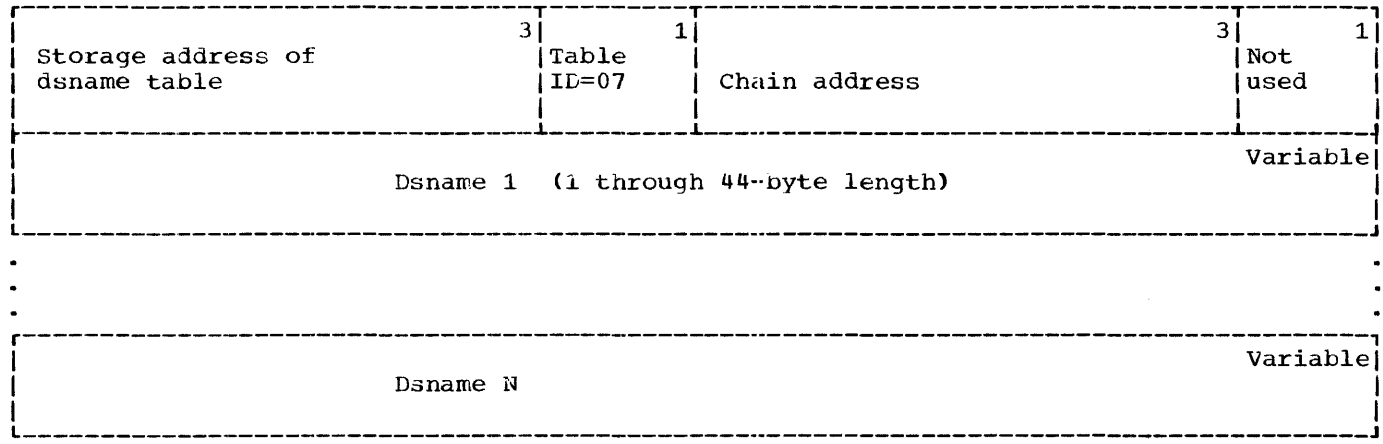


Figure 36. Dsname Table

Generation Data Group Bias Count

The generation data group (GDG) bias count table, shown in Figure 37, makes GDG information available to the data management portion of the system, and allows the user to refer to a particular GDG member by the same number in different steps of the same job. The programmer refers to GDG members serially from the start of a job, but data management refers to GDG members serially from the last-cataloged member. The last member cataloged in a previous job, if any, is referenced as member number zero. The programmer will refer to the first new data set in the present job as number +1. This table is used to convert a reference that is relative to the start of the present job, as specified by the programmer, to a reference that is relative to the last-cataloged member, as required by data management.

An entry to the GDG bias count table is created by the GDG single processing routine of JFCB housekeeping when a single GDG is requested by the user. When a step is completed by JFCB housekeeping, the JFCB housekeeping control routine transfers the GDG work bias byte to the GDG bias byte location if the value of the work byte is greater than that of the bias byte. In subsequent steps of the same job, any reference by the programmer to a GDG member will be decremented by the value of the bias count, which is contained in the GDG bias byte, to obtain a corrected member number for data management reference.

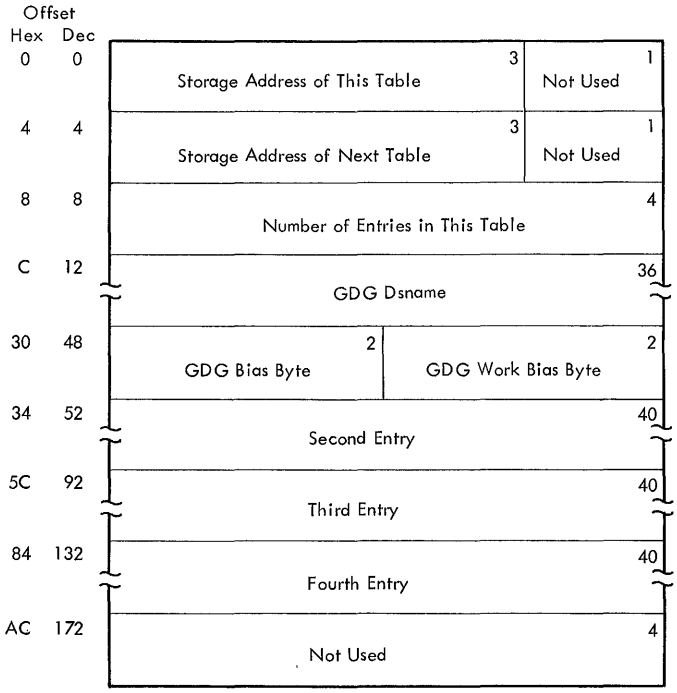
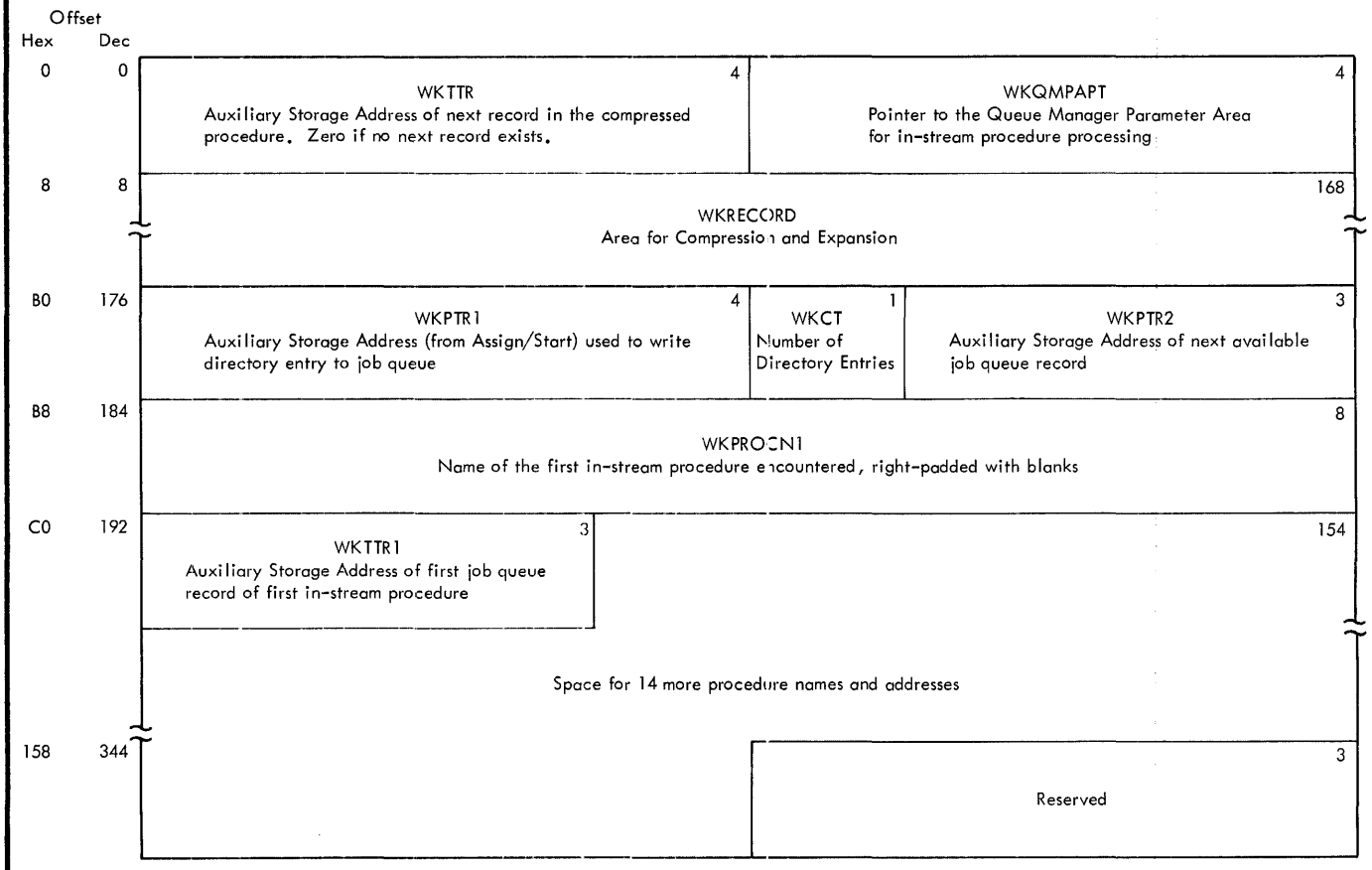


Figure 37. GDG Bias Count Table

In-Stream Procedure Work Area

The 352-byte work area shown in Figure 38 functions as two 176-byte halves in processing procedures found in the job stream. The first half, the work buffer, is used in compressing and expanding procedure statements. The second half, the directory, is used to store from one to fifteen entries, each containing the name of a procedure and the auxiliary storage address of the first job queue record of that procedure. Directory entries are created as in-stream procedures are encountered in a job input stream and processed. Storage for the area is obtained when the first procedure is processed, and is freed when the next JOB statement is read.



• Figure 38. In-Stream Procedure Work Area

Job Control Table

The job control table (JCT), shown in Figure 39, is created by the job routine of the reader/interpreter upon receipt of a job statement. It contains information taken from the job statement, and also storage addresses of major tables. After all steps within a job have been interpreted, the JCT is stored by the interpreter. The JCT is used by the initiator/terminator in preparing a job step for execution, and is stored by the step initiation routine of the initiator/terminator, before control is passed to the job step.

The JCT includes the following entries:

Job Serial Number (JCTJSRNO): Always contains 1 in the primary control program.

Job Status Indicators (JCTJSTAT):

Bit 0: The job library indicator contains a 1 if a JOBLIB DD statement is included with the job.

Bit 1: is set to 1 if the job is flushed because of an error condition.

Bit 2: is set to 1 if the job step is cancelled by condition codes.

Bit 3: is set to 1 if the job step is flushed because of an error condition.

Bit 4: The ABEND indicator contains a 1 if one or more steps have been terminated through the ABEND routine.

Bit 5: The job-failed indicator contains a 1 if an error condition caused the job to be terminated.

Bit 6: is set to 1 if the job includes a cataloged procedure.

Bit 7: is set to 1 for a job which does not require the mounting of volumes; it contains 0 if volume mounting is necessary.

Message Level (JCTJMGLV-1/2 byte):

Bit 0 contains 0 if message level for allocation is 0.

Bit 0 contains 1 if message level for allocation is 1.

Bits 2-3 contain 00 if message level for JCL is 0.

Bits 2-3 contain 01 if message level for JCL is 1.

Bits 2-3 contain 10 if message level for JCL is 2.

The second half-byte, JCTJPRTY (Job priority) is not used in the primary control program.

Restart Switches:

JCTRSW1:
Bit 1 contains a 1 when step termination has begun.

Bit 3 contains a 1 if a checkpoint has been taken for the step.

Bit 4 contains a 1 for a checkpoint/restart to be done.

Bit 5 contains a 1 for a step restart to be done.

Bits 6 and 7 must be zero.

JCTRSW2:
Bit 0 contains a 1 if a SYSCHK DD statement is present.

Bit 1 contains a 1 if the RD parameter is other than NC.

Bit 2 contains a 1 if the RD parameter is NR.

Bit 3 contains a 1 if the RD parameter is NC or RNC.

Bit 4 contains a 1 if the RD parameter is R or RNC.

Bit 7 contains a 1 if module IEFDSDRP has encountered an unrecoverable error.

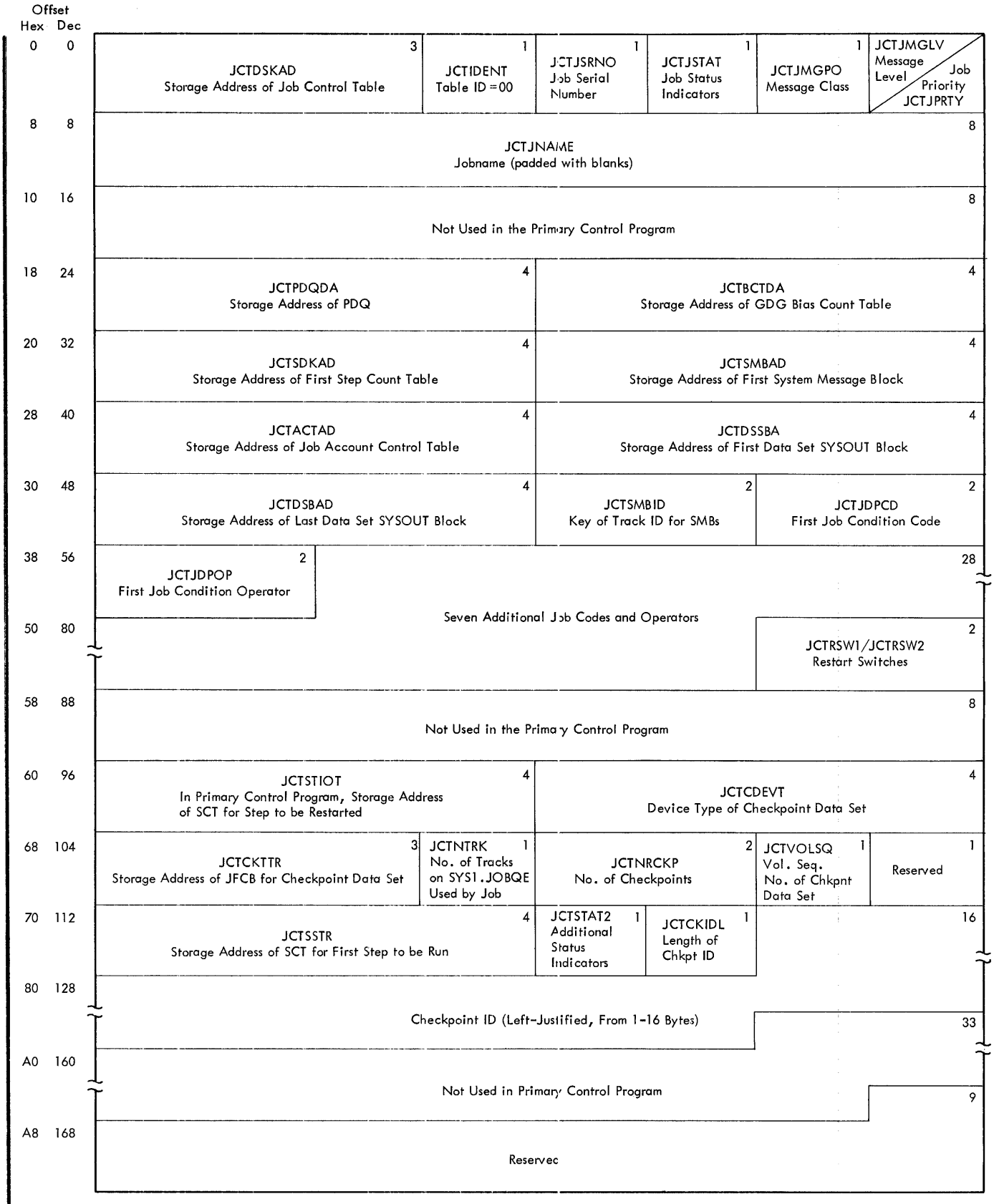


Figure 39. Job Control Table

Job File Control Block

A job file control block (JFCB) is constructed and written on auxiliary storage by the job management routines for each ddname specified in a job step. A JFCB is brought into main storage when a DCB with the corresponding ddname is opened. Information in a JFCB may be modified during OPEN. Figure 40 shows the format of the JFCB. See Figure 12 for the fields used for DD statement parameter dispositions.

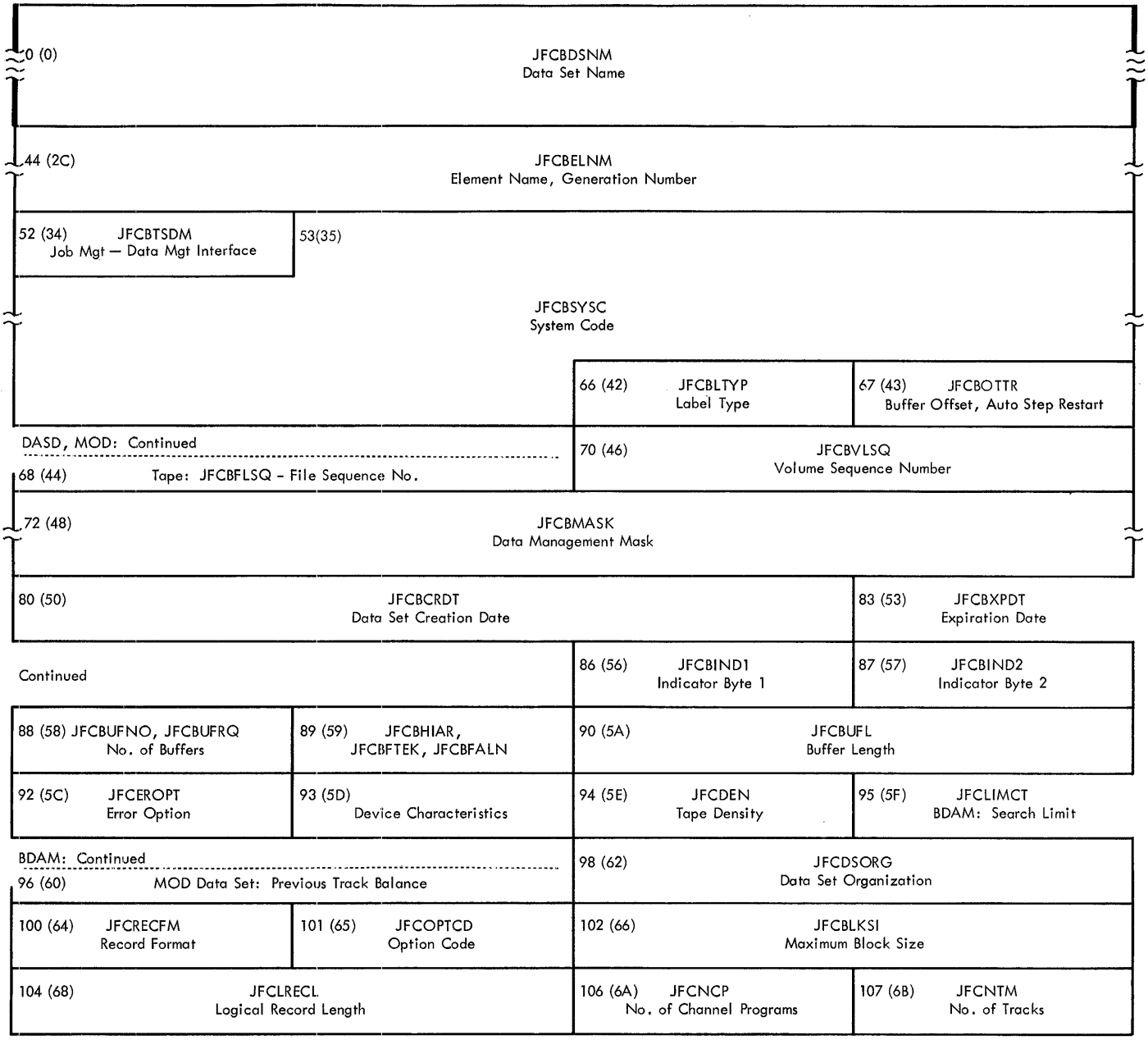


Figure 40. Job File Control Block (Part 1 of 2)

Segments

Normal 108 Segment

108 (6C) JFCRKP Relative Key Position	109 (6D) JFCCYLOF No. of Tracks	110 (6F) JFCDBUFN Reserved
112 (70) JFCINTVL Seconds of Delay		

UCS Segment

108 (6C) JFCUCSID UCS Image Name
112 (70) JFCUCSOP UCS Image Operation

	113 (71) JFCCPRI Send/Receive Priority	114 (72) JFCSOWA Size of Work Area
116 (74) Reserved	117 (75) JFCBNVOL No. of Serial Numbers	118 (76)
JFCBVOLS Volume Serial Numbers		
148 (94) JFCBEXTL Reserved	149 (95) JFCBEXAD Relative Track Address for First JFCB Extension	
152 (98) JFCBPQTY Primary Quantity of Direct-Access Storage		155 (9B) JFCBCTRI Space Parameters
156 (9C) JFCBSQTY Secondary Quantity of Direct-Access Storage		159 (9F) Reserved
160 (A0) JFCBDQTY Direct-Access Storage Required for Index		163 (A3) JFCBSPNM Split Cyl: Address of JFCB
Continued	166 (A6) JFCBABST Relative Address of First Track	
168 (A8) JFCBSBNM Main Storage Address of JFCB - Suballocate		171 (AB) JFCBDRLH Data Block Length
Continued	174 (AE) JFCBVLCT Volume Count	175 (AF) JFCBSPTN Split Cyl: No. of Tracks

• **Figure 40. Job File Control Block (Part 2 of 2)**

Master Scheduler Resident Data Area

The master scheduler resident data area is a 196-byte portion of the nucleus used as a communications area between the master scheduler and the rest of the operating system. (See Figure 41.) The CVTMSLT field of the communication vector table contains its address. In the PCP configuration of the operating system, the first 136 bytes comprise a four-byte control program header and a 132-byte buffer into which console commands are read. The buffer's first four bytes contain a V-type header address, and the last two bytes mark the end of the buffer; console messages may therefore occupy a maximum area of 126 bytes.

The remaining sixty bytes of the master scheduler resident data area constitute a system independent space known as the master common area. The two message communication fields contained within it are each used for passing indicators between two message modules. The command pointer always points to the current console command; the command is initially read into the remote command buffer at offset 8 in Figure 41, but it is moved out of the master scheduler resident data area into the local buffer for processing.

Preceding the master common area's control blocks, addresses, and pointers are six bytes of switches and flags:

- Initialization Switches

Bit	Definition	Name
0	IPL Switch	MSNIP
1	Sysout IPL	
2	Sysout Job Start	
3-4	Reserved	
5	34 Security Bit	MSCURE34
6	Queue Initialized	MSQNIP
7	Procedure Catalog Initialized	MSPNIP

- PCP System Exclusive Flags

Bit	Definition	Name
0	Console Flag	MSCONFLG
1	Cancel Flag for ABENDT	MSCANFLG
2	Rollout Flag	MSROLFLG
3	Spinoff Flag (Cancel)	MSSO
4	Display Dataset Name	MSSSDSN
5	Display Space	MSSSPACE
6-7	Reserved	

Note: Bits 4 and 5 may be used by other control programs.

- Pending Flags

Bit	Definition	Name
0	IPL Date	MSDATE
1	Partition Busy	MSPNB
2	Command Move Completed	MSCMC
3	Interpreter Command Completed	MSICR
4	System Input Control Purge Request	MSSYN
5	System Output Control Purge Request	MSSYT
6	Blank Start Pending (REQ=1, START BLANK=0)	MSBSP
7	Console Command Suppressed	MSCCS

- ECB Flags

Bit	Definition	Name
0	External Interrupt	MSEXT
1	Write to Operator	MSWTO
2	Write to Log	MSWTL
3	Pending Console Attention	MSATTN
4	System Input	MSYSIN
5	System Output	MSYSOUT
6	Master Command Routine	MSMCR
7	Summary Bit, Vary UCB Scan Required	MSSUM

- Status Flags

Bit	Definition	Name
0	Master Initialization Switch (IPL)	MSINLSW (or) MSSSSIPL
1	WTO Pending	MSWRPEN
2	Console Usage, Principal or Alternate	MSNUPSW
3	Log Purge Request	MSWRLOG
4	Reader End of File (or) Start Reader	MSREOF (or) MSSRDR
5	New Reader Pending	MSNRP
6	New Writer Pending (or) New Writer Pending (MODIFY)	MSNWP (or) MSYOUT
7	Job Notification Flag (1=YES)	MSJNF

- Fetch Flags

Bit	Definition	Name
0	Named Fetch	MSNMF
1	Defer Current Command Execution Sequence	MSCSD
2	TCB Tree Trace Fetch (LOCATE)	MSTTT
3	Auxiliary Fetch Given	MSFAX
4	Reply Bit to Request Attention	MSREPLYB
5	Pseudo-Sysout Flag	MSPSDT
6	Reserved	
7	Queue Hold-Release	MSQHR

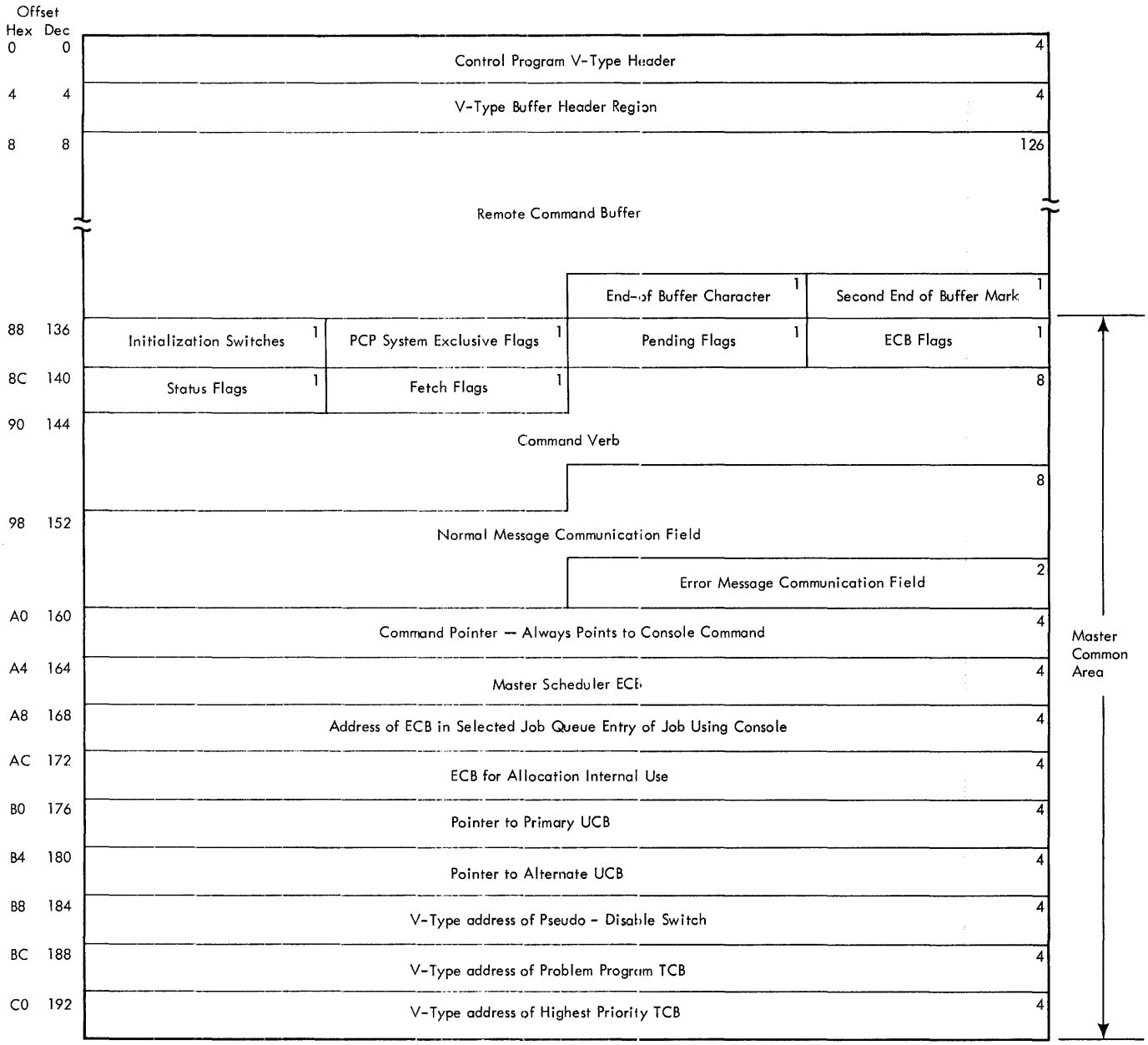


Figure 41. Master Scheduler Resident Data Area

New Reader or Writer Table

The new reader or writer table (NRWT), shown in Figure 42, is a control block that contains OPEN requirements for reader and writer routines. At initial program load time, the table is written onto auxiliary storage. The table is read into main storage from auxiliary storage and is used by the interpreter and SYSOUT routines. Each entry (except jobname) consists of an active section and an inactive section. Whether the lower or higher order part of the entry is active is indicated by a 1 in bit 0 of the flag 1 byte in the active section. When a NRWT entry is active, the data set has been opened, and the device indicated by the applicable UCB pointer is active. The currently inactive section of the entry receives information from new START commands. The table is always available in the SYS1.SYSJOBQE data set.

The bits in location Flags 1 have the following meanings:

Bit 0	I/O active/inactive
Bit 1	I/O jobname/no jobname
Bit 2	Pending start
Bit 3	Pending stop
Bit 4	Separator/no separator
Bits 5-7	Not used in primary control program

The bits in location Flags 2 have the meanings:

Bit 0	Local flag
Bit 1	I/O DSNAME/no DSNAME in START command
2-7	Not used in primary control program

Start Reader Entry

3	1	2	2
Track address	Flags 1	Flags 2 (not used)	UCB pointer
Track address	Flags 1	Flags 2 (not used)	UCB pointer

Jobname Entry

8
Jobname from START command of operator

Start Writer Entry
(One entry for each possible active class-a maximum of eight)

3	1	1	1	2
Track address	Flags 1	Flags 2	Class name	UCB pointer
Track address	Flags 1	Flags 2	Class name	UCB pointer

Cataloged Procedures Entry

3	1	2	2
Track address	Flags 1	Flags 2 (not used)	UCB pointer
Track address	Flags 1	Flags 2 (not used)	UCB pointer

Figure 42. New Reader or Writer Table

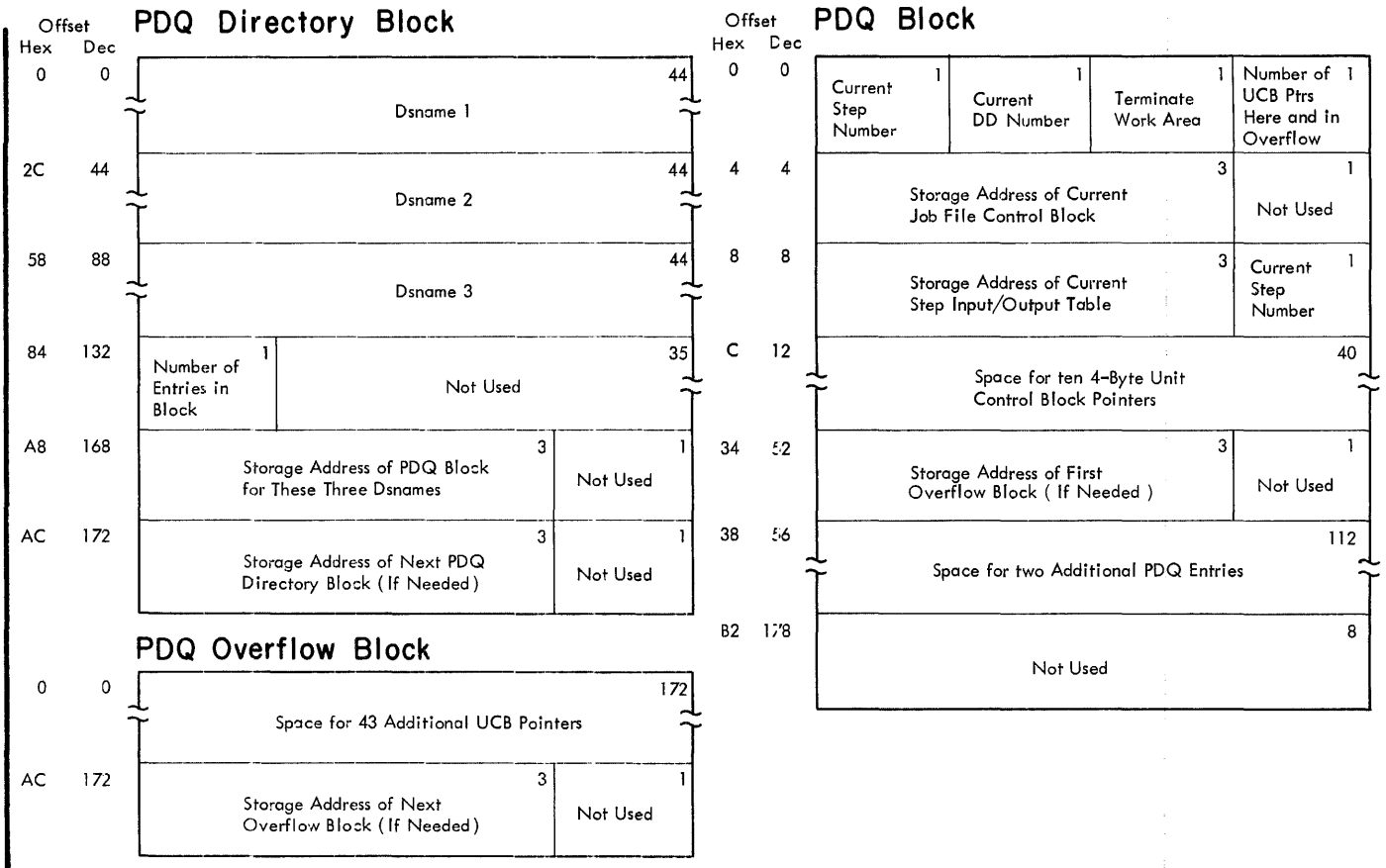


Figure 43. Passed Data Set Queue Tables

Passed Data Set Queue

The passed data set queue (PDQ), shown in Figure 43, contains information regarding previously processed data sets which have been passed from executed steps of the job, that may be referenced by subsequent steps of the same job. Each PDQ contains a set of tables, consisting of three types of blocks: the PDQ directory block, the PDQ block, and the PDQ overflow block (if required). The PDQ directory block and the PDQ block are created by the initiator/terminator JFCB housekeeping routine. The directory blocks are chained together with pointers, and each PDQ directory block also points to its respective PDQ block. If more than ten additional UCB pointers are needed for any one PDQ entry, one or more PDQ overflow blocks are added in a chain to each such PDQ block entry by allocation routines.

Initiator/terminator routines use the PDQ to obtain pointers to UCBs when allocating devices to passed data sets. Step termination routines use the PDQ to obtain UCB allocation pointers and disposition information.

When control passes to the initiator/terminator, the JFCB housekeeping routine inspects the disposition field of the SIOT for the disposition "PASS" to determine whether a new entry may be required in the PDQ.

If a PASS disposition is found and the dsname is not in the PDQ directory because it was not placed into the directory by a prior PASS, a fifty-six byte entry is made in the PDQ for this dsname. If the last PDQ directory block and PDQ block already contain the maximum number of three entries, auxiliary storage space is assigned for a new PDQ directory block and a new PDQ block, thereby providing space for three more dsname entries.

When a passed data set is to be referenced by a subsequent step in the same job, the dsname is specified in the DD statement. The JFCB housekeeping routine checks for the dsname in the PDQ directory to see if the data set was received (passed from a previous step).

If the dsname is found in the PDQ directory, the existing PDQ entry for this dsname is updated to identify the reference

as the latest reference to this dsname and the data set is marked as being received in the PDQ entry. If no entry is found, the data set must have been cataloged, so the JFCB routine searches the catalog for this dsname, assuming that this is an initial reference for this job to a cataloged data set.

Bits of the terminate work area byte of the PDQ block have the following status significance:

Bit	Significance	Status
0	Initial status	1 = old
1	Current status	1 = old
2	Pass satisfied	1 = passed 0 = received
3	SYSIN specified	1 = SYSIN
4	SYSOUT specified	1 = SYSOUT

Step Control Table

The step control table (SCT), shown in Figure 44, is used to pass control information to the DD routine of the interpreter and to the initiator/ terminator routines, which also contribute information to the table. This table is created and initialized by the execute routine of the interpreter when an EXEC statement is read. One SCT is created for each step of a job, and is stored by the interpreter control routine and the initiator/terminator step initiation routine.

When the EXEC statement includes the optional PARM field, the information is placed in a specially created SCT extension block, whose storage address is maintained in a four-byte field at offset 68 (44 hex) of the SCT. Zeroes in this field indicate that the EXEC statement provided no PARM information, and hence that no SCT extension block was created.

If the step is part of a previously cataloged procedure, the name of the step that called the procedure, if any, is entered. The following variable-content and indicator fields are included in the table:

1. Internal Step Status Indicators (offset 4 hex):
   Bit 2 contains a 1 if RD = NR is specified on the JOB or EXEC statement.
   Bit 3 contains a 1 if RD = RNC or RD = NC is specified on the JOB or EXEC statement.
   Bit 4 contains a 1 if RD = R or RD = RNC is specified on the JOB or EXEC statement.
   Bit 7 contains a 1 if an error condition caused the step to be terminated.
2. PARM Count or Step Status Code (offset 8 hex):
   a. Interpreter: The number of characters specified in the PARM parameter of the EXEC statement is placed in this entry.
   b. Initiator/Terminator: This table entry contains the condition code returned by the processing program.
3. Step Type Indicators (offset 43 hex):
   Bit 0 contains a one if the following parameter definition appears in the EXEC statement:
   PGM=*.stepname.ddname
   Bit 1 indicates SYSIN is specified (DD *).
   Bit 2 indicates SYSOUT is specified.
   Bit 3 contains a 1 if JFCB housekeeping is complete.
   Bits 4, 5, and 6 are unused in PCP. Bit 7 is reserved.
4. Extension of Internal Step Status Indicators (offset 68 hex):
   Bit 6 contains a 1 if the job has ended.
   Bit 7 contains a 1 if the GDG Bias Table needs to be updated.
5. Execute Step after ABEND or Eighth Condition Code (offset A0 hex):
   (Execute step after ABEND)
   First byte (offset A0):
   Bit 0 is not used.
   Bit 1 is not used.
   Bit 2 is not used.
   Bit 3 contains a 1 if the step is bypassed because of one or more prior ABEND macros.
   Bit 4 contains a 1 if the step is bypassed because COND=ONLY was specified and no ABEND has occurred.
   Bit 5 contains a 1 if the step was terminated by the ABEND routine during problem program execution.
   Bit 6 contains a 1 if the interpreter encountered the EVEN parameter in the COND field of the EXEC statement.
   Bit 7 contains a 1 if the interpreter encountered the ONLY parameter in the COND field of the EXEC statement.
   The remainder of the field (offset A1-A5) must contain zeros.
   (Eighth condition code)
   First two bytes (offset A0-A1) contain the eighth step condition code, or zeros.
   Third byte (offset A2) contains the eighth step condition operator, or zeros.
   Fourth through sixth bytes (offset A3-A5) contain the storage address of the eighth condition SCT, or zeros.

Step Control Table

Offset Hex	Offset Dec				
0	0	Storage address of step control table (3)	Table ID = 02 (1)	Internal step status indicators (1)	Maximum step running time (3)
8	8	PARM count or step status code (2)	Length of allocate work area or number of SIOTs (2)	Storage address of first SIOT entry (3)	Not used (1)
10	16	Storage address of allocate work area (3)	Not used (1)	Storage address of next SCT (3)	Not used (1)
18	24	Storage address of first SMB for next step (3)	Not used (1)	Storage address of last SMB for this step (3)	Not used (1)
20	32	Storage address of first ACT entry for this step (3)	Not used (1)	Storage address of volume table (3)	Not used (1)
28	40	Storage address of dsname table for this step (3)	Not used (1)	Name of step that called procedure (if any)	
30	48	Name of step that called procedure (cont'd) (8)		Stepname	
38	56	Stepname (cont'd) (8)		Relative pointer to step entry in ACT (2)	Length of volume table (2)
40	64	No. of SIOTs in this step (1)	No. of setup messages (1)	No. of JFCBs to allocate (1)	Step type indicators (1)
48	72	X'00' (1)	Hierarchy 0 region address (3)	X'01' (1)	Hierarchy 1 region address (3)
50	80	Queue address of first write-to-programmer SMB for automatic checkpoint/restart use (3)	Count of WTP SMBs for step (1)	Storage address of SCT extension block (4)	
58	88	Hierarchy 0 region size (2)	Hierarchy 1 region size (2)	Reserved (2)	Not used in PCP (2)
60	96	Not used in PCP (6)		Queue address of SIOT for PGM=*.STEPNAME.DDNAME (4)	
68	104	Extension of internal step status indications (1)	Storage address of step TIOT (3)	Programname	
70	112	Programname (cont'd) (8)		Length of dsname table in bytes (2)	First step condition code (2)
78	120	First step condition operator (1)	Storage address of first condition SCT (3)	Second through seventh step condition entries . . . . . (36)	
A0	160	Execute step after ABEND or eighth condition code (6)		Reserved (2)	

SCT Exension Block

Hex Offset	Offset Dec		
0	0	TTR of this record (3)	ID = X'0C' (1)
4	4	Parameter value	
68	68	Reserved	
80	80		

Figure 44. Step Control Table and SCT Extension Block

Step Input/Output Table

The Step Input/Output Table (SIOT), shown in Figure 45, makes DD statement information available to the initiator/terminator for use as a source of information for the TIOT and for providing DD information to allocation and disposition routines. When a DD statement is read, the interpreter creates a new SIOT and places the DD information into it. The individual bits of indicator bytes 56 through 59 and byte 92 in the SIOT are set to one to indicate the following conditions:

BYTE 55: Scheduler Data Set Disposition Switches (SCTSDISP)

- Bit 0 Nonshareable volume
- Bit 1 Retain volume
- Bit 2 Private volume
- Bit 3 Pass data set
- Bit 4 Keep data set
- Bit 5 Delete data set
- Bit 6 Catalog data set
- Bit 7 Uncatalog data set

BYTE 56: Status Byte 1 (SCTSBYT1)

- Bit 0 Dummy data set
- Bit 1 SYSIN data set
- Bit 2 Split (primary)
- Bit 3 Split (secondary)
- Bit 4 Suballocate
- Bit 5 Parallel mount indicator
- Bit 6 Unit affinity
- Bit 7 Unit separation

BYTE 57: Status Byte 2 (SCTSBYT2)

- Bit 0 Channel affinity
- Bit 1 Channel separation
- Bit 2 Volume affinity
- Bit 3 JOBLIB DD statement
- Bit 4 Unlabeled
- Bit 5 Nonstandard label
- Bit 6 Defer mounting
- Bit 7 Received data set

BYTE 58: Status Byte 3 (SCTSBYT3)

- Bit 0 Volume reference is dsname
- Bit 1 SYSIN expected (procedures only)
- Bit 2 No associated volume serial in volume table
- Bit 3 Intra-step suballocate
- Bit 4 SYSOUT was specified
- Bit 5 New data set
- Bit 6 Modified data set
- Bit 7 Old data set

BYTE 59: Status Byte 4 (SCTSBYT4)

- Bit 0 Set by reader/interpreter to indicate GDG single
- Bit 1 SIOT created for GDG all
- Bit 2 Volume serial was found in passed data set queue (PDQ)
- Bit 4 Step processed
- Bit 5 Intra-step volume affinity
- Bit 6 Data set is in PDQ
- Bit 7 1 = old or modified data set
  0 = new data set

BYTE 92: Conditional Disposition Status Byte (SIOTALTD)

- Bits 0-2 Reserved
- Bit 3 This bit is set at Restart time to indicate that this DD is not private.
- Bit 4 Keep data set
- Bit 5 Delete data set
- Bit 6 Catalog data set
- Bit 7 Uncatalog data set

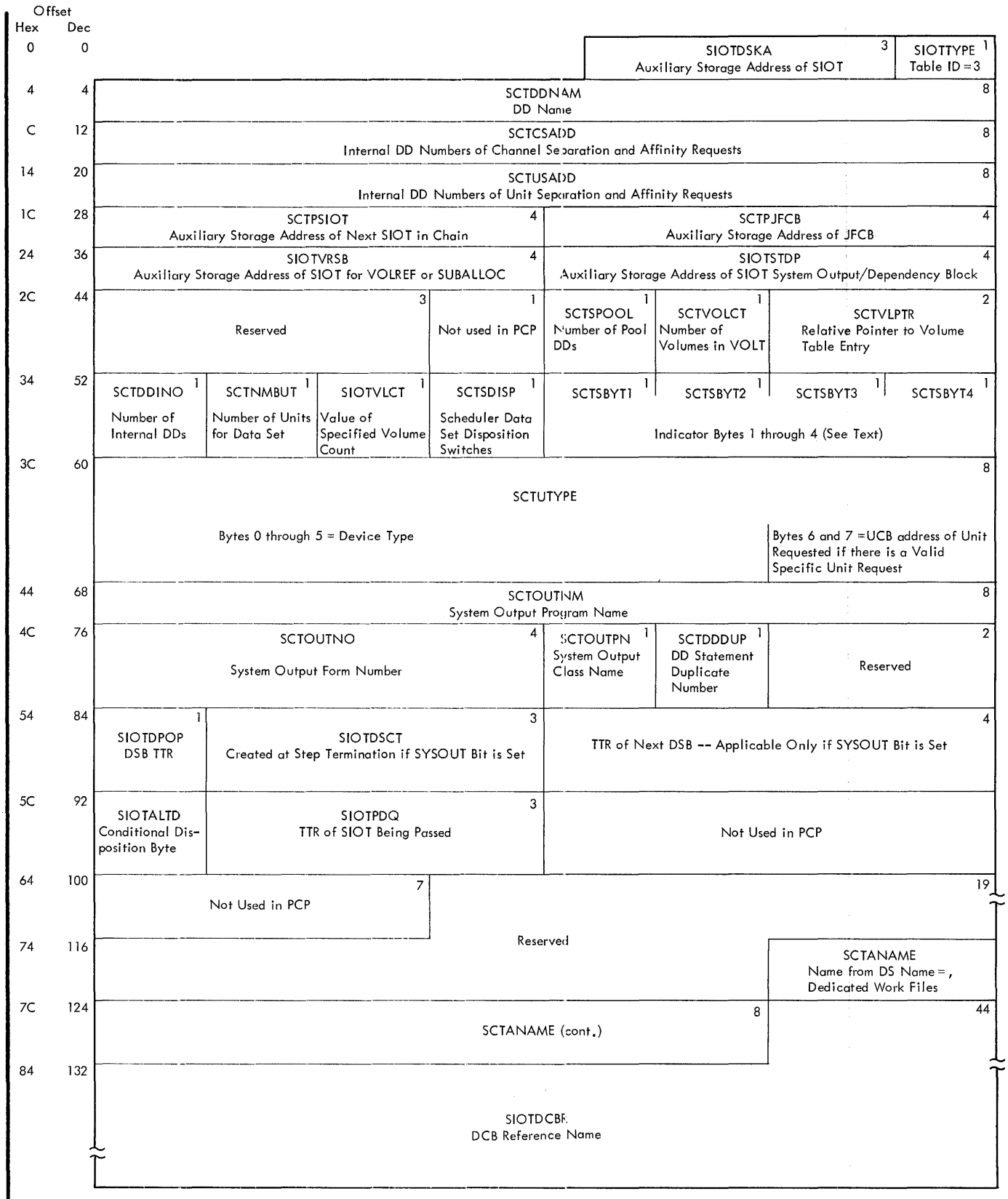


Figure 45. Step Input/Output Table

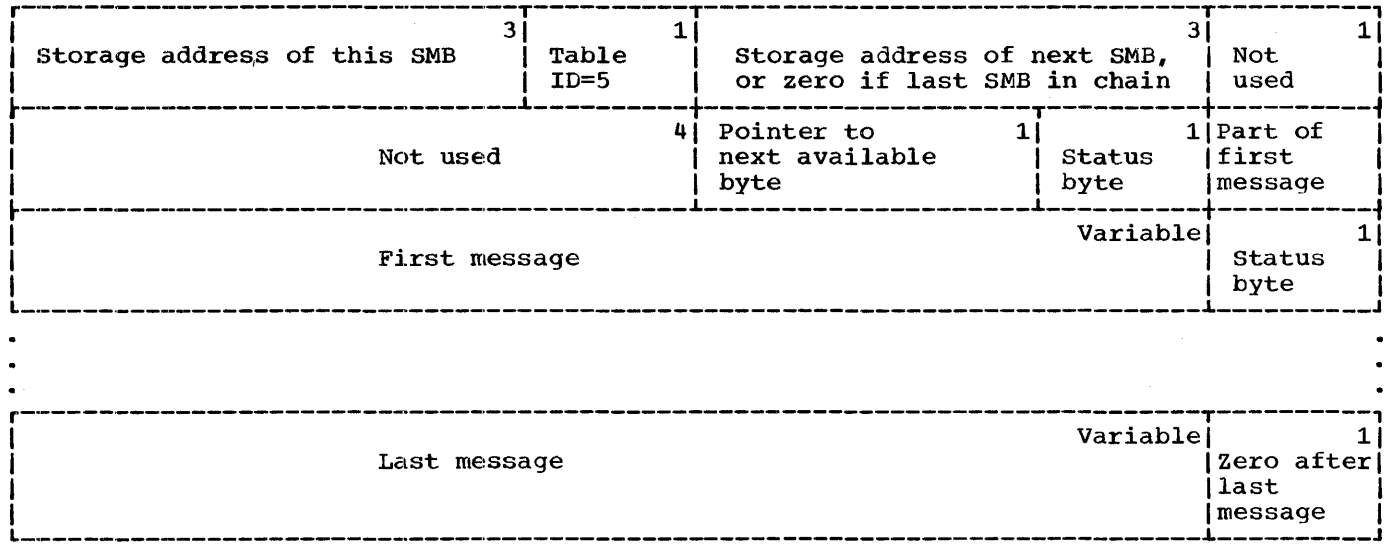


Figure 46. System Message Block

System Message Block

The system message block (SMB), shown in Figure 46, temporarily stores all control statements, programmer messages, and diagnostic error messages before they are printed via the system output writer routine. The interpreter control routine creates and initializes one or more SMBs for each job step. Initiator/terminator routines also may add messages to the SMB. The chain address of the next SMB is given in bytes 4 through 6 of each table but the last, resulting in a chain of SMBs for each job. The status byte of each block concerns the following block, and contains the message length, zero if there are no more messages, or all ones if a data set block follows.

Volume Table

The volume table (VOLT), shown in Figure 47, consists of a series of chained blocks, and contains the list of volume serial numbers to be used in a given step. Use of the list reduces the number of times that the SYS1.SYSJOBQE data set must be referenced during allocation. The table is built by the DD routine for each step, and is modified by the JFCB housekeeping routine. The maximum extent of each block of the table is 176 bytes, and the maximum number of volumes listed per block is 28.

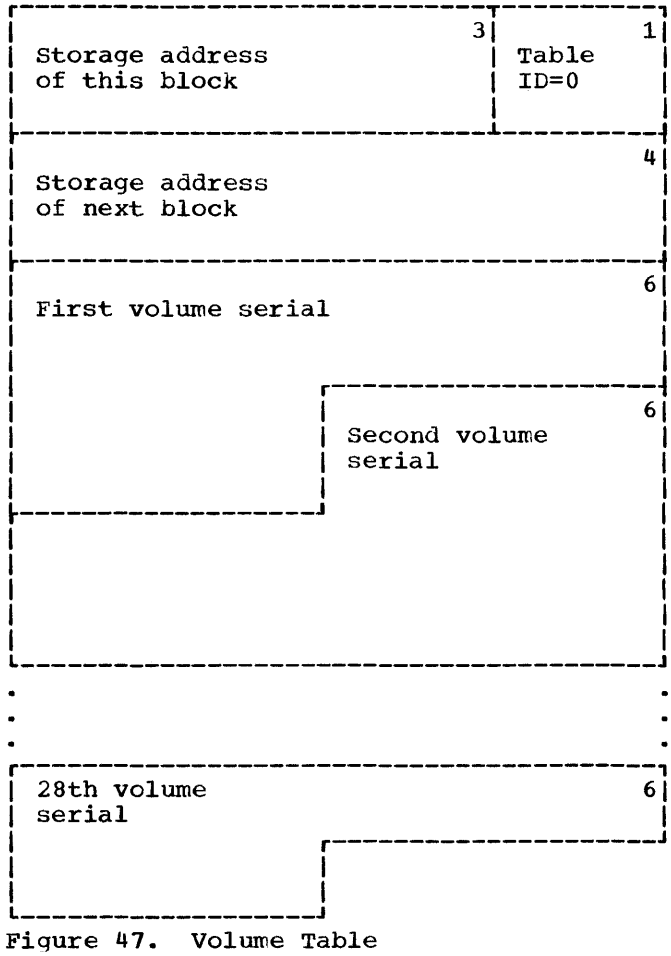


Figure 47. Volume Table

Write-to-Programmer Control Block

The write-to-programmer control block (WTPCB) depicted in Figure 48 is a 16-byte area containing the information required for adding programmer messages to the processing program's message class output data set. This block is located through a pointer contained in the Job Step Control Block (JSCB) which, in turn, is located through a pointer contained in the Task Control Block (TCB). Conditions reflected by the presence of 1 (switch on) in each bit of the WTPFLGSA field are:

BIT	CONDITION
0	Job Queue I/O problem.
1	"Limit exceeded" message issued.
2	Step contains SYSOUT.
3	Return from WTP third load to second load.
4	"No more SMBs" message issued.
5	Last SMB allowable for this job has been used.
6	WTP has been invoked for this step.
7	Routing code other than WTP encountered.

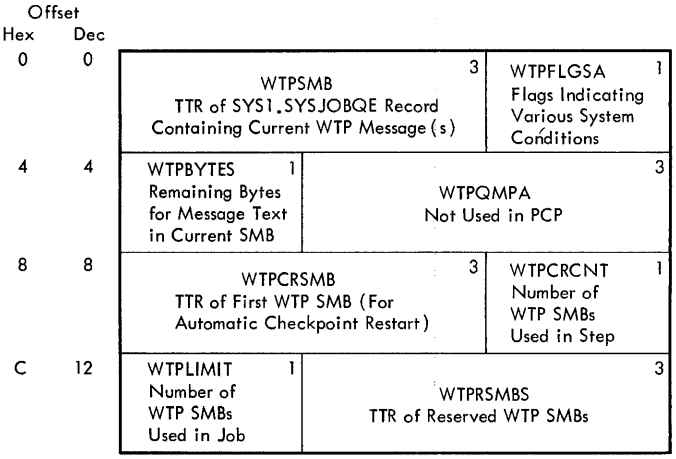


Figure 48. Write-To-Programmer Control Block

Appendix C: Load Modules and Assembly Modules

This appendix lists job management load modules and indicates the assembly modules that are processed by the linkage editor into each load module during system generation. Included is a separate list that shows the load modules in which each assembly module is contained.

Job management routines for sequential scheduling systems are packaged in three configurations: 18K, 44K, and 100K (where K is 1024 bytes of main storage). The numbers represent the maximum amount of main storage occupied by job management routines and work areas at any time. All three job management configurations function identically but differ in both the number of their load modules and the number of assembly modules within each load module. Job management routines occupy the dynamic portion of main storage alternately with processing programs, and therefore these size designations bear a direct relationship to the main storage required for each configuration.

Load Modules

In each configuration, all load modules are contained in three data sets: SYS1.NUCLEUS, SYS1.SVCLIB, and SYS1.LINKLIB. These data sets also contain other parts of the control program. The load modules in the first two data sets remain the same for all three job management configurations, but the SYS1.LINKLIB data set contains a different set of load modules for each configuration, depending on which one was selected at system generation time. In the 18K configuration, LINKLIB contains 56 load modules; in the 44K configuration, it contains 42 load modules; and in the 100K configuration, 37 load modules.

Charts 54 through 56 show the control flow among load modules. The decision to transfer control (XCTL) to a particular succeeding load module is made in the previous load module. Each subsequent module loaded in response to an XCTL macro instruction is read into main storage directly over the previous load module. Such load modules are read into the low-numbered end of the dynamic, or problem-program, area of main storage.

Modules that are brought into storage with LINK macro instructions and LOAD macro instructions occupy separate storage areas within the problem program area; such modules are shown on the control-flow charts. Because storage is used in this manner, the load module lists may be used with Charts 54, 55 or 56 to determine the approximate layout of main storage at different times during the execution of job management routines. Other items present in the problem program area at the same time as the load modules are not shown on the control flow charts because, although these items are necessary, control is not passed among them. They are, generally, the tables and control blocks, work areas, access methods, buffers, and register save areas.

In the following load module lists, entry points are shown if a load module contains more than one assembly module. If only one assembly module is named, the entry point is the same as the assembly module's control section (CSECT) name given in the Assembly Modules and Control Sections table in this appendix.

LOAD MODULES CONTAINED IN THE SYS1.NUCLEUS DATA SET

The load modules and assembly modules in the following list are contained in the SYS1.NUCLEUS data set, and are always present in the nucleus, or fixed area of main storage, regardless of the job management configuration.

Load Module Name: IEANUC01

Assembly Modules:

IEEBC1PE	External interrupt routine.
IEECIR01	Console interrupt routine.
IEERSC01	Master scheduler buffers, switches, input/output block (IOB), event control block (ECB), channel control word (CCW), and data extent block (DEB). This load module forms master scheduler resident main storage in the nucleus area when the primary or alternate console (1052) is used.
IEERSR01	Master scheduler buffers, switches, IOB, ECB, CCW, and DEB. This load module forms master scheduler resident main storage in the nucleus area when the composite console is used.
IEFDPOST	Unsolicited interrupt routine.
IEFKRESA	Table store subroutine work area.
IEFWTP0A	Write-to-programmer control block (WTPCB) and job step control block (JSCB).

LOAD MODULES CONTAINED IN THE SYS1.SVCLIB DATA SET

The load modules and assembly modules in the following list are contained in the SYS1.SVCLIB data set, and are called in response to SVC instructions.

Load Module Name: IGC0003D
Assembly Modules:
IEEMXC01 Master command EXCP routine (Part 1) -- primary/alternate console.
IEEMXR01 Master command EXCP routine (Part 1) -- composite console.

Load Module Name: IGC0003E
Assembly Modules:
IEEWTC00 Write-to-operator (WTO) routine -- primary/alternate console.
IEEWTR00 Write-to-operator (WTO) routine -- composite console.

Load Module Name: IGC0003F
Assembly Module:
IEEBH1PE Not used in sequential scheduling system.

Load Module Name: IGC00090
Assembly Module:
IEFXMPCP Transient queue manager I/O and record assignment routine. Used by WTP.

Load Module Name: IGC0103D
Assembly Modules:
IGC0103D Command processing routine for 'MOUNT, VARY ONLINE/OFFLINE, and UNLOAD. This routine issues an XCTL to IGC0203D if command is other than listed.'
IGC0203D Command processing routine for 'DISPLAY JOBNAMES, STOP JOBNAMES, CANCEL' (SHIFT command not used primary control program.)

Load Module Name: IGC0103E
Assembly Modules:
IEEWTC01 Write-to-operator-with-reply (WTOR) routine -- primary/alternate console.
IEEWTR01 Write-to-operator-with-reply (WTOR) routine -- composite console.

Load Module Name: IGC0203E
Assembly Module:
IEFWTP00 Write-to-programmer (WTP) initialization.

Load Module Name: IGC0303E
Assembly Module:
IEFWTP01 Write-to-programmer (WTP) message processing.

Load Module Name: IGC0403E
Assembly Module:
IEFWTP02 Write-to-programmer (WTP) error routine.

MODULES CONTAINED IN THE SYS1.LINKLIB DATA SET

The load modules and assembly modules in the following lists are contained in the SYS1.LINKLIB data set. Separate lists are provided for each of the three Job Management packaging configurations. The load modules within each configuration and the assembly modules within each load module are listed in alphameric order.

Any load module which contains IEFACTLK, IEFACTRT, and IEFWAD, may contain instead IEFACTFK if the system generation option for no accounting routine is specified.

18K CONFIGURATION

Load Module Name: DEVNAMET
Entry Point: DEVNAMET
Assembly Module:
IEFWMAS1 Device Name Table.

Load Module Name: DEVMASKT
Entry Point: DEVMASKT
Assembly Module:
IEFWMSKA Device Mask Table.

Load Module Name: IEEFAULT
Alias: IEEGK1GM
Assembly Module:
IEEGK1GM Fault routine, issues Master Scheduler messages.

Load Module Name: IEEJFCB
Alias: IEEIC3JF
Assembly Module:
IEEIC3JF Contains preformatted JFCB for one START command.

Load Module Name: IEESET
Alias: IEEGES01
Assembly Module:
IEEGES01 Master Scheduler SET Command routine.

Load Module Name: IEESJFCB
Alias: IEEIC2NQ
Entry Point: IEEIC2NQ

Assembly Module:
IEEIC2NQ Saves START command JFCBs.
IESQMSSS Table Store subroutine.

Load Module Name: IEESTART
Alias: IEEIC1PE
Entry Point: IEEIC1PE
Assembly Modules:
IEEREADER Start Reader routine.
IEESTART Process START and STOP WTR commands.
IEEWRITER Start Writer routine.

Load Module Name: IEETIME
Alias: IEEQOT00
Assembly Module:
IEEQOT00 Sets time and date.

Load Module Name: IEFALOC1
Alias: IEFXA
Alias: IEFXJ000
Entry Point: IEFXA
Assembly Modules:
IEFCVFAK Linkage to IEFMCVOL
IEFQMSSS Table Store subroutine.
IEFWAFAK Linkage to IEFWA000 (in IEFALOC2 load module).
IEFWCFAK Linkage to IEFWCIMP (in IEFALOC3 load module).
IEFXAMSG Contains Initiator/Terminator messages.
IEFXCSSS Allocation Control routine.
IEFXJIMP Allocation Error Recovery routine.
IEFXJMSG Contains Initiator/Terminator messages.
IEFYNFAK Linkage to IEFYNIMP (in IEFSTERM load module).
IEFYSSMB Message Enqueuing routine.

Load Module Name: IEFALOC2
Alias: IEFWA000
Entry Point: IEFWA000
Assembly Modules:
IEFDEVPT Device bit pattern.
IEFSCAN Bit pattern scan routine.
IEFSD006 Converts record number to logical track address (TTR).
IEFSGOPT System generation option indicators.
IEFV15XL Allocation Error routine.
IEFWA000 Demand Allocation routine.
IEFWCFAK Linkage to IEFWCIMP (in IEFALOC3 load module).
IEFWMSKA Device mask table.
IEFWSWIN Passes control to Decision Allocation or Automatic Volume Recognition (AVR) routine.
IEFXJFAK Linkage to IEFXCSSS (in IEFALOC1 load module).
IEFXVFAK Linkage to IEFXV001 (in IEFALOC4 load module).
IEFX300A Device Strikeout routine.
IEFX5FAK Linkage to IEFX5000L (in IEFX5000 load module).

Load Module Name: IEFALOC3
Alias: IEFWC000
Entry Point: IEFWC000
Assembly Modules:
IEFWCIMP TIOT Construction routine.
IEFWDFAK Linkage to IEFWD000 (in IEFALOC4 load module).
IEFXH000 Separation Strikeout routine.
IEFXJFAK Linkage to IEFXCSSS (in IEFALOC1 load module).

Load Module Name: IEFALOC4
Alias: IEFWD000
Alias: IEFXV001
Entry Point: IEFWD000
Assembly Modules:
IEFCVFAK Linkage to IEFMCVOL.
IEFDEVPT Device bit pattern.
IEFQMSSS Table Store subroutine.
IEFSCAN Bit pattern scan routine.
IEFSD006 Converts record number to logical track address (TTR).
IEFV15XL Allocation Error routine.
IEFWD000 External Action routine.
IEFWD001 Message directory for External Action routine.
IEFXKIMP Allocation Error Non-recovery routine.
IEFXKMSG Contains Initiator/Terminator messages.
IEFXTFAK Linkage to IEFXT000 (in IEFALOC5 load module).
IEFVMSG Automatic Volume Recognition (AVR) Message routine.
IEFXVNSL Automatic Volume Recognition (AVR) Nonstandard Label routine.
IEFXV001 Automatic Volume Recognition (AVR) routine.
IEFXV002 AVR Volume Serial Number Reading routine.
IEFX1FAK Linkage to IEFXJIMP (in IEFALOC1 load module).
IEFX2FAK Linkage to IEFX5000 (in IEFALOC2 load module).
IEFX3FAK Linkage to IEFWCIMP (in IEFALOC3 load module).
IEFX300A Device Strikeout routine.
IEFYNFAK Linkage to IEFYNIMP (in IEFSTERM load module).
IEFYSSMB Message Enqueuing routine.

Load Module Name: IEFALOC5
Alias: IEFXT000
Entry Point: IEFXT000
Assembly Modules:
IEFCVFAK Linkage to IEFMCVOL
IEFQMSSS Table Store subroutine.
IEFSD006 Converts record number to logical track address (TTR).
IEFWDFAK Linkage to IEFWD000 (in IEFALOC4 load module).
IEFW41SD Exit to IEK04FAK (in this load module).
IEFXKIMP Allocation Error Non-recovery routine.
IEFXKMSG Contains Initiator/Terminator messages.
IEFXTDMY Queue Overflow routine.

IEFXTMSG Contains Initiator/Terminator messages.
IEFXT00D Space Request routine.
IEFXT002 TIOT Compression routine.
IEFXT003 DASDM Error Recovery routine.
IEFYNFAK Linkage to IEFYNIMP (in IEFSTERM load module).
IEFYSSMB Message Enqueuing routine.
IEF04FAK Linkage to IEFSD004 (in IEFACTACH load module).

Load Module Name: IEFATACH
Alias: IEFSD004
Entry Point: IEFSD004
Assembly Modules:
IEFQMSSS Table Store subroutine.
IEFSD004 Step Initiation routine, with exit to processing program.
IEFSD006 Converts record number to logical track address (TTR).
IEFSD007 Call to Table Store subroutine.
IEFSD010 Dequeues and writes out system message blocks (SMBs).

Load Module Name: IEFBR14
Assembly Module:
IEFBR14 Branch 14.

Load Module Name: IEFCNTRL
Alias: IEFVHA
Alias: IEFVHAA
Alias: IEFVHC3
Alias: IEFVHE
Alias: IEFVHEB
Entry Point: IEFVHEB
Assembly Modules:
IEFFAFAK Linkage to IEFVFA (in IEFVHH load module).
IEFHBFAK Linkage to IEFVHB (in IEFVHH load module).
IEFHECFK Linkage to IEFVHEC (in IEFVHH load module).
IEFHHFAK Linkage to IEFVHH (in IEFVHH load module).
IEFHLFAK Linkage to IEFVHL (in IEFVHH load module).
IEFHMFAK Linkage to IEF7KPXX (in IEFCOMMD load module).
IEFQMSSS Table Store subroutine.
IEFVGMSS Builds Interpreter error system message blocks (SMBs).
IEFVHA Performs input stream or proclib I/O.
IEFVHAA Sets reader end-of-file (EOF) conditions.
IEFVHC Checks input for valid continuation.
IEFVHCB Identifies control statement verbs and performs procedure modification.
IEFVHE Job Router routine.
IEFVHEB Pre-scan routine.
IEFVHGSS DD* Error routine.
IEFVHQ Table Store Interface routine.
IEFVHRSS Writes operator error messages.
IEFVIND In-stream procedures expansion interface routine.

Load Module Name: IEFCOMMD
Alias: IEFVHM
Entry Point: IEFVHM
Assembly Modules:
IEECNDUN Prevents unresolved external reference to IEEICN01.
IEEILCDM Prevents unresolved IEEICCAN symbol after initialization.
IEEMCR01 Master Command routine.
IEFHAAFK Linkage to IEFVHAA (in IEFCNTRL load module).
IEFHAFAK Linkage to IEFVHA (in IEFCNTRL load module).
IEFQMSSS Table Store subroutine.
IEFSD006 Converts record number to logical track address (TTR).
IEFVGMSS Builds Interpreter error system message blocks (SMBs).
IEFVHQ Table Store Interface routine.
IEFVHRSS Writes operator error messages.
IEF7KPXX Input Stream Command routine.

Load Module Name: IEFCSA
Entry Point: IEFCSA
Assembly Module:
IEFCSA Reads JCL from console.

Load Module Name: IEFDD
Alias: IEFVDA
Entry Point: IEFVDA
Assembly Modules:
IEFGMFAK Saves messages codes from IEFVDA.
IEFQMSSS Table Store subroutine.
IEFSD006 Converts record number to logical track address (TTR).
IEFSD012 DD* Statement routine.
IEFSD090 Assigns unit for system output (SYSOUT).
IEFVDA DD Card Scan routine.
IEFVDDUM Prevents unresolved IEFVDBSD symbol.
IEFVGI Interpreter Dictionary Entry routine.
IEFVGK Obtains parameter from internal table built by IEFVFA.
IEFVGS Interpreter Dictionary Search routine.
IEFVGT Checks validity of control card parameters.
IEFVHF Entry point to IEFGMFAK. Final exit from IEFVDA. Linkage to IEFVGMEP (in IEFVGMSS load module).
IEFVHQ Table Store Interface routine.
IEFVHRSS Writes operator error messages.

Load Module Name: IEFERROR
Alias: Iefvm6ls
Entry Point: IEFVMSGR
Assembly Modules:
IEFQMSSS Table Store subroutine.
IEFVMLS6 JFCB housekeeping, Error Message routine.
IEFVMLS7 Contains Initiator/Terminator messages.

IEFYNFAK Linkage to IEFYNIMP (in IEFSTERM load module).
IEFYSSMB Message Enqueuing routine.

Load Module Name: IEFEXEC
Alias: IEFVEA
Entry Point: IEFVEA
Assembly Modules:
IEFHFFAK Linkage to IEFVHF (in IEFVHH load module).
IEFQMSSS Table Store subroutine.
IEFVEA EXEC Card Scan routine.
IEFVGI Interpreter Dictionary Entry routine.
IEFVGK Obtains parameter from internal table built by IEFVFA.
IEFVGMSS Builds Interpreter error system message blocks (SMBs).
IEFVGS Interpreter Dictionary Search routine.
IEFVGT Checks validity of control card parameters.
IEFVHQ Table Store Interface routine.
IEFVHRSS Writes operator error messages.

Load Module Name: IEFIDUMP
Entry Point: IEFIDUMP
Assembly Modules:
IEFIDMPM Contains Initiator/Terminator messages.
IEFIDUMP Indicative Dump routine.
IEFQMSSS Table Store subroutine.
IEFYNFAK Linkage to IEFYNIMP (in IEFSTERM load module).
IEFYSSMB Message Enqueuing routine.

Load Module Name: IEFINTFC
Alias: IEFKG
Alias: IEFSD001
Alias: IEFSD008
Entry Point: IEFSD008
Assembly Modules:
IEECNDUM Prevents unresolved external reference to IEEICN01.
IEEILCDM Prevents unresolved IEEICCAN symbol after initialization.
IEEMCR01 Master Command routine.
IEFHAAFK Linkage to IEFVHAA (in IEFCNTRL load module).
IEFHAFAK Linkage to IEFVHA (in IEFCNTRL load module).
IEFHCBFK Linkage to IEFVHCB (in IEFCNTRL load module).
IEFQMSSS Table Store subroutine.
IEFSD001 Interpreter entry to IEF09FAK or to IEF23FAK. In case of restart, tests to determine if restarting step has been interpreted; if not, returns to interpreter.
IEFSD006 Converts record number to logical track address (TTR).
IEFSD007 Call to Table Store subroutine.
IEFSD008 Initiator/Terminator to Interpreter interface. Enters interpreter to prepare for restart if necessary.
IEFVHQ Table Store Interface routine.
IEFVHRSS Writes operator error messages.
IEF09FAK Linkage to IEFSD009 (in IEFSELCT load module).
IEF23FAK Linkage to IEFW23SD (in IEFJTRM1 load module).
IEF7KGXX Interpreter-Initiator interface.

Load Module Name: IEFJOB
Alias: IEFVJA
Entry Point: IEFVJA
Assembly Modules:
IEFHFFAK Linkage to IEFVHF (in IEFVHH load module).
IEFQMSSS Table Store subroutine.
IEFVGK Obtains parameter from internal table built by IEFVFA.
IEFVGMSS Builds Interpreter error system message blocks (SMBs).
IEFVGT Checks validity of control card parameters.
IEFVHQ Table Store Interface routine.
IEFVHRSS Writes operator error messages.
IEFVJA Job Card Scan routine.

Load Module Name: IEFJOBQE
Alias: IEFINTQS
Assembly Modules:
IEFINTQA Initializes SYS1.SYSJOBQE data set.
IEFSGOPT System generation option indicators.

Load Module Name: IEFJTRM1
Alias: IEFW23SD
Alias: IEFZA
Entry Point: IEFZA
Assembly Modules:
IEFACTLK Linkage to user accounting routine.
IEFACTRT Dummy, to be replaced by user accounting routine.
IEFQMSSS Table Store subroutine.
IEFWAD Writes accounting information to SYS1.ACCT data set.
IEFW23SD Initializes for job termination, exits to IEFZAJB3 (in this load module).
IEFW31FK Linkage to IEFW31SD (in IEFJTRM2 load module).
IEFYSSMB Message Enqueuing routine.
IEFZAJB3 Job Termination routine.
IEFZGJB1 Disposition and Unallocation subroutine.
IEFZGMSG Contains Initiator/Terminator messages.
IEFZHFAK Call to ZPOQMGR1 subroutine (in IEFZGJB1 assembly module of this load module).
IEFZHMSG Contains Initiator/Terminator messages.

Load Module Name: IEFJTRM2
Alias: IEFW31SD
Entry Point: IEFW31SD
Assembly Modules:
IEFQMSSS Table Store subroutine.

IEFSD003 Passes control to IEFSD010, then to IEK08FAK (both in this load module).
IEFSD006 Converts record number to logical track address (TTR).
IEFSD008 Call to Table Store subroutine. Enters interpreter to prepare for restart if necessary.
IEFSD010 Dequeues and writes out system message blocks (SMBs).
IEFWTERM Job Ended Message routine.
IEFW31SD Exit to IEFSD003 (in this load module).
IEF08FAK Linkage to IEFSD008 (in IEFINTFC load module).
IEF35DUM Prevents unresolved external reference to IEFS0035.

Load Module Name: IEFMCVOL
Alias: IEFCVOL1
Alias: IEFCVOL2
Alias: IEFCVOL3
Entry Point: IEFCVOL1
Assembly Modules:
IEFMCVOL Sets up tables for mounting control volume.
IEFQMSSS Queue Manager Table Store subroutine.
IEFVMFAK Linkage to IEFVMCVL (in IEFVMLS1 assembly module).
IEFVMLS6 JFCB Housekeeping Error Message Processing routine.
IEFVMLS7 Contains Initiator/Terminator messages.
IEFVMMS1 Linkage to IEFVM1 (in IEFVMLS1 assembly module).
IEFYNFAK Linkage to IEFYNIMP.
IEFYSSMB Message Enqueuing routine, enqueues SMBs.

Load Module Name: IEFPRES
Entry Point: IEFPRES
Assembly Modules:
IEFDEVPT Device bit pattern.
IEFK1MSG IEFPRES messages.
IEFPRES Volume Attribute Initialization routine.
IEFSCAN Bit pattern scan routine.

Load Module Name: IEFPRINT
Alias: IEFPRT
Alias: SPRINTER
Assembly Module:
IEFPRTXX Tape SYSOUT to printer or punch.

Load Module Name: IEFSELCT
Alias: IEFSD009, IEFVM1, IEFVMCVL
Entry Point: IEFSD009
Assembly Modules:
IEFACTLK Linkage to user accounting routine.
IEFACTRT Dummy, to be replaced by user accounting routine.
IEFCVFAK Linkage to IEFMCVOL.
IEFQMSSS Table Store subroutine.
IEFSD006 Converts record number to logical track address (TTR).
IEFSD009 Initializes Initiator/Terminator.
IEFSD059 Checks that all SYSOUT classes requested by a job step have been made active. Passes control to Job Separator routines if so indicated.
IEFSD088 Contains transition routine for SYSOUT Job Separator. Sets control characters, etc.
IEFSD089 Contains PUT for Job Separator and error exit.
IEFSD094 Set up for Job Separator routine. Control is given for classes A and B only.
IEFSD095 Block Letter routine for Job Separator.
IEFSEPAR Dummy Job Separator routine to be replaced by user separator routine.
IEFSGOPT System generation option indicators.
IEFVKIMP Execute Statement Condition Code routine.
IEFVKMSG Contains Initiator/Terminator messages.
IEFVMLK5 Linkage to IEFVMLS6 (in IEFERROR load module).
IEFVMLS1 JFCB housekeeping, Control routine.
IEFVM2LS JFCB housekeeping, Fetch DCB routine.
IEFVM3LS JFCB housekeeping, Generation Data Group (GDG) Single routine.
IEFVM4LS JFCB housekeeping, Generation Data Group (GDG) All routine.
IEFVM5LS JFCB housekeeping, Pattern Data Set Control Block (DSCB) routine.
IEFVM76 Processes passed, non-labeled tape data sets.
IEFWAD Writes accounting information to SYS1.ACCT data set.
IEFWSTRT Job started and job termination message routine.
IEFW21SD System Control routine. In case of restart, restores TT pointers from CVT and reads modified JCT from old queue. In case of step restart, moves tables from old to new queue.
IEFXAFAK Linkage to IEFXCSSS (in IEFALOC1 load module).
IEFYNFAK Linkage to IEFYNIMP (in IEFSTERM load module).
IEFYSSMB Message Enqueuing routine.

Load Module Name: IEFSTERM
Alias: GO
Alias: IEFVMCVL
Alias: IEFVM1
Alias: IEFYN
Entry Point: IEFSD011
Assembly Modules:
IEFACTLK Linkage to user accounting routine.
IEFACTRT Dummy, to be replaced by user accounting routine.

IEFIDFAK	Linkage to IEFIDUMP (in IEFIDUMP load module).
IEFQMSSS	Table Store subroutine.
IEFSD002	Exit to IEF08FAK or IEF09FAK (both in this load module).
IEFSD006	Converts record number to logical track address (TTR).
IEFSD007	Call to Table Store subroutine.
IEFSD011	Entry to Job Management from Supervisor.
IEFSD017	Places logical track address (TTR) of first system message block (SMB) into job control table (JCT).
IEFVJIMP	Job Statement Condition Code routine.
IEFVJMSG	Contains Initiator/Terminator messages.
IEFWAD	Writes accounting information to SYS1.ACCT data set.
IEFW22SD	Passes control to IEFYNIMP (in this load module), then to IEFSD002 (in this load module) or to IEFZAJB3 (in IEFJTRM1 load module).
IEFW42SD	Passes control to IEFIDUMP (in IEFIDUMP load module) if necessary, or to IEFYNIMP (in this module).
IEFYNIMP	Step Termination routine.
IEFYNMSG	Contains Initiator/Terminator messages.
IEFYPJB3	Step Data Set Driver routine.
IEFYPMSG	Contains Initiator/Terminator messages.
IEFYSSMB	Message Enqueuing routine.
IEFZAFAK	Linkage to IEFZAJB3 (in IEFJTRM1 load module).
IEFZGMSG	Contains Initiator/Terminator messages.
IEFZGST1	Disposition subroutine. Performs special disposition processing for step to be restarted.
IEFZGST2	Unallocation subroutine. Performs special unallocation processing for step to be restarted.
IEFZHMSG	Contains Initiator/Terminator messages.
IEF08FAK	Linkage to IEFSD008 (in IEFINTFC load module).
IEF09FAK	Linkage to IEFSD009 (in IEFSELCT load module).

Load Module Name: IEFVGMSS
Alias: IEFVGMEP
Entry Point: IEFVGMEP
Assembly Modules:

IEFQMSSS	Table Store subroutine.
IEFVGMEP	Calls IEFVGMSS to write messages for IEFVDA.
IEFVGMSS	Builds Interpreter error system message blocks (SMBs).
IEFVHQ	Table Store Interface routine.
IEFVHRSS	Writes operator error messages.

Load Module Name: IEFVGM1
Assembly Module:

IEFVGM1	Contains Interpreter messages.

Load Module Name: IEFVGM2
Assembly Module:

IEFVGM2	Contains Interpreter messages.

.
.
.

Load Module Name: IEFVGM17
Assembly Module:

IEFVGM17	Contains Interpreter messages.

Load Module Name: IEFVGM18
Assembly Module:

IEFVGM18	Contains Interpreter messages.

Load Module Name: IEFVGM70
Assembly Module:

IEFVGM70	Contains Interpreter messages.

Load Module Name: IEFVGM71
Assembly Module:

IEFVGM71	Contains Interpreter messages.

Load Module Name: IEFVGM78
Assembly Module:

IEFVGM78	Contains Interpreter messages.

Load Module Name: IEFVHH
Alias: IEFVFA
Alias: IEFVHB
Alias: IEFVHEC
Alias: IEFVHF
Alias: IEFVHL
Entry Point: IEFVHH
Assembly Modules:

IEFACT	User exit at Interpreter time.
IEFDAFAK	Linkage to IEFVDA (in IEFDD load module).
IEFEAFAK	Linkage to IEFVEA (in IEFEXEC load module).
IEFHAFAK	Linkage to IEFVHA (in IEFCNTRL load module).
IEFHCBFK	Linkage to IEFVHCB (in IEFCNTRL load module).
IEFHCFAK	Linkage to IEFVHC (in IEFCNTRL load module).
IEFHEBFK	Linkage to IEFVHEB (in IEFCNTRL load module).
IEFHEFAK	Linkage to IEFVHE (in IEFCNTRL load module).
IEFJAFAK	Linkage to IEFVJA (in IEFJOB load module).
IEFKGDUM	Linkage to IEF7KGXX (in IEFINTFC load module).
IEFQMSSS	Table Store subroutine.
IEFVFA	Interpreter Scan routine.
IEFVFB	Symbolic parameter processing.
IEFVGMSS	Builds Interpreter error system message blocks (SMBs).
IEFVHB	Generates DD* statement for data in the input stream.
IEFVHEC	Enqueues job requests.
IEFVHF	Post-processing Control routine.
IEFVHGSS	DD* Error routine.

IEFVHH Sets up table for queuing and provides Initiator/Terminator interface.
IEFVHHB Job and step enqueue housekeeping.
IEFVHL Null Statement routine.
IEFVHQ Table Store Interface routine.
IEFVHRSS Writes operator error messages.

Load Module Name: IEFVHN
Entry Point: IEFVHN
Assembly Modules:
IEEICN01 Builds new Reader-Writer table by inserting TTRs obtained by conversion of record numbers. These are the TTRs of the SYSOUT JFCBs in the preempted track area.
IEEILCDM Prevents unresolved IEEICCAN symbol after initialization.
IEEMCR01 Master Command routine.
IEFK1FAK Linkage to IEF7K1XX (in IEFVH1 load module).
IEFQMSSS Table Store subroutine.
IEFRAPCP Restart Activation routine.
IEFSD006 Converts record number to logical track address (TTR).
IEFVHN Interpreter Termination routine.
IEF7K3XX Interpreter Exit routine.

Load Module Name: IEFVH1
Alias: IEFINITL
Alias: IEFK1
Entry Point: IEFK1
Assembly Modules:
IEEICN01 Builds new Reader/Writer table by inserting TTRs obtained by conversion of record numbers. These are the TTRs of the SYSOUT JFCBs in the preempted track area.
IEEILC01 Automatic Command routine.
IEEMCR01 Master Command routine.
IEEVSMDM Prevents unresolved external reference to IEEVSMSB.
IEFQMSSS Table Store subroutine.
IEFSD006 Converts record number to logical track address (TTR).
IEFSD007 Call to Table Store subroutine.
IEFSGOPT System generation option indicators.
IEFVHQ Table Store Interface routine.
IEFVHRSS Writes error messages to operator.
IEFVH1 Interpreter work area (IWA).
IEFVH2 Opens input reader and procedure libraries.
IEFWSDIP Linkage control table (LCT) initialization.
IEF7K1XX Entry to Job Management from Nucleus Initialization Program (NIP).
IEF7K2XX PCP-dependent Interpreter initialization.
IEFK3XX Interpreter exit routine. Calls IEFRAPCP if restart is to be done.

Load Module Name: IEFVINA
Entry Point: IEFVINA
Assembly Modules:
IEFQMSSS Table Store routine.
IEFVGMSS Builds interpreter message blocks.
IEFVHQ Table Store Interface routine.
IEFVHRSS Writes in-stream error messages to the operator.
IEFVINA Processes in-stream procedures.
IEFVINB Searches directory for the TTR of an in-stream procedure.
IEFVINC Builds a directory entry for an in-stream procedure.
IEFVINE Checks syntax of the PROC and PEND statements.
IEZNCODE Compresses blanks from in-stream procedure statements.

Load Module Name: IEFX5000
Entry Point: IEFX5000
Assembly Modules:
IEFV15XL Allocation Error routine.
IEFWCFAK Linkage to IEFWCIMP (in IEFALOC3 load module).
IEFXH000 Separation Strikeout routine
IEFXJFAK Linkage to IEFXCSSS (in IEFALOC1 load module).
IEFX300A Device Strikeout routine.
IEFX5000 Decision Allocation routine.

Load Module Name: IEZDCODE
Assembly Module:
IEZDCODE Expands in-stream procedures.

Load Module Name: IEZNCODE
Assembly Module:
IEZNCODE Compresses in-stream procedures.

44K CONFIGURATION

Load Module Name: DEVNAMET
Entry Point: DEVNAMET
Assembly Module:
IEFWMAS1 Device Name Table.

Load Module Name: DEVMASKT
Entry Point: DEVMASKT
Assembly Module:
IEFWMSKA Device Mask Table.

Load Module Name: IEEFAULT
Alias: IEEGK1GM
Assembly Module:
IEEGK1GM Fault routine, issues Master Scheduler messages.

Load Module Name: IEEJFCB
Alias: IEEIC3JF
Assembly Module:
IEEIC3JF Contains preformatted JFCB for one START command.

Load Module Name: IEESET
Alias: IEEGES01
Assembly Module:
IEEGES01 Master Scheduler SET Command routine.

Load Module Name: IEESJFCB
Alias: IEEIC2NQ
Entry Point: IEEIC2NQ
Assembly Module:
IEEIC2NQ Saves START command JFCBs.
IESQMSSS Table Store subroutine.

Load Module Name: IEESTART
Alias: IEEIC1PE
Entry Point: IEEIC1PE
Assembly Modules:
IEEREADR Start Reader routine.
IEESTART Process START and STOP WTR commands.
IEEWRITR Start Writer routine.

Load Module Name: IEETIME
Alias: IEEQOT00
Assembly Module:
IEEQOT00 Sets time and date.

Load Module Name: IEFALOC1
Alias: IEFXA
Entry Point: IEFXA
Assembly Modules:
IEFDEVPT Device bit pattern.
IEFQMSSS Table Store subroutine.
IEFSCAN Bit pattern scan routine.
IEFSD006 Converts record number to logical track address (TTR).
IEFSGOPT System generation option indicators.
IEFSWIN Passes control to Descision Allocation or Automatic Volume Recognition (AVR) routine.
IEFV15XL Prevents unresolved external symbol for IEFS15XL.
IEFWA000 Demand Allocation routine.
IEFWCFAK Linkage to IEFWC000 (in IEFALOC2 load module).
IEFWD000 External Action routine.
IEFWD001 Message directory for External Action routine.
IEFXAMSG Contains Initiator/Terminator messages.
IEFXCSSS Allocation Control routine.
IEFXJIMP Allocation Error Recovery routine.
IEFXJMSG Contains Initiator/Terminator messages.
IEFXKIMP Allocation Error Non-recovery routine.
IEFXKMSG Contains Initiator/Terminator messages.
IEFXTFAK Linkage to IEFXTCCD (in IEFALOC2 load module).
IEFXVMSG Automatic Volume Recognition (AVR) Message routine.
IEFXVNSL Automatic Volume Recognition (AVR) Nonstandard Label routine.
IEFXV001 Automatic Volume Recognition (AVR) routine.
IEFXV002 AVR Volume Serial Number Reading routine.
IEFX300A Device Strikeout routine.
IEFX5FAK Linkage to IEFX5000 (in IEFALOC2 load module).
IEFYNFAK Linkage to IEFYNIMP (in IEFSTERM load module).
IEFYSSMB Message Enqueuing routine.

Load Module Name: IEFALOC2
Alias: IEFWC000
Alias: IEFX5000
Entry Point: IEFX5000
Assembly Modules:
IEFQMSSS Table Store subroutine.
IEFSD004 Step Initiation routine with exit to processing program.
IEFSD006 Converts record to logical track address (TTR).
IEFSD007 Call to Table Storage subroutine.
IEFSD010 Dequeues and writes out system message blocks (SMBs).
IEFV15XL Prevents unresolved external reference for IEFS15XL.
IEFWCIMP TIOT Construction routine.
IEFWD000 External Action routine.
IEFWD001 Message directory for External Action routine.
IEFW41SD Exit to Step Initiation routine.
IEFXAFAK Linkage to IEFXCSSS (in IEFALOC1 load module).
IEFXH000 Separation Strikeout routine.
IEFXJIMP Allocation Error Recovery routine.
IEFXJMSG Contains Initiator/Terminator messages.
IEFXKIMP Aloocation Error Non-recovery routine.
IEFXKMSG Contains Initiator/Terminator messages.
IEFXTDMY Queue Overflow routine.
IEFXTMSG Contains Initiator/Terminator messages.
IEFXT00D Space Request routine.
IEFX300A Divide Strikeout routine.
IEFX5000 Decision Allocation routine.
IEFYNFAK Linkage to IEFYNIMP (in IEFSTERM load module).
IEFYSSMB Message Enqueuing routine.

Load Module Name: IEFBR14
Assembly Module:
IEFBR14 Branch 14.

Load Module Name: IEFCNTRL
Alias: IEFKG
Alias: IEFSD008
Alias: IEFVHA
Alias: IEFVHAA
Alias: IEFVHCB
Entry Point: IEFVHA
Assembly Modules:

IEEMCRFK	Linkage to IEEMCREP (in IEFCOMMD load module).
IEFACT	User exit at Interpreter time.
IEFHHB	Job and step enqueue housekeeping.
IEFHMFAK	Linkage to IEF7KPXX (in IEFCOMMD load module).
IEFQMSSS	Table Store subroutine.
IEFSD001	Interpreter entry to IEFSD009 or to IEFW23SD. In case of restart, tests to determine if restarting step has been interpreted; if not, returns to interpreter.
IEFSD006	Converts record number to logical track address.
IEFSD007	Call to Table Store subroutine.
IEFSD008	Initiator/Terminator to Interpreter interface. Enters interpreter to prepare for restart if necessary.
IEFSD010	Dequeues and writes out system message blocks (SMBs).
IEFSD012	DD* Statement routine.
IEFSD090	Assign unit for system output (SYSOUT).
IEFVDA	DD Card Scan routine.
IEFVDDUM	Prevents unresolved IEFVDBSD symbol.
IEFVEA	EXEC Card Scan routine.
IEFVFA	Interpreter Scan routine.
IEFVFB	Symbolic parameter processing.
IEFVGI	Interpreter Dictionary Entry routine.
IEFVGK	Interpreter Get Parameter routine.
IEFVGMSS	Builds system message blocks (SMBs).
IEFVGS	Interpreter Dictionary Search routine.
IEFVGT	Interpreter Test and Store routine.
IEFVHA	Performs input stream or proclib I/O.
IEFVHAA	Sets reader end-of-file (EOF) conditions.
IEFVHB	Generates DD* statement for data in the input stream.
IEFVHC	Checks input for valid continuation.
IEFVHCB	Identifies control statement verbs and performs procedure modification.
IEFVHE	Job Router routine.
IEFVHEB	Pre-scan routine.
IEFVHEC	Enqueues job request.
IEFVHF	Post-processing Control routine.
IEFVHGSS	DD* Error routine.
IEFVHH	Sets up tables for queuing and provides Initiator/Terminator interface.
IEFVHL	Null Statement Processing routine.
IEFVHQ	Table Store Interface routine.
IEFVHRSS	Writes error messages to operator.
IEFVIND	In-stream procedures expansion interface routine.
IEFVJA	JOB Card Scan routine.
IEF09FAK	Linkage to IEFSD009 (in IEFSTERM load module).
IEF23FAK	Linkage to IEFW23SD (in IEFJTERM load module).
IEF7KGXX	Output table for step.

Load Module Name: IEFCOMMD
Alias: IEFVHM
Alias: IEEMCREP
Entry Point: IEFKP
Assembly Modules:

IEECNDUM	Prevents unresolved external reference to IEEICN01.
IEEILCDM	Prevents unresolved IEEICAN symbol after initialization.
IEEMCREP	Links to IEEMCR01 and returns to IEF7KGXX (in IEFCNTRL load module).
IEEMCR01	Master command routine.
IEFHAAFK	Linkage to IEFVHAA in IEFCNTRL load module).
IEFHAFAK	Linkage to IEFVHA (in IEFCNTRL load module).
IEFQMSSS	Table store subroutine.
IEFSD006	Converts record number to logical track address (TTR).
IEFVGMSS	Builds system message blocks (SMBs).
IEFVHQ	Table store interface routine.
IEF7KPXX	Command in the input stream routine.

Load Module Name: IEFCSA
Entry Point: IEFCSA
Assembly Module:

IEFCSA	Reads JCL from console.

Load Module Name: IEFERROR
Alias: IEFVM6LS
Entry Point: IEFVMSGR
Assembly Modules:

IEFQMSSS	Table Store subroutine.
IEFVMLS6	JFCB housekeeping, Error Message routine.
IEFVMLS7	Contains Initiator/Terminator messages.
IEFYNFAK	Linkage to IEFYNIMP (in IEFSTERM load module).
IEFYSSMB	Message Enqueuing routine.

Load Module Name: IEFIDUMP
Entry Point: IEFIDUMP
Assembly Modules:

IEFIDMPM	Contains Initiator/Terminator messages.
IEFIDUMP	Indicative Dump routine.
IEFQMSSS	Table Store subroutine.

IEFYNFAK Linkage to IEFYNIMP (in IEFSTERM load module).
IEFYSSMB Message Enqueuing routine.

Load Module Name: IEFJOBQE
Alias: IEFINTQS
Assembly Modules:
IEFINTQA Initializes SYS1.SYSJOBQE data set.
IEFSGOPT System generation option indicators.

Load Module Name: IEFJTERM
Alias: IEFZA
Alias: IEFW23SD
Entry Point: IEFZA
Assembly Modules:
IEFACTLK Linkage to user's accounting routine.
IEFACTRT Dummy routine to be replaced by user's account routine.
IEFQMSSS Table Store subroutine.
IEFSD006 Converts record number to logical track address (TTR).
IEFSD007 Call to Table Store subroutine.
IEFSD010 Dequeues and writes out system message blocks (SMBs).
IEFWAD Writes accounting information to SYS1.ACCT data set.
IEFWTERM Job ended message routine.
IEFW23SD Initializes for job termination and exits to IEFZAJB3 (this load module).
IEFW31SD Job termination exit to IEFSD003.
IEFYSSMB Message Enqueuing routine, enqueues SMBs.
IEFZAJB3 Job Termination routine.
IEFZGJB1 Disposition and Unallocation subroutine.
IEFZGMSG Contains initiator/terminator messages.
IEFZHFAK Call to ZPOQMGR1 subroutine in IEFZGST1 (in IEFSTERM load module).
IEFZHMSG Contains Initiator/Terminator messages.

Load Module Name: IEFMCVOL
Alias: IEFCVOL1
Alias: IEFCVOL2
Alias: IEFCVOL3
Entry Point: IEFCVOL1
Assembly Modules:
IEFMCVOL Sets up tables for mounting control volume.
IEFQMSSS Queue manager table store subroutine.
IEFVMFAK Linkage to IEFVMCVL (in IEFVMLS1 assembly module).
IEFVMLS6 JFCB housekeeping error message processing routine.
IEFVMLS7 Contains Initiator/Terminator messages.
IEFVMMS1 Linkage to IEFVM1 (in IEFVMLS1 assembly module).
IEFYNFAK Linkage to IEFYNIMP.
IEFYSSMB Message Enqueuing routine, enqueues SMBs.

Load Module Name: IEFPRES
Entry Point: IEFPRES
Assembly Modules:
IEFK1MSG IEFPRES messages
IEFPRES Volume Attribute Initialization routine.

Load Module Name: IEFPRINT
Alias: IEFPRT
Alias: SPRINTER
Assembly Module:
IEFPRTXX Tape SYSOUT to printer or punch.

Load Module Name: IEFSTERM
Alias: GO
Alias: IEFSD009
Alias: IEFYN
Assembly Modules:
IEFACTLK Linkage to user accounting routine.
IEFACTRT Dummy, to be replaced by user accounting routine.
IEFIDFAK Linkage to IEFIDUMP (in IEFIDUMP load module).
IEFQMSSS Table Store subroutine.
IEFSD002 Exit to EIF08FAK or IFSD00. (both in this load module).
IEFSD006 Converts record number to logical track address (TTR).
IEFSD007 Call to Table Store subroutine.
IEFSD009 Initializes Initiator/Terminator, passes control to IEFW21SD (in this load module).
IEFSD011 Entry to Job Management from Supervisor.
IEFS017 Places logical track address (TTR) of first system message block (SMB) in job control table (JCT).
IEFSD059 Checks that all SYSOUT classes requested by a job step have been made active. Passes control to Job Separator routines if so indicated.
IEFSD088 Contains transition routine for SYSOUT Job Separator. Sets control characters, etc.
IEFSD089 Contains PUT for Job Separator and error exit.
IEFSD094 Set up for Job Separator routine. Control is given for classes A and B only.
IEFSD095 Block Letter routine for Job Separator.
IEFSEPAR Dummy Job Separator routine to be replaced by user separator routine.
IEFSGOPT System generation option indicators.
IEFVJIMP Job Statement Condition Code routine.
IEFVJMSG Contains Initiator/Terminator messages.
IEFVKIMP Execute Statement Condition Code routine.

IEFVKMSG	Contains Initiator/Terminator messages.
IEFVMLK5	Linkage to IEFVMLS6 (in IEFERROR load module).
IEFVMSL1	JFCB housekeeping, Control routine.
IEFVM2LS	JFCB housekeeping, Fetch DCB routine.
IEFVM3LS	JFCB housekeeping, Generation Data Group (GDG) Single routine.
IEFVM4LS	JFCB housekeeping, Generation Data Group (GDG) All routine.
IEFVM5LS	JFCB housekeeping, Pattern Data Set Control Block (DCB) routine.
IEFVM76	Processes passed, non-labeled data sets.
IEFWAD	Writes accounting information to SYS1.ACCT data set.
IEFWSTRT	Job started and job termination message routine.
IEFW21SD	System Control routine. In case of restart, restores TT pointers form CVT and reads modified JCT from old queue. In case of step restart, moves tables from old to new queue.
IEFW22SD	Passes control to UEFYNIMP (in this load module), then to IEFSD002 (in this load module) or to IEFZAJB3 (in IEFJTERM load module).
IEFW42SD	Passes control to IEFIDUMP (in IEFIDUMP load module) if necessary, or to IEFYNIMP (in this load module).
IEFXAFAK	Linkage to IEFXCSSS (in IEFALOC1 load module).
IEFYNIMP	Step Termination routine.
IEFYNMSG	Contains Initiator/Terminator messages.
IEFYPJB3	Step Data Set Driver routine.
IEFYPMSG	Contains Initiator/Terminator messages.
IEFYSSMB	Messages Enqueuing routine.
IEFZAFAK	Linkage to IEFZAJB3 (in IEFJTERM load module).
IEFZGMSG	Contains Initiator/Terminator messages.
IEF2GST1	Disposition subroutine. Performs special disposition processing for step to be restarted.
IEF2GST2	Unallocation subroutine. Performs special unallocation processing for step to be restarted.
IEFZHMSG	Contains Initiator/Terminator messages.
IEF08FAK	Linkage to IEFSD008 (in IEFCNTRL load module).

Load Module Name: IEFVGM1
Assembly Module:
IEFVGM1 Contains Interpreter messages.

Load Module Name: IEFVGM2
Assembly Module:
IEFVGM2 Contains Interpreter messages.
.
.
.

Load Module Name: IEFVGM17
Assembly Module:
IEFVGM17 Contains Interpreter messages.

Load Module Name: IEFVGM18
Assembly Module:
IEFVGM1, Contains Interpreter messages.

Load Module Name: IEFVGM70
Assembly Module:
IEFVGM70 Contains Interpreter messages.

Load Module Name: IEFVGM71
Assembly Module:
IEFVGM71 Contains Interpreter messages.

Load Module Name: IEFVGM78
Assembly Module:
IEFVGM78 Contains Interpreter messages.

Load Module Name: IEFVH1
Alias: IEFK1
Alias: IEFVHN
Alias: IEFINITL
Entry Point: IEFK1
Assembly Modules:

IEEICN01	Initialize new reader writer table by inserting TTRs obtained by conversion of record numbers. These are the TTRs of the SYSOUT JFCBs in the preempted track area.
IEEILC01	Automatic command routine.
IEEMCR01	Master command routine.
IEEVSMDM	Prevents unresolved external symbol for IEEVSMSG.
IEFDEVPT	Device bit pattern.
IEFK1MSG	Reader/Interpreter message routine.
IEFQMSSS	Table store subroutine.
IEFRAPCP	Prepares for restart.
IEFSCAN	Bit pattern scan routine.
IEFSD006	Converts record number to logical track address (TTR).
IEFSD007	Call to table store subroutine.
IEFSGOPT	System generation option indicators.
IEFVHN	Interpreter termination routine.
IEFVHQ	Table store interface routine.
IEFVHRSS	Writes error messages to operator.
IEFVH1	Interpreter work area (IWA) initialization routine.
IEFVH2	Opens input reader and procedure libraries.
IEF7K1XX	Entry to job management from nucleus initialization program (NIP).

IEF7K2XX	PCP dependent interpreter initialization.
IEF7K3XX	Interpreter exit routine. Calls IEFRAPCP if restart is to be done.

Load Module Name: IEFVINA
Entry Point: IEFVINA
Assembly Modules:

IEFQMSSS	Table Store subroutines.
IEFVGMSS	Builds interpreter message blocks.
IEFVHQ	Table Store Interface routine.
IEFVHRSS	Writes in-stream error messages to the operator.
IEFVINA	Processes in-stream procedures.
IEFVINB	Searches directory for the TTR of an in-stream procedure.
IEFVINC	Builds a directory entry for an in-stream procedure.
IEFVINE	Checks syntax of the PROC and PEND statements.
IEZNCODE	Compresses blanks from in-stream procedure statements.

Load Module Name: IEZDCODE
Assembly Module:

IEZDCODE	Expands in-stream procedures.

Load Module Name: IEZNCODE
Assembly Module:

IEZNCODE	Compresses in-stream procedures.

100K CONFIGURATION

Load Module Name: DEVNAMET
Entry Point: DEVNAMET
Assembly Module:

IEFWMAS1	Device Name Table.

Load Module Name: DEVMASKT
Entry Point: DEVMASKT
Assembly Module:

IEFWMSKA	Device Mask Table.

Load Module Name: GO
Alias: IEFVHA
Alias: IEFVINA
Alias: IEFVMCVL
Alias: IEFVM1
Alias: IEFYN
Alias: IEFZA
Alias: IEZDCODE
Alias: IEZNCODE
Entry Point: IEFSD011
Assembly Modules:

IEECNDUM	Prevents unresolved external reference to IEEICN01.
IEEFZGJB1	Disposition and unallocation subroutine.
IEEILCDM	Prevents unresolved external reference.
IEEMCR01	Master command routine.
IEFACT	User exit at interpreter time.
IEFACTLK	Linkage to user's accounting routine.
IEFACTRT	Dummy routine to be replaced by user's accounting routine.
IEFCVFAK	Linkage to IEFMCVOL (in IEFMCVOL load module).
IEFIDFAK	Linkage to IEFIDUMP (in IEFIDUMP load module).
IEFQMSSS	Table Store subroutine.
IEFRPREP	Restart preparation.
IEFSD001	Interpreter entry to IEFSD009 or to IEFW23SD (both in this load module). In case of restart, tests to determine if restarting step has been interpreted; if not, returns to interpreter.
IEFSD002	Exit to IEFSD008 or IEFSD009 (both in this load module).
IEFSD003	Passes control to IEFSD010 and then goes to IEFSD008 (both in this load module).
IEFSD006	Converts record number to logical track address (TTR).
IEFSD007	Call to table store subroutine.
IEFSD008	Initiator to interpreter interface. Enters interpreter to prepare for restart if necessary.
IEFSD009	Initiator/terminator initialization of output unit.
IEFSD010	Dequeues and writes out system message blocks (SMBs).
IEFSD011	Entry to job management from supervisor.
IEFSD012	DD* statement routine. SMB) into job control table (JCT).
IEFSD059	Checks that all SYSOUT classes requested by a job step have been made active. Passes control to Job Separator routine if so indicated.
IEFSD088	Contains transition routine for SYSOUT job separator. Sets control characters, etc.
IEFSD089	Contains PUT for job separator and error exit.
IEFSD090	Assigns unit for system output (SYSOUT).
IEFSD094	Set up for job separator routine. Control is given for classes A and B only.
IEFSD095	Block letter routine for job separator.
IEFSEPAR	Dummy job separator routine to be replaced by user separator routine.
IEFSGOPT	SYSGEN option flags.
IEFVDA	DD Card Scan routine.
IEFVDDUM	Prevents unresolved IEFVDBSD symbol.
IEFVEA	Exec Card Scan routine.

IEEVFA	Interpreter Scan routine.
IEFVFB	Symbolic parameter processing.
IEFVGI	Interpreter dictionary entry routine.
IEFVGK	Interpreter get parameter routine.
IEFVGMSS	Builds interpreter system message blocks (SMBs).
IEFVGS	Interpreter dictionary search routine.
IEFVGT	Interpreter test and store routine.
IEFVHA	Performs input stream or proclib I/O.
IEFVHAA	Sets reader end-of-file conditions.
IEFVHB	Generates DD* for data in the input stream.
IEFVHC	Checks input for valid continuation.
IEFVHCB	Identifies control statement verbs and performs procedure modification.
IEFVHE	Interpreter Router routine.
IEFVHEB	Pre-scan routine.
IEFVHEC	Enqueues job request.
IEFVHF	Post-processing Control routine.
IEFVHGSS	DD* Error routine.
IEFVHH	Sets up tables for queuing and provides initiator/terminator interface.
IEFVHHB	Job and step enqueuing housekeeping.
IEFVHL	Null statement processing routine.
IEFVHQ	Table store interface routine.
IEFVHRSS	Writes error messages to operator.
IEFVINA	Processes in-stream procedures.
IEFVINB	Searches directory for the TTR of an in-stream procedure.
IEFVINC	Builds a directory entry for an in-stream procedure.
IEFVIND	In-stream procedures expansion interface routine.
IEFVINE	Checks syntax of the PROC and PEND statements.
IEFVJA	Job card scan routine.
IEFVJIMP	JOB statement condition code routine.
IEFVJMSG	Contains initiator/terminator messages.
IEFVKIMP	EXEC statement conditional execution routine.
IEFVKMSG	Contains initiator/terminator messages.
IEFVMLS1	JFCB housekeeping control routine.
IEFVMLS6	JFCB housekeeping error message processing routine.
IEFVMLS7	Contains initiator/terminator messages.
IEFVM2LS	JFCB housekeeping fetch DCB routine.
IEFVM3LS	JFCB housekeeping generation data group single routine.
IEFVM4LS	JFCB housekeeping generation data group all routine.
IEFVM5LS	JFCB housekeeping patterning data set control block (DSCB) subroutine.
IEFVM76	Processes passed, non-labeled tape data sets.
IEFWAD	Writes accounting information to SYS1.ACCT data set.
IEFWSTRT	Job started and job termination message routine.
IEFWTERM	Job ended message routine.
IEFW21SD	System control routine. In case of restart, restore TT pointers from CVT and reads modified JCT from old queue. In case of step restart, moves tables from old to new queue.
IEFW22SD	Passes control to IEFYNIMP assembly module, and then to IEFSD002 or IEFZAJB3 (all in this load module).
IEFW23SD	Initializes for job termination and exits to IEFZAJB3 (in this load module).
IEFW31SD	Job termination exit to IEFSD003.
IEFW42SD	Passes control to IEFIDUMP if needed, or to IEFYNIMP.
IEFXAFAK	Linkage to IEFXCESS (in IEFALLOC load module).
IEFYNIMP	Step termination routine.
IEFYNMSG	Contains initiator/terminator messages.
IEFYPJB3	Step data set driver routine.
IEFYRCDS	Table of Abend codes eligible for restart.
IEFYSSMB	Message enqueuing routine.
IEFZAJB3	Job termination routine.
IEFZGST1	Disposition subroutine. Performs special disposition processing for step to be restarted.
IEFZGST2	Unallocation subroutine. Performs special unallocation processing for step to be restarted.
IEFZHMSG	Contains initiator/terminator messages.
IEF2GMSG	Contains initiator/terminator messages.
IEF7KGXX	Output tables for step.
IEF7KPXX	Command in the input stream routine.
IEZDCODE	Expands in-stream procedures.
IEZNCODE	Compresses in-stream procedures.

Load Module Name: IEEFAULT
Alias: IEEGK1GM
Assembly Module:

IEEGK1GM	Fault routine, issues Master Scheduler messages.

Load Module Name: IEEJFCB
Alias: IEEIC3JF
Assembly Module:

IEEIC3JF	Contains preformatted JFCB for one START command.

Load Module Name: IEESET
Alias: IEEGES01
Assembly Module:
IEEGES01 Master Scheduler SET Command routine.

Load Module Name: IEESJFCB
Alias: IEEIC2NQ
Entry Point: IEEIC2NQ
Assembly Module:
IEEIC2NQ Saves START command JFCBs.
IESQMSSS Table Store subroutine.

Load Module Name: IEESTART
Alias: IEEIC1PE
Entry Point: IEEIC1PE
Assembly Modules:
IEEREADR Start Reader routine.
IEESTART Process START and STOP WTR commands.
IEEWRITR Start Writer routine.

Load Module Name: IEETIME
Alias: IEEQOT00
Assembly Module:
IEEQOT00 Sets time and date.

Load Module Name: IEFALLOC
Alias: IEFXA
Entry Point: IEFXA
Assembly Modules:
IEFCVFAK Linkage to IEFMCVOL (in IEFMCVOL load module).
IEFDEVPT Device bit pattern.
IEFQMSSS Table store subroutine.
IEFSCAN Bit pattern scan routine.
IEFSD004 Step initiation routine, with exit to processing program.
IEFSD006 Converts record number to logical track address (TTR).
IEFSD007 Call to table store subroutine.
IEFSD010 Dequeues and writes out system message blocks (SMBs).
IEFSGOPT System generation option indicators.
IEFVJMSG Contains initiator/terminator messages.
IEFVKMSG Contains initiator/terminator messages.
IEFV15XL Prevents unresolved external symbol for IEFS15XL.
IEFWA000 Demand allocation routine.
IEFWCIMP Task input/output table (TIOT) construction routine.
IEFWD000 External action routine.
IEFWD001 Message directory for external action routine.
IEFWSWIN Passes control to decision allocation or AVR routine.
IEFW41SD Exit to step initiation routine.
IEFXAMSG Contains initiator/terminator messages.
IEFXCSSS Allocation control routine.
IEFXH000 Separation strikeout routine.
IEFXJIMP Allocation error recovery routine.
IEFXJMSG Contains initiator/terminator messages.
IEFXKIMP Allocation error non-recovery routine.
IEFXKMSG Contains initiator/terminator messages.
IEFXTDMY Queue overflow routine.
IEFXTMSG Contains initiator/terminator messages.
IEFXT00D Space request routine.
IEFXT002 TIOT compression routine.
IEFXT00O DADSM error recovery routine.
IEFXVMSG AVR message routine.
IEFXVNSL AVR Nonstandard Label routine.
IEFXV001 Automatic volume recognition.
IEFXV002 AVR Volume Serial Number Reading routine.
IEFX300A Device strikeout routine.
IEFX5000 Decision allocation routine.
IEFYNFAK Linkage to IEFYNIMP (in GO load module).
IEFYSSMB Message enqueuing routine, enqueues SMBs.
IEF35DUM Prevents unresolved IEFSD035 symbol.

Load Module Name: IEFBR14
Assembly Module:
IEFBR14 Branch 14.

Load Module Name: IEFCSA
Entry Point: IEFCSA
Assembly Module:
IEFCSA Reads JCL from console.

Load Module Name: IEFIDUMP
IEFIDMPM Contains Initiator/Terminator messages.
IEFIDUMP Indicative Dump routine.
IEFQMSSS Table Store subroutine.
IEFYNFAK Linkage to IEFYNIMP (in IEFSTERM load module).
IEFYSSMB Message Enqueuing routine.

Load Module Name: IEFJOBQE
Alias: IEFINTQS
Assembly Modules:
IEFINTQA Initializes SYS1.SYSJOBQE data set.
IEFSGOPT System generation option indicators.

Load Module Name: IEFMCVOL
Alias: IEFCVOL1
Alias: IEFCVOL2
Alias: IEFCVOL3
Entry point: IEFCVOL1
Assembly Modules:
IEFMCVOL Sets up tables for mounting control volume.
IEFQMSSS Queue manager table store subroutine.
IEFVMFAK Linkage to IEFVMCVL (in IEFVMLS1 assembly module).
IEFVMLS6 JFCB housekeeping error message processing routine.
IEFVMLS7 Contains initiator/terminator messages.

IEFVMMS1 Linkage to IEFVM1 (in IEFVMLS1 assembly module).
IEFYNFAK Linkage to IEFYNIMP.
IEFYSSMB Message enqueuing routine, enqueues SMBs.

Load Module Name: IEFPRES
Entry Point: IEFPRES
Assembly Modules:
IEFK1MSG IEFPRES messages
IEFPRES Volume Attribute Initialization routine.

Load Module Name: IEFPRINT
Alias: IEFPRT
Alias: SPRINTER
Assembly Module:
IEFPRTXX Tape SYSOUT to printer or punch.

Load Module Name: IEFVGM1
Assembly Module:
IEFVGM1 Contains Interpreter messages.

Load Module Name: IEFVGM2
Assembly Module:
IEFVGM2 Contains Interpreter messages.
.
.
.

Load Module Name: IEFVGM17
Assembly Module:
IEFVGM17 Contains Interpreter messages.

Load Module Name: IEFVGM18
Assembly Module:
IEFVGM18 Contains Interpreter messages.

Load Module Name: IEFVGM70
Assembly Module:
IEFVGM70 Contains Interpreter messages.

Load Module Name: IEFVGM71
Assembly Module:
IEFVGM71 Contains interpreter messages.

Load Module Name: IEFVH1
Alias: IEFK1
Alias: IEFVHN
Alias: IEFINITL
Entry Point: IEFK1
Assembly Modules:
IEEICN01 Builds new reader writer table by inserting TTRs obtained by conversion of record numbers. These are the TTRs of the SYSOUT JFCBs in the preempted track area.
IEEILC01 Automatic command routine.
IEEMCR01 Master command routine.
IEEVSMDM Prevents unresolved external reference for IEEFSMSG.
IEFDEVPT Device bit pattern.
IEFHAFAK Linkage to IEFVHA (in GO load module).
IEFK1MSG Interpreter message routine.
IEFQMSSS Table Store subroutine.
IEFRAPCP Prepares for restart.
IEFSCAN Bit pattern scan routine.
IEFSD006 Converts record number to logical track address (TTR).
IEFSD007 Call to table store subroutine.
IEFSGOPT System generation option indicators.
IEFVHN Interpreter termination routine.
IEFVHQ Table store interface routine.
IEFVHRSS Writes error messages to operator.
IEFVH1 Interpreter Initialization routine.
IEFVH2 Opens input stream data set and procedure library.
IEFWSDIP Linkage control table (LCT) initialization.
IEF7K1XX Initial entry to job management from nucleus initialization program (NIP).
IEF7K2XX PCP interpreter system-dependent initialization.
IEF7K3XX Interpreter exit routine. Calls IEFRAPCP if restart is to be done.

Assembly Modules and Control Sections

The following table shows in which load modules each assembly module is used in the three configurations of job management. The first column lists the assembly module names in alphameric order. Except as indicated, all assembly modules are contained in load modules in the SYS1.LINKLIB data set. The third column lists the control section names that correspond to the assembly module names in the first column. The next three columns of the table indicate which load modules of each configuration contain each assembly module. The two right-hand columns refer to the CHARTS section. If a control section is shown as a subroutine block, the flowchart number is listed in the "Appears As Subr. Block" column; if the flow within a control section is given in a chart, the flowchart number is listed in the "Flow is Defined" column.

Assembly Modules and Control Sections (Part 1 of 7)

Assembly Module Name	Notes	Control Section Name	Load Modules in Which Assembly Modules are Used			Chart Number	
			18K	44K	100K	Appears As Subr. Block	Flow is Defined
IEECNDUM		IEEICN01	IEFINTFC	IEFCOMMD	GO		
			IEFCOMMD				
IEEGES01		IEEGESTO	IEESET	IEESET	IEESET		
IEEGK1GM		IEEGK1GM	IEEFAULT	IEEFAULT	IEEFAULT	05,23	
IEEICN01		IEEICN01	IEFVHN	IEFVH1	IEFVH1		
IEEIC2NQ		IEEIC2NQ	IEESJFCB	IEESJFCB	IEESJFCB		
IEEIC3JF	**	IEEIC3JF	IEEJFCB	IEEJFCB	IEEJFCB		
IEEILCDM		IEEICCAN	IEFVHN			05	
			IEFCOMMD	IEFCOMMD	GO		
IEEILC01	**	IEEICCAN	IEFVH1	IEFVH1	IEFVH1		
IEEMCREP		IEEMCREP		IEFCOMMD			
IEEMCRFK		IEEBB1		IEFCOMMD		02	05
IEEMCR01		IEEBB1	IEFINTFC		GO	02,53,54,55	05
			IEFCOMMD	IEFVH1	IEFVH1		
			IEFVH1	IEFCOMMD			
			IEFVHN				
IEEQOT00		IEEQOT00	IEETIME	IEETIME	IEETIME		
IEEREADR		IEEICRDR	IEESTART	IEESTART	IEESTART		
IEERSC01	*	IEEMSLT					
IEERSR01	*	IEEMSLT					
IEESTART		IEEIC1PE	IEESTART	IEESTART	IEESTART	53,54,55	
IEEVSMDM		IEEVSMSG	IEFVH1	IEFVH1	IEFVH1		
IEEWRITR		IEECWTR	IEESTART	IEESTART	IEESTART		
IEFACT		IEFACT	IEFVHH	IEFCNTRL	GO		
IEFACTFK	****	IEFACTFK	IEFSTERM	IERSTERM	GO		
			IEFSELCT				
			IEFJTRM1	IEFJTERM			
IEFACTLK	****	IEFACTLK	IEFSTERM	IEFSTERM	GO	46	48
			IEFSELCT				
			IEFJTRM1	IEFJTERM			
IEFACTRT	****	IEFACTRT	IEFSTERM	IEFSTERM	GO	47	
			IEFSELCT				
			IEFJTRM1	IEFJTERM			
IEFBR14		IEFBR14	IEFBR14	IEFBR14	IEFBR14		
IEFCSA		IEFCSA	IEFCSA	IEFCSA	IEFCSA		
IEFCVFAK		IEFCVOL1	IEFSELCT	IEFSTERM	GO		
			IEFALOC1	IEFALOC1	IEFALLOC		
			IEFALOC4	IEFALOC2			
			IEFALOC5				
IEFDAFAK		IEFVDA	IEFVHH				
IEFDEVPT		IEFDEVPT	IEFALOC2	IEFALOC1	IEFALLOC		
			IEFALOC4	IEFVH1	IEFVH1		
			IEFPRES				
IEFDPOST	*	IEFDPOST					
IEFEAFAK		IEFVEA	IEFVHH				
IEFFAFAK		IEFVFA	IEFCNTRL				

Assembly Modules and Control sections (Part 2 of 7)

Assembly Module Name	Notes	Control Section Name	Load Modules in Which Assembly Modules are Used			Chart Number	
			18K	44K	100K	Appears As Subr. Block	Flow is Defined
IEFGMFAK		IEFVGM	IEFDD				
IEFHAAFK		IEFVHAA	IEFCOMMD	IEFCOMMD			
			IEFINTFC				
IEFHAFAK		IEFVHA	IEFINTFC	IEFCOMMD			
			IEFCOMMD				
			IEFVHH	IEFVH1	IEFVH1		
IEFHBFAK		IEFVHB	IEFCNTRL				
IEFHCFAK		IEFVHC	IEFVHH				
IEFHCBFK		IEFVHCB	IEFINTFC				
			IEFVHH				
IEFHEFAK		IEFVHE	IEFVHH				
IEFHEBFK		IEFVHEB	IEFVHH				
IEFHECFK		IEFVHEC	IEFCNTRL				
IEFHFFAK		IEFVHF	IEFEXEC				
			IEFJOB				
IEFHHFAK		IEFVHH	IEFCNTRL				
IEFHLFAK		IEFVHL	IEFCNTRL				
IEFHMFAK		IEFVHM	IEFCNTRL	IEFCNTRL			
IEFIDFAK		IEFIDUMP	IEFSTERM	IEFSTERM	GO		
IEFIDMPM		IEFIDMPM	IEFIDUMP	IEFIDUMP	IEFIDUMP		
IEFIDUMP		IEFIDUMP	IEFIDUMP	IEFIDUMP	IEFIDUMP	46,53,54	
IEFINTQA		IEFINTQS	IEFJOBQE	IEFJOBQE	IEFJOBQE		
IEFJAFAK		IEFJA	IEFVHH				
IEFKGDUM		IEFKG	IEFVHH			16	
IEFK1FAK		IEFK1	IEFVHN			14	
IEFMCVOL		IEFCVOL1	IEFMCVOL	IEFMCVOL	IEFMCVOL	25,53,54,55	27
		IEFCVOL2					
		IEFCVOL3					
IEFPRES		IEFPRES	IEFPRES	IEFPRES	IEFPRES	53	
IEFPRTXX		SPRINTER	IEFPRINT	IEFPRINT	IEFPRINT		
IEFQMSSS		IEFQMSSS	IEFSTERM	IEFSTERM	GO	24	
			IEFSELCT	IEFALOC1	IEFVH1		
			IEFALOC1	IEFCNTRL			
			IEFALOC4	IEFALOC2	IEFIDUMP		
			IEFALOC5				
			IEFATACH		IEFSJFCB		
			IEFCNTRL		IEFALLOC		
			IEFDD	IEFERROR	IEFMCVOL		
			IEFINTFC	IEFIDUMP			
			IEFEXEC	IEFVH1			
			IEFJOB	IEESJFCB			
				IEFCOMMD			
			IEFCOMMD	IEFJTERM			
			IEFERROR	IEFMCVOL			
			IEFIDUMP				
			IEESJFCB				
			IEFVGMSS				
			IEFVHH				
			IEFVHN				
			IEFVH1				
			IEFVINA	IEFVINA			
			IEFJTRM1				
			IEFJTRM2				
			IEFMCVOL				
IEFSCAN		IEFSCAN	IEFALOC2	IEFALOC1	IEFALLOC		
			IEFALOC4	IEFVH1	IEFVH1		
			IEFPRES				
IEFSD001		IEFSD001	IEFINTFC	IEFCNTRL	GO		
IEFSD002		IEFSD002	IEFSTERM	IEFSTERM	GO		
IEFSD003		IEFSD003	IEFJTERM	IEFJTERM	GO		

Assembly Modules and Control Sections (Part 3 of 7)

Assembly Module Name	Notes	Control Section Name	Load Modules in Which Assembly Modules are Used			Chart Number	
			18K	44K	100K	Appears As Subr. Block	Flow is Defined
IEFSD004		IEFSD004	IEFATACH	IEFALOC	IEFALLOC		46
IEFSD006		IEFSD006	IEFSTERM	IEFSTERM	GO		
			IEFALOC2	IEFALOC1	IEFVH1		
			IEFALOC4	IEFCNTRL			
			IEFALOC5	IEFALOC2	IEFALLOC		
			IEFATACH		GO		
			IEFSELCT				
			IEFDD	IEFVH1			
			IEFINTFC	IEFCOMMD			
			IEFJTERM	IEFJTERM			
			IEFVHN				
			IEFVH1				
IEFSD007		IEFSD007	IEFSTERM	IEFSTERM	GO		
			IEFATACH	IEFALOC2	IEFVH1		
			IEFINTFC	IEFCNTRL			
			IEFJTERM	IEFVH1			
			IEFVH1	IEFJTERM			
IEFSD008		IEFSD008	IEFINTFC	IEFCNTRL	GO		
IEFSD009		IEFSD009	IEFSELCT	IEFSTERM	GO		
IEFSD010		IEFSD010	IEFATACH	IEFALOC2	GO		
			IEFJTRM2	IEFJTERM			
IEFSD011		IEFSD011	IEFSTERM	IEFSTERM	GO	46	48
IEFSD012		IEFSD012	IEFDD	IEFCNTRL	GO		
IEFSD017		IEFSD017	IEFSTERM	IEFSTERM	GO		
IEFSD059		IEFSD059	IEFSELCT	IEFSTERM	GO		
IEFSD088		IEFSD088	IEFSELCT	IEFSTERM	GO		
IEFSD089		IEFSD089	IEFSELCT	IEFSTERM	GO		
		IEFSD89M					
IEFSD090		IEFSD090	IEFDD	IEFCNTRL	GO		
IEFSD094		IEFSD094	IEFSELCT	IEFSTERM	GO		
IEFSD095		IEFSD095	IEFSELCT	IEFSTERM	GO		
IEFSEPAR		IEFSEPAR	IEFSELCT	IEFSTERM	GO		
IEFSGOPT	**	IEFSGOPT	IEFSELCT	IEFSTERM	GO		
			IEFALOC2	IEFALOC1	IEFVH1		
			IEFVH1	IEFVH1	IEFJOBQE		
			IEFJOBQE	IEFJOBQE			
IEFVDA		IEFVDA	IEFDD	IEFCNTRL	GO	14	
IEFVDDUM		IEFVDBSD	IEFDD	IEFCNTRL	GO		
IEFVEA		IEFVEA	IEFEXEC	IEFCNTRL	GO	14	
IEFVFA		IEFVFA	IEFVHH	IEFCNTRL	GO	14	
IEFVFB		IEFVFB	IEFVHH	IEFCNTRL	GO		
IEFVGI		IEFVGI	IEFDD	IEFCNTRL	GO		
			IEFEXEC				
IEFVGK		IEFVGK	IEFDD	IEFCNTRL	GO	18	
			IEFEXEC				
			IEFJOB				
IEFVGMEP		IEFVGM	IEFVGMSS				
IEFVGMSS		IEFVGM	IEFVHH	IEFCNTRL	GO	53	
			IEFVINA	IEFVINA			
			IEFCOMMD	IEFCOMMD	IEFCOMMD		
			IEFVGMSS				
			IEFEXEC				
			IEFJOB				
IEFVGM1		IEFVGM1	IEFVGM1	IEFVGM1	IEFVGM1	53,54,55	
IEFVGM2		IEFVGM2	IEFVGM2	IEFVGM2	IEFVGM2	53,54,55	
IEFVGM3		IEFVGM3	IEFVGM3	IEFVGM3	IEFVGM3	53,54,55	
IEFVGM4		IEFVGM4	IEFVGM4	IEFVGM4	IEFVGM4	53,54,55	
.							
.							
.							

Assembly Modules and Control Sections (Part 4 of 7)

Assembly Module Name	Notes	Control Section Name	Load Modules in Which Assembly Modules are Used			Chart Number	
			18K	44K	100K	Appears As Subr. Block	Flow is Defined
IEFVGM18		IEFVGM18	IEFVGM18	IEFVGM18	IEFVGM18	53,54,55	
IEFVGM70		IEFVGM70	IEFVGM70	IEFVGM70	IEFVGM70	53,54,55	
IEFVGM71		IEFVGM71	IEFVGM71	IEFVGM71	IEFVGM71		
IEFVGM78		IEFVGM78	IEFVGM78	IEFVGM78	IEFVGM78	53,54,55	
IEFVGS		IEFVGS	IEFEXEC	IEFCNTRL	GO		
			IEFDD				
IEFVGT		IEFVGT	IEFDD	IEFCNTRL	GO	18	
			IEFEXEC				
			IEFJOB				
IEFVHA		IEFVHA	IEFCNTRL	IEFCNTRL	GO	14,16	
IEFVHAA		IEFVHAA	IEFCNTRL	IEFCNTRL	GO	16	
IEFVHB		IEFVHB	IEFVHH	IEFCNTRL	GO	16	
IEFVHC		IEFVHC	IEFCNTRL	IEFCNTRL	GO	16	
IEFVHCB		IEFVHCB	IEFCNTRL	IEFCNTRL	GO	16	
IEFVHE		IEFVHE	IEFCNTRL	IEFCNTRL	GO	16	
IEFVHEB		IEFVHEB	IEFCNTRL	IEFCNTRL	GO	16	
IEFVHEC		IEFVHEC	IEFVHH	IEFCNTRL	GO	16	
IEFVHF		IEFVHF	IEFVHH	IEFCNTRL	GO	16	
IEFVHGSS		IEFVHG	IEFVHH	IEFCNTRL	GO	16	
IEFVHH		IEFVHH	IEFVHH	IEFCNTRL	GO	16,53	
IEFVHHB		IEFVHHB	IEFVHH	IEFCNTRL	GO		
IEFVHL		IEFVHL	IEFVHH	IEFCNTRL	GO	16	
IEFVHN		IEFVHN	IEFVHN	IEFVH1	IEFVH1	14,53	20
IEFVHQ		IEFVHQ	IEFCNTRL	IEFCNTRL	GO		
			IEFINTFC				
			IEFDD	IEFCOMMD	IEFVH1		
			IEFEXEC	IEFVH1			
			IEFJOB	IEFVINA			
			IEFCOMMD				
			IEFVHH				
			IEFVGMSS				
			IEFVH1				
			IEFVINA				
IEFVHRSS		IEFVHR	IEFCNTRL	IEFCNTRL	GO		
			IEFDD	IEFVH1	IEFVH1		
			IEFEXEC	IEFCOMMD			
			IEFINITFC				
			IEFVH1				
			IEFVINA	IEFVINA			
			IEFCOMMD				
			IEFJOB				
			IEFVGMSS				
IEFVH1		IEFVH1	IEFVH1	IEFVH1	IEFVH1	14,53,54,55	
IEFVH2		IEFVH2	IEFVH1	IEFVH1	IEFVH1	14	
IEFVINA		IEFVINA	IEFVINA	IEFVINA	GO		
IEFVINB		IEFVINB	IEFVINA	IEFVINA	GO		
IEFVINC		IEFVINC	IEFVINA	IEFVINA	GO		
IEFVIND		IEFVIND	IEFCNTRL	IEFCNTRL	GO		
			IEFEXEC				
IEFVINE		IEFVINE	IEFVINA	IEFVINA	GO		
IEFVJA		IEFVJA	IEFJOB	IEFCNTRL	GO	14	
IEFV15XL		IEFV15XL	IEFALOC2	IEFALOC1	IEFALLOC		
			IEFX5000	IEFALOC2			
			IEFALOC4				
IEFVJIMP		IEFVJ	IEFSTERM	IEFSTERM	GO	46,47	50
IEFVJMSG		IEFVJMSG	IEFSTERM	IEFSTERM	GO		
IEFVKIMP		IEFVK	IEFSELCT	IEFSTERM	GO	22	24
IEFVKMSG		IEFVKMSG	IEFSELCT	IEFSTERM	GO		
IEFVMFAK		IEFVMCVL	IEFMCVOL	IEFMCVOL	IEFMCVOL		
IEFVMLK5		IEFVM6	IEFSELCT	IEFSTERM			

Assembly Modules and Control Sections (Part 5 of 7)

Assembly Module Name	Notes	Control Section Name	Load Modules in Which Assembly Modules are Used			Chart Number	
			18K	44K	100K	Appears As Subr. Block	Flow is Defined
IEFVMLS1		IEFVM1	IEFSELCT	IEFSTERM	GO	24,25	25
IEFVMLS6		IEFVM6	IEFERROR	IEFERROR	GO	25,26	33
IEFVMLS7		IEFVM7	IEFERROR	IEFERROR	GO		
IEFVMMS1		IEFVM1	IEFMCVOL	IEFMCVOL	IEFMCVOL		
IEFVM2LS		IEFVM2	IEFSELCT	IEFSTERM	GO	25,26	29
IEFVM3LS		IEFVM3	IEFSELCT	IEFSTERM	GO	25,26	30
IEFVM4LS		IEFVM4	IEFSELCT	IEFSTERM	GO	25,26	31
IEFVM5LS		IEFVM5	IEFSELCT	IEFSTERM	GO	25,26	32
IEFVM76		IEFVM76	IEFSELCT	IEFSTERM	GO		
IEFWAFAK		IEFWA000	IEFALOC1				
IEFWAD	****	IEFWAD	IEFSTERM	IEFSTERM	GO		
			IEFSELCT	IEFJTERM			
			IEFJTRM1				
		IEFWA002					
IEFWA000		IEFWA7	IEFALOC2	IEFALOC1	IEFALLOC	34	36
IEFWCFAK		IEFWC000	IEFALOC1	IEFALOC1			
			IEFX5000				
			IEFALOC2				
IEFWCIMP		IEFWC000	IEFALOC3	IEFALOC2	IEFALLOC	34	41
		IEFWC002	IEFALOC3	IEFALOC2	IEFALLOC		
IEFWDFAK		IEFWD000	IEFALOC3				
			IEFALOC5				
IEFWD000		IEFWD000	IEFALOC4	IEFALOC1	IEFALLOC	34,35,38	42
				IEFALOC2			
IEFWD001		IEFWD001	IEFALOC4	IEFALOC1	IEFALLOC		
				IEFALOC2			
IEFWMAS1	**	DEVNAMET	DEVNAMET	DEVNAMET	DEVNAMET		
IEFWMSKA	**	DEVMASKT	DEVMASKT	DEVMASKT	DEVMASKT		
IEFWSDIP		IEFWSDIP	IEFVH1	IEFVH1	IEFVH1		
IEFWSTRT		IEFWSTRT	IEFSELCT	IEFSTERM	GO		
IEFWSWIN		IEFWSWIT	IEFALOC2	IEFALOC1	GO		
IEFWTERM		IEFWTERM	IEFJTRM1	IEFJTERM	GO		
IEFW21SD		IEFW21SD	IEFSELCT	IEFSTERM	GO	22	23
IEFW22SD		IEFW22SD	IEFSTERM	IEFSTERM	GO	46	
IEFW23SD		IEFW23SD	IEFJTRM1	IEFJTERM	GO	46	
IEFW31FK		IEFW31SD	IEFJTRM1				
IEFW31SD		IEFW31SD	IEFJTRM2	IEFJTERM	GO	46	
IEFW41SD		IEFW41SD	IEFALOC5	IEFALOC2	IEFALLOC		
IEFW42SD		IEFW42SD	IEFSTERM	IEFSTERM	GO	46	
IEFXAFAK		IEFXA	IEFSELCT	IEFSTERM	IEFALERR		35
				IEFALOC2			
IEFXAMSG		IEFXAMSG	IEFALOC1	IEFALOC1	IEFALLOC		
IEFXCSSS		IEFXA	IEFALOC1	IEFALOC1	IEFALLOC	32,38	33
		IEFXAB00					
IEFXH000		IEFXH000	IEFX5000	IEFALOC2	IEFALLOC		
			IEFALOC3				
IEFXJFAK		IEFXJ000	IEFALOC2		IEFALLOC		
			IEFX5000				
			IEFALOC3				
IEFXJIMP		IEFXJ000	IEFALOC1	IEFALOC1	IEFALLOC	38	
				IEFALOC2			
IEFXJMSG		IEFXJMSG	IEFALOC1	IEFALOC1	IEFALLOC		
				IEFALOC2			
IEFXKFAK		IEFXK000			IEFALLOC		
IEFXKIMP		IEFXK000	IEFALOC4	IEFALOC1	IEFALLOC		
			IEFALOC5	IEFALOC2			
IEFXKMSG		IEFXKMSG	IEFALOC4	IEFALOC1	IEFALERR		
			IEFALOC5	IEFALOC2			
IEFXTFAK		IEFXT000	IEFALOC4				
IEFXTDMY		IEFXTDMY	IEFALOC5	IEFALOC2	IEFALLOC		

Assembly Modules and Control Sections (Part 6 of 7)

Assembly Module Name	Notes	Control Section Name	Load Modules in Which Assembly Modules are Used			Chart Number	
			18K	44K	100K	Appears As Subr. Block	Flow is Defined
IEFXTMSG		IEFXTMSG	IEFALOC5	IEFALOC2	IEFALLOC		
IEFXT00D		IEFXT000	IEFALOC5	IEFALOC2	IEFALLOC	34	43
IEFXT002		IEFXT002	IEFALOC5	IEFALOC2	IEFALLOC		45
IEFXT003		IEFXT003	IEFALOC5	IEFALOC2	IEFALLOC		44
IEFXVMSG		IEFXVMSG	IEFALOC4	IEFALOC1	IEFALLOC		
IEFXVNSL	*****	IEFXVNSL	IEFALOC4	IEFALOC1	IEFALLOC		
IEFXV001		IEFXV001	IEFALOC4	IEFALOC1	IEFALLOC	34	37
IEFXV002		IEFXV002	IEFALOC4	IEFALOC1	IEFALLOC	37	38
IEFXVFAK		IEFXV001	IEFALOC2				
IEFX1FAK		IEFXJ000	IEFALOC4				
IEFX2FAK		IEFX5000	IEFALOC4				
IEFX3FAK		IEFWC000	IEFALOC4			34	40
IEFX300A		IEFX3000	IEFALOC2	IEFALOC1	IEFALLOC		
			IEFX5000				
			IEFALOC4	IEFALOC2			
IEFX5FAK		IEFX5000	IEFALOC2	IEFALOC1			
IEFX5000		IEFX5000	IEFX5000	IEFALOC2	IEFALLOC	34,53	40
IEFYNFAK		IEFYN	IEFSELCT	IEFALOC1			
			IEFALCO1	IEFERROR	IEFALLOC		
			IEFALOC4	IEFIDUMP			
			IEFALOC5	IEFALOC2	IEFMCVOL		
			IEFERROR	IEFMCVOL			
			IEFIDUMP				
			IEFMCVOL				
IEFYNIMP		IEFYN	IEFSTERM	IEFSTERM	GO	46	
IEFYNMSG		IEFYNMSG	IEFSTERM	IEFSTERM	GO		
IEFYPJB3		IEFYP	IEFSTERM	IEFSTERM	GO	45,47	49
IEFYPMSG		IEFYPMSG	IEFSTERM	IEFSTERM	GO		
IEFYSSMB		IEFYS	IEFSTERM	IEFSTERM	GO		
			IEFSELCT	IEFALOC1	IEFIDUMP		
			IEFALOC1	IEFALOC2	IEFALLOC		
			IEFALOC4	IEFERROR			
			IEFALOC5	IEFJTERM	IEFMCVOL		
			IEFJTRM1	IEFIDUMP			
			IEFERROR	IEFMCVOL			
			IEFIDUMP				
			IEFMCVOL				
IEFZAFAK		IEFZA	IEFSTERM	IEFSTERM			
IEFZAJB3		IEFZA	IEFJTRM1	IEFJTERM	GO	46	51
IEFZGJB1		IEFZGJ	IEFJTRM1	IEFCNTRL	GO	47	53
IEFZGMSG		IEFZGMSG	IEFSTERM	IEFSTERM	GO		
			IEFJTRM1	IEFJTERM			
IEFZGST1		IEFZG	IEFSTERM	IEFSTERM	GO	46,50	52
IEF2GST2		IEF2G2	IEFSTERM	IEFSTERM	GO	46	
IEFZHFAK		IEFZPOQM	IEFJTRM1	IEFJTERM			
IEFZHMSG		IEFZH	IEFSTERM	IEFSTERM	GO	46	
			IEFJTRM1	IEFJTERM			
IEF04FAK		IEFSD004	IEFALOC5				
IEF08FAK		IEFSD008	IEFSTERM	IEFSTERM			
			IEFINTFC				
			IEFJTRM2				
IEF09FAK		IEFSD009	IEFSTERM	IEFCNTRL			
			IEFINTFC				
IEF23FAK		IEFW23SD	IEFINTFC				
IEF35DUM			IEFJTERM	IEFCNTRL	GO		
IEF7KGXX		IEFKG	IEFINTFC	IEFCNTRL	GO		
IEF7KPXX		IEFVHM	IEFCOMMD	IEFCOMMD	GO	16	
IEF7K1XX		IEFK1	IEFVH1	IEFVH1	IEFVH1	14	
IEF7K2XX		IEFK2	IEFVH1	IEFVH1	IEFVH1	14	
IEF7K3XX		IEFK3	IEFVHN	IEFVH1	IEFVH1	14	

Assembly Modules and Control Sections (Part 7 of 7)

Assembly Module Name	Notes	Control Section Name	Load Modules in Which Assembly Modules are Used			Chart Number	
			18K	44K	100K	Appears As Subr. Block	Flow is Defined
IEZDCODE IEZNCODE		IEZDCODE IEZNCODE	IEZDCODE IEFVINA IEZNCODE	IEZDCODE IEFVINA IEZNCODE	GO GO		

Notes:
*Assembly modules in SYS1.NUCLEUS data set.
**Modules are assembled during system generation.
***Assembly modules in SYS1.SVCLIB data set.
****IEFACTFK may replace IEFACTLK, IEFACTRT, and IEFWAD during system generation.
*****IEFXVNSL is a simple exit and return subroutine that the user may replace with his own subroutine for processing nonstandard labels.

Control Sections and Assembly Modules

The following list provides a cross-reference between job management control section (CSECT) names, which appear in alphameric order, and the corresponding assembly module names. Control section names are also listed in the preceding assembly module to load module cross reference table.

CSECT NAME	ASSEMBLY MODULE NAME
DEVMASKT	IEFWMSKA
DEVNAMET	IEFWMAS1
IEEBB1	IEEMCRFK
IEEBB1	IEEMCR01
IEEGESTO	IEEGES01
IEEGK1GM	IEEGK1GM
IEEICCAN	IEEILCDM
IEEICCAN	IEEILC01
IEEICN01	IEECNDUM
IEEICN01	IEEICN01
IEEICRDR	IEEREADR
IEEICWTR	IEEWRITR
IEEIC1PE	IEESTART
IEEIC2NQ	IEEIC2NQ
IEEIC3JF	IEEIC3JF
IEEMCREP	IEEMCREP
IEEQOT00	IEEQOT00
IEEVSMSG	IEEVSMDM
IEFACTLK	IEFACTLK
IEFACTLK	IEFACTFK
IEFACTRT	IEFACTRT
IEFBR14	IEFBR14
IEFCVOL1	IEFMCVOL
IEFCVOL2	IEFMCVOL
IEFCVOL3	IEFMCVOL
IEFDEVPT	IEFDEVPT
IEFIDMPM	IEFIDMPM
IEFIDUMP	IEFIDFAK
IEFIDUMP	IEFIDUMP
IEFINTQS	IEFINTQA
IEFKG	IEFKGDUM
IEFKG	IEF7KGXX
IEFK1	IEF7K1XX
IEFK1	IEFK1AK
IEFK1MSG	IEFK1MSG
IEFK2	IEF7K2XX
IEFK3	IEF7K3XX
IEFPRES	IEFPRES
IEFQMSSS	IEFQMSSS
IEFSCAN	IEFSCAN
IEFSD001	IEFSD001
IEFSD002	IEFSD002
IEFSD003	IEFSD003
IEFSD004	IEFSD004
IEFSD004	IEF04FAK
IEFSD006	IEFSD006
IEFSD007	IEFSD007
IEFSD008	IEFSD008
IEFSD008	IEF08FAK
IEFSD009	IEFSD009
IEFSD009	IEF09FAK
IEFSD010	IEFSD010
IEFSD011	IEFSD011
IEFSD012	IEFSD012
IEFSD017	IEFSD017
IEFSD035	IEF35DUM
IEFSD059	IEFSD059
IEFSD088	IEFSD088
IEFSD089	IEFSD089
IEFSD090	IEFSD090
IEFSD094	IEFSD094
IEFSD095	IEFSD095
IEFSD89M	IEFSD089
IEFSEPAR	IEFSEPAR
IEFSGOPT	IEFSGOPT
IEFVDA	IEFDAFAK
IEFVDA	IEFVDA
IEFVDBSD	IEFVDDUM
IEFVEA	IEFEAFAK
IEFVEA	IEFVEA
IEFVFA	IEFVFA
IEFVFA	IEFFAFAK
IEFVFB	IEFVFB
IEFVHB	IEFVHB
IEFVGI	IEFVGI
IEFVGK	IEFVGK
IEFVGM	IEFVGMSS
IEFVGM	IEFVGMEP
IEFVGM	IEFGMFAK
IEFVGM1	IEFVGM1
IEFVGM2	IEFVGM2
IEFVGM3	IEFVGM3
IEFVGM4	IEFVGM4
IEFVGM5	IEFVGM5
IEFVGM6	IEFVGM6
IEFVGM7	IEFVGM7
IEFVGM8	IEFVGM8
IEFVGM9	IEFVGM9
IEFVGM10	IEFVGM10
IEFVGM11	IEFVGM11
IEFVGM12	IEFVGM12
IEFVGM13	IEFVGM13
IEFVGM14	IEFVGM14
IEFVGM15	IEFVGM15
IEFVGM16	IEFVGM16
IEFVGM17	IEFVGM17
IEFVGM18	IEFVGM18
IEFVGM70	IEFVGM70
IEFVGM71	IEFVGM71
IEFVGM78	IEFVGM78
IEFVGS	IEFVGS
IEFVGT	IEFVGT
IEFVHA	IEFHAFAK
IEFVHA	IEFVHA
IEFVHAA	IEFHAAFK
IEFVHAA	IEFVHAA
IEFVHB	IEFVHB
IEFVHB	IEFHBFAK
IEFVHC	IEFVHC
IEFVHC	IEFHCFAK
IEFVHCB	IEFHCBFK
IEFVHCB	IEFVHCB
IEFVHE	IEFVHE
IEFVHE	IEFHEFAK
IEFVHEB	IEFVHEB
IEFVHEB	IEFHEBFK
IEFVHEC	IEFVHEC
IEFVHEC	IEFHECFK

CSECT NAME	ASSEMBLY MODULE NAME
IEFVHF	IEFHFFAK
IEFVHF	IEFVHF
IEFVHG	IEFVHGSS
IEFVHH	IEFVHH
IEFVHH	IEFHHFAK
IEFVHHB	IEFVHHB
IEFVHL	IEFVHL
IEFVHL	IEFHLFAK
IEFVHM	IEFHMFAK
IEFVHM	IEF7KPXX
IEFVHN	IEFVHN
IEFVHQ	IEFVHQ
IEFVHR	IEFVHRSS
IEFVH1	IEFVH1
IEFVH2	IEFVH2
IEFVINA	IEFVINA
IEFVINB	IEFVINB
IEFVINC	IEFVINC
IEFVIND	IEFVIND
IEFVINE	IEFVINE
IEFVJA	IEFJAFAK
IEFVJA	IEFVJA
IEFVJMSG	IEFVJMSG
IEFVJ	IEFVJIMP
IEFVJ	JTERM030
IEFVKMSG	IEFVKMSG
IEFVMCVL	IEFVMLS1
IEFVK	IEFVKIMP
IEFVMQMI	IEFVMLS1
IEFVMPOQ	IEFVMLS1
IEFVM1	IEFVMLS1
IEFVM2	IEFVM2LS
IEFVM3	IEFVM3LS
IEFVM4	IEFVM4LS
IEFVM5	IEFVM5LS
IEFVM6	IEFVMLK5
IEFVM6	IEFVMLS6
IEFVM76	IEFVM76
IEFVM7	IEFVMLS7
IEFV15XL	IEFV15XL
IEFWAD	IEFWAD
IEFWA000	IEFWAFAK
IEFWA000	IEFWA000
IEFWA002	IEFWA000
IEFWA7	IEFWA000
IEFWC000	IEFWCFAK
IEFWC000	IEFWCIMP
IEFWC000	IEFX3FAK
IEFWC002	IEFWCIMP
IEFWDMSG	IEFWD000
IEFWD000	IEFWDFAK
IEFWD000	IEFWD000
IEFWD001	IEFWD001
IEFWSDIP	IEFWSDIP
IEFWSTRT	IEFWSTRT
IEFWSWIT	IEFWSWIN
IEFWTERM	IEFWTERM
IEFW21SD	IEFW21SD
IEFW22SD	IEFW22SD
IEFW23SD	IEFW23SD
IEFW23SD	IEF23FAK
IEFW31SD	IEFW31SD
IEFW31SD	IEFW31FK
IEFW41SD	IEFW41SD
IEFW42SD	IEFW42SD
IEFXAMSG	IEFXAMSG
IEFXA	IEFXAFAK
IEFXA	IEFXCSSS
IEFXAB00	IEFXCSSS
IEFXH000	IEFXH000
IEFXJMSG	IEFXJMSG
IEFXJ000	IEFXJFAK
IEFXJ000	IEFXJIMP
IEFXJ000	IEFX1FAK
IEFXKMSG	IEFXKMSG
IEFXK000	IEFXKIMP
IEFXTDMY	IEFXTDMY
IEFXTMSG	IEFXTMSG
IEFXT000	IEFXT00D
IEFXT002	IEFXT002
IEFXT003	IEFXT003
IEFXT000	IEFXTFAK
IEFXVMSG	IEFXVMSG
IEFXVNSL	IEFXVNSL
IEFXV001	IEFXVFAK
IEFXV001	IEFXV001
IEFXV002	IEFXV002
IEFX3000	IEFX300A
IEFX5000	IEFX2FAK
IEFX5000	IEFX5FAK
IEFX5000	IEFX5FAK
IEFYNIMP	IEFYNIMP
IEFYNMSG	IEFYNMSG
IEFYN	IEFYNFAK
IEFYN	IEFWTERM020
IEFYPMSG	IEFYPMSG
IEFYP	IEFYPJB3
IEFYS	IEFYSSMB
IEFZA	IEFZAFAK
IEFZA	IEFZAJB3
IEFZGMSG	IEFZGMSG
IEFZGJ	IEFZGJB1
IEFZG	IEFZGST1
IEFZG2	IEFZGST2
IEFZH	IEFZHMSG
IEFZPOQM	IEFZHFAK
IEZDCODE	IEZDCODE
IEZNCODE	IEZNCODE
SPRINTER	IEFPRTXX

Appendix D: List of Acronyms

The following list contains the full name associated with each acronym used in this publication:

Acronym	Name
ACB	Allocate control block
ACT	Account control table
AVR	Automatic volume recognition
AVT	Allocate volume table
AWA	Auxiliary work area
AWT	Allocate work table
BPAM	Basic partitioned access method
CCW	Channel control word
CLT	Channel load table
CSCB	Command scheduling control block
CSECT	Control section
CVT	Communications vector table
DADSM	Direct access device space management
DCB	Data control block
DEB	Data extent block
DMT	Device mask table
DNT	Device name table
DSCB	Data set control block
DSNAME	Data set name
ECB	Event control block
GDG	Generation data group
I/O	Input/output
IPL	Initial program load
IRB	Interrupt request block
IWA	Interpreter work area
JCL	Job control language
JCT	Job control table
JFCB	Job file control block
JSCB	Job step control block
KBT	Keyword branch table
LCT	Linkage control table
LWA	Local work area
NEL	Interpreter entrance list
NIP	Nucleus initialization program
NRWT	New reader/writer table
NSL	Non-standard label
PCP	Primary control program
PDQ	Passed data set queue
PDS	Partitioned data set
PDT	Parameter descriptor table
PUD	Potential user on device
QSAM	Queued sequential access method
SCT	Step control table
SIOT	Step input/output table
SMB	System message block
SYSGEN	System generation
SYSIN	System input device
SYSOUT	System output device
TCB	Task control block
TIOT	Task input/output table
TTR	Auxiliary storage address on direct access device
UCB	Unit control block
VCON	variable constant
VOLT	Volume table
WTO	Write-to-operator
WTOR	Write-to-operator with reply
WTP	Write-to-programmer
WTPCB	Write-to-programmer control block

Chart 01. Job Management

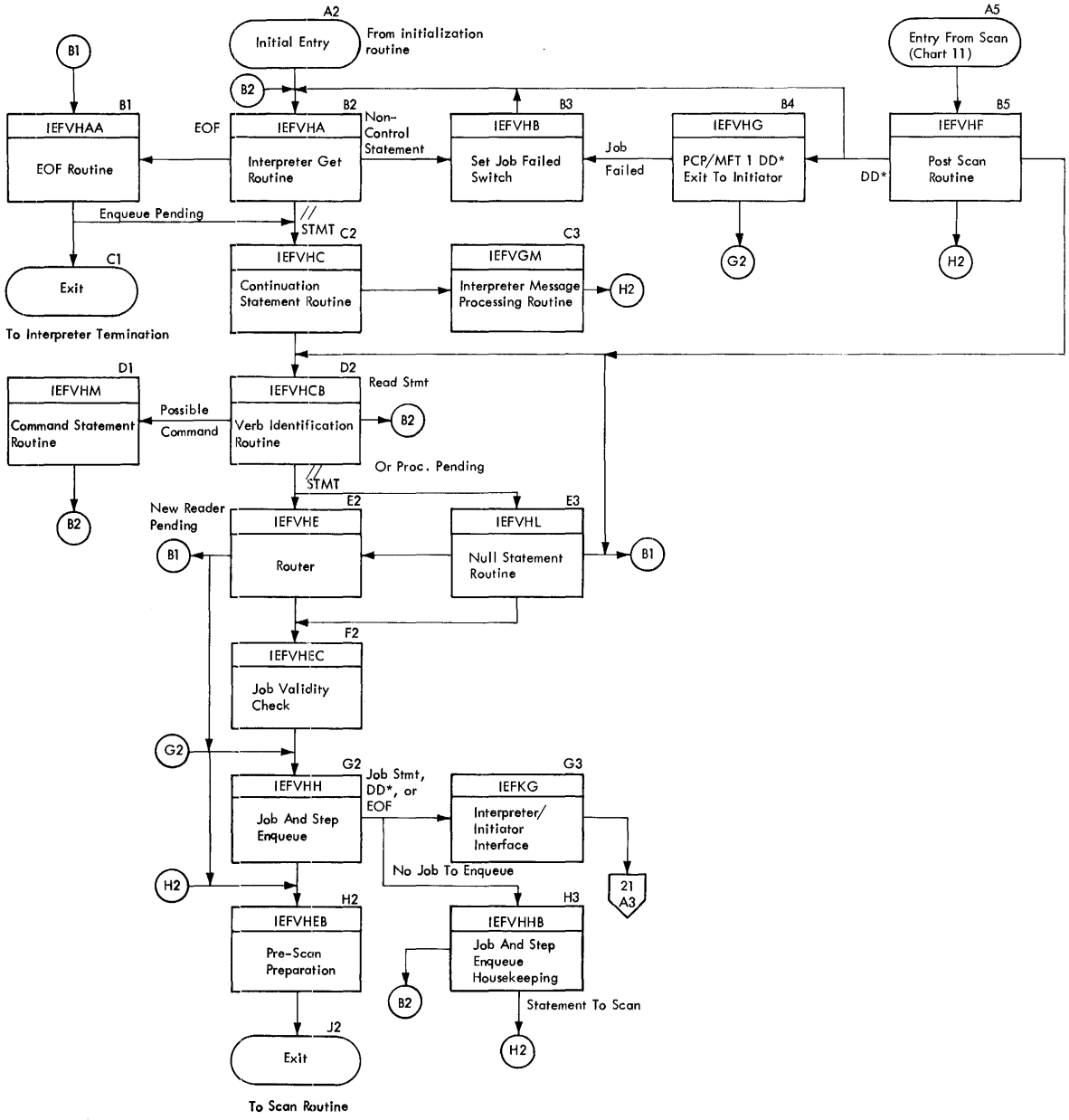


Chart 02. Master Scheduler

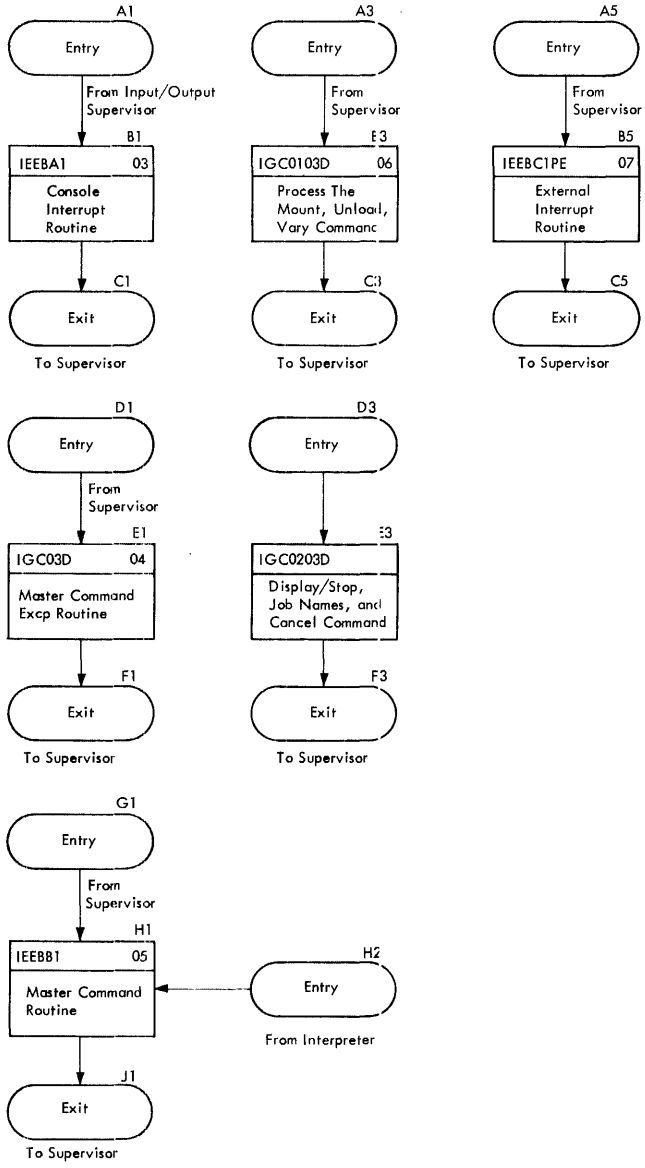
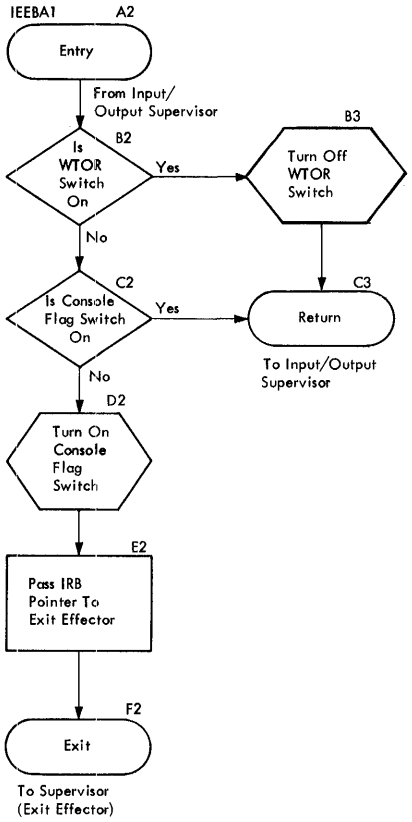


Chart 03. Console Interrupt Routine



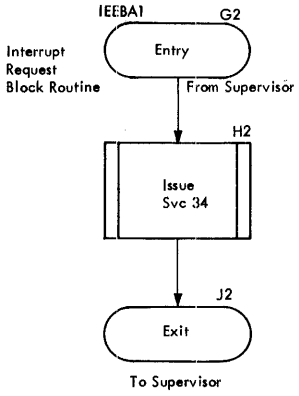


Chart 04. Master Command EXCP Routine

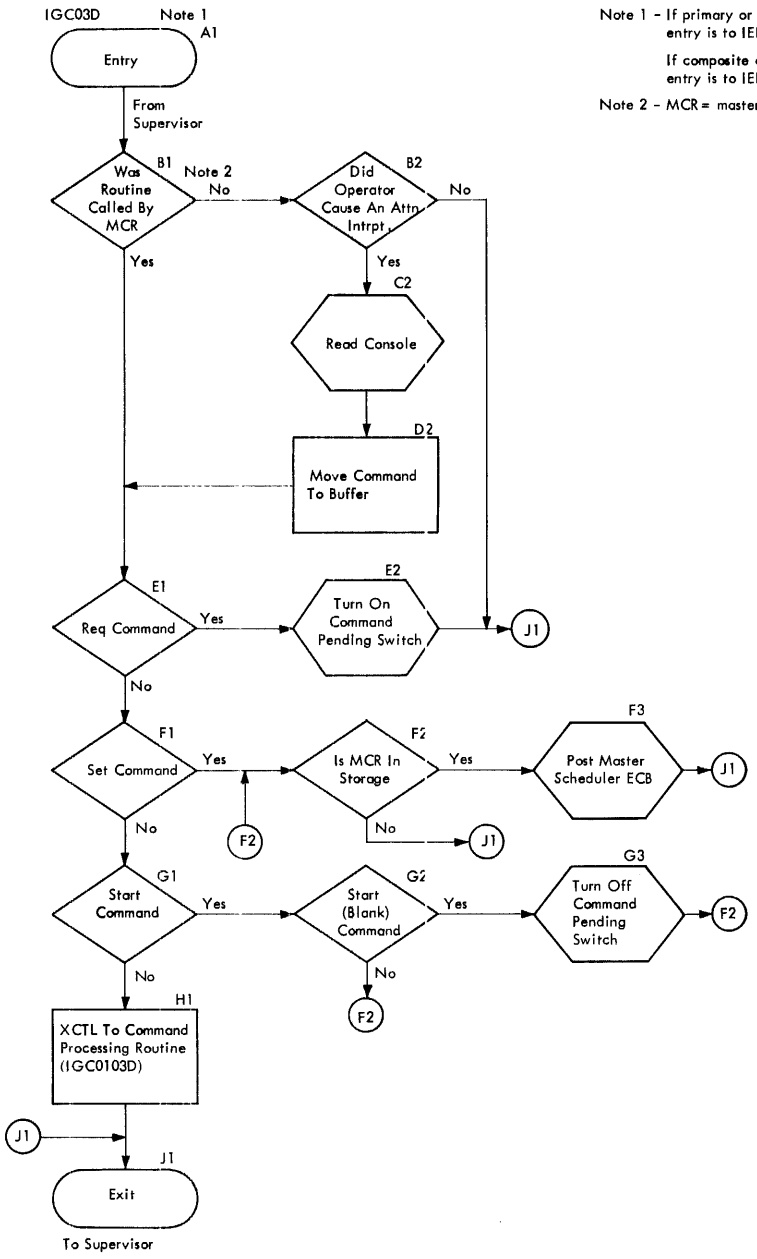
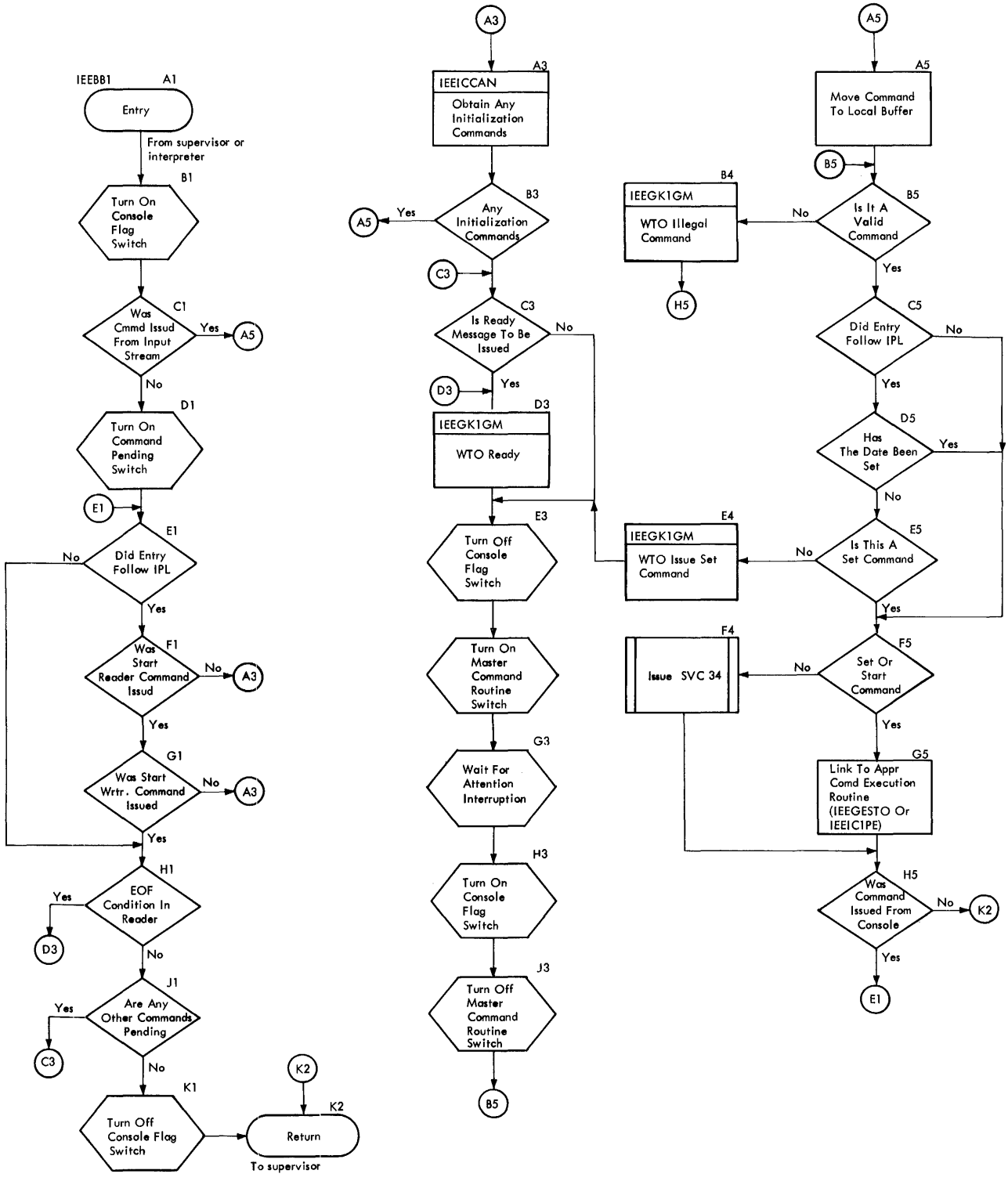
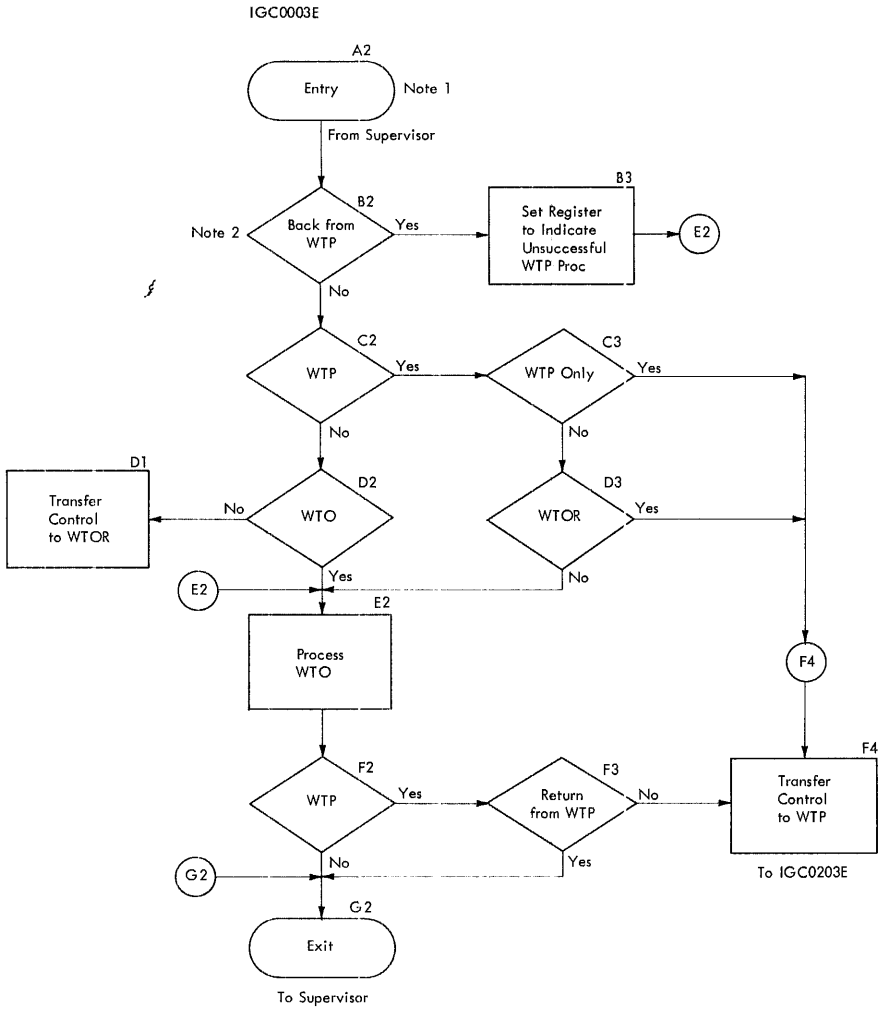


Chart 05. Master Command Routine



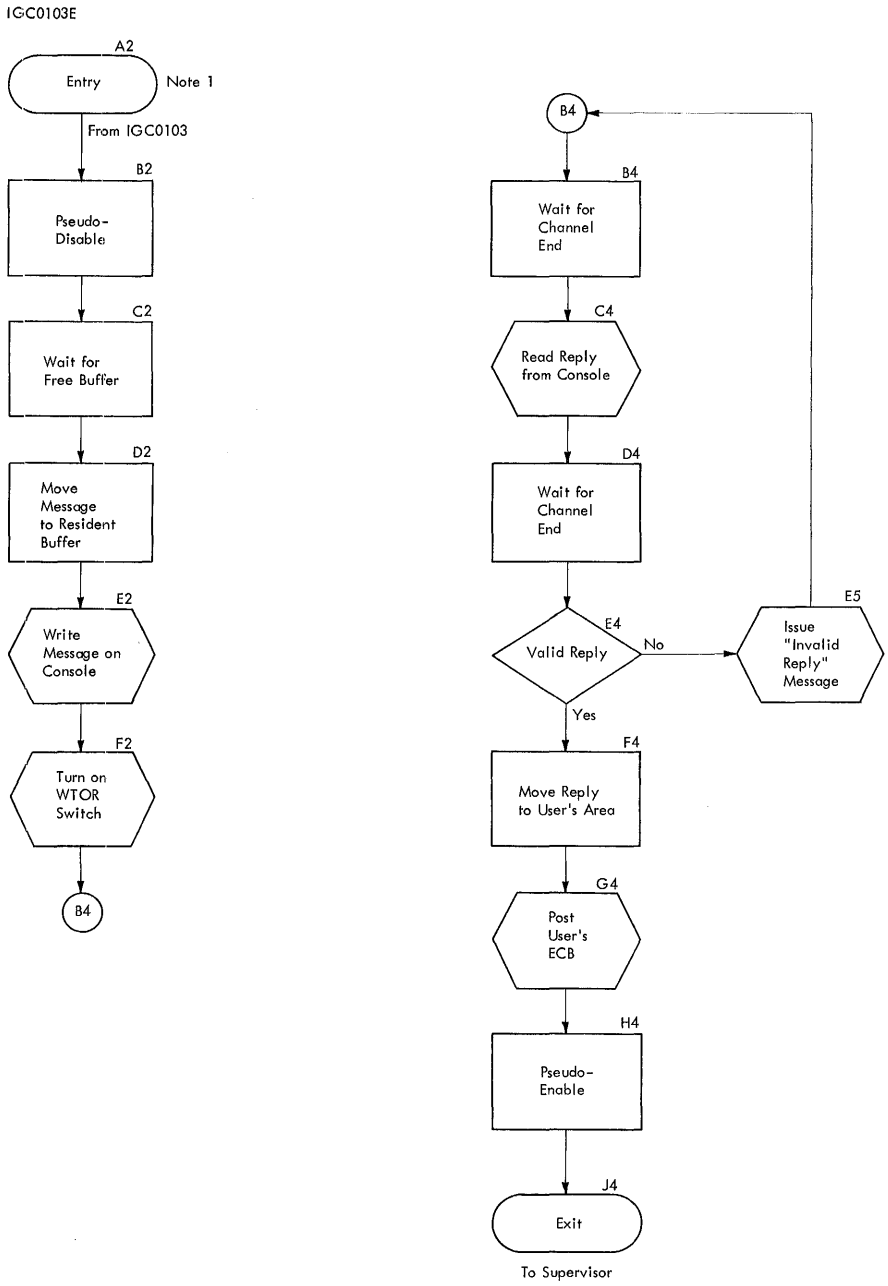
• Chart 06. Write-to-Operator Routine



Note 1: If primary or alternate console is in use, entry is to IEEWTC00.
If composite console is in use, entry is to IEEWTR00.

Note 2: WTP returns to WTO only if message processing was unsuccessful.

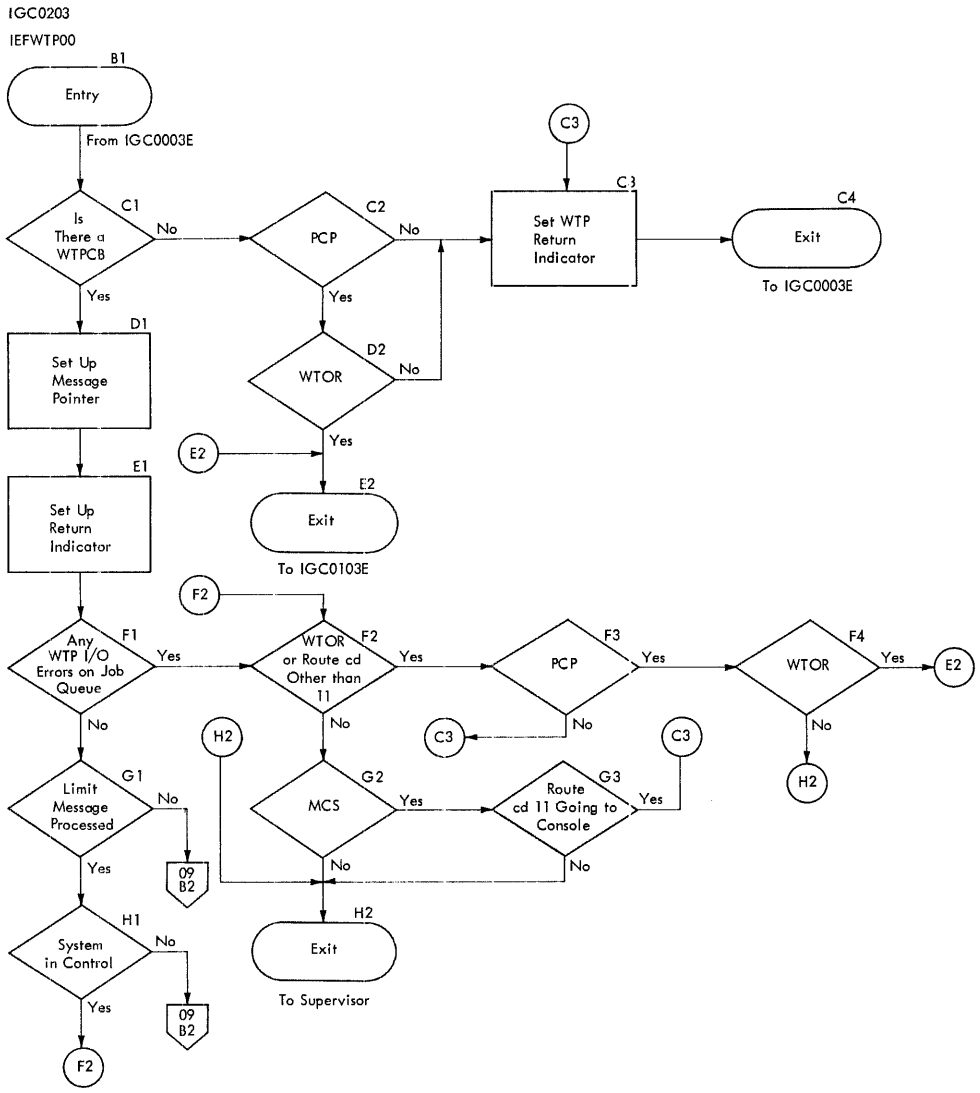
• Chart 07. Write-to-Operator with Reply Routine



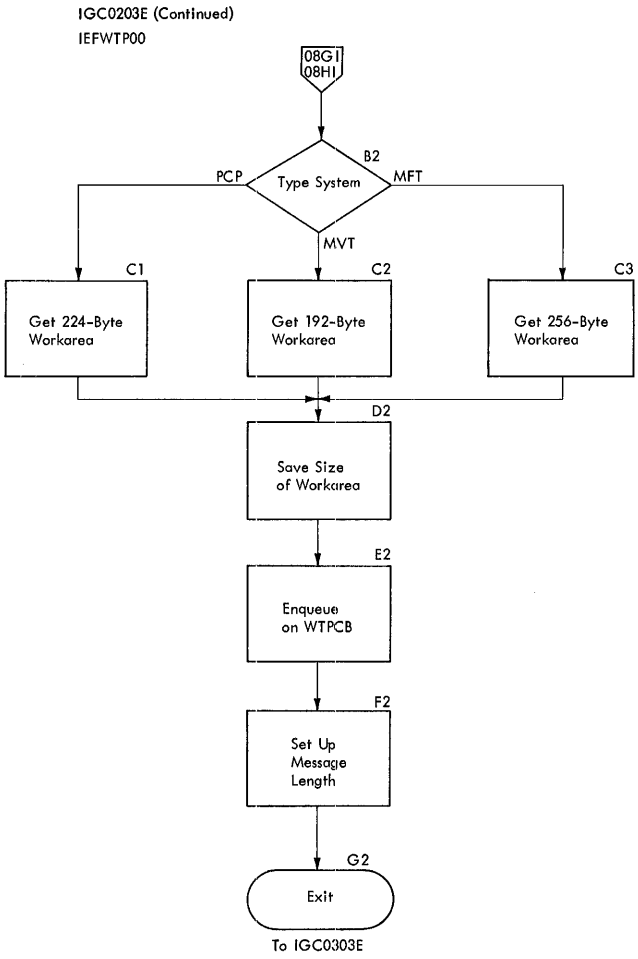
Note 1: If primary or alternate console is in use, entry is to IEEWTC01
If composite console is in use, entry is to IEEWTR01.

Note also that this routine can ABEND to user if parameter list is invalid or if reply area is out of bounds.

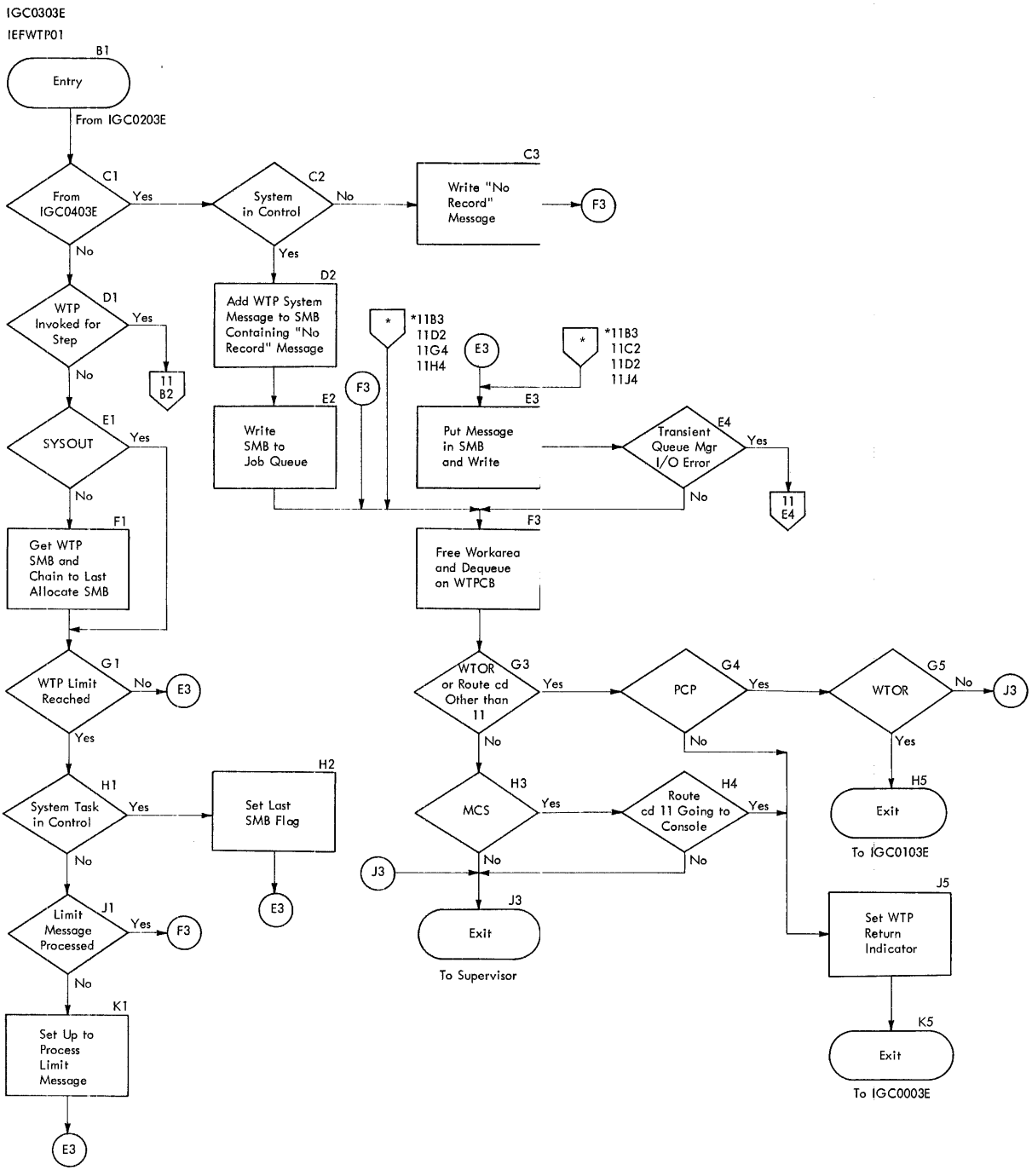
• Chart 08. Write-to-Programmer Routines (Part 1 of 5)



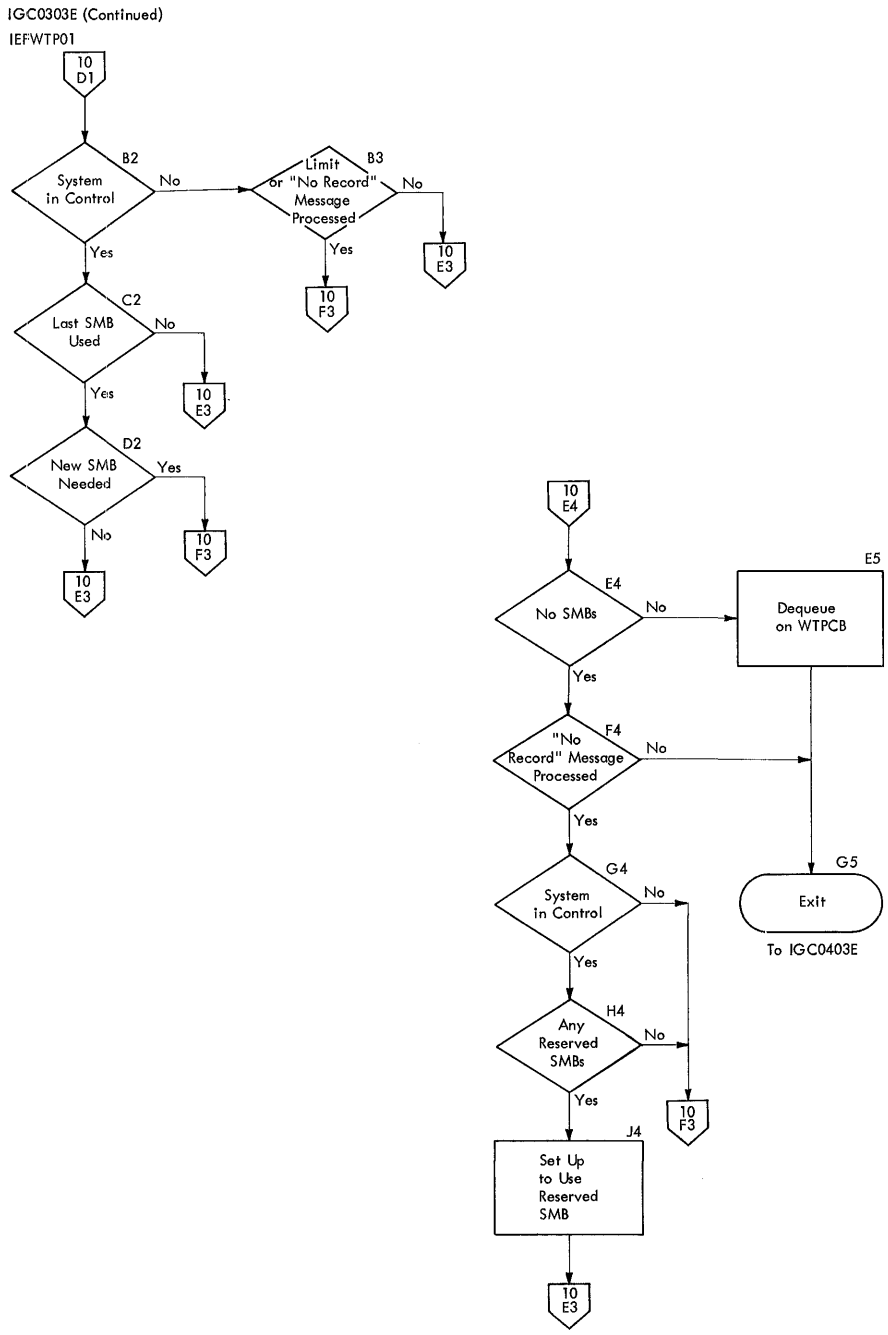
• Chart 09. Write-to-Programmer Routines (Part 2 of 5)



• Chart 10. Write-to-Programmer Routines (Part 3 of 5)



• Chart 11. Write-to-Programmer Routines (Part 4 of 5)



• Chart 12. Write-to-Programmer Routines (Part 5 of 5)

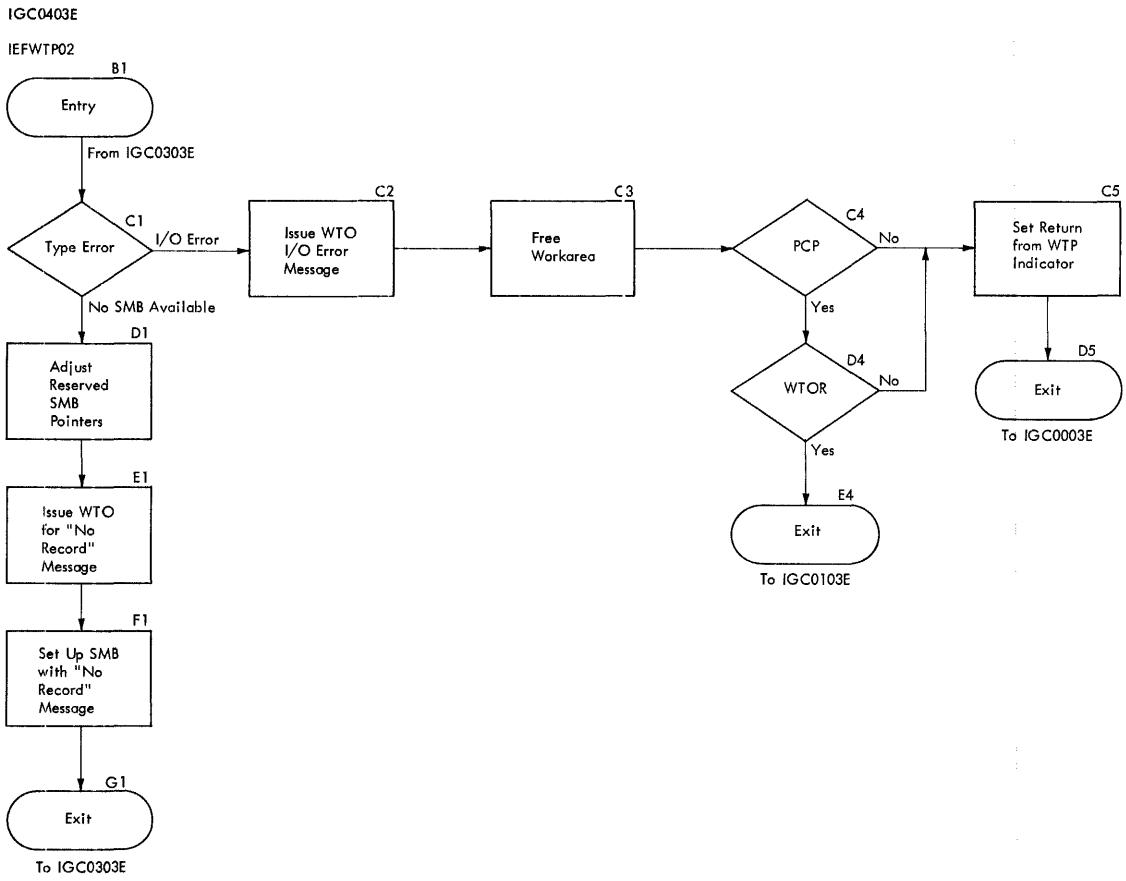


Chart 13. External Interrupt Routine

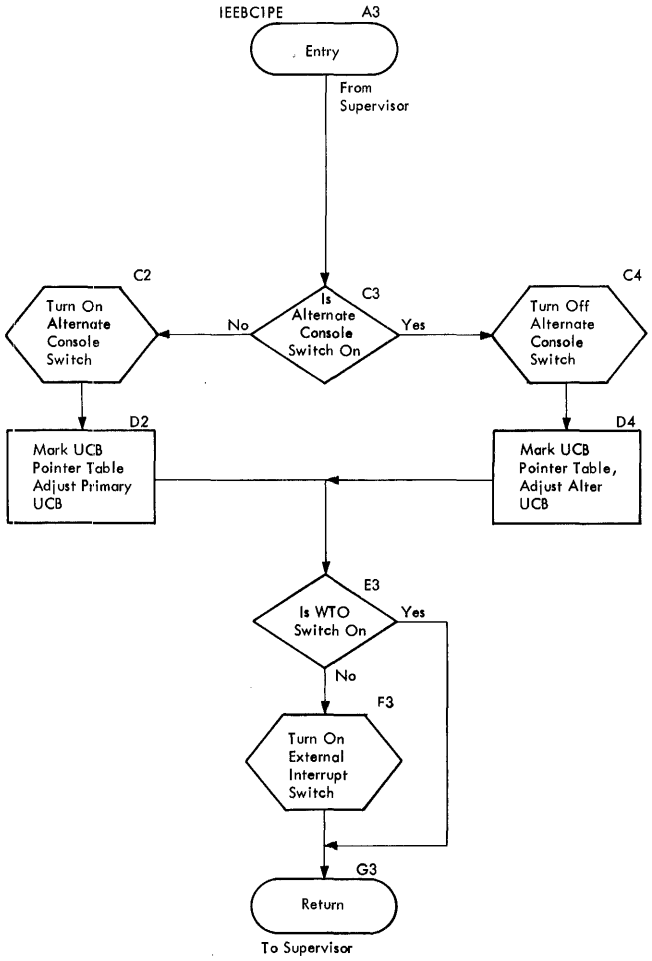


Chart 14. Interpreter Control Flow

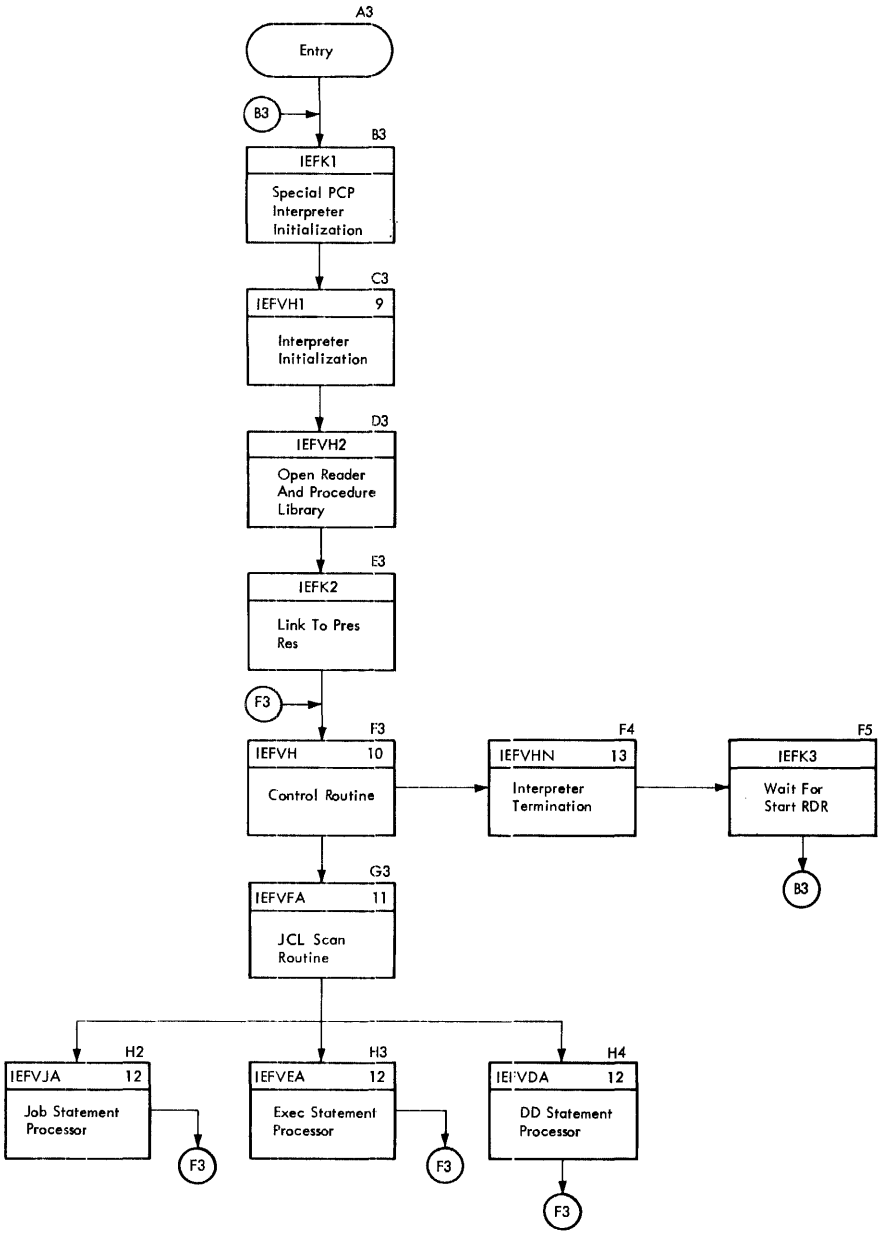


Chart 15. Interpreter Initialization

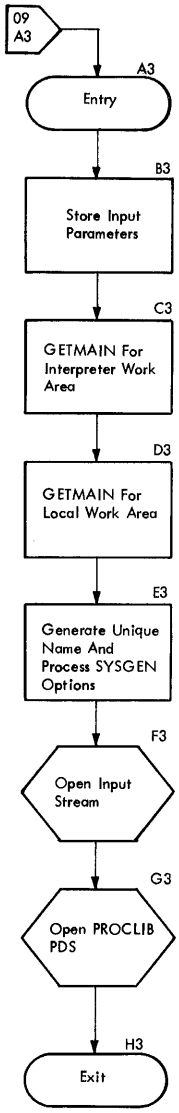


Chart 16. Interpreter Control Routine

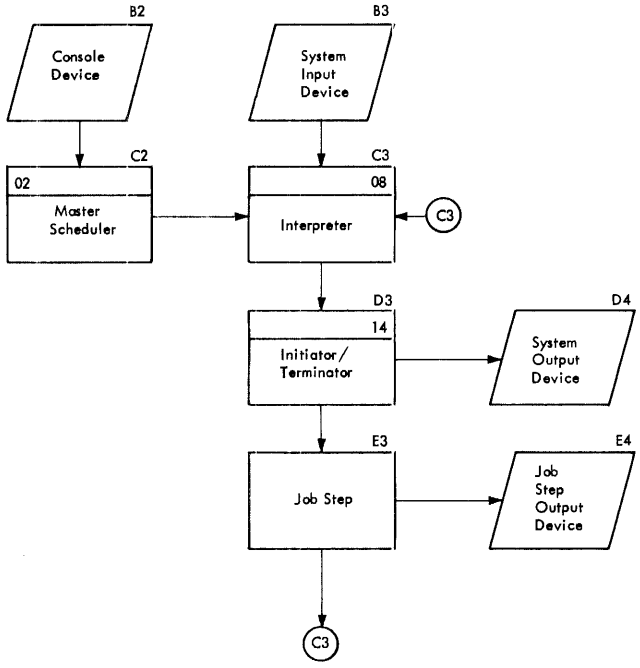
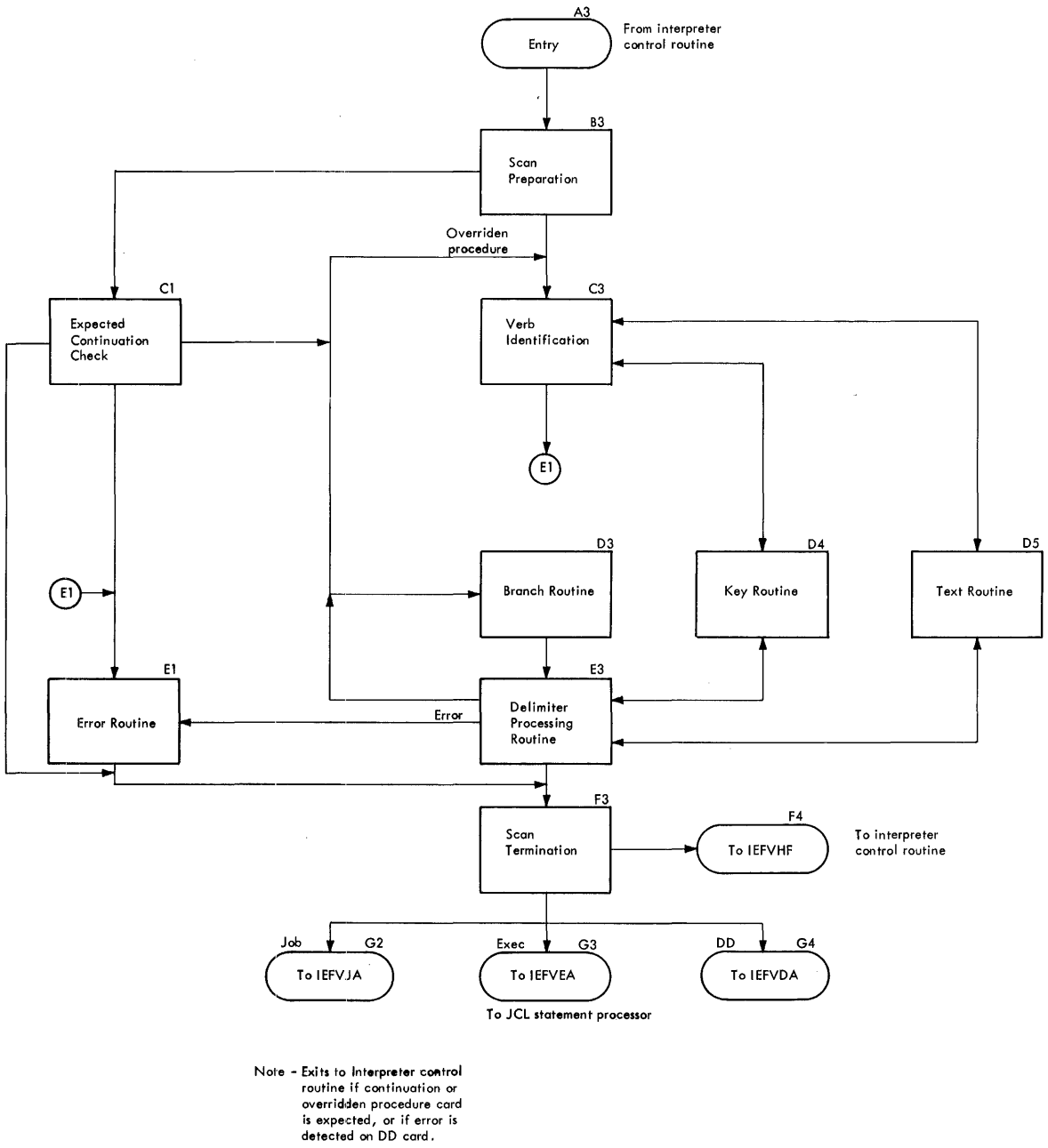


Chart 17. Interpreter Scan Routine

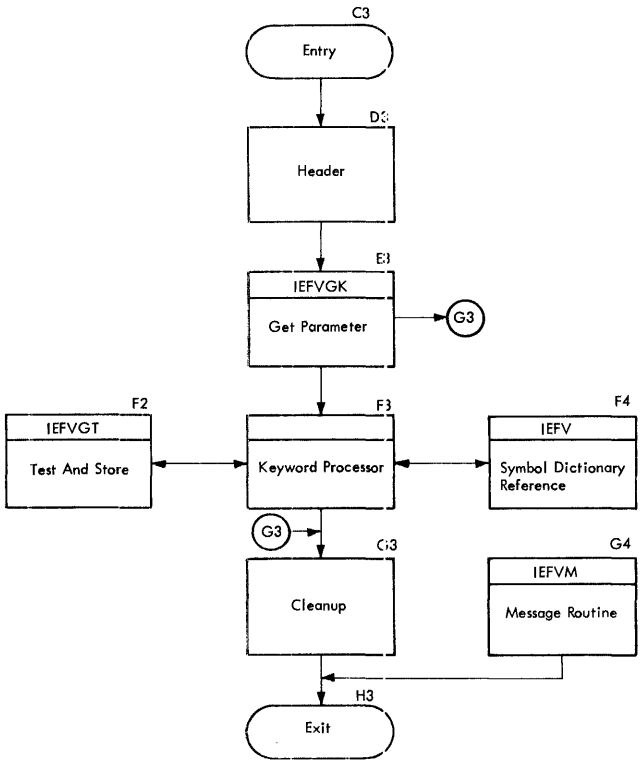


Note - Exits to interpreter control routine if continuation or overridden procedure card is expected, or if error is detected on DD card.

Chart 18. JCL Statement Processors

Note - This chart shows control flow in all three statements processors. Each statement processor is entered from the JCL scan routine and passes control to the control routine.

Each statement processor includes several keyword processors, most of which use IEFVGT and IEFV as subroutines.



• Chart 19. In-Stream Procedure Routines

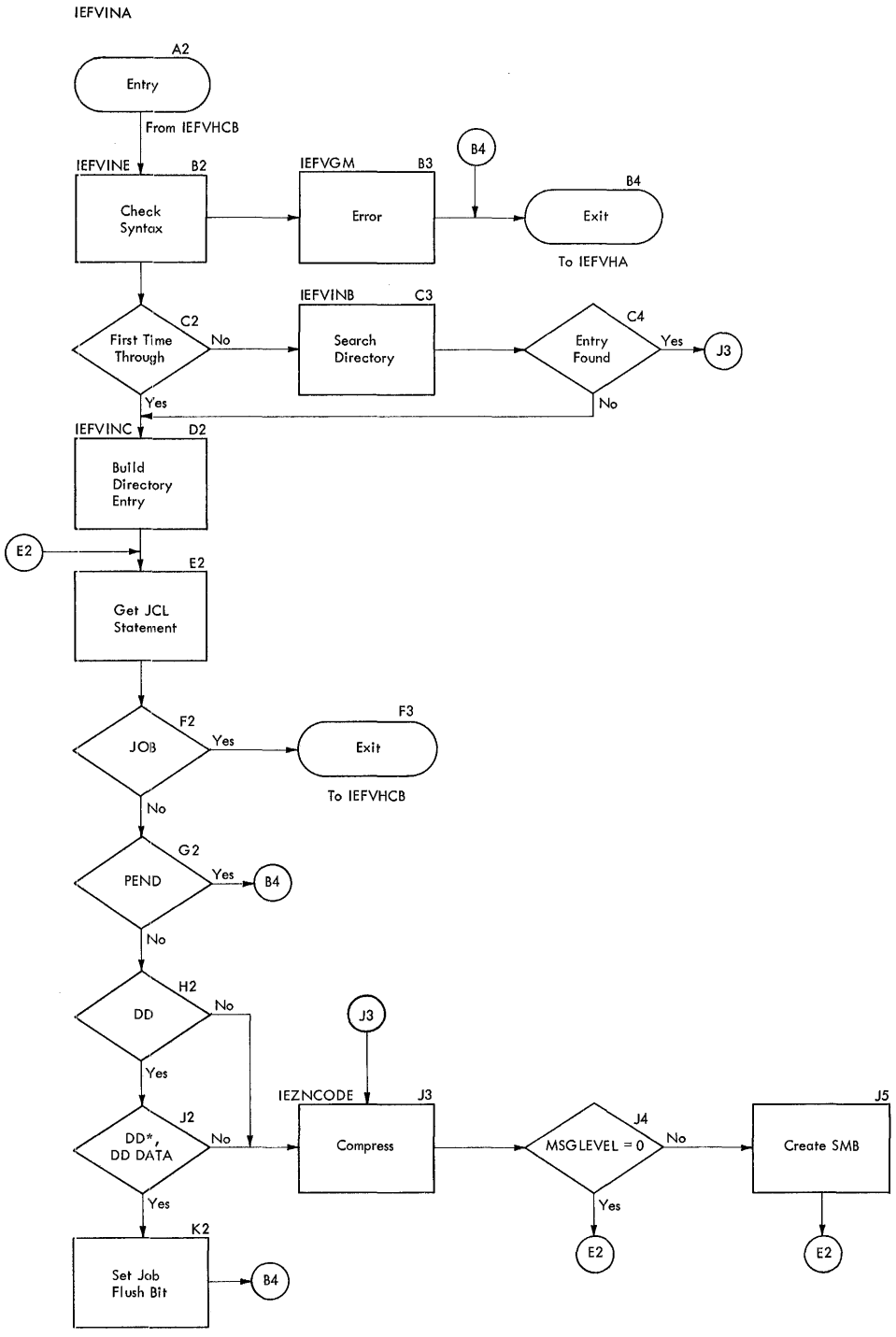


Chart 20. Interpreter Termination

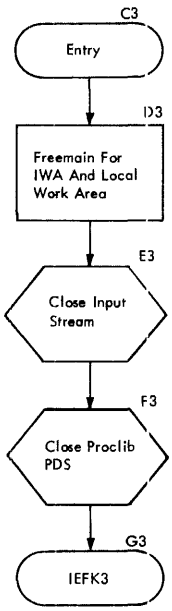


Chart 21. Initiator/Terminator

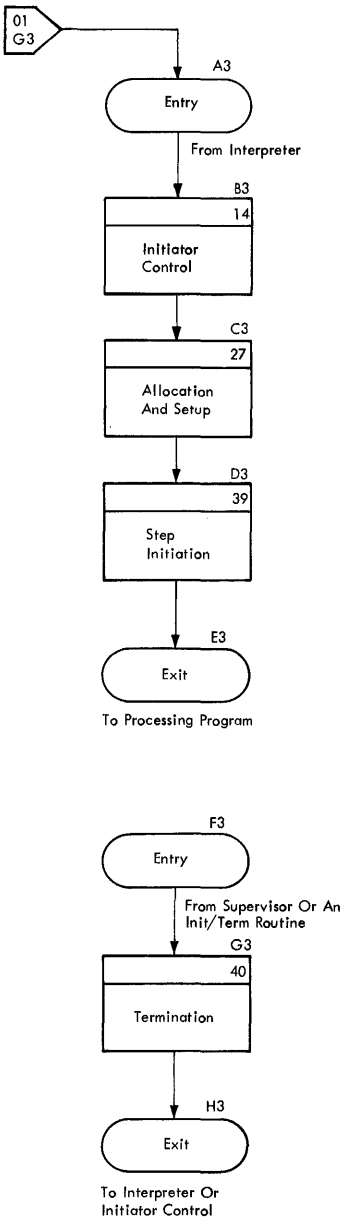
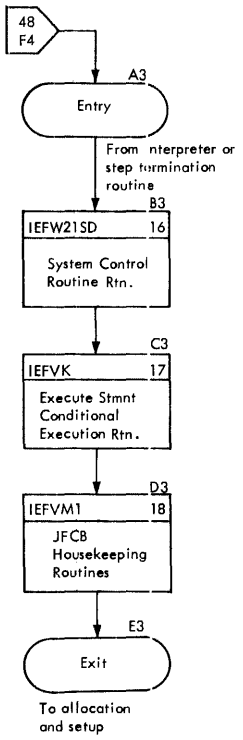


Chart 22. Initiator Control



Entry is from the interpreter when a JOB, NULL, DD *, or DD DATA statement is encountered in the input job stream

Chart 23. System Control Routine

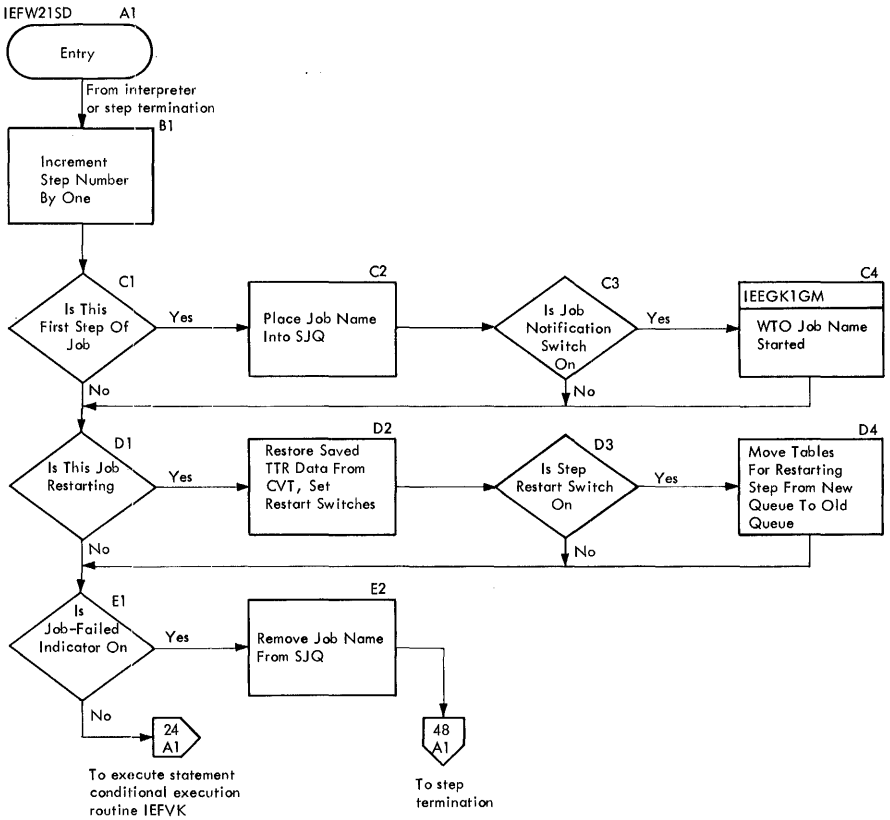


Chart 24. Execute Statement Conditional Execution Routine

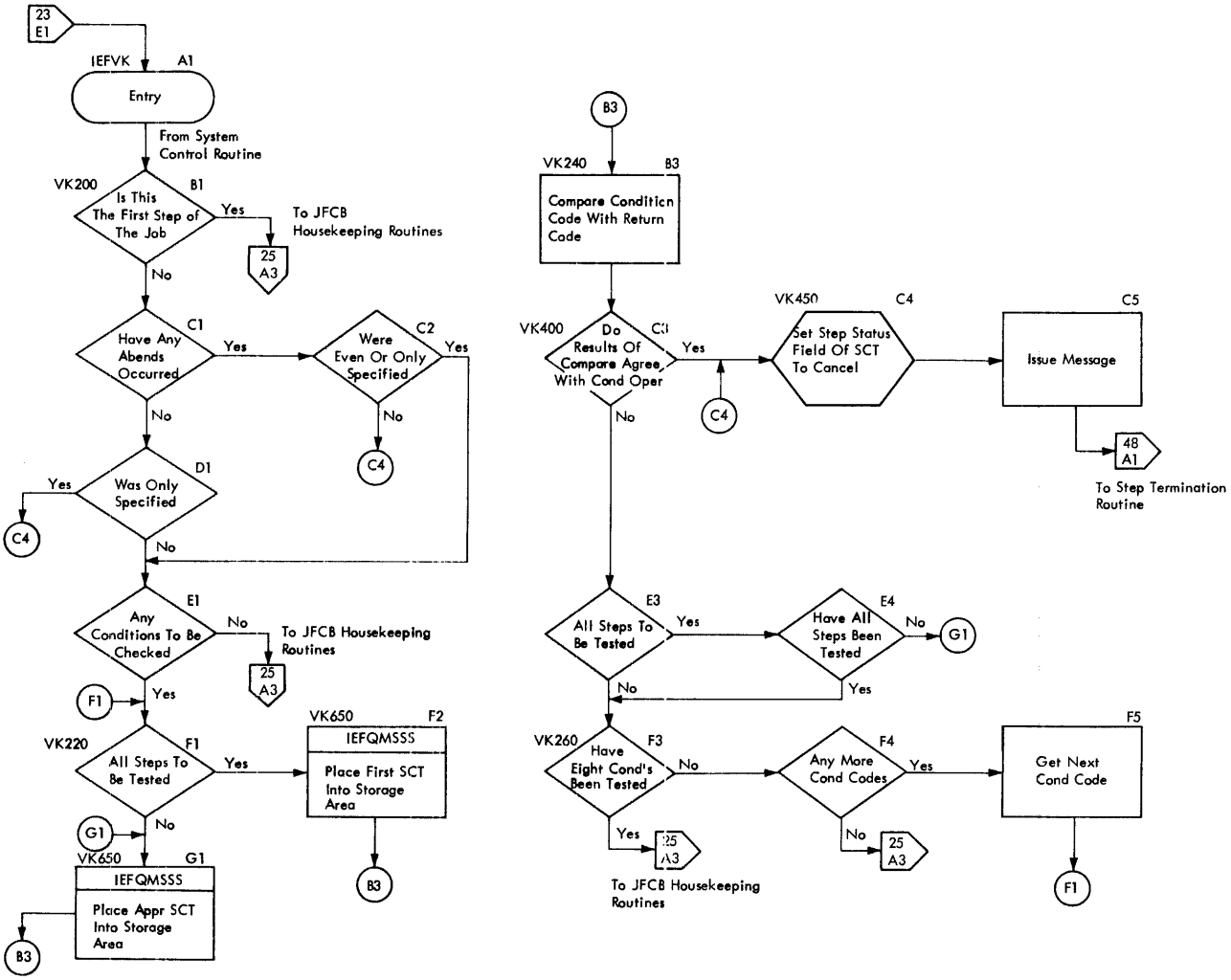


Chart 25. JFCB Housekeeping Routines

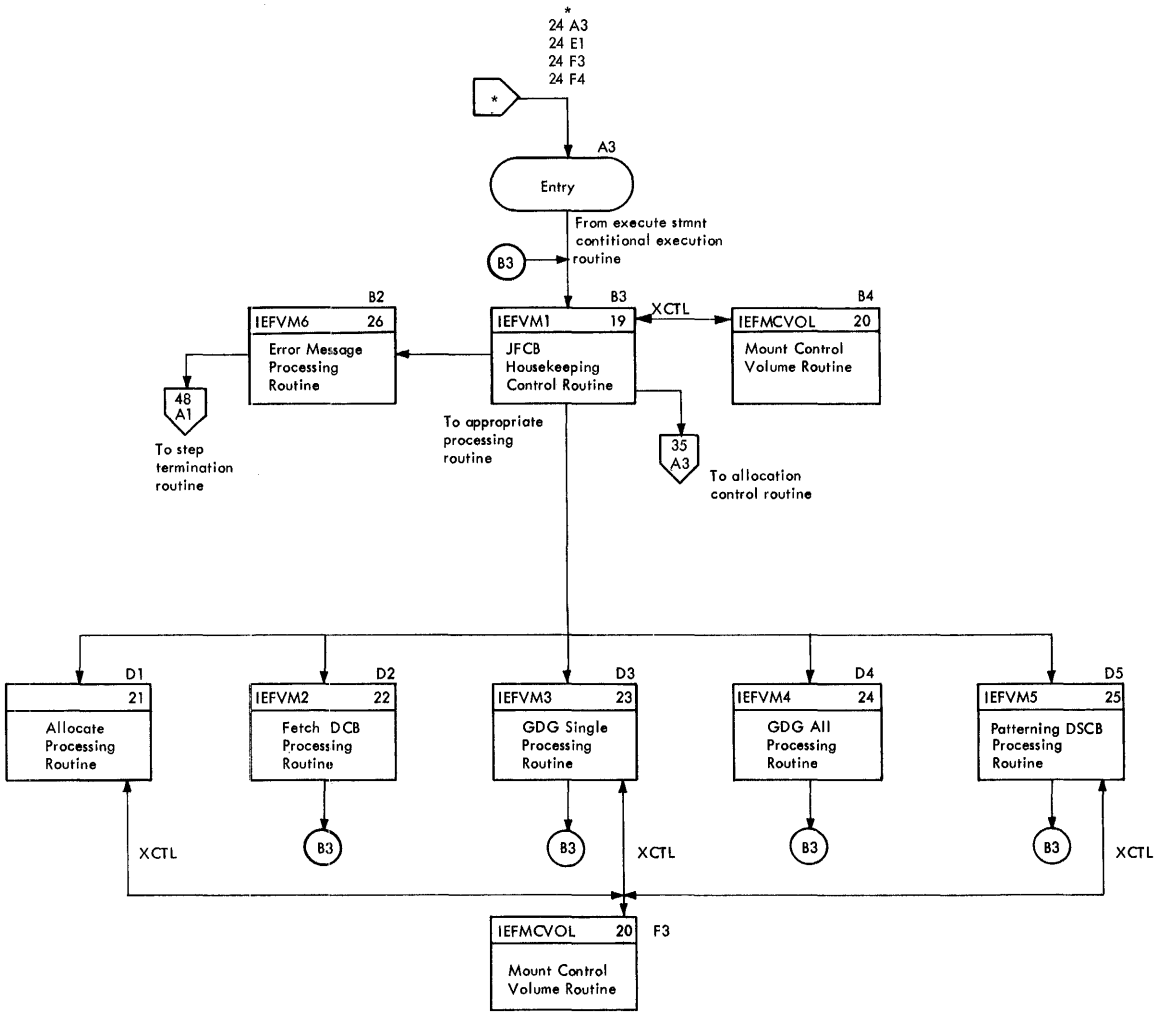


Chart 26. JFCB Housekeeping Control Routine

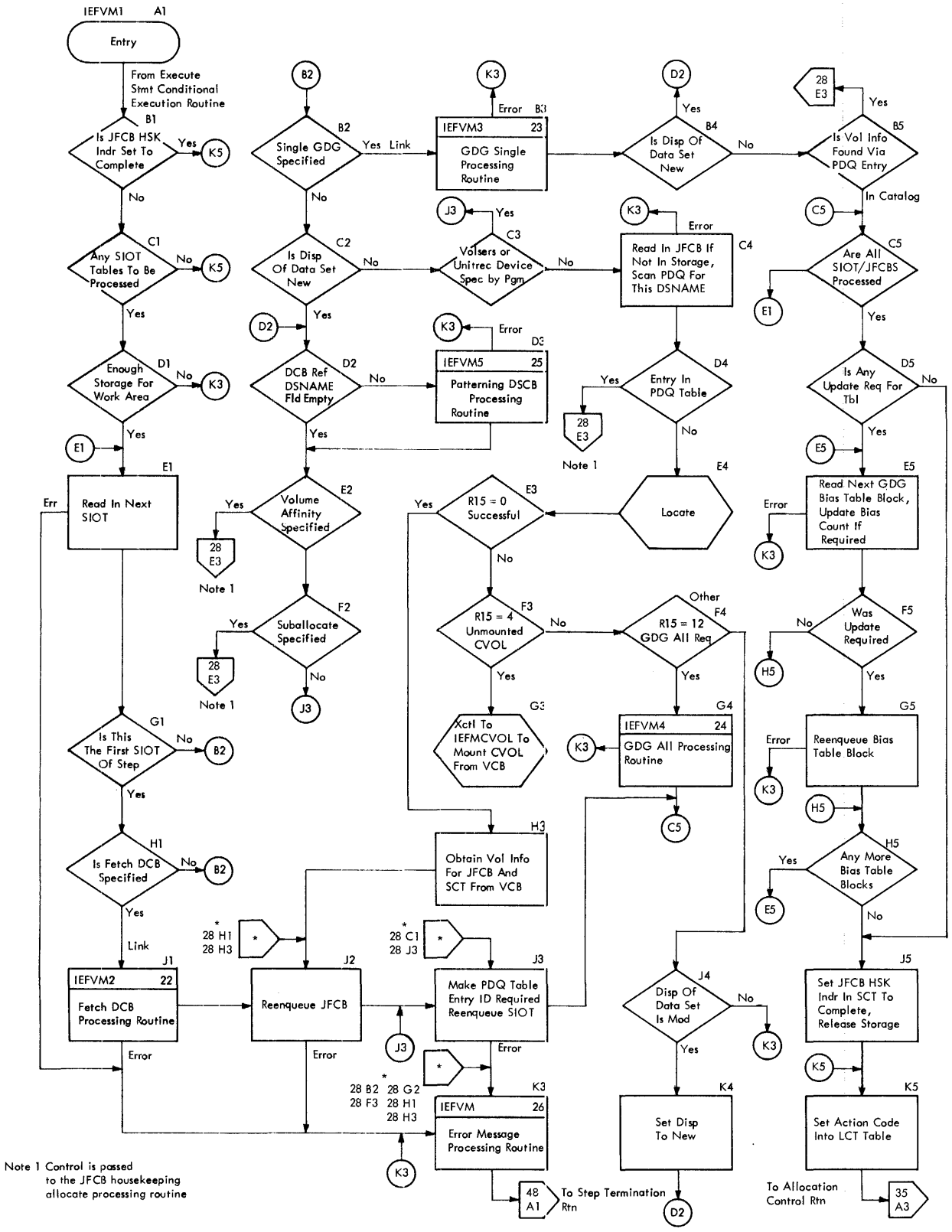


Chart 27. Mount Control Volume Routine

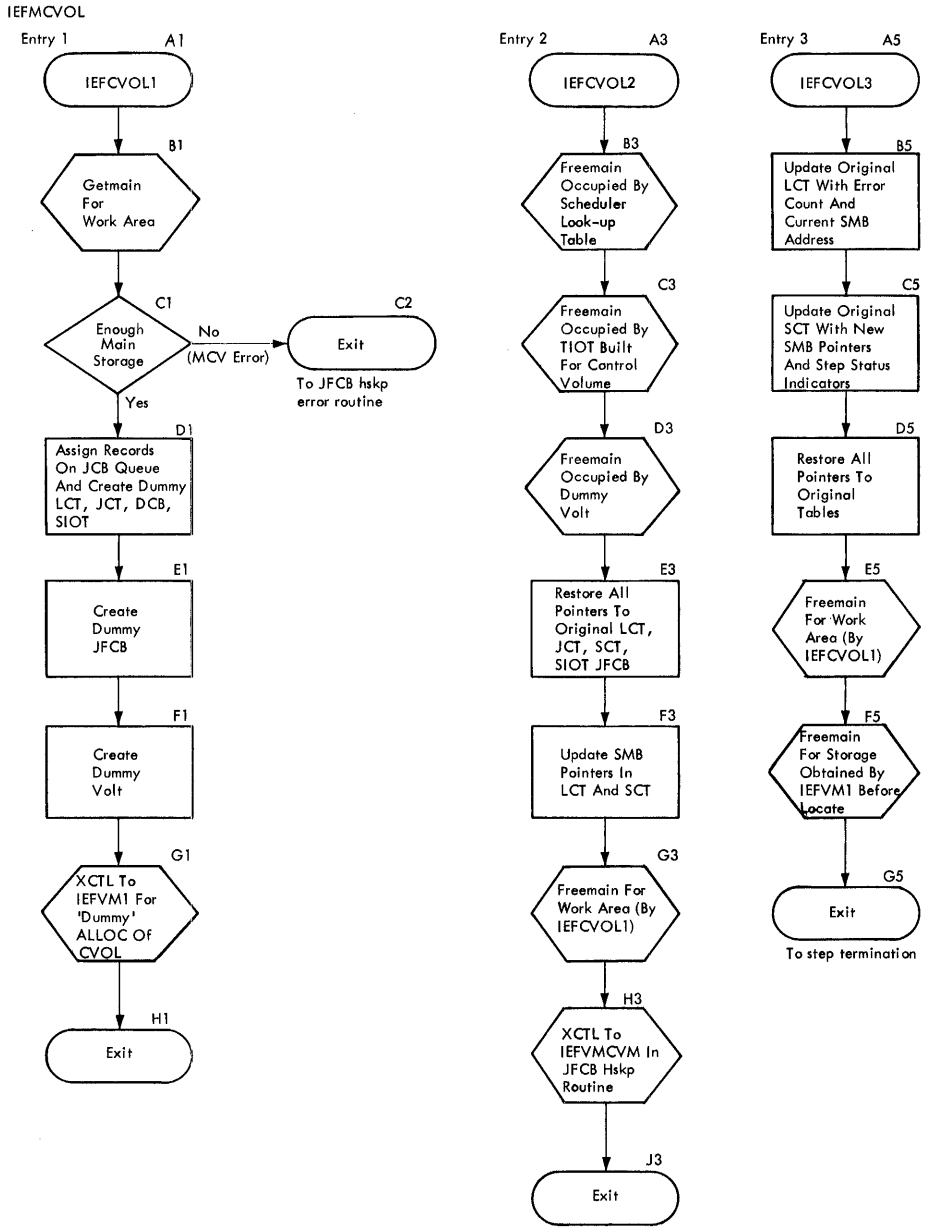
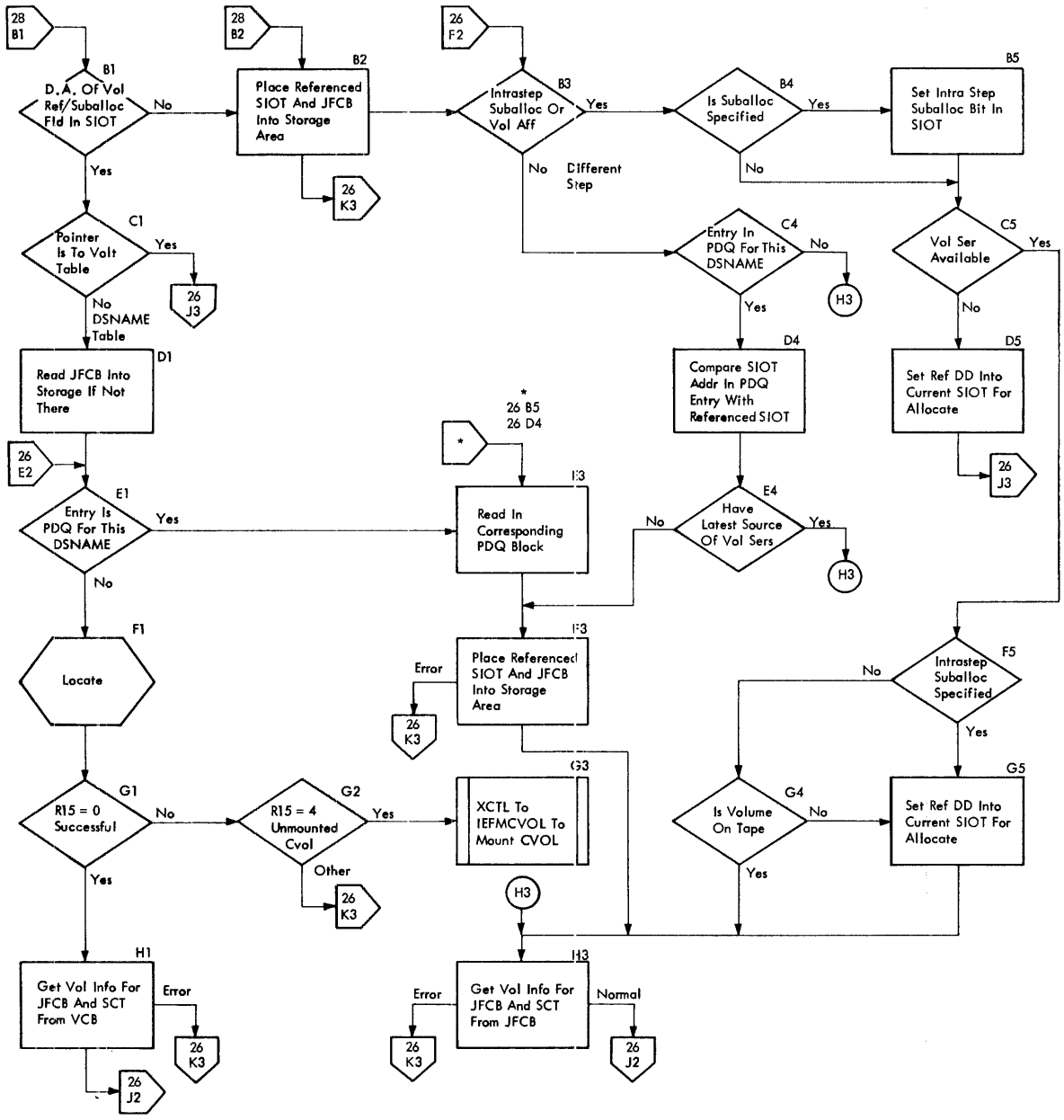


Chart 28. Allocate Processing Routine



All Entries/exits are from/to the JFCB housekeeping control routine

Chart 29. Fetch DCB Processing Routine

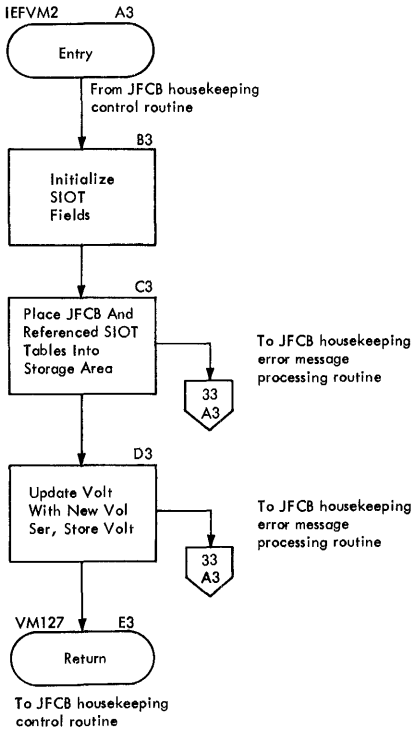


Chart 30. GDG Single Processing Routine

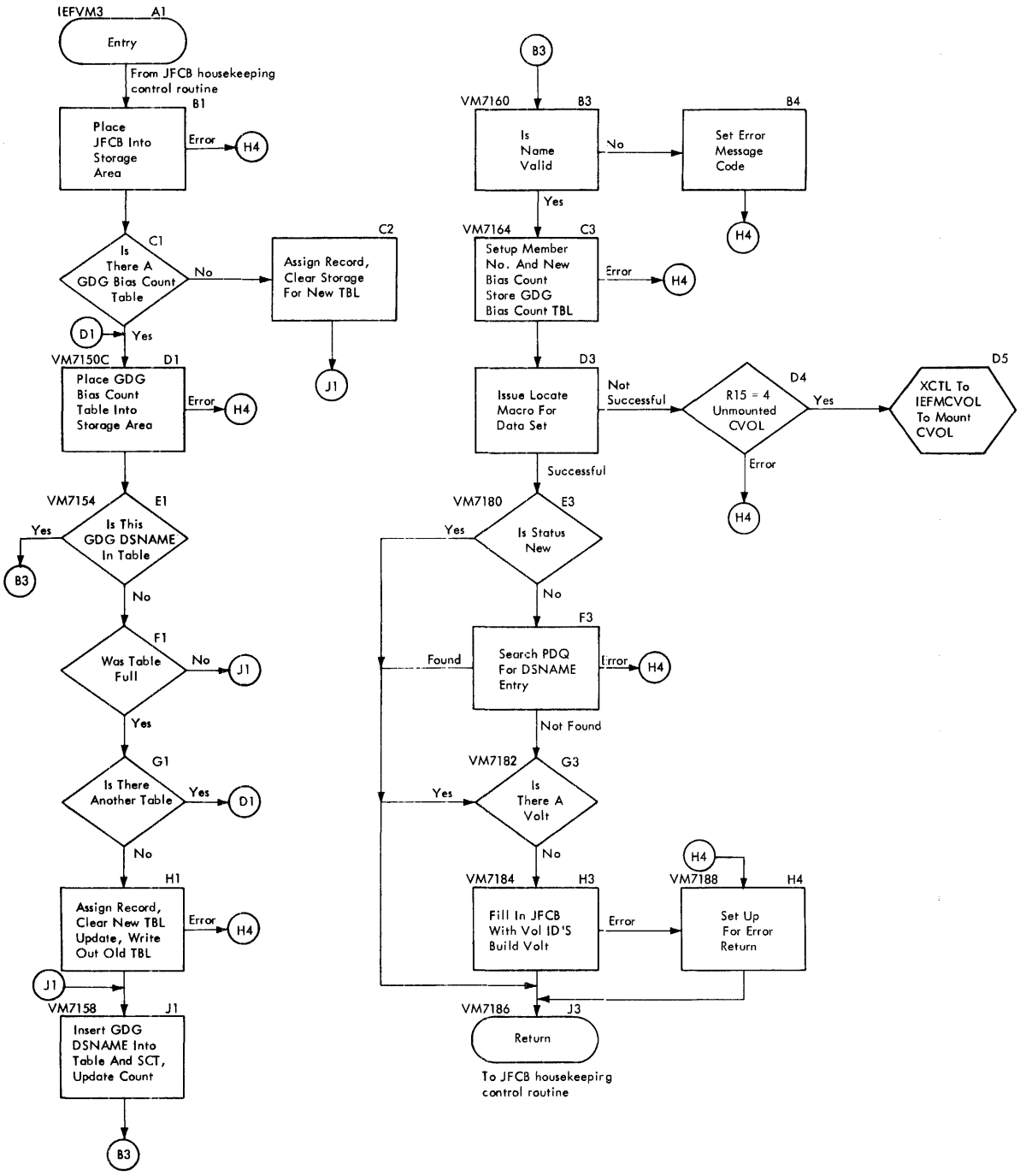


Chart 31. GDG All Processing Routine

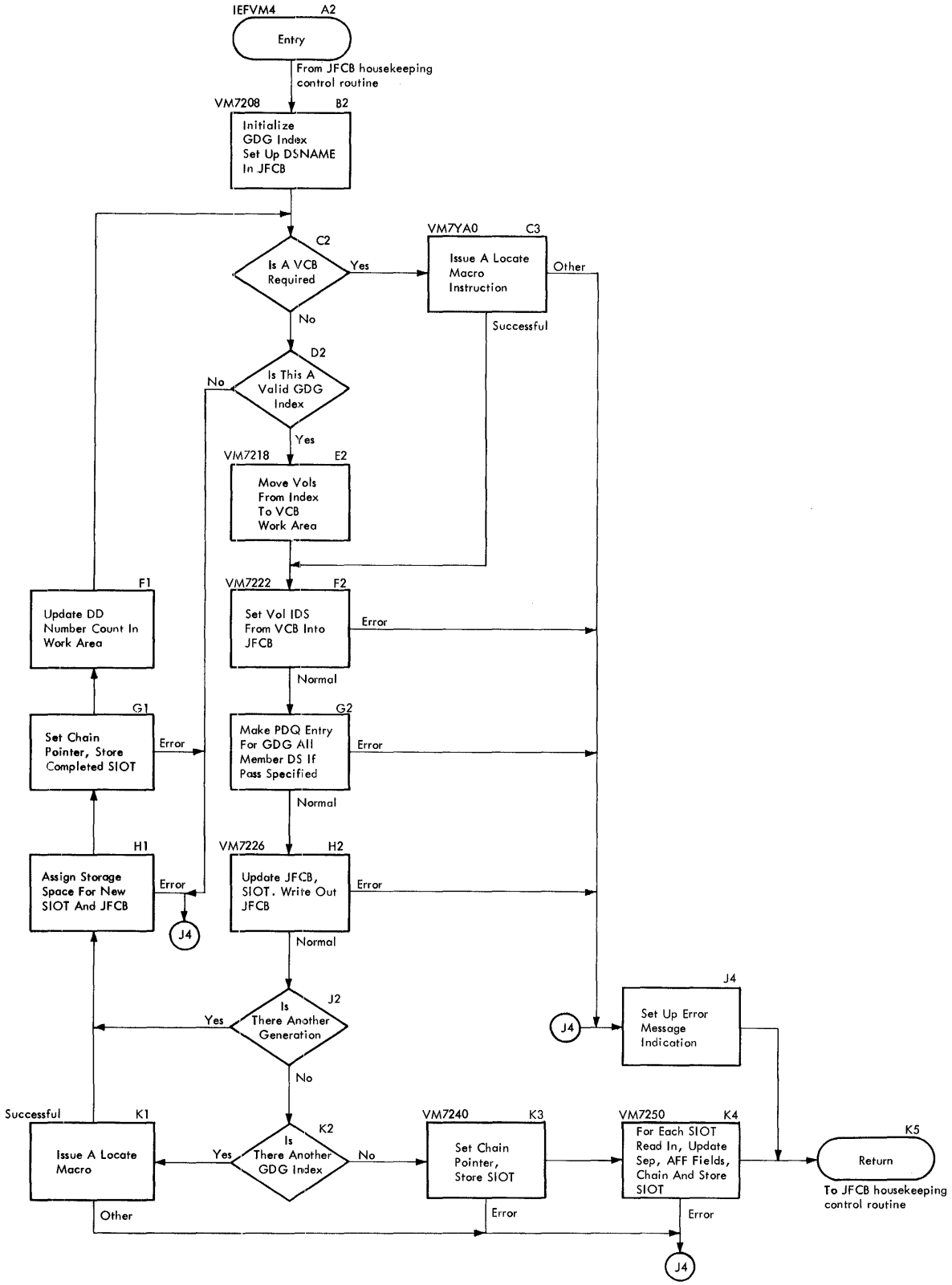


Chart 32. Patterning DSCB Processing Routine

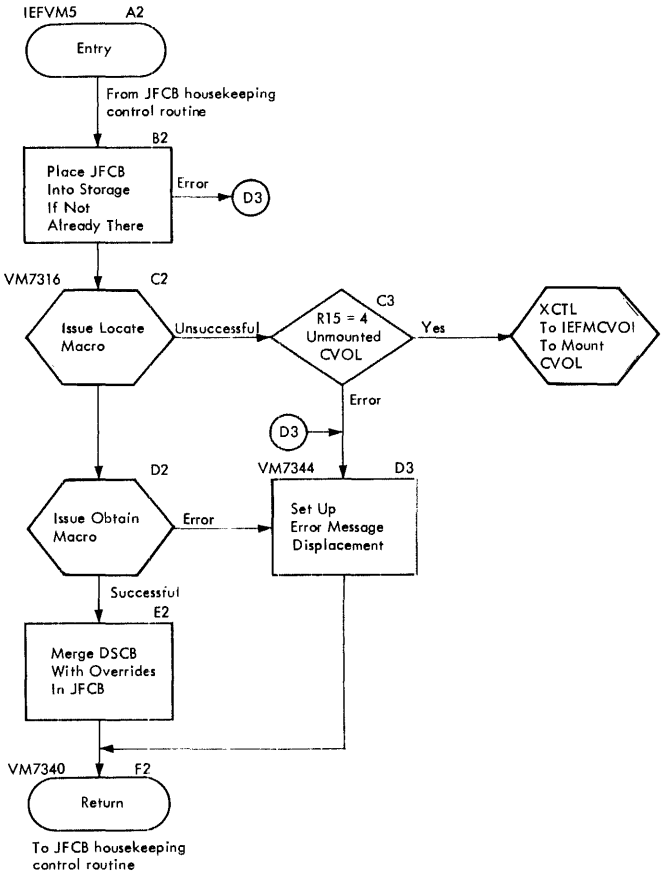


Chart 33. Error Message Processing Routine

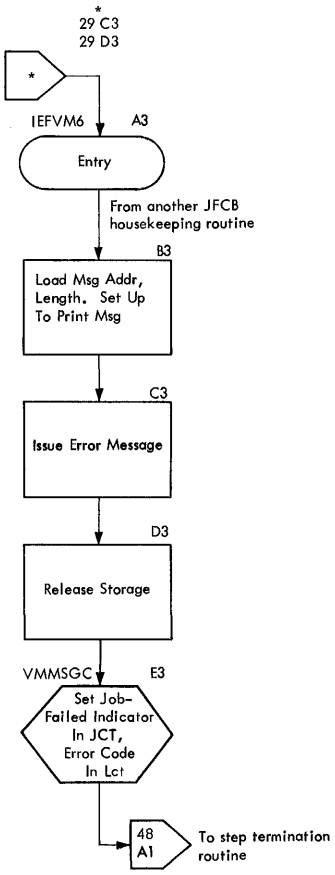


Chart 34. Allocation and Setup

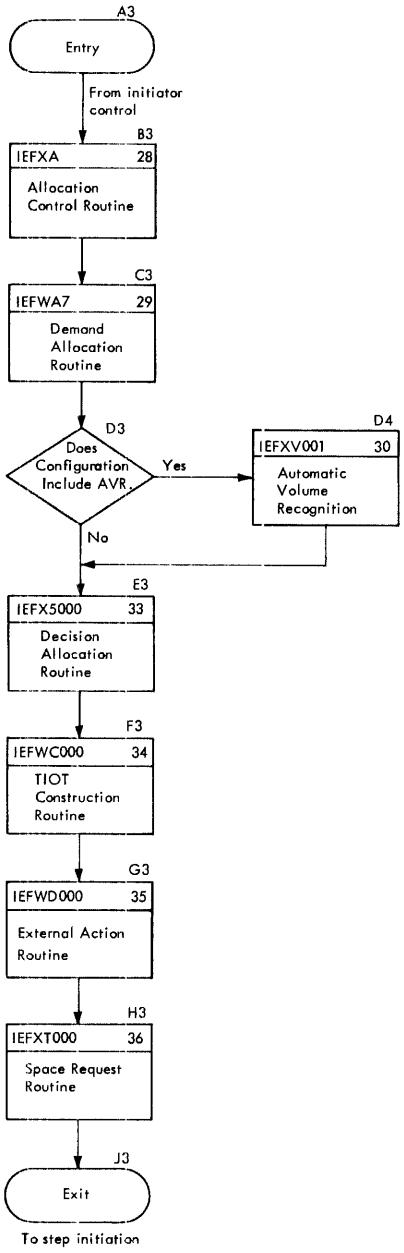


Chart 35. Allocation Control Routine

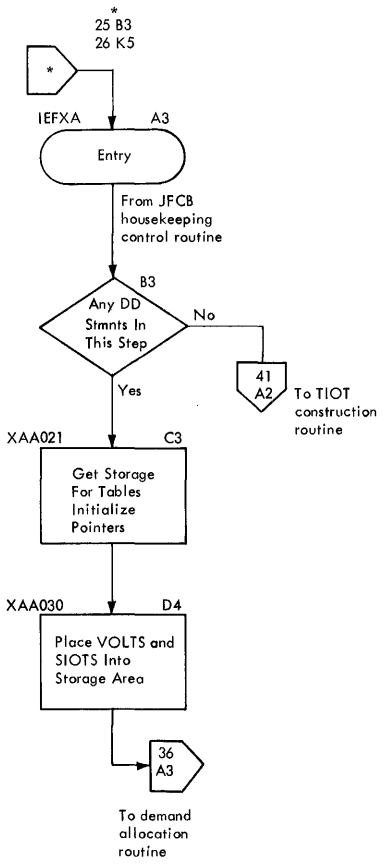
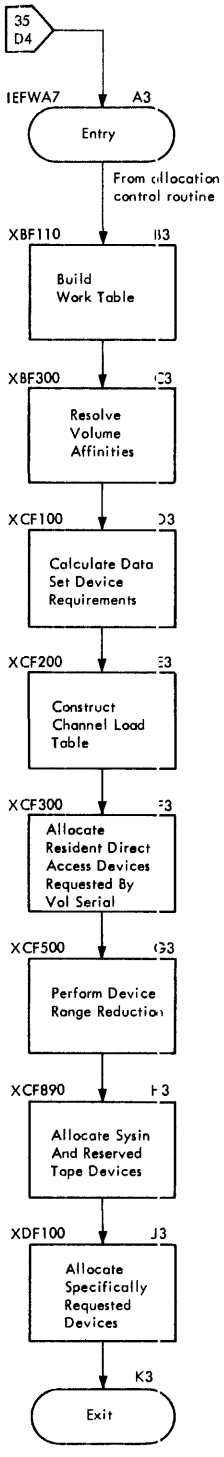


Chart 36. Demand Allocation Routine



Exits are to the decision allocation routine (or AVR when included in configuration) if allocation is incomplete.

The TIOT construction routine if allocation is complete.

Chart 37. Automatic Volume Recognition (IEFXV001)

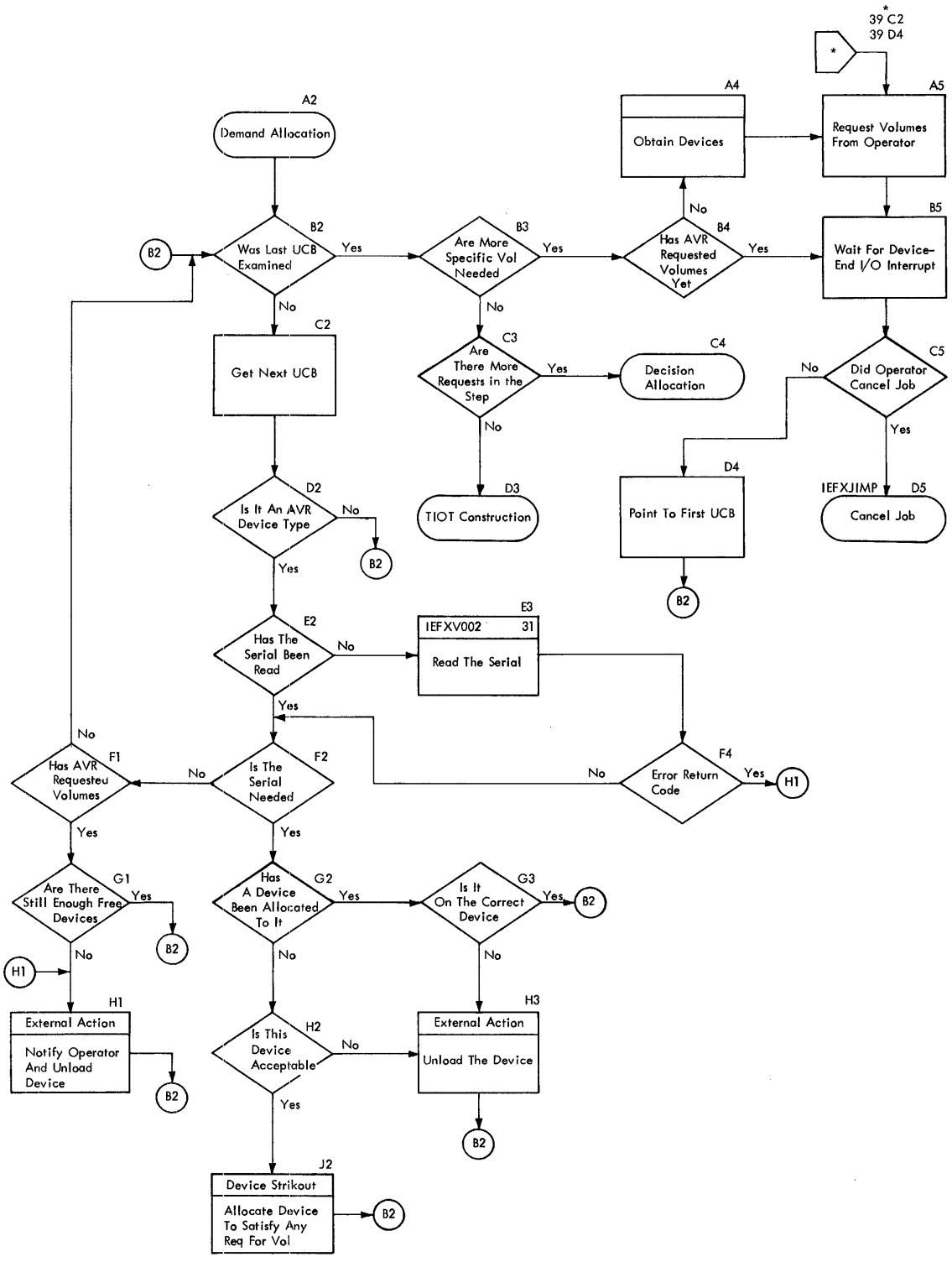


Chart 38. Automatic Volume Recognition (IEFXV002)

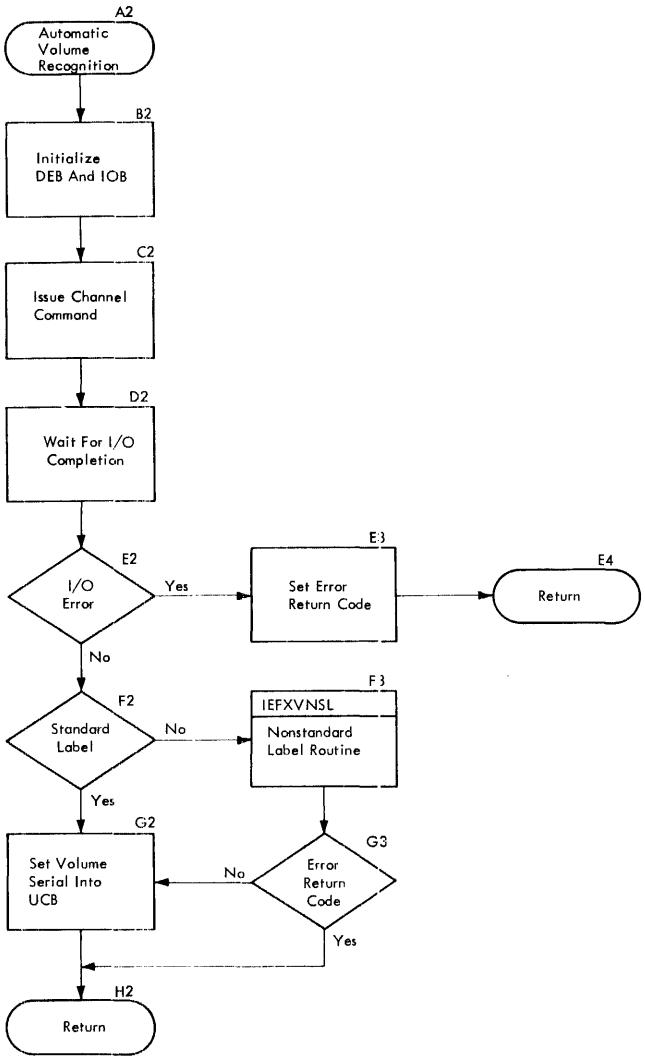


Chart 39. Obtain Devices

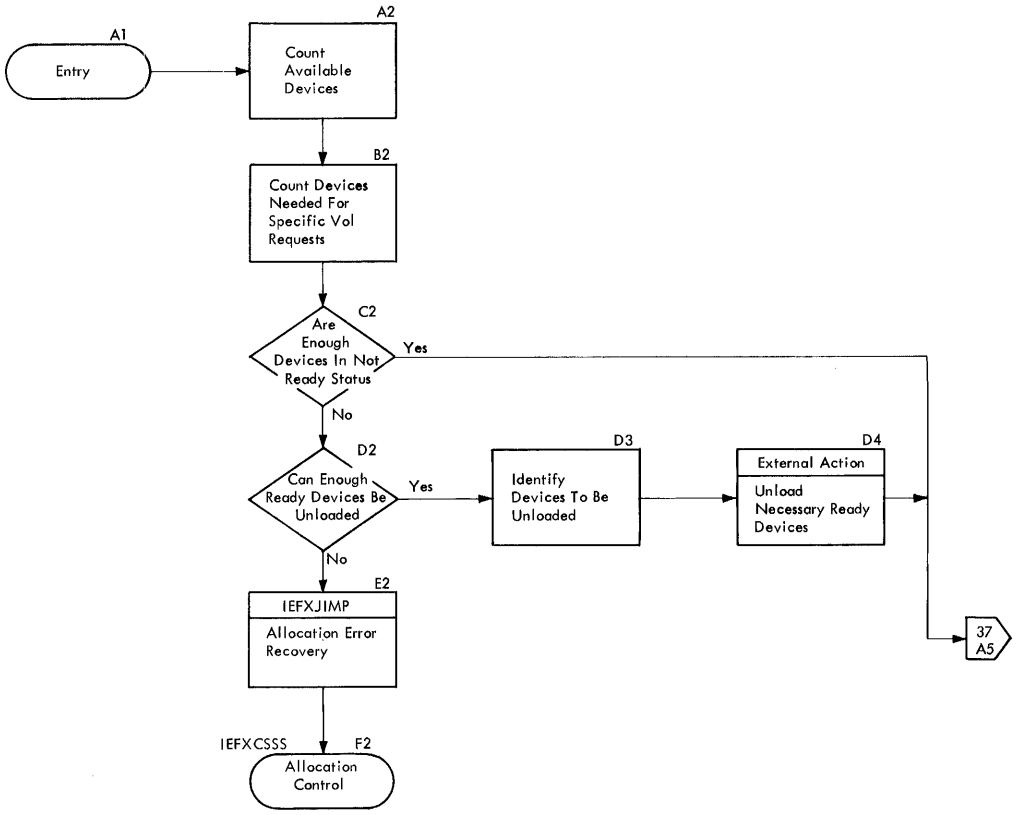


Chart 40. Decision Allocation Routine

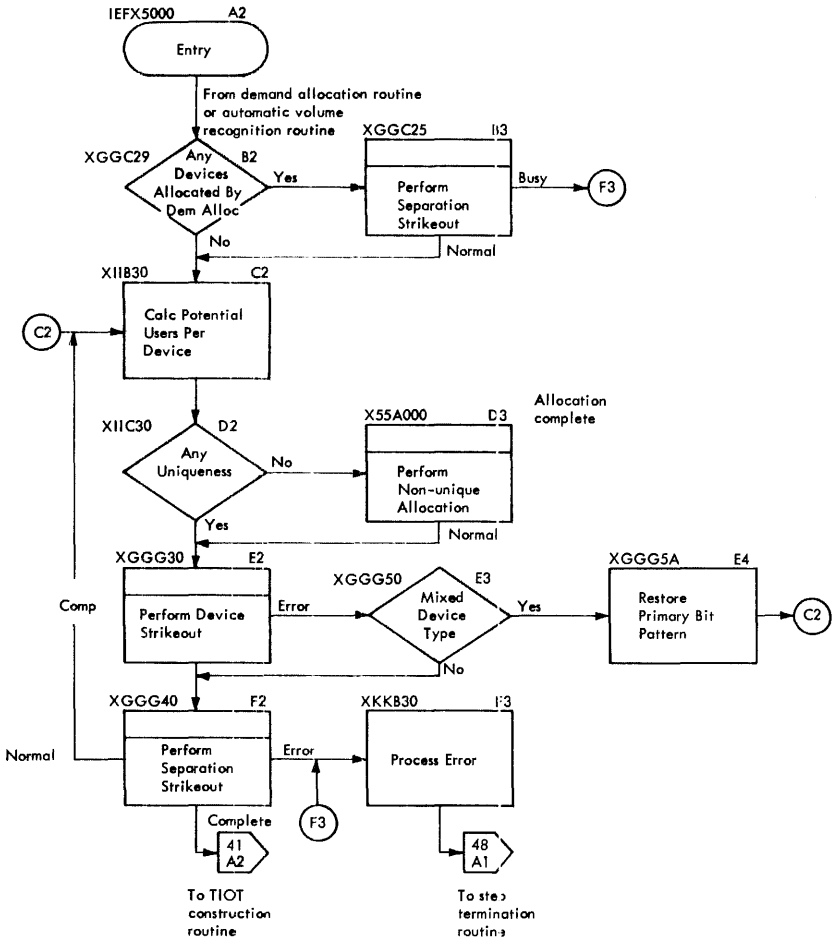


Chart 41. TIOT Construction Routine

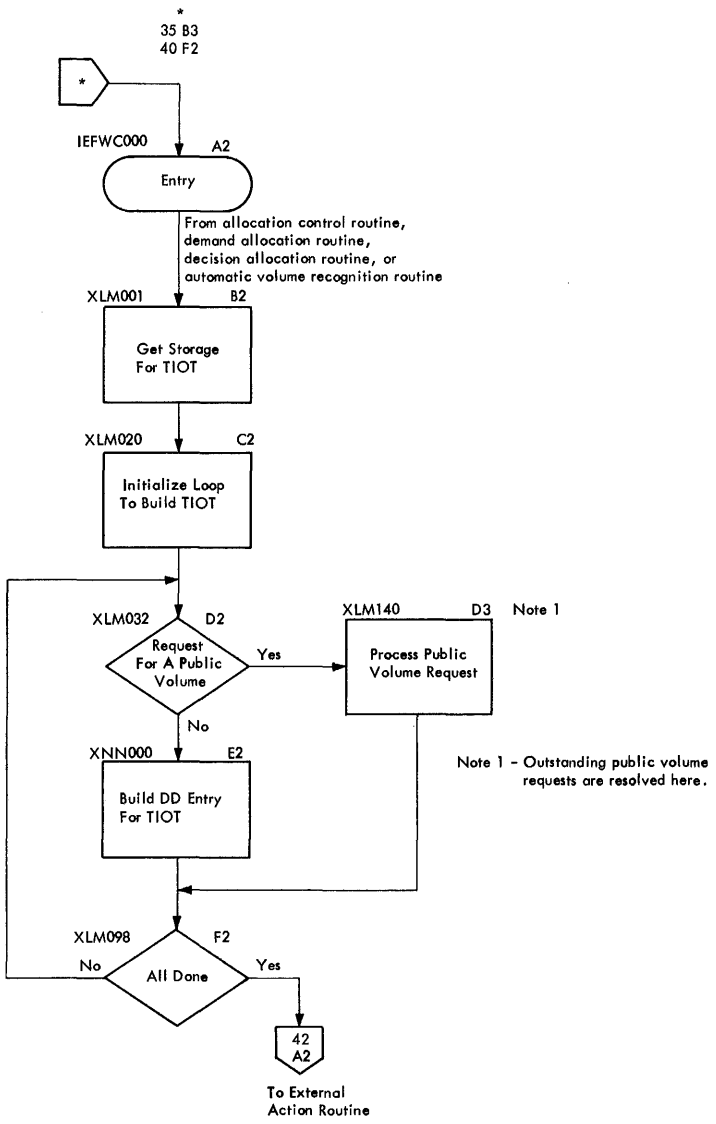
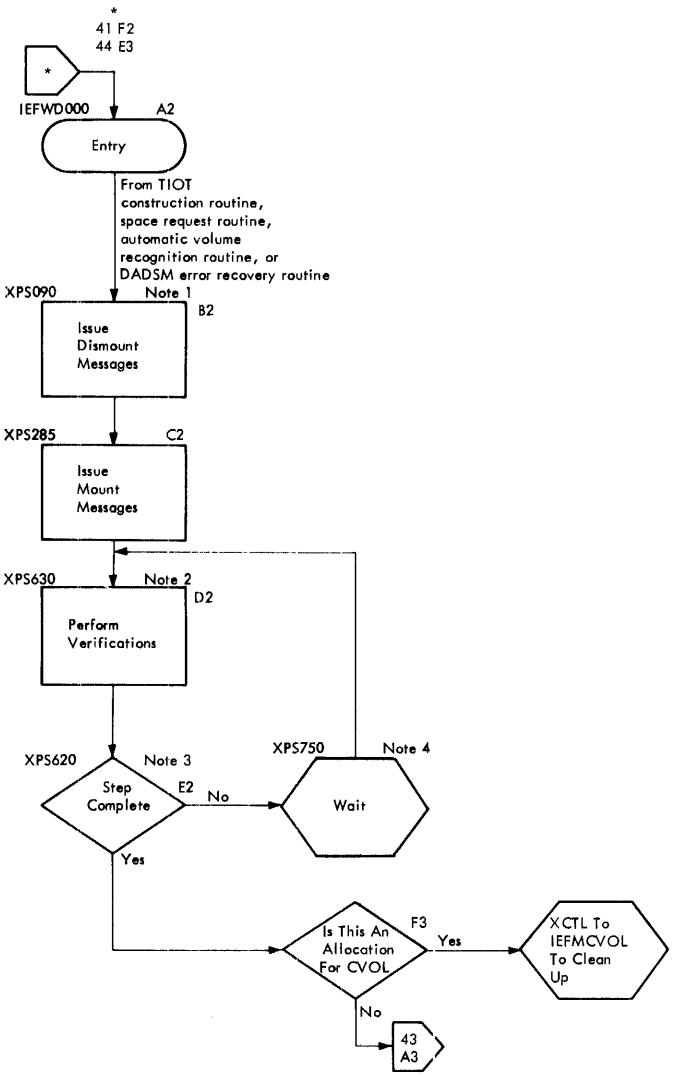


Chart 42. External Action Routine



Note 1 Includes a scan to determine if the required volumes are mounted on unallocated devices.

Note 2 Unload commands from master scheduler are honored. UCB'S are updated if required.

Note 3 The step is complete when all setup messages have been issued and verification has been performed. Counts in the SCT control this mechanism.

Note 4 Either of two events is waited upon. Issuance of a cancel command or a device being made ready.

Chart 43. Space Request Routine

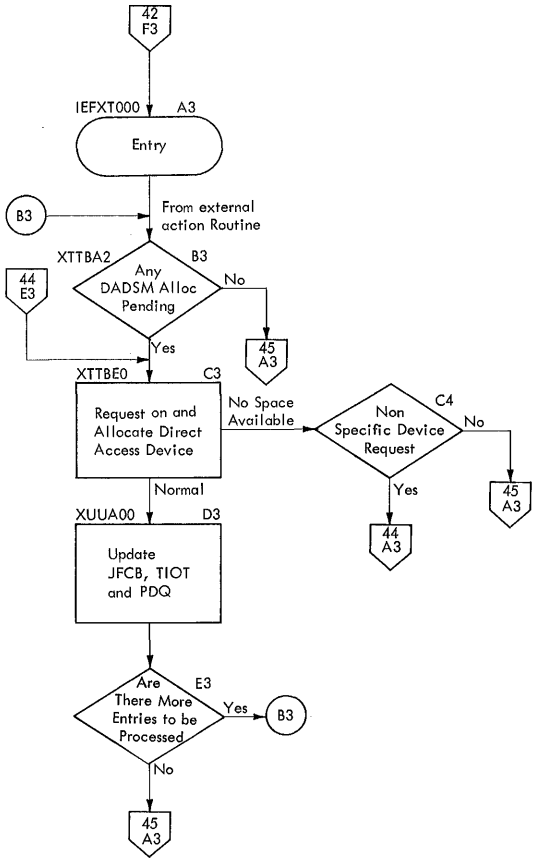


Chart 44. DADSM Error Recovery Routine

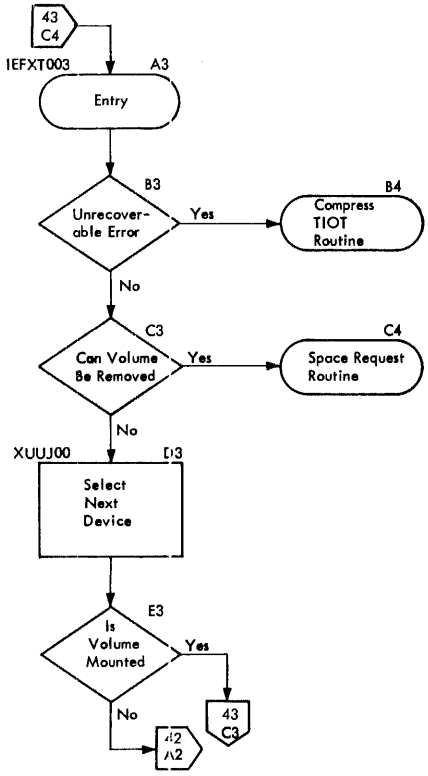


Chart 45. TIOT Compression Routine

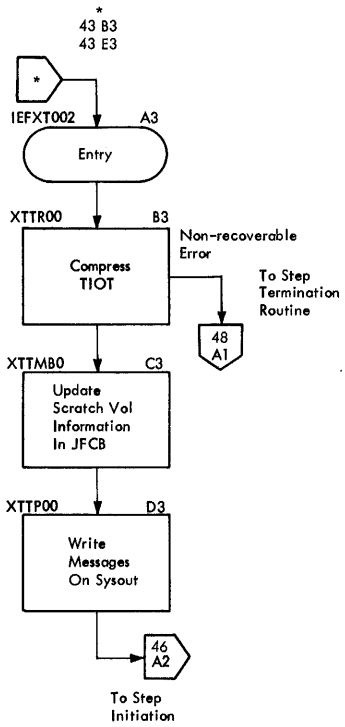
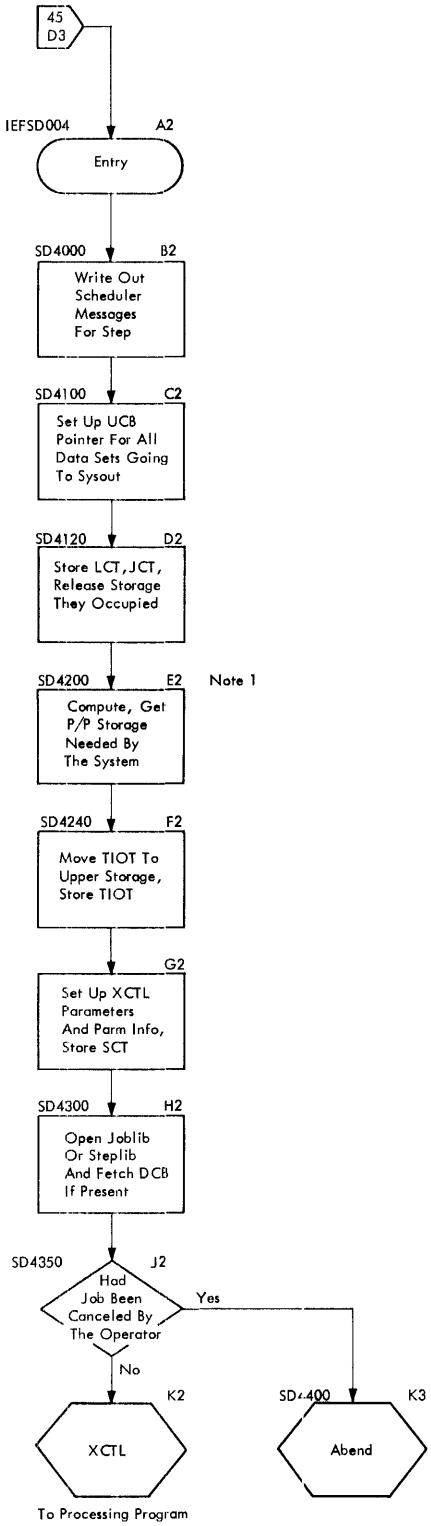
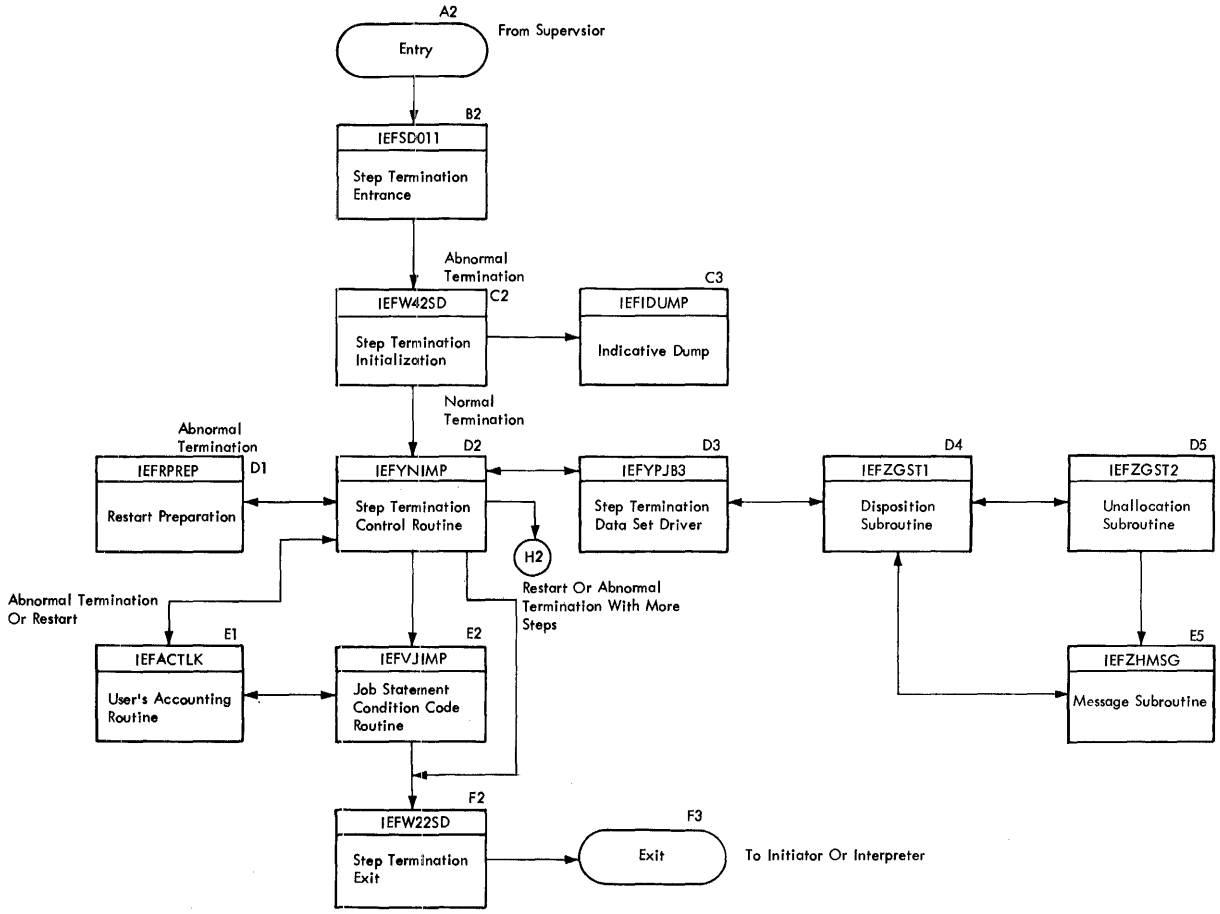


Chart 46. Step Initiation



Note 1 - During step execution, processing program (P/P) storage is needed for
- A steplib or joblib DCB, if present
- A fetch DCB, if present
- An XCTL parameter list
- Parm field information
- A step TIOT
- A P/P register save area

Chart 47. Termination



Step Termination

Job Termination

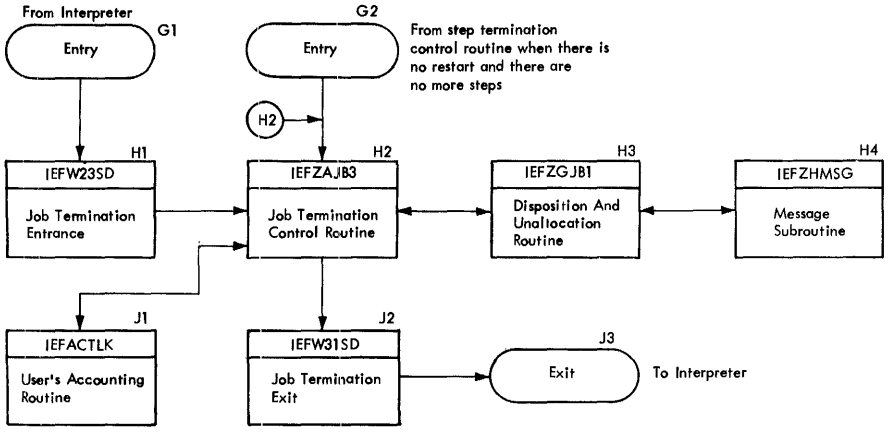


Chart 48. Step Termination Routine

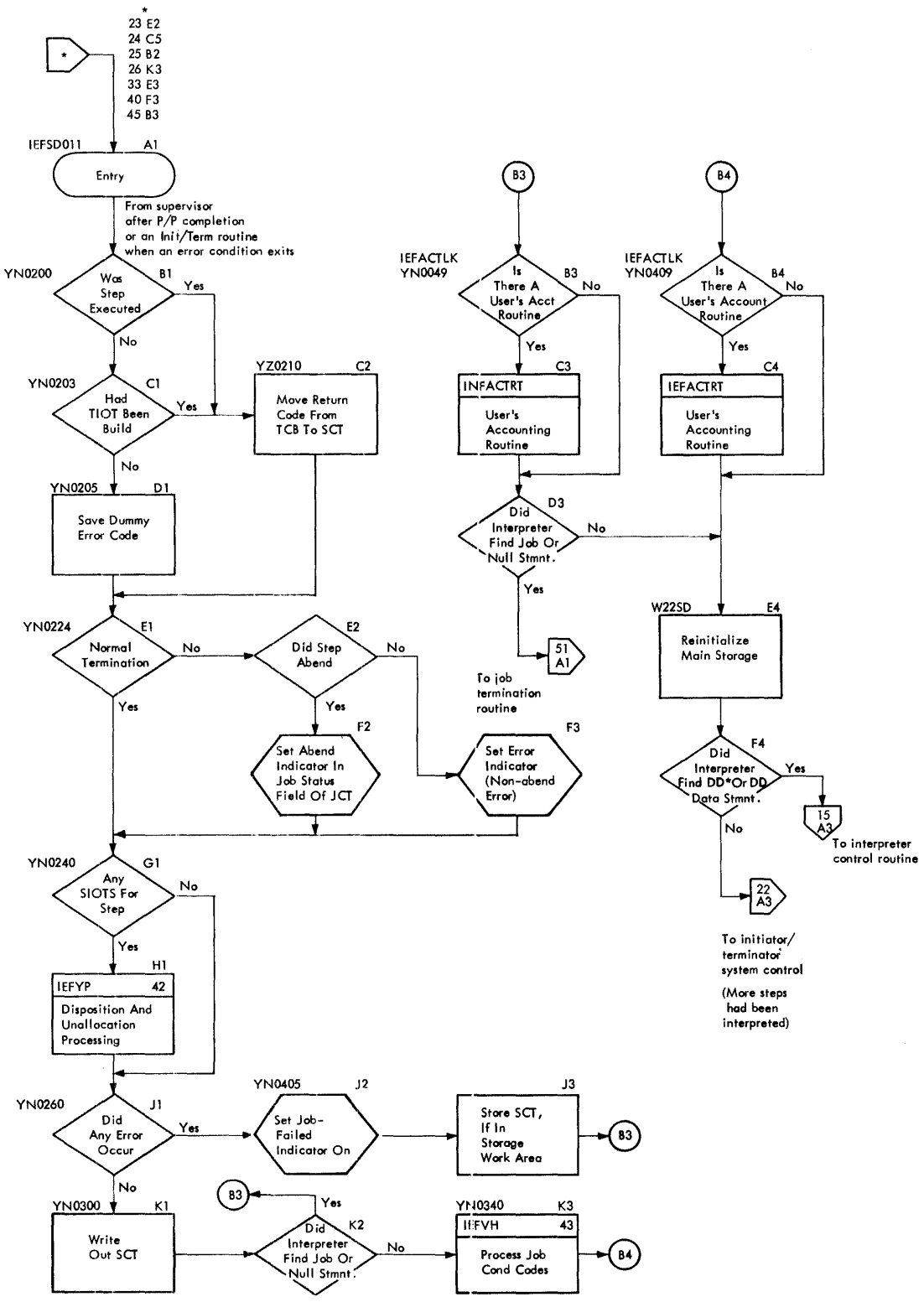


Chart 49. Restart Preparation Routine

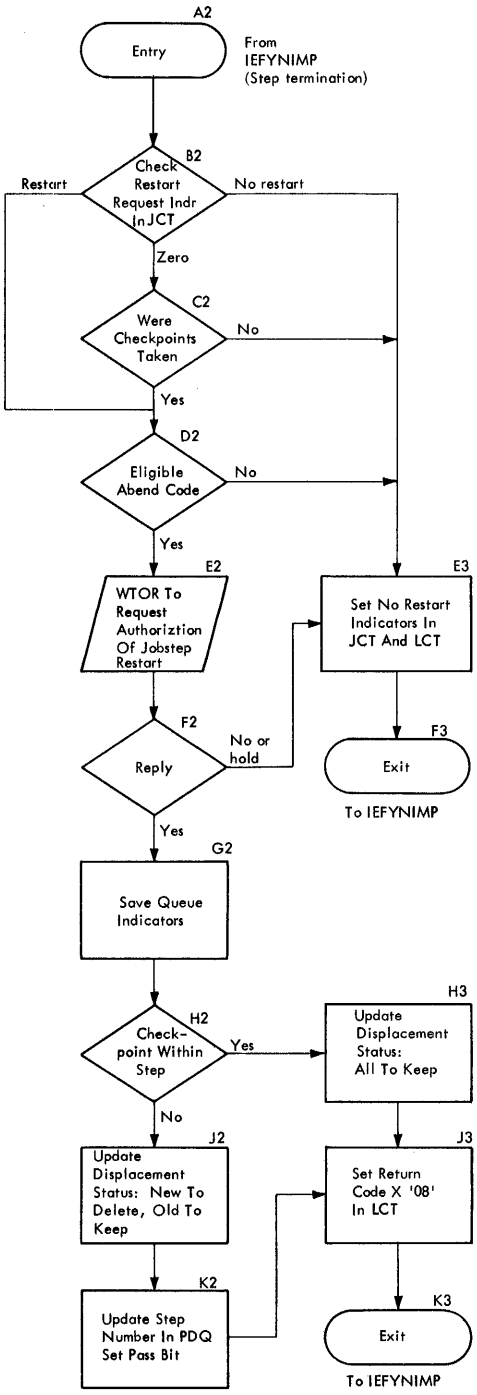


Chart 50. Job Statement Condition Code Routine

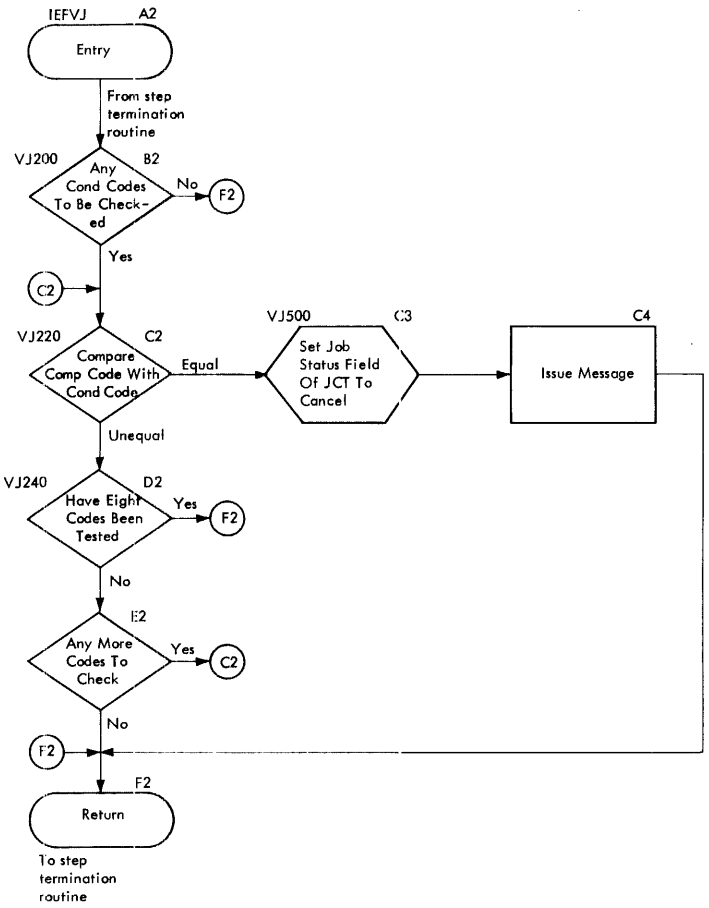


Chart 51. Job Termination Routine

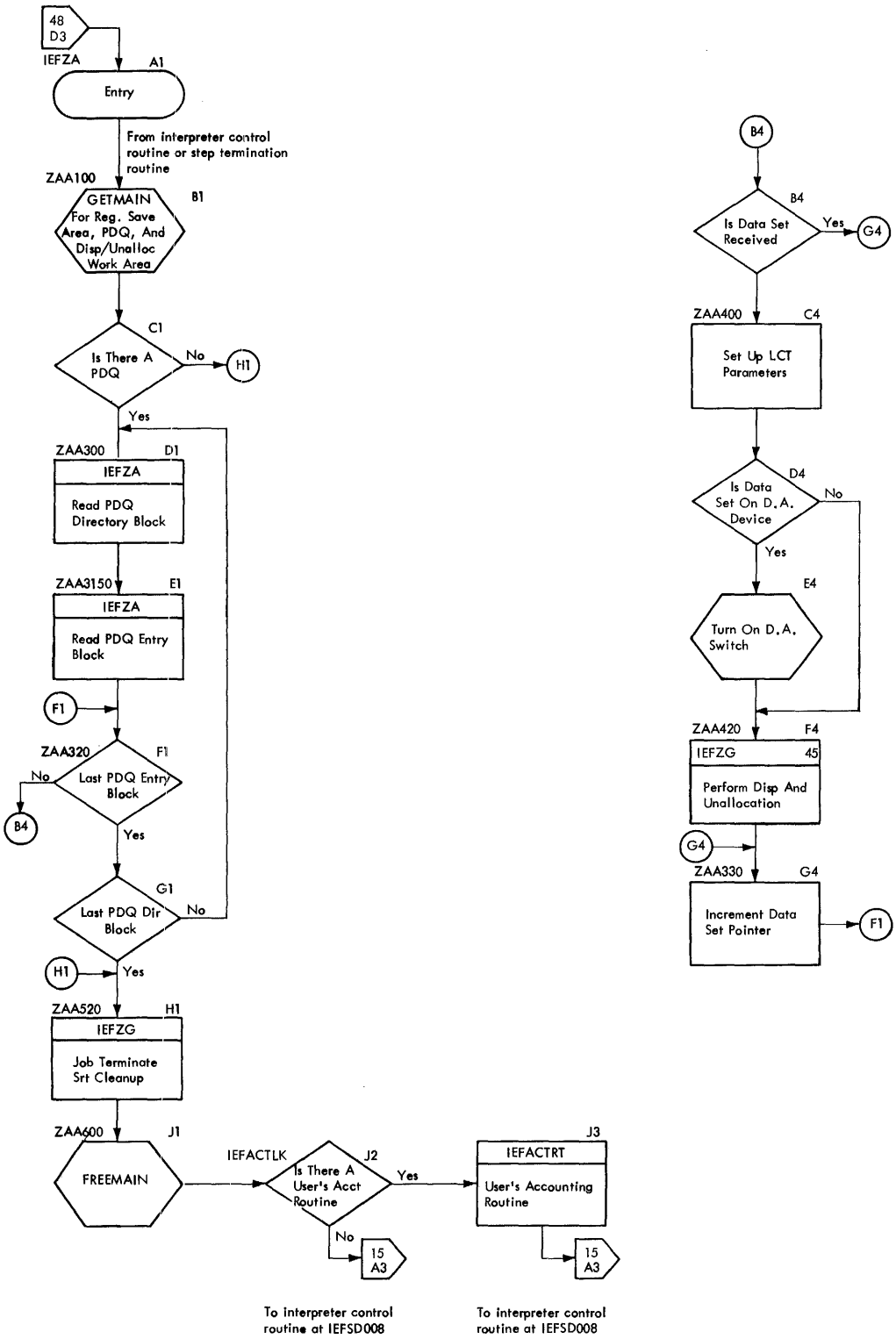


Chart 52. Disposition and Unallocation Subroutine -- Entry From Step Termination Routine

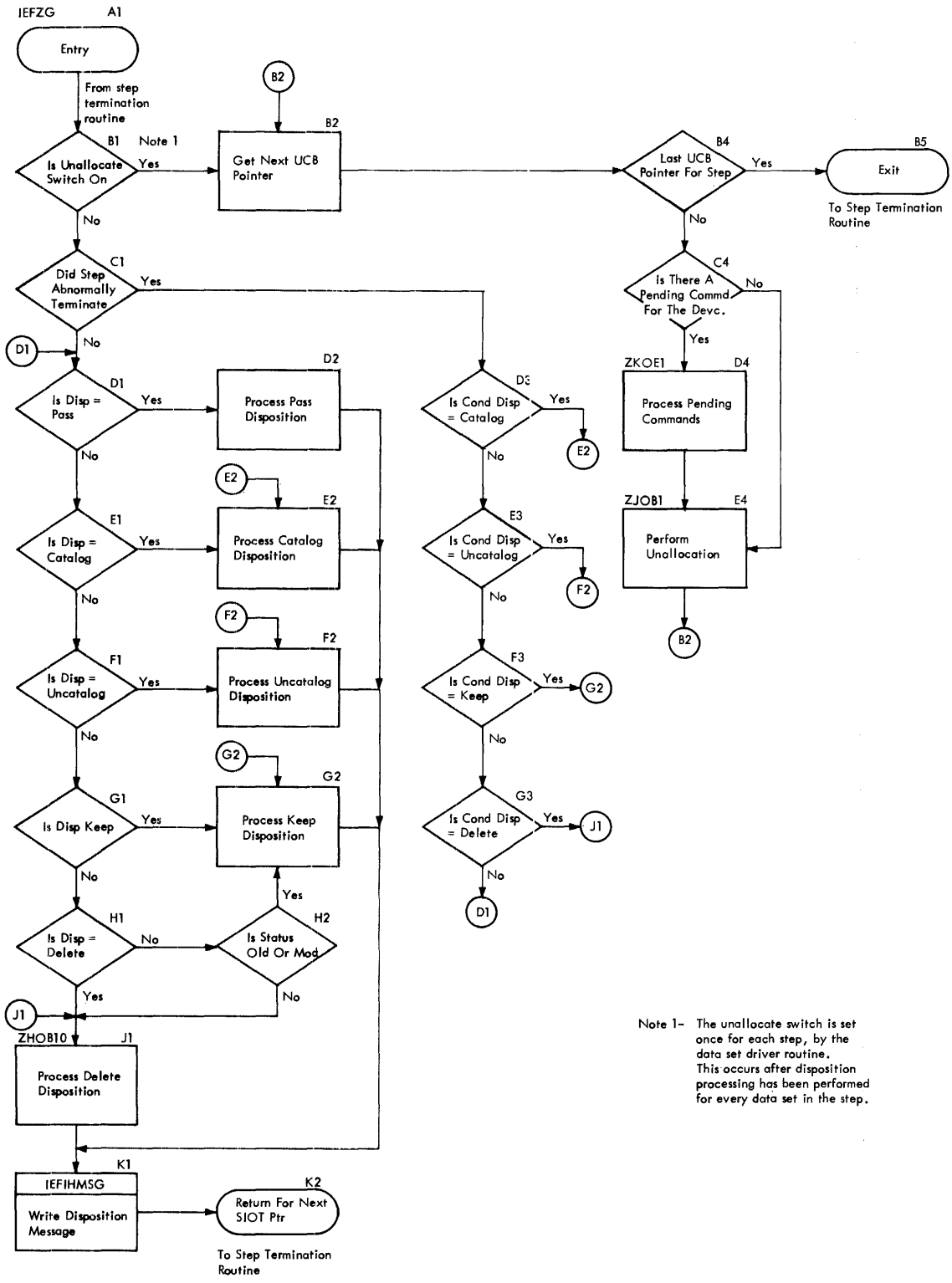
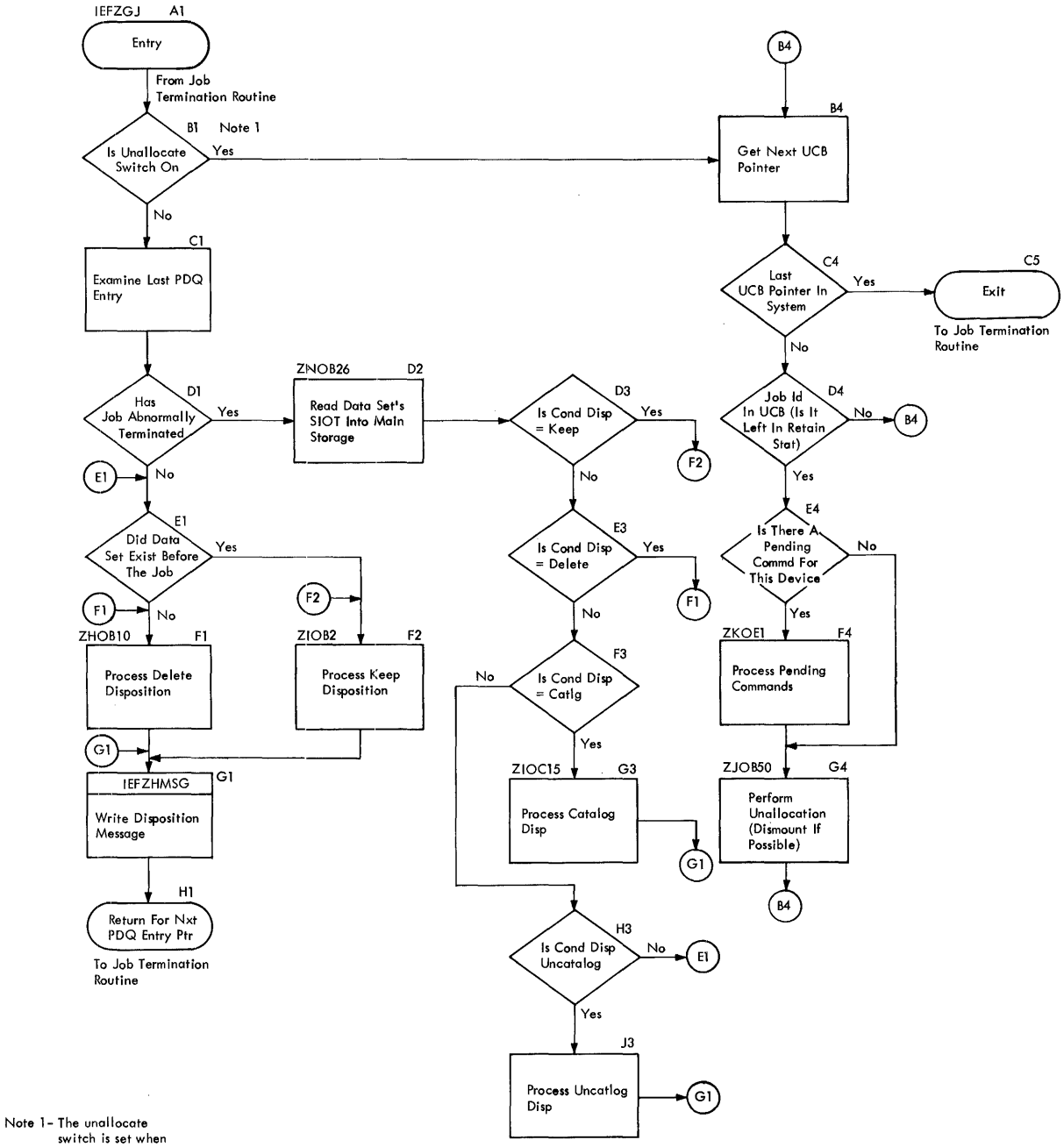
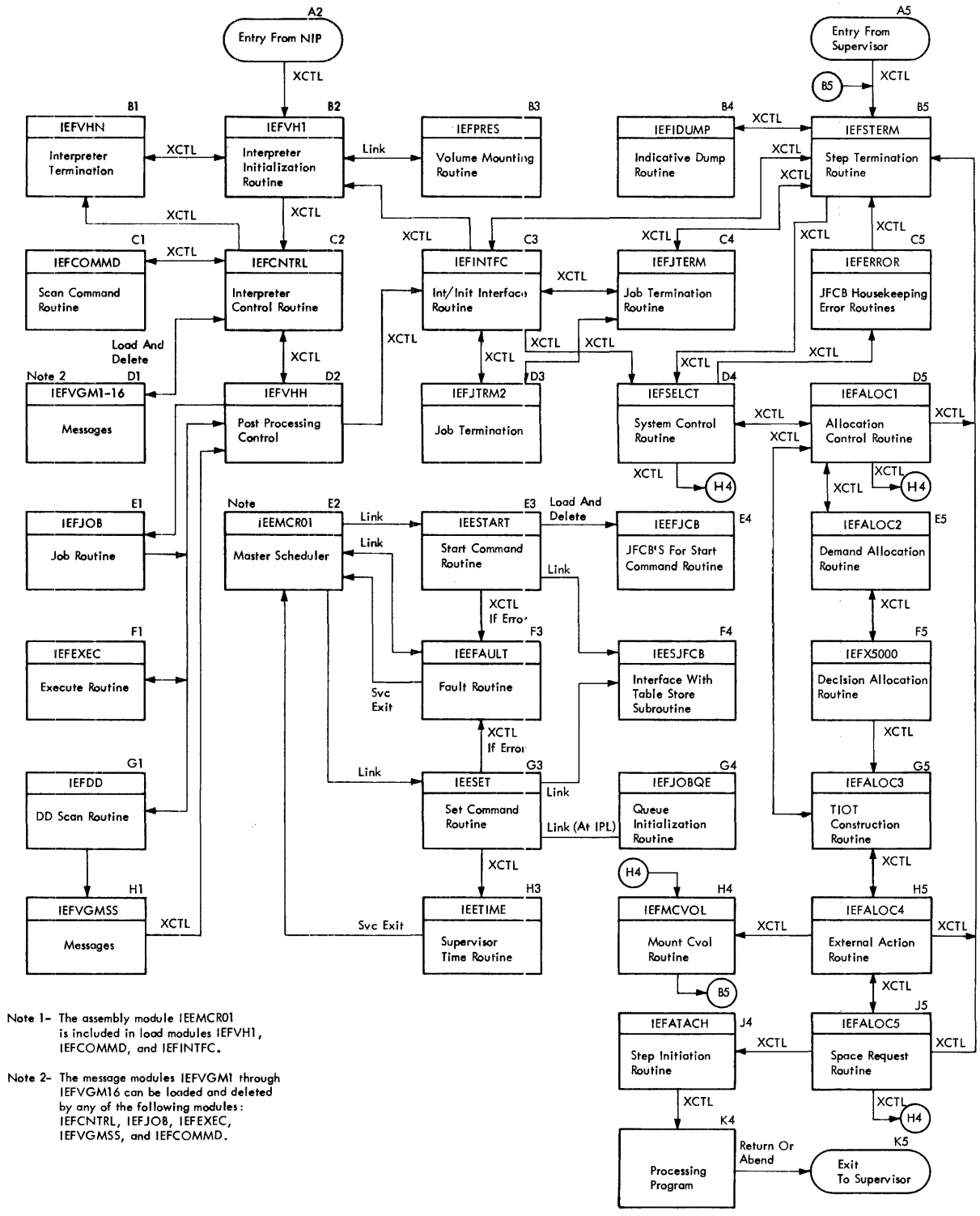


Chart 53. Disposition and Unallocation Subroutine -- Entry From Job Termination Routine



Note 1- The unallocate switch is set when all PDQ entries have been examined.

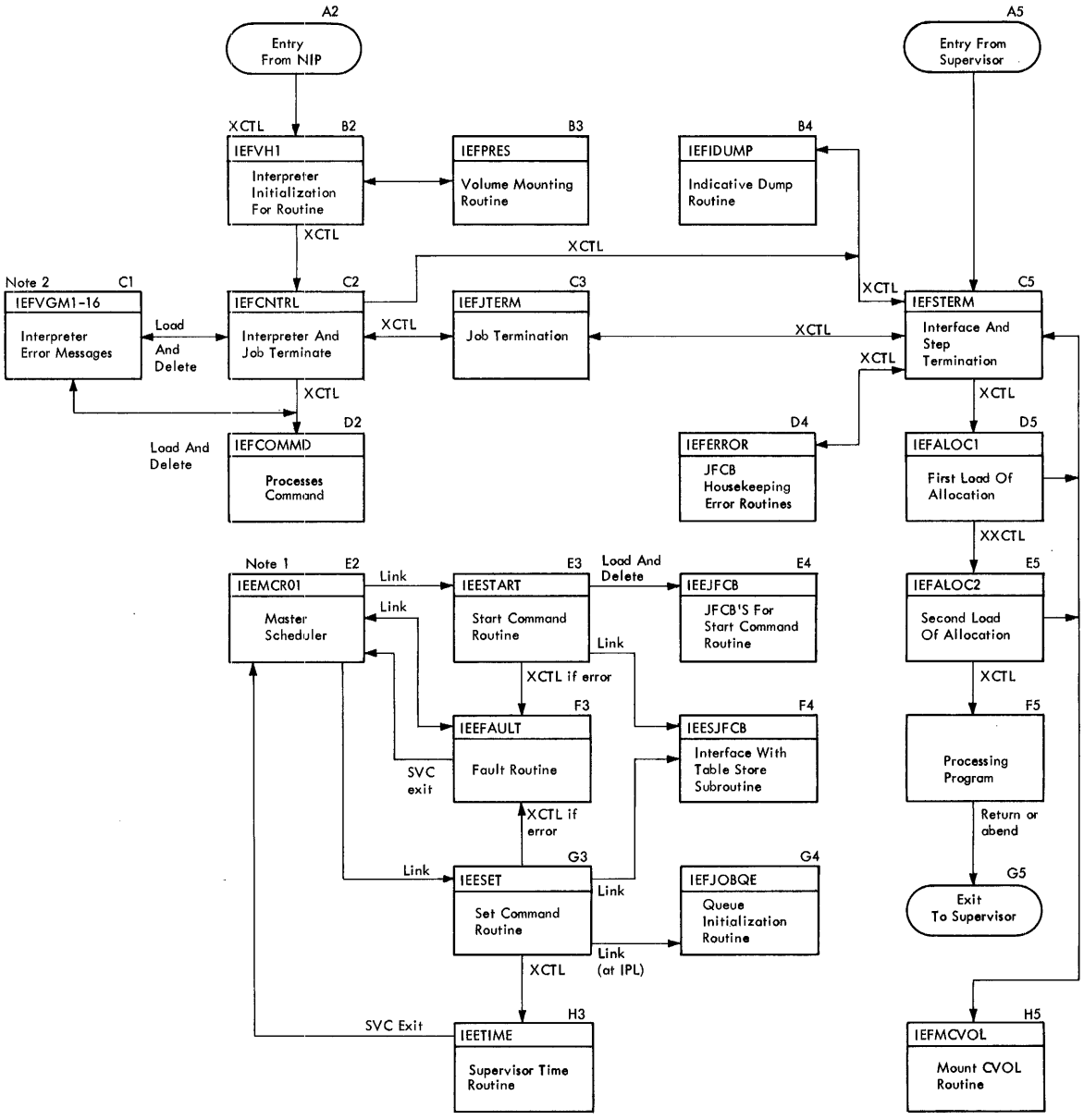
Chart 54. 18K Configuration Load Module Control Flow



Note 1- The assembly module IEEMCR01 is included in load modules IEFVH1, IEFCOMMD, and IEFINTFC.

Note 2- The message modules IEFVGM1 through IEFVGM16 can be loaded and deleted by any of the following modules: IEFCNTRL, IEFJOB, IEFEXEC, IEFVGMSS, and IEFCOMMD.

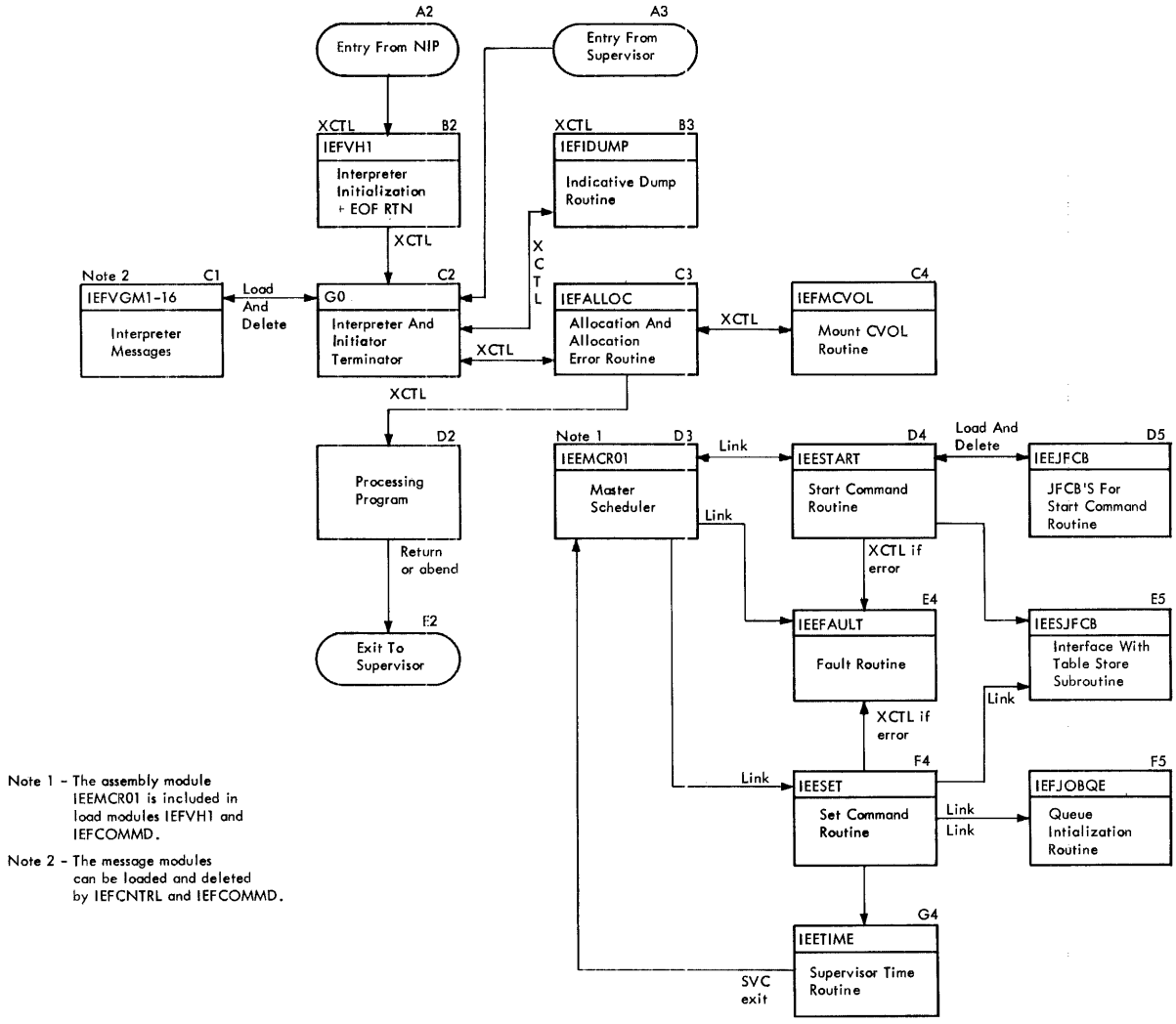
Chart 55. 44K Configuration Load Module Control Flow



Note 1 - The assembly module IEEMCR0 1 is included in load modules IEFVH1 and IEFCOMMD.

Note 2 - The message modules can be loaded and deleted by IEFCNTRL and IEFCOMMD.

Chart 56. 100K Configuration Load Module Control Flow



Note 1 - The assembly module IEEMCR01 is included in load modules IEFVH1 and IEFCOMMD.

Note 2 - The message modules can be loaded and deleted by IEFCNTRL and IEFCOMMD.

Index

READER'S COMMENT FORM

IBM System/360 Operating System
Job Management
Program Logic Manual

Order No. GY28-6613-5

- Is the material:

	Yes	No
Easy to read?	☐	☐
Well organized?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Accurate?	☐	☐
Suitable for its intended audience?	☐	☐

- How did you use this publication?
  - ☐ As an introduction to the subject
  - ☐ For additional knowledge

  Other .....

- Please check the items that describe your position:
  - ☐ Customer personnel
  - ☐ IBM personnel
  - ☐ Manager
  - ☐ Systems Analyst
  - ☐ Operator
  - ☐ Programmer
  - ☐ Customer Engineer
  - ☐ Instructor
  - ☐ Sales Representative
  - ☐ Systems Engineer
  - ☐ Trainee

  Other .....

- Please check specific criticism(s), give page number(s), and explain below:
  - ☐ Clarification on page(s) .....
  - ☐ Addition on page(s) .....
  - ☐ Deletion on page(s) .....
  - ☐ Error on page(s) .....

Explanation:

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

YOUR COMMENTS, PLEASE . . .

This manual is part of a library that serves as a reference source for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Note: Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Fold Fold

FIRST CLASS
PERMIT NO. 81
POUGHKEEPSIE, N.Y.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY ...

IBM Corporation
P.O. Box 390
Poughkeepsie, N.Y. 12602

Attention: Programming Systems Publications
Department D58

Fold Fold

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
(USA Only)

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

Cut Along Line

System/360 OS Job Management (S360-36) Printed in U.S.A. GY28-6613-5

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]