

Program Logic

Version 8.1

IBM System/360 Time Sharing System Resident Supervisor

Describes the internal logic of the resident supervisor, and provides a brief overview of its operation. It is intended as a reference for anyone involved in maintaining or altering resident supervisor logic.

The resident supervisor schedules and dispatches tasks, provides services that might endanger system integrity if a task were allowed to execute them, handles interruptions, and deals with system errors. It is permanently resident in main storage.

The first and second sections of the book are intended to introduce the reader to the functions of the resident supervisor, and to provide him with an easily used, overall presentation of resident supervisor logic. Interruption handling is described from the point at which the interruption occurs until it has been completely processed. Paging and queuing are discussed. The third and fourth sections describe in some detail the individual modules that make up the resident supervisor. These modules are discussed under the following headings: interruption classification, queue scanning and processing, storage allocation, SVC processing, paging, I/O handling, task selection and scheduling, and error recovery. The appendixes contain module IDs and names of supervisor routines, and SVC codes.

The prerequisites for this publication are: IBM System/360 Principles of Operation, GA22-6821, and IBM System/360 Time Sharing System: Concepts and Facilities, GC28-2003.

PREFACE

This publication describes the logic and operation of the TSS/360 resident supervisor. It is divided into four sections and two appendixes. Section 1, the introduction, describes the purpose and major components of the resident supervisor, and how these components interact with the rest of TSS/360. Section 2, the method of operation, describes resident supervisor interruption handling, queue processing, storage allocation, and task selection and scheduling. Section 3, program organization, discusses the internal logic of the resident supervisor. Each routine is described in detail. Section 4 contains flowcharts for the more complex supervisor modules. Appendix A contains a list of module IDs and entry point names for all modules in the resident supervisor. Appendix B is a table of defined SVC codes and their meaning.

This publication is intended for use by anyone involved in maintaining or altering

resident supervisor logic. It will be particularly useful to systems programmers.

PREREQUISITE PUBLICATIONS

Familiarity with the material contained in the following publications is essential to the use of this manual:

IBM System/360 Principles of Operation, GA22-6821

IBM System/360 Time Sharing System: Concepts and Facilities, GC28-2003

In addition, IBM System/360 Time Sharing System: System Control Blocks PLM, GY28-2011, and IBM System/360 Time Sharing System: Assembler User Macro Instructions, GC28-2004, should be available for reference purposes.

Sixth Edition (September 1971)

This is a major revision of, and makes obsolete, GY28-2012-4.

This edition reflects changes to the resident supervisor intended to improve its performance and make it even more efficient. Several task selection and scheduling methods have been changed, the paging error recovery function has been expanded, and XTSI paging has been revised. In addition, the structure of the book itself has been changed for easier reference and a method of operation section added.

This edition is current with Version 8, Modification 1, and remains in effect for all subsequent versions or modifications of IBM System/360 Time Sharing System unless otherwise indicated. Significant changes or additions to this publication will be provided in new editions or Technical Newsletters. Before using this publication, refer to the latest edition of IBM System/360 Time Sharing System: Addendum, GC28-2043, which may contain information pertinent to the topics covered in this edition. The Addendum also lists the editions of all TSS/360 publications that are applicable and current.

This publication was prepared for production using an IBM computer to update the text and to control the page and line format. Page impressions for photo-offset printing were obtained from an IBM 1403 printer using a special print chain.

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

A form is provided at the back of the publication for reader's comments. If the form has been removed, comments may be addressed to the IBM Corporation, Time Sharing System/360 Programming Publications, Department 643, Neighborhood Road, Kingston, New York 12401

© Copyright International Business Machines Corporation 1967, 1968, 1969, 1970, 1971

CONTENTS

SECTION 1. INTRODUCTION 1
 Relationship to the System 1
 Purpose and Functions 1
 Major Components of the Resident Supervisor 2

SECTION 2. METHOD OF OPERATION 4
Overview of Resident Supervisor Operations 4
 Processing of Interruptions 4
 Scheduling of Tasks 5
 Control Blocks Used by the Supervisor 5
Interruption Handling 6
 Entry to the Supervisor 6
 General Queue Entry (GQE) 7
 Queue Scanning 8
 Scan Table (SCANT) 8
 Scan Table Master Control Table (SMC) 8
 Queue Scanner Functions 8
 Queue Processing 9
 Timer Interruption Processing 9
 Segment and Page Tables 10
 Program Interruption Processing 10
 I/O Interruption Processing 11
 Pathfinding 12
 Paging 12
 Disk Paging 13
 Drum Paging 13
 Main Storage Allocation 13
 User Core Allocation 14
 Supervisor Core Allocation 14
 Auxiliary Storage Allocation 15
 SVC Interruption Processing 15
 Task Scheduling and Selection 16
 Schedule Table (CHASTE) 16
 Active and Inactive Lists 16
 Task Selection 17
 Task Scheduling 17

SECTION 3: PROGRAM ORGANIZATION 20
Interruption Classification 20
 Interrupt Stacker Module (CEAJI) Chart AA 20
Queue Scanning and Processing 25
 Queue Scanner (CEAJQ) Chart AB 25
 Queue-Control Subroutines 26
 Enqueue GQE Subroutine (CEAJQ Entered at CEAJEN) 26
 Dequeue GQE Subroutine (CEAJQ Entered at CEAJDE) 27
 Move GQE Subroutine (CEAJQ Entered at CEAJMG) 28
 Set Suppress Flag Subroutine (CEAJQ Entered at CEAJSF) Chart AC 28
Queue Processors 29
 Timer Interrupt Queue Processor (CEAKT) Chart AD 29
 Page Drum Queue Processor (CEAA8) Chart AE 33
 Page Drum Interrupt Queue Processor (CEAA9) Chart AF 35
 Program Interrupt Queue Processor (CEANA) Chart AG 38
 I/O Device Queue Processor (CEAA3) Chart AH 41
 Page Direct Access Interrupt Subprocessor (CEAA7) Chart AK 43
 Page Direct Access Queue Subprocessor (CEAA6) Chart AI 45
 Channel Interrupt Queue Processor (CEAA4) Chart AL 47
 Remote Job Entry Asynchronous I/O Interrupt Subroutine (CEABA)
 Chart AL 53
 Remote Job Entry Synchronous I/O Error Interrupt Subroutine
 (CEABB) 56
 Terminal Communications Subprocessor (CEATC) Chart AM 58
Storage Allocation Processors 63

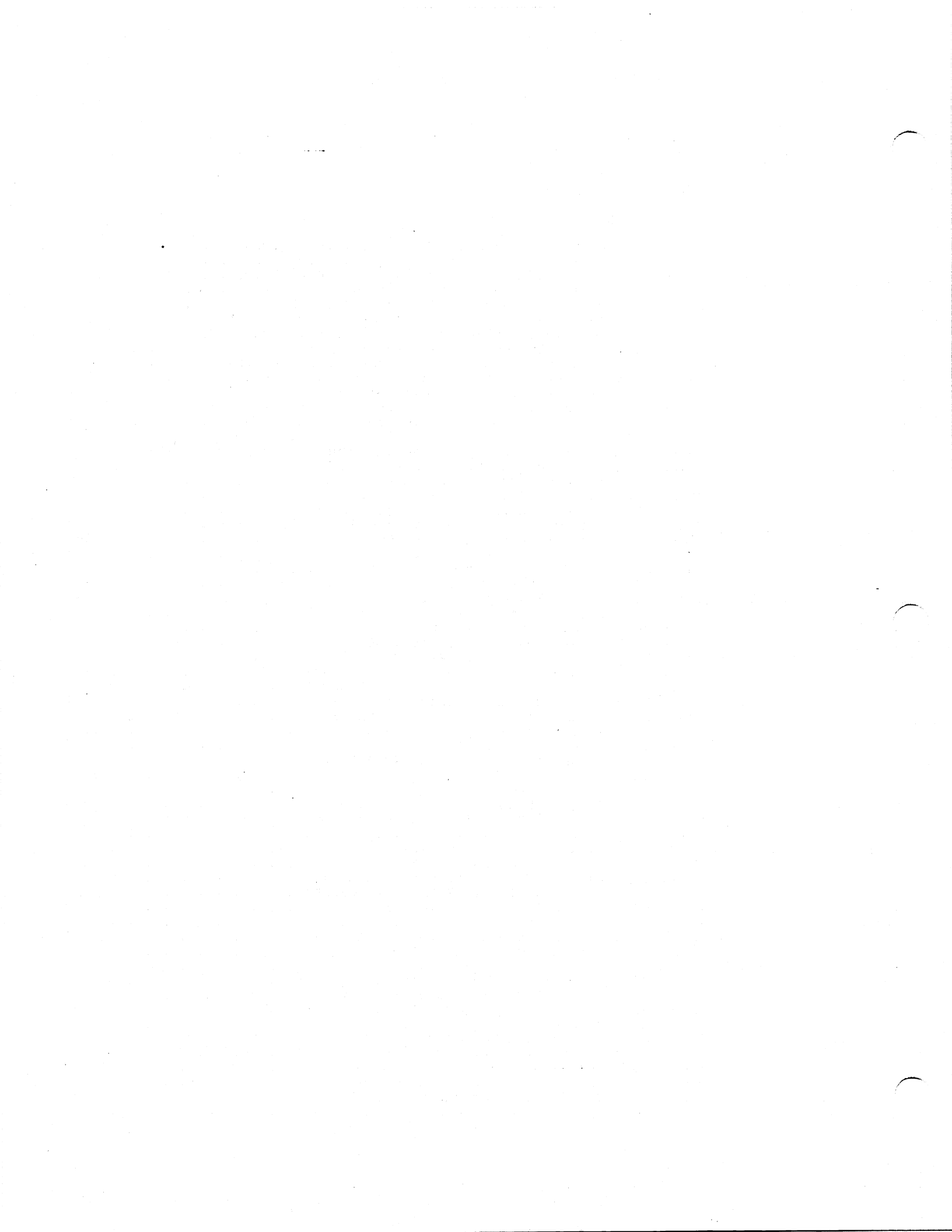
User Core Allocation Queue Processor (CEANB) Chart AN	63
Auxiliary Storage Allocation Queue Processor (CEAIA) Chart AO	65
Contiguous Core Allocation Queue Processor (CEANF)	67
Supervisor Core Allocation Subroutine (CEAL1 Entered at CEAL01)	
Chart AP	69
Supervisor Core Release Subroutine (CEAL1 Entered at CEAL02)	
Chart AP	70
User Core Release Subroutine (CEAL1 Entered at CEAL04) Chart AP	71
Auxiliary Storage Release Subroutine (CEAIA) Chart AO	71
Suppress Auxiliary Allocation Subroutine (CEAAP)	72
SVC Queue Processor and Service Routines	72
Supervisor Call Queue Processor (CEAHQ) Chart AQ	72
Add Page Subroutine (CEAHQ entered at CEAHQA) Chart AR	73
Add Shared Pages Subroutine (CEAQ6) Chart AS	75
Delete Page Subroutine (CEAND)	76
Set External Page Table Entries Subroutine (CEAH7)	77
Move External Page Table Entries Subroutine (CEAP0)	78
Connect Segment to Shared Page Table Subroutine (CEAQ7) Chart AT	78
Disconnect Segment From Shared Page Table Subroutine (CEAQ8)	79
Check Protection Class Subroutine (CEAQ4) Chart AU	79
Create-TSI Subroutine (CEAMC)	80
Special Create TSI Processor (CEAT2)	80
Delete TSI Processor (CEAMD) Chart AV	80
Set up XTSI Field Subroutine (CEAS4)	81
Set up TSI Field Subroutine (CEAH2)	81
Extract TSI Field Subroutine (CEAH2)	82
Extract XTSI Field Subroutine (CEAS4)	82
Time Slice End Subroutine (CEAHQ entered at CEAHQF)	83
AWAIT SVC Subroutine (CEAP7) Chart AW	83
TWAIT Subprocessor (CEAR0)	84
Pulse Schedule Table Entry Processor (CEAR2)	84
Change Schedule Table Entry Processor (CEAR3)	84
Set User Interval Timer Subroutine (CEAQ2)	85
Set Real Time Interval Subroutine (CEAS7) Chart AX	85
Restore Elapsed Time Subroutine (CEAS8)	87
Read-Time Subroutine (CEAS6)	87
System Table Modification and Extraction Processors	88
Set up System Table Field Subroutine (CEAS2 Entered at CEAH42)	88
Extract System Table Field Subroutine (CEAS2 Entered at CEAH43)	88
Extract Accumulated Time Routine (CEAT1)	88
Extract from Auxiliary Storage Allocation Table (CEAT4)	89
I/O Call Subroutine (CEAA0) Chart AY	89
Pageout Service Subroutine (CEAA1) Chart AZ	90
Remote Job Entry Line Control Subroutine (CEABC) Chart BA	92
Reset Device Suppression Flag Subroutine (CEAAH)	94
Set Path Subroutine (CEAAB)	94
Queue Device on Task Subroutine (CEAAC)	94
Remove Device From Task Subroutine (CEAAD)	95
Set Asynchronous Entry Subroutine (CEAAK)	96
Terminal SVC Processor (CEAR4) Chart BB	97
Reset Drum Interlock Subroutine (CEAAZ)	99
Inter-Task Communication Subroutine (CEAQ5)	99
TSS Dynamic Status (CEASS)	100
Supervisor Subroutines	101
Page-Handling Subroutines	101
Find Page Subroutine (CEANC)	101
Locate Page Subroutine (CEAML)	102
Page Posting Subroutine (CEAMP) Chart BC	102
Write Shared Pages Subroutine (CEAMW)	106
External Page Location Address Translator Subroutine (CEAAE)	108
Search-RSPI-Table Subroutine (CEAMS)	109
Segment Block Remover Subroutine (CEANG) Chart BD	109
XTSI Page Packing Subroutine (CEAMY)	110
Real Core Statistical Data Recording Subroutine (CEAI6) Chart BE	110
Real Core Error Recording Subroutine (CEAI7) Chart BF	110
Paging Error Recovery Routines	111
Paging Failure Recovery Subroutine (CEAAQ) Chart BG	111
Paging I/O Error Recovery Routine (CEAAM)	113

Start Retry Operation Subroutine (CEAAX)115
Standard Area Retry Subroutine (CEAAT)116
Alternate Path Retry Subroutine (CEAAS) Chart BH117
Same Path Retry Subroutine (CEAAV) Chart BI119
I/O Service Subroutines123
Pathfinding Subroutine (CEAA5) Chart BJ123
Start I/O Subroutine (CEAAG) Chart BK125
Halt I/O Subroutine (CEAAI)126
Dequeue I/O Requests Subroutine (CEAAJ) Chart BL128
Generate and Enqueue Interrupt-GQE Subroutine (CEABQ)129
Command Word Relocator Subroutine (CEAAA)130
Purge Subroutine (CEAAL) Chart BM131
Terminal Control Table Entry Slot Allocation Subroutine (CEATS) Chart BN133
Special Task Service Subroutines133
Task Initiation Subroutine (CEAMC) Chart BO134
XTSI Overflow Subroutine (CEAMX)134
Queue GQE on TSI Subroutine (CEAAF) Chart BP137
Task Communication Control Subroutine (CEAAN)139
General Service Subroutines140
Inter-CPU Communication Subroutine (CEAIC) Chart BQ140
Create Real Time Interrupt Subroutine (CEAKR) Chart BR142
Task Selection and Scheduling Routines143
Internal Scheduler (CEAKI) Chart BS143
The Dispatcher (CEAKD) Chart BT144
Task Interrupt Control Subroutine (CEAA2) Chart BU145
Entrance Criteria Subroutine (CEAKE) Chart BV146
Rescheduling Subroutine (CEAKZ) Chart BW147
Major Error Recovery Procedures148
Recovery Nucleus-67 (CEAIR) Chart BX148
Reconfiguration Routine (CGCMA)152
External Machine Check Interrupt Processor (CEABE) Chart BY153
System Environment Recording and Retry Programs154
SERR Bootstrap (CMASA) Chart BZ156
Environment Recording Program (CMASB)158
Immediate Print Program (CMASC)158
Checker Program (CMASD) Chart CA161
Pointer Program (CMASE)162
Restore and Validate Program (CMASF) Chart CB162
Instruction Retry Execution Program (CMASG) Chart CC163
CPU/Memory Checkout 1 Program (CMASH) Chart CD163
CPU/Memory Checkout 2 Program (CMASI) Chart CE164
CPU/Memory Checkout 3 Program (CMASJ) Chart CF164
System Error Processor (CEAIS) Chart CG164
SECTION 4: FLOWCHARTS167
APPENDIX A: MODULE IDS AND NAMES313
APPENDIX B: TSS/360 SVC CODES317
INDEX322

ILLUSTRATIONS

Figure 1.	General flow of resident supervisor functions	1
Figure 2.	Resident supervisor components and their functions	3
Figure 3.	Interruption receiving	4
Figure 4.	Interruption processing	5
Figure 5.	Task scheduling and selection	6
Figure 6.	Queue Scanning and processing module interface	9
Figure 7.	Page table relationship	11
Figure 8.	Activities of the SVC Queue Processor	16
Figure 9.	Interrupt stacker module overview	20
Figure 10.	The interrupt log	21
Figure 11.	Timer Interrupt Queue Processor activities	31
Figure 12.	Page Drum Interrupt Queue Processor activities	36
Figure 13.	Page Drum Interrupt Queue Processor checking and response to conditions specified in the CSW	37
Figure 14.	Activities of the Program Interrupt Queue Processor	39
Figure 15.	PDAQ Processor cross-referencing between the GQE, PCB, and the DAIB	46
Figure 16.	Activities of the Auxiliary Storage Allocation Queue Processor	67
Figure 17.	SCA control information block	69
Figure 18.	Device type table format	108
Figure 19.	Track and record ID format	108
Figure 20.	Format of translated symbolic addresses	108
Figure 21.	Format of path availability check results	124
Figure 22.	Contents of Halt I/O return registers	128
Figure 23.	CCW - page list structure	130
Figure 24.	XTSI states	136
Figure 25.	Action Matrix for TSI Flag Settings	138
Figure 26.	General flow through SERR after a machine-check interruption	156
Figure 27.	Record format for an internal machine check (call types 01 and 09 from the recovery nucleus)	159
Figure 28.	Record format for an external machine check (call type 29)	159
Figure 29.	Record format for a paging device SDR overflow or a solid paging I/O outboard error (call types 27, 28, and 2E)	160
Figure 30.	Record format for a system error (call type 41)	160
Figure 31.	Record format for a paging I/O inboard failure (call type 2D)	161
Table 1.	Major control blocks used by the supervisor	7
Table 2.	Queue Processors that work off the scan table in their order of priority	8
Table 3.	Paging and storage allocation control blocks	12
Table 4.	Schedule table entry parameters	18
Table 5.	QUEUE Scanner operations in processing of GQE	29

Chart AA.	Interrupt Stacker (CEAJI)	.168
Chart AB.	Queue Scanner (CEAJQ)	.174
Chart AC.	Set Suppress Flags subroutine (CEAJSF)	.175
Chart AD.	Timer Interrupt Queue Processor (CEAKT)	.176
Chart AE.	Page Drum Queue Processor (CEAA8)	.182
Chart AF.	Page Drum Interrupt Queue Processor (CEAA9)	.184
Chart AG.	Program Interrupt Queue Processor (CEANA)	.186
Chart AH.	I/O Device Queue Processor (CEAA3)	.188
Chart AI.	Page Direct Access Interrupt subroutine (CEAA7)	.192
Chart AJ.	Page Direct Access Queue Processor (CEAA6)	.193
Chart AK.	Channel Interrupt Queue Processor (CEAA4)	.194
Chart AL.	RJE Asynchronous Interrupt subroutine (CEABA)	.200
Chart AM.	Terminal Communications Subprocessor (CEATC)	.204
Chart AN.	User Core Allocation (CEANB)	.223
Chart AO.	Auxiliary Storage Allocation Queue Processor (CEAIA)	.227
Chart AP.	Core Control subroutines (CEAL1)	.231
Chart AQ.	SVC Queue Processor (CEAAQ)	.235
Chart AR.	Add Page SVC (CEAHQA)	.236
Chart AS.	Add Shared Page subroutine (CEAQ6)	.239
Chart AT.	Connect segment to shared segment (CEAQ7)	.240
Chart AU.	Check Protection Class SVC (CEAH8)	.241
Chart AV.	Delete TSI (CEAMD)	.242
Chart AW.	AWAIT SVC Processor (CEAP7)	.245
Chart AX.	Set Real Time Interval subroutine (CEAS7)	.246
Chart AY.	I/O Call subroutine (CEAAO)	.247
Chart AZ.	Pageout Service subroutine (CEAA1)	.248
Chart BA.	RJE line control (CEABC)	.249
Chart BB.	Terminal SVC Processor (CEAR4)	.250
Chart BC.	Page posting (CEAMP)	.256
Chart BD.	Segment Block Remover subroutine (CEANG)	.259
Chart BE.	Real Core Statistical Data Recording (CEAI6)	.260
Chart BF.	Real Core Error Recording (CEAI7)	.261
Chart BG.	Paging Failure Recovery (CEAAQ)	.263
Chart BH.	Alternate Path Retry (CEAAS)	.264
Chart BI.	Same Path Retry (CEAAV)	.265
Chart BJ.	Pathfinding subroutine (CEAA5)	.267
Chart BK.	Start I/O subroutine (CEAAG)	.274
Chart BL.	Dequeue I/O Requests (CEAAJ)	.275
Chart BM.	Purge (CEAAL)	.276
Chart BN.	Terminal Control Table Entry Slot Allocation subroutine (CEATS)	.277
Chart BO.	Task Initiation (CEAMC)	.283
Chart BP.	Queue GQE on TSI (CEAAF)	.284
Chart BQ.	Inter-CPU Communications (CEAIC)	.287
Chart BR.	Create Real Time Interrupt (CEAKR)	.288
Chart BS.	Internal Scheduler (CEAKI)	.289
Chart BT.	Dispatcher (CEAKD)	.290
Chart BU.	Task Interrupt Control (CEAA2)	.291
Chart BV.	Entrance Criteria (CEAKE)	.292
Chart BW.	Rescheduling (CEAKZ)	.293
Chart BX.	Recovery Nucleus (CEAIR)	.297
Chart BY.	External Machine Check Interrupt Processor (CEABE)	.302
Chart BZ.	SERR Bootstrap (CMASA)	.303
Chart CA.	Checker Program (CMASD)	.304
Chart CB.	Restore and Validate (CMASF)	.305
Chart CC.	Instruction Retry Execution (CMASG)	.306
Chart CD.	CPU/Memory Checkout 1 (CMASH)	.307
Chart CE.	CPU/Memory Checkout 2 (CMASI)	.308
Chart CF.	CPU/Memory Checkout 3 (CMASJ)	.309
Chart CG.	System Error Processor (CEAIS)	.310



Relationship to the System

The TSS/360 resident supervisor is the only TSS/360 component that is permanently resident in main storage after startup. By contrast with other system components, which reside in virtual storage and are paged in when needed, the supervisor is nonpageable and nonrelocatable; its instruction operands are main storage addresses, not logical addresses. No location within the resident supervisor may be addressed by a program operating in virtual storage; thus it is protected from being altered by other system components or user programs. Supervisor modules may execute privileged instructions; they therefore use Type I linkage to communicate with each other.

Purpose and Functions

The resident supervisor is responsible for accepting and processing interruptions and scheduling the use of system resources. The latter involves such operations as:

- Time slicing and task dispatching.
- Storage allocation.
- Handling of paging and non-paging I/O requests.
- CPU and paging I/O error retry and recovery.

Entry to the supervisor is made by means of an interruption (see Figure 1). These interruptions represent such things as: error conditions, time-slice end, I/O hardware completions, and requests from tasks for services. An interruption may be received before the processing of previous interruptions has been completed; therefore, on entry, interruptions are stacked (remembered) until they can be processed. Since it is possible to receive more than one interruption at a time, each interruption is attached to the queue of the appropriate interrupt processor (according to interruption type); the queues are then processed on a priority basis. Complete information on the status of the interrupted task is saved during processing.

When there is no further work to be processed (that is, all the queues have been examined and emptied, if possible), the resident supervisor selects the next task to be given CPU time. Each task has assigned to it a set of scheduling parameters; the value of its time slice starts at the value indicated in one of these parameters. When the task is interrupted, a record is kept of the CPU time used. When the task is set into execution again, its time slice is set equal to the initial value minus the CPU time the task has already used.

A time slice is seldom used on a straight through basis; generally there is

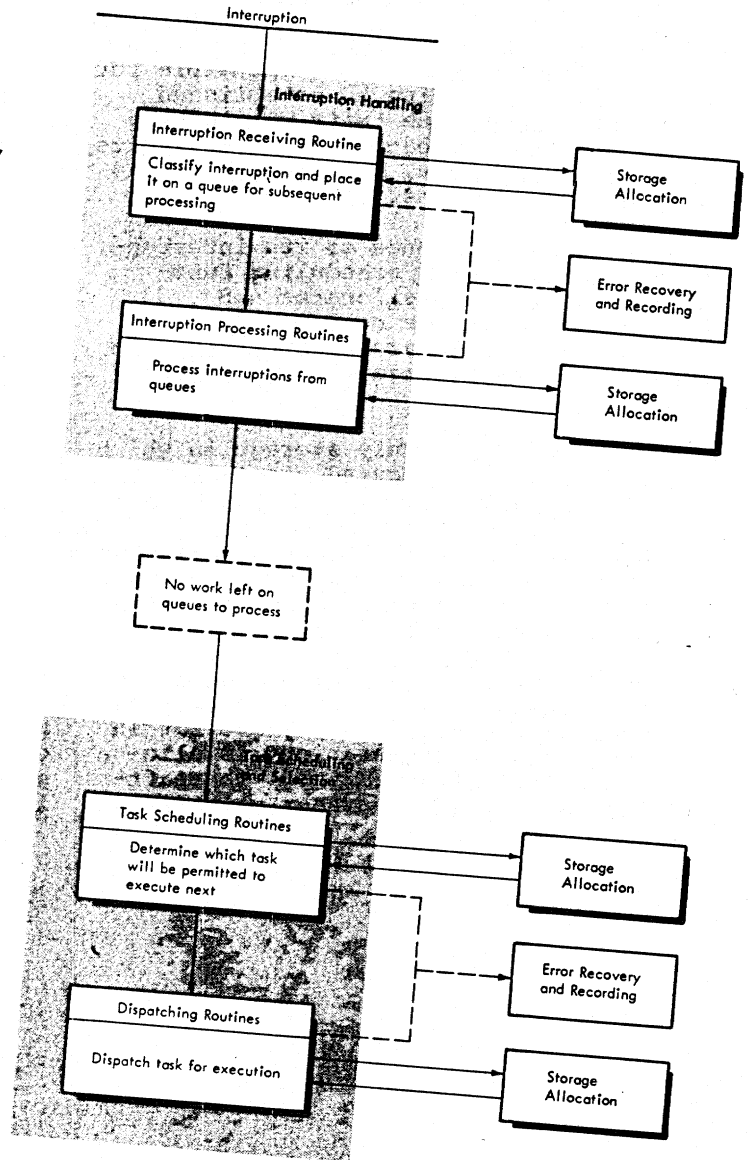


Figure 1. General flow of resident supervisor functions

some waiting time for input/output, terminal, or paging operations. During these waiting periods, other tasks may begin their time slices. However, when an interruption signals that the first task is ready to proceed, it is again readied for execution. The scheduling components of the resident supervisor are responsible for ordering tasks according to established priorities to determine their eligibility for execution. Thus each task receives its fair share of CPU time, and maximum use is made of system elements.

During the performance of its interruption handling and task scheduling functions, the supervisor allocates and releases storage as the need arises. Two categories of storage are allocated by the supervisor: main storage and auxiliary storage.

- Main storage, the only storage in which programs can be executed, is initially allocated at startup in 4096-byte units called pages. During system operation, the supervisor reallocates this storage in either page or smaller block units for specific uses by supervisor components and user tasks.
- Auxiliary storage, which consists of the drums and disks used as temporary storage for pages when they are not in use during system operation, is allocated initially at startup. Thereafter, the resident supervisor maintains a

constantly updated count of available auxiliary storage, allocates it to user tasks as needed, and releases and returns it to available status when its use is no longer required.

The resident supervisor exercises its error recovery and retry capabilities, when necessary, to dynamically correct errors or to minimize the effect of errors on the system as a whole. The general approach to error recovery is to retry failing operation, where possible; when an operation cannot be retried or is retried without success, or when a hardware element cannot be made to perform correctly, the failing element or device is removed from the system in an orderly manner so as not to disrupt system operation. Only as a last resort, when recovery is not possible or removal of the failing element would render the system inoperative, is the system shut down. System environment recording facilities maintain a continuous error history, including complete hardware environment at the time of failure, on the paging drum for subsequent use by the customer engineer.

Major Components of the Resident Supervisor

The major components of the resident supervisor can be grouped according to the functions they perform (see Figure 2). In addition to its major components, a group of subroutines provide services throughout supervisor operations; these are described in Section 3.

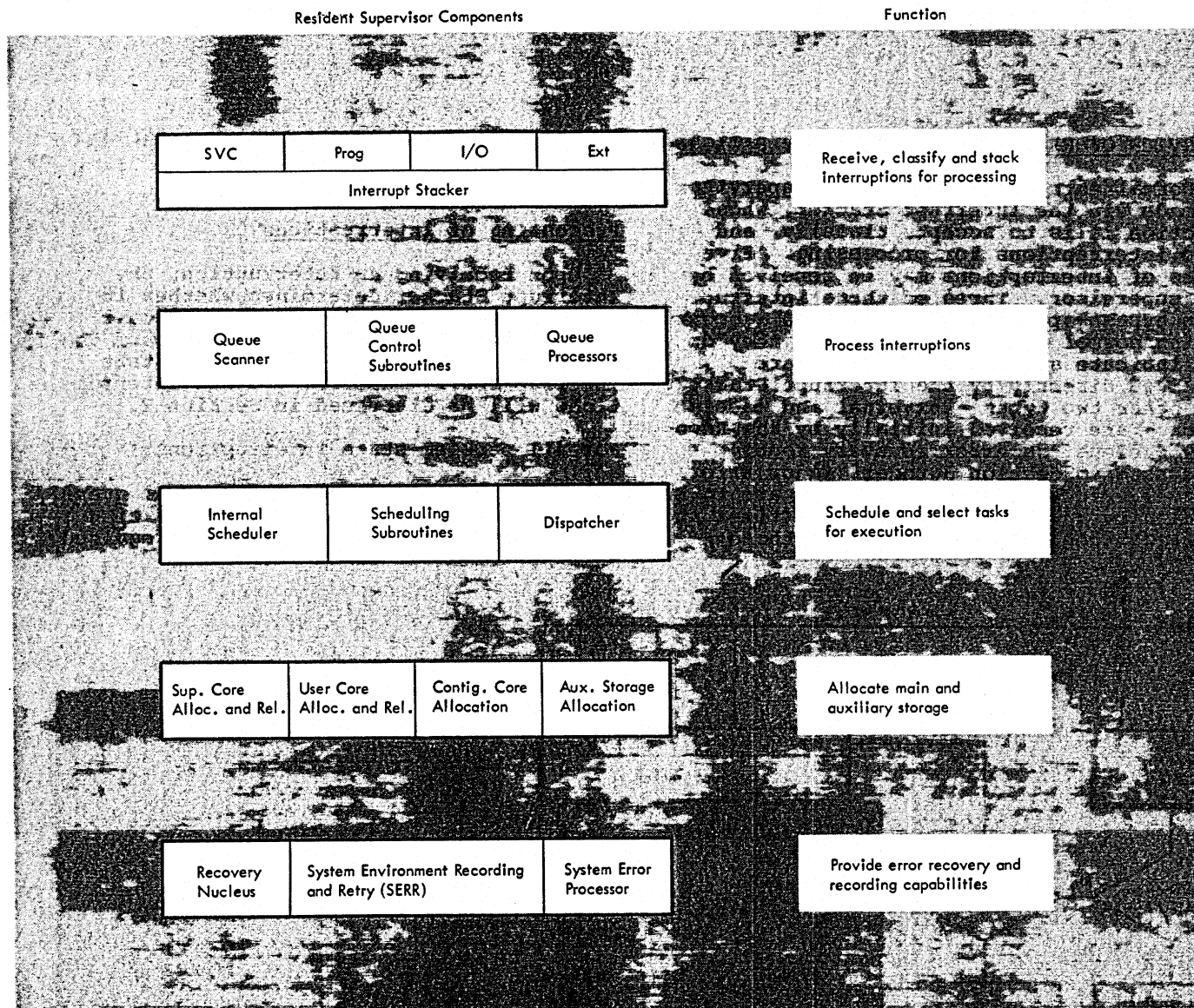


Figure 2. Resident supervisor components and their functions

SECTION 2. METHOD OF OPERATION

OVERVIEW OF RESIDENT SUPERVISOR OPERATIONS

Normal entry to the resident supervisor is made via the Interrupt Stacker, whose function it is to accept, classify, and save interruptions for processing. Five types of interruptions may be received by the supervisor. Three of these interruption types - program, SVC, and I/O - occur during normal operation (that is, they do not indicate system errors), and are received directly by the Interrupt Stacker. The other two types - external and machine check - are received initially by the Recovery Nucleus, an error recovery routine discussed in Section 3 under "Major Error Recovery Procedures". Those that are to be handled by the Interrupt Stacker (timer and interrupt key) are passed to it; those

indicating a malfunction are handled by error recovery routines (see Figure 3).

Processing of Interruptions

Upon receiving an interruption, the Interrupt Stacker determines whether it occurred during the execution of a task (problem-state interruption) or one of the supervisor components (supervisor-state interruption). Supervisor-state interruptions will be discussed in Section 2.

For all problem-state interruptions:

- A 64-byte block called a general queue entry (GQE) is generated, containing a description of the processing required.

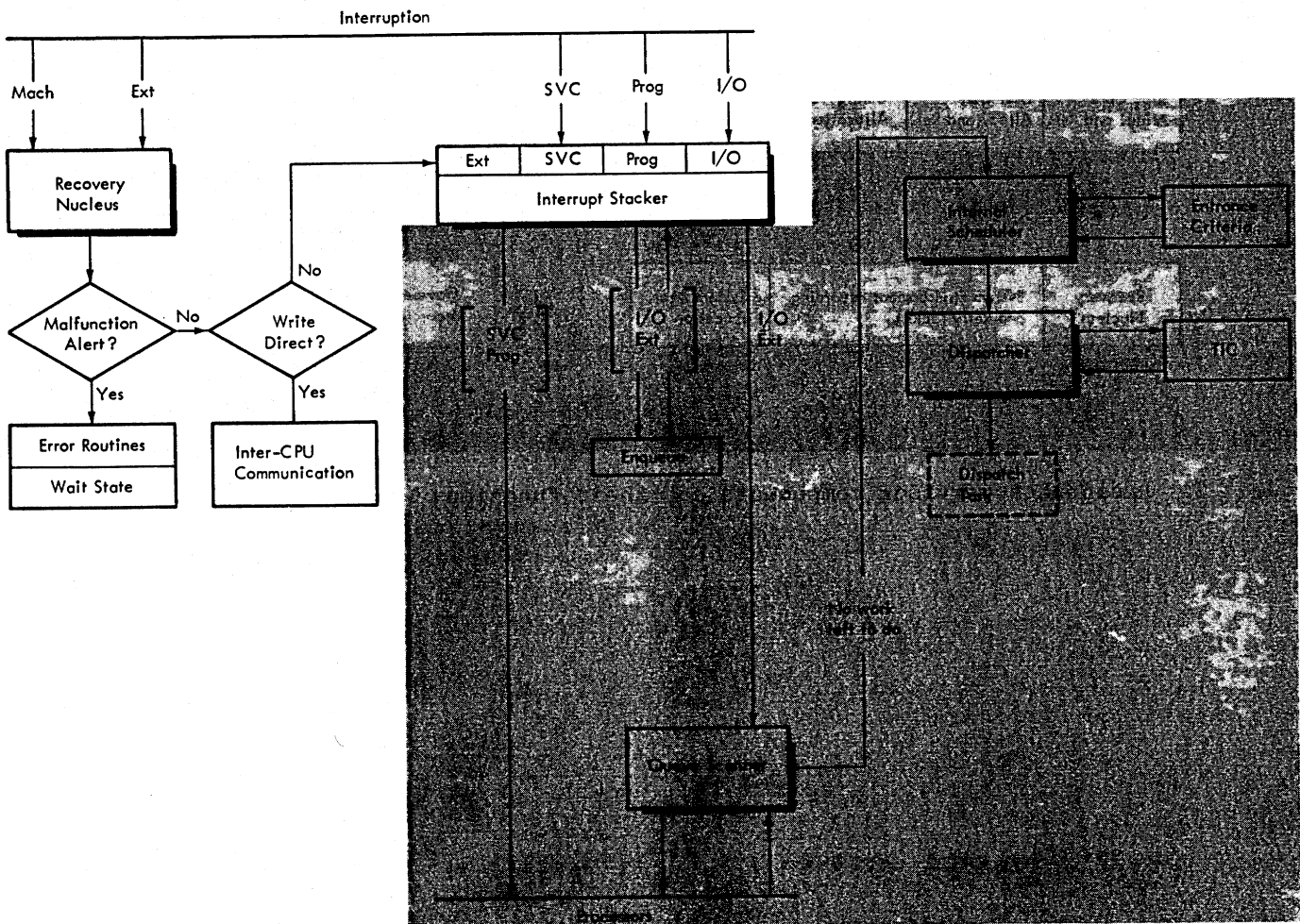


Figure 3. Interruption receiving

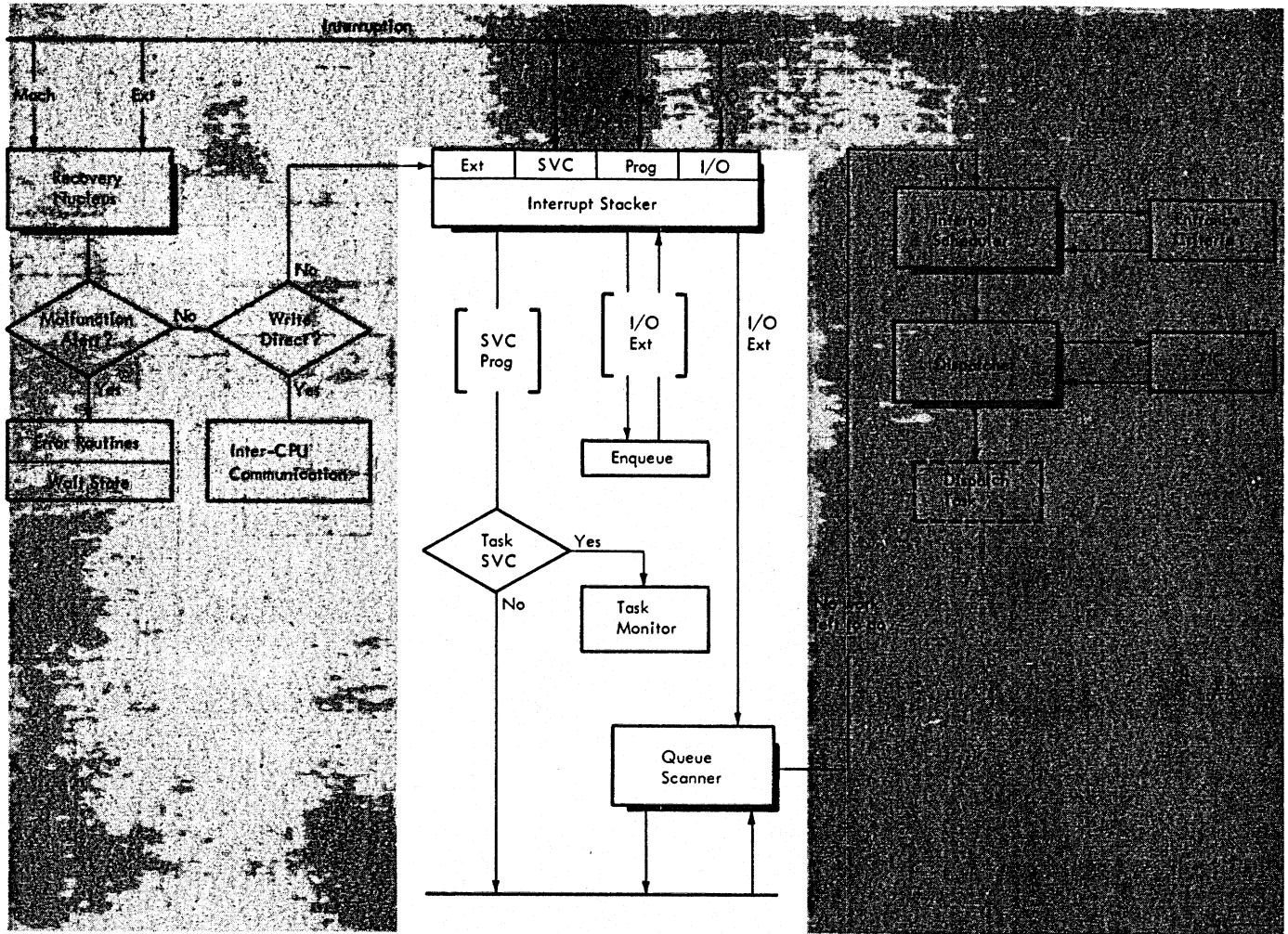


Figure 4. Interruption processing

- I/O and timer interruptions - GQEs for these interruptions are queued on a table (scan table) used to contain pointers to the GQEs representing work in progress or waiting to be performed; control is then transferred to the Queue Scanner.
- SVC and program interruptions - GQEs for these interruptions are not queued on the scan table; control is transferred directly to the appropriate interrupt processor or to the task via LPSW.

The Queue Scanner locates queue entries in the scan table, then transfers control to the appropriate queue processor, until no processable work remains. See Figure 4.

Scheduling of Tasks

The task scheduling and selection mechanism of the supervisor is entered when the Queue Scanner can find no work left to process in the scan table. Control is

transferred to the Internal Scheduler, which moves tasks from the list of those eligible for CPU time (eligible list) to the list of those ready to execute (dispatchable list). It is assisted by the Entrance Criteria subroutine, which verifies or denies the task's eligibility to be moved. The Internal Scheduler then passes control to the Dispatcher, which selects from the dispatchable list the task to be executed. Before the task is dispatched, the Task Interrupt Control subroutine checks for pending interruptions to the task, and arranges for them to be serviced by the task monitor (see Figure 5). Once a task has used up its allotted CPU time, it is moved to a list of inactive tasks, after which the Rescheduling routine is invoked to compute a new time value for scheduling the task.

Control Blocks Used by the Supervisor

Each supervisor component keeps detailed records of the status of interruptions

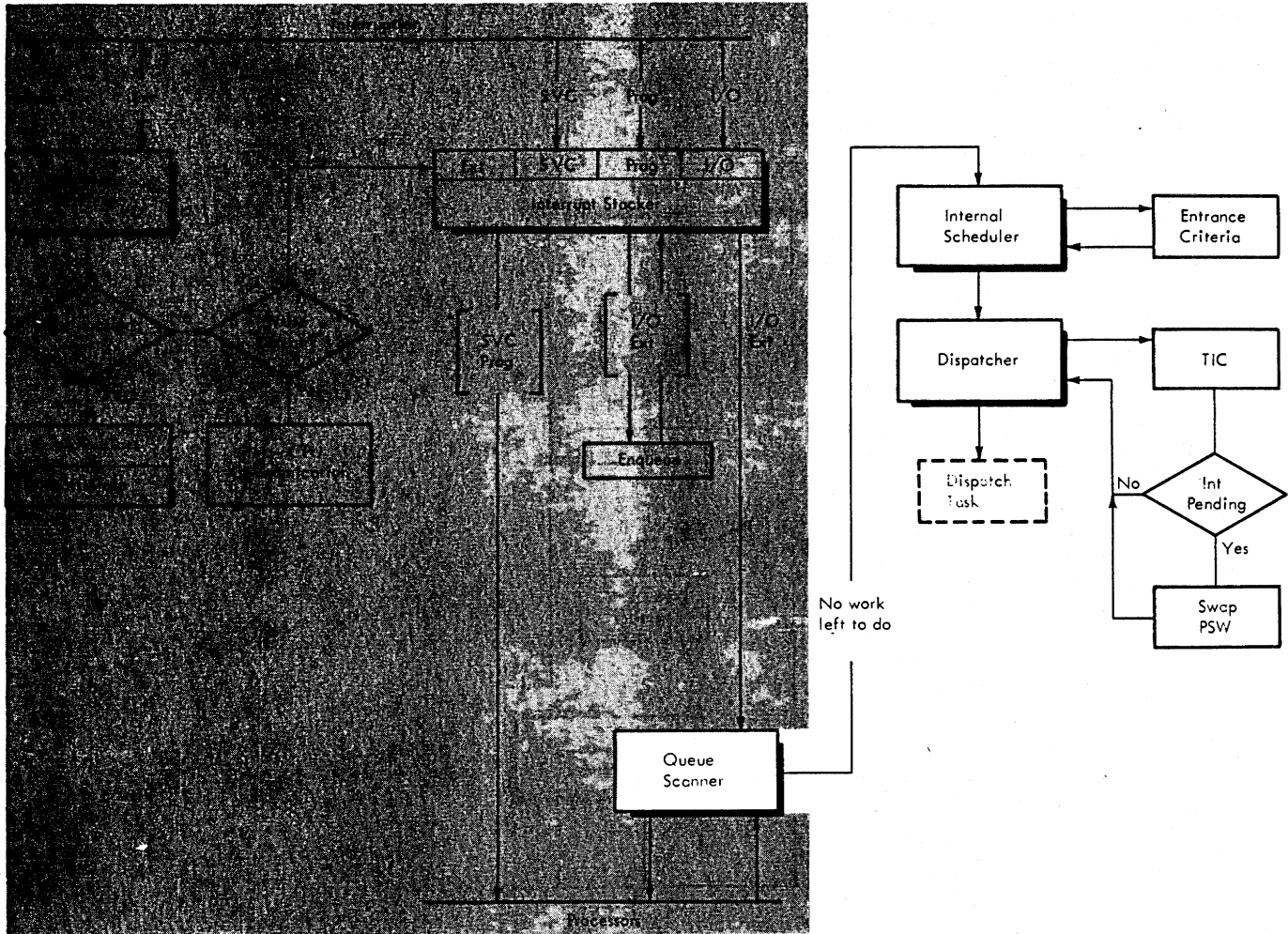


Figure 5. Task scheduling and selection

throughout its processing. These records are kept in tables, or control blocks, which are available to each supervisor component required to process an interruption. Detailed descriptions of these tables are presented in the System Control Blocks manual. General descriptions of tables used by specialized groups of supervisor components are presented within the descriptions of these modules. Table 1 provides a brief summary of the functions of these control blocks. (Another table of control blocks, concerned exclusively with paging and storage allocation activities, can be found in Section 2 under "Program Interruption Processing").

INTERRUPTION HANDLING

Entry to the Supervisor

Five types of interruptions cause the resident supervisor to be entered:

- Program interruption - occurs when any of 17 program interruption codes are generated by the System/360 Model 67. When it occurs while the CPU is operating in the supervisor state (that is, it is caused by the supervisor), an error is indicated; occurring in the problem state (during execution of a task), a program interruption is handled by the task monitor (codes 1 - 15) or the resident supervisor (codes 16 and 17).
- I/O interruption - represents the method by which an I/O device signals the CPU that I/O is completed. These are classified and queued for processing according to type: interruptions involving drum paging operations are distinguished from all others and transferred to a separate processor.
- SVC interruption - the supervisor call (SVC) is the normal method of communication between a task and the supervi-

Table 1. Major control blocks used by the supervisor

Control Block	Function
General Queue Entry (GQE)	64-byte control block created at interruption time to contain information describing processing required by queue processors.
Task Status Index (TSI)	Contains nucleus of task information that <u>must</u> be retained in main storage; also a pointer to the XTSI. A TSI exists for each task.
Extended Task Status Index (XTSI)	Contains that portion of task's information that is <u>not required</u> to be permanently resident in main storage.
Page Control Block (PCB)	Created each time a paging operation is indicated. Controls movement of virtual storage pages between main and auxiliary or external storage.
Scan Table (SCANT)	Serves as common anchor point for all GQEs representing work in progress or waiting to be performed inside supervisor.
Scan Table Master Control Table (SMC)	Contains information used by Queue Scanner for more efficient search for work in scan table.
Schedule Table (CHASTE)	Parameter table used by scheduling mechanism of supervisor to determine what scheduling characteristics to apply to each task in TSS/360.
System Table (SYS)	Contains a variety of system and installation parameters used by scheduling and paging mechanisms; anchor point for chain of TSIs.

sor, caused by the execution of an SVC instruction that is usually embedded in the expansion of a macro instruction. Depending upon the SVC code, these interruptions are serviced by the Time Sharing Support System (TSSS), the task monitor, or the resident supervisor SVC processing routines.

- External interruption - can result from a timer interruption, an inter-CPU communication in the form of a write-direct message, or a machine malfunction alert. External interruptions are initially intercepted by the Recovery Nucleus (discussed in Section 3 under "Major Error Recovery Procedures"); those that are to be handled by the Interrupt Stacker (timer and interrupt key) are passed on to it.
- Machine check interruption - indicates that a hardware detected error has occurred; error recovery procedures are initiated by the Recovery Nucleus.

The Interrupt Stacker comprises four separate stackers, one for each interruption type it receives. Interruptions are classified as they are received, and the information associated with them (old PSW, interrupt code, etc.) is saved. The

stacker receiving the interruption determines whether the system is under the control of the resident support system (RSS). If it is, the interruption is processed and control is returned to the appropriate RSS/VSS handling module. Those interruptions that do not involve RSS are then classified as problem state or supervisor state, depending upon whether the interruption occurred during the execution of a task or a supervisor component.

If the interruption occurred in the supervisor state, the Interrupt Stacker returns to the point of interruption using the old PSW, so that the interrupted supervisor routine can complete the work it began. For all problem-state interruptions, the Interrupt Stacker builds a record, called a general queue entry (GQE), to contain information describing the interruption.

General Queue Entry (GQE)

The general queue entry (GQE) is built in a 64-byte block obtained by a call to the Supervisor Core Allocation subroutine. It contains a description of the work to be done by a device or facility controlled by the resident supervisor. Contents generally include:

• Pointers to:

- 1) The task status index (TSI).
- 2) Preceding and succeeding GQEs on the same queue.
- 3) A page control block (PCB) if paging is involved.

• GQE movement information.

• Flags.

• Interruption code.

GQEs are attached to the appropriate interruption processor's queue, and are subsequently processed in a logical order, on a first-in-first-out basis within each queue. The queues themselves are processed on a priority basis (discussed under "Queue Scanning"). For SVC and program interruptions, control is transferred directly to the SVC or program interrupt processor to ensure fast processing of these interruptions which occur quite frequently. I/O and timer interruptions are queued on a table, called the scan table, which is private to the Queue Scanner and determines the order in which the queues are processed. Once classification and queuing is completed, the Queue Scanner receives control.

Queue Scanning

It is the responsibility of the Queue Scanner to provide a sequencing mechanism that decides the order in which individual queue processors are permitted to execute. To accomplish this, the Queue Scanner uses two tables: the scan table (SCANT) and the scan table master control table (SMC). The scan table is used in the processing of all I/O and external interruptions; program and SVC interruptions (with a few exceptions) are sent directly to the appropriate queue processor by the Interrupt Stacker.

Scan Table (SCANT)

The scan table, residing in main storage, contains one 16-byte entry for each I/O device or supervisor facility. Four-byte fields within each entry relate the supervisor queue processors to their facilities. A processor pointer field points to a unique processor for each entry, except for I/O device entries. Since only one I/O device queue processor exists in the supervisor, all device processor scan table entries point to the same processor program. The order in which device entries appear in the scan table, hence their priority, is specified from the symbolic device address (SDA) assigned to each device during system generation.

Table 2 lists the queue processors that work off the scan table in their order of priority.

Scan Table Master Control Table (SMC)

The information maintained in the scan table master control table (SMC) facilitates the Queue Scanner's search of the scan table. SMC comprises a set of device interaction groups (DIG). A DIG is a subset of entries in the scan table containing either one queue processor or a group of I/O devices having a common device controller (control unit). The SMC header contains a master count of the GQEs awaiting an available processor. This count is incremented or decremented as GQEs are added to or removed from scan table queues. The header also contains a count of DIG entries and a master count of matched facilities. Matched facilities are a processor, that is neither locked nor suppressed, and its queue of one or more GQEs. The DIG fields also include a DIG busy flag.

Queue Scanner Functions

When the Queue Scanner receives control, it inspects individual queues within a DIG only if the master count indicates that there is work queued within the DIG, and then only if other flags indicate that the appropriate queue processor is not busy and an I/O path to the device is available (see "Pathfinding"). To prevent one active device in a group from monopolizing an I/O

Table 2. Queue Processors that work off the scan table in their order of priority

Timer Interrupt Queue Processor (CEAKT)
Page Drum Queue Processor (CEAA8) - Page Drum Interrupt Queue Processor (CEAA9)
Auxiliary Storage Allocation Queue Processor (CEAIA)
User Core Allocation Queue Processor (CEANB)
Channel Interrupt Queue Processor (CEAA4)
I/O Device Queue Processor (CEAA3)
.
.
.
.
.
.
one queue for each device on the system
.
Pageout Service Subroutine (CEAA1)
I/O Call Subroutine (CEAA0)
Program Interrupt Queue Processor (CEANA)
Contiguous Core Allocation Queue Processor (CEANF)

path and greatly delaying the processing of other requests within the DIG, the Queue Scanner processes the queues within each DIG in a round-robin order.

When it finds a DIG with processable work queued, the Queue Scanner locates the queue entry in the scan table and transfers control to its associated processor(s). The queue control subroutines queue and dequeue entries, and move entries from queue to queue until all processing specified in the GQE is accomplished. These subroutines maintain control fields in SMC and protect the scan table and the queues by setting suppress flags (see Figure 6).

The queue processors locate the GQEs pointed to by the queue entries, analyze the processing requirements specified in the GQEs, and set up the necessary storage space, tables, controls, and subroutine linkages to effect the processing. When all queues are empty, or when the necessary processors are busy, the Queue Scanner transfers control to the Internal Scheduler to select the next task to be put in execution.

Queue Processing

Associated with the Queue Scanner are four groups of specialized queue processors: timer, drum paging, I/O, and storage allocation queue processors. The processors perform detailed checks on conditions reflected by the GQE fields and determine the appropriate action to be taken to process the GQE. If the processing requires

the attention of several processors, the GQE is transferred from one processor's queue to the next through the services of one of the queue control subroutines (see Figure 6).

The queue control subroutines examine the first routing field in a GQE. This field will either contain a location-on-queue (Loc-on-Q) value or all ones. The Loc-on-Q value designates the relative location on the scan table of the queue to which the GQE is to be transferred (see Table 2). A value of all ones indicates that no further processing is to be performed for the GQE, and the main storage occupied can be released.

In general, a queue processor locks its associated queue upon entry and unlocks it as soon as the processor has dequeued a GQE from the queue for processing. In certain cases a queue processor may wish to lock a queue until some specific future event or condition has occurred. Indicators, called suppress flags, contained in each scan table entry are set and reset by the Set Suppress Flag subroutine to prevent unwanted recursion.

Timer Interruption Processing

The GQE for a timer interruption is placed on the Timer Interrupt Queue Processor (TIP) queue in the scan table. The interruption can be the result of a task having:

- Reached normal time-slice end.
- Been forced to time-slice end.
- Been selected to have its pages migrated from auxiliary drum to auxiliary disk storage (see "Auxiliary Storage Allocation").

The latter two are the more usual reasons since tasks in TSS/360 seldom reach normal time-slice end, but are more often forced to time-slice end to satisfy a variety of conditions.

A user timer field in the task's XTISI contains the length of time the task is to execute during one time slice. At each forced or normal time-slice end, the task's timers are decremented. When the user timer field goes to zero, and it's a normal time-slice end, a task interruption is created and queued on the task's TSI interruption queue, the task's pages are left in main storage, and the task remains in the dispatchable list (for eventual dispatching to the task monitor for processing). In all other situations (that is, when a user timer interruption has not occurred or the task has been forced to time-slice end),

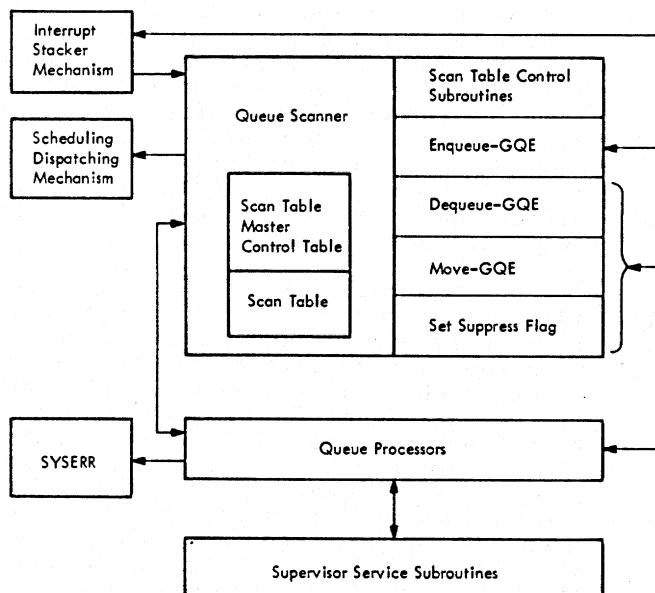


Figure 6. Queue Scanning and processing module interface

the task is rescheduled according to its scheduling parameters, and its pages are written from main storage.

An exception to this is caused by the issuance of a TSEND SVC (a request by a task to delay execution until some event has occurred), which results in the task being forced to time-slice end. In this case the task is set in delay status and placed on the inactive list, its pages are written from main storage, and a timer interruption is set up. Once the event has occurred, the timer interruption is processed, causing the Rescheduling subroutine to be called to place the task on the eligible list with its time recomputed.

When a timer interruption occurs as a result of the completion of user I/O for a page, TIP scans the task's page tables (see "Program Interruption") for a page that is available. If the page is unchanged, Supervisor Core Release is called to release its main storage space; changed pages must be written to auxiliary storage and the auxiliary storage space previously occupied by the page must be released.

When a task has been selected to have some of its pages migrated from auxiliary drum to auxiliary disk storage (see "Auxiliary Storage Allocation"), a timer interruption occurs. TIP determines which of the task's pages are to be migrated and, in the case of private pages, the GQE is queued on the Page Drum Queue Processor queue for processing. If shared pages are to be migrated, the Write Shared Pages subroutine provides the pages to migrate.

Segment and Page Tables

Each task keeps track of the location of its pages in storage by means of segment and page tables. Within each task's XTSTI there is a segment table (SGT), consisting of groups of four-byte entries. Each entry contains a pointer to the beginning of a page table (PGT), the count of the number of entries in that page table, and its availability. Each entry in the page table points to the location of a page in storage. Immediately following each segment table is an auxiliary segment table (AST) containing pointers to page tables on auxiliary storage; an external page table (XPT) immediately following each page table points to pages not in main storage.

A list of the location of all shared page tables (SPT) currently in the system is maintained in the resident shared page index (RSPI). The RSPI, permanently resident in main storage, indicates the main storage location (if available), the in-transit state, and the length of shared page tables.

The relationship of these tables to each other is illustrated in Figure 7. Table 3 provides a brief summary of the functions of paging and storage allocation control blocks referred to by the supervisor. A more detailed description of these control blocks is contained in System Control Blocks.

Program Interruption Processing

Of the 17 program interruption codes generated by System/360 Model 67, only codes 16 and 17 are processed by the resident supervisor. When a program interruption with a code of 0-15 occurs in the problem state, the Interrupt Stacker queues the interruption GQE on the task's TSI; before the task is next given CPU control by the Dispatcher, the Task Interrupt Control (TIC) subroutine arranges for the interruption to be processed by the task monitor.

Program interruption code 16 is a segment relocation exception, indicating that a task's page table (see "Segment and Page Tables") or shared page table is unavailable. Program interruption code 17 is a page relocation exception, indicating that a page is unavailable (that is, not in main storage). The Interrupt Stacker links directly to the Program Interrupt Queue Processor (PIP) to provide faster service for these interruptions which occur frequently.

When a segment relocation exception (code 16) occurs, it is first determined whether it is a shared page table that is unavailable. Since all of a task's private pages must be in main storage during execution, if it is not a shared page table, a system error is indicated. For shared page tables, a search is made of the resident shared page index (RSPI) to determine the location of the shared page in storage. When the table is found, the task is again returned to a "ready" status. When the page table is not available in RSPI, the task is put in "page-wait" status. In either case, control is returned to the Queue Scanner.

For page relocation exceptions (code 17), the number of page reads that have been performed for the task is compared with the maximum number permitted (a parameter in the task's schedule table entry). If they are equal (the maximum number has been reached), the GQE is queued on the Timer Interrupt Queue processor's queue and the task is forced to time-slice end. When the maximum number of page reads has not been reached, and the page is a shared page that is in transit, the GQE is added to the external shared page table (XSPT) queue and the task is placed in a page-wait status.

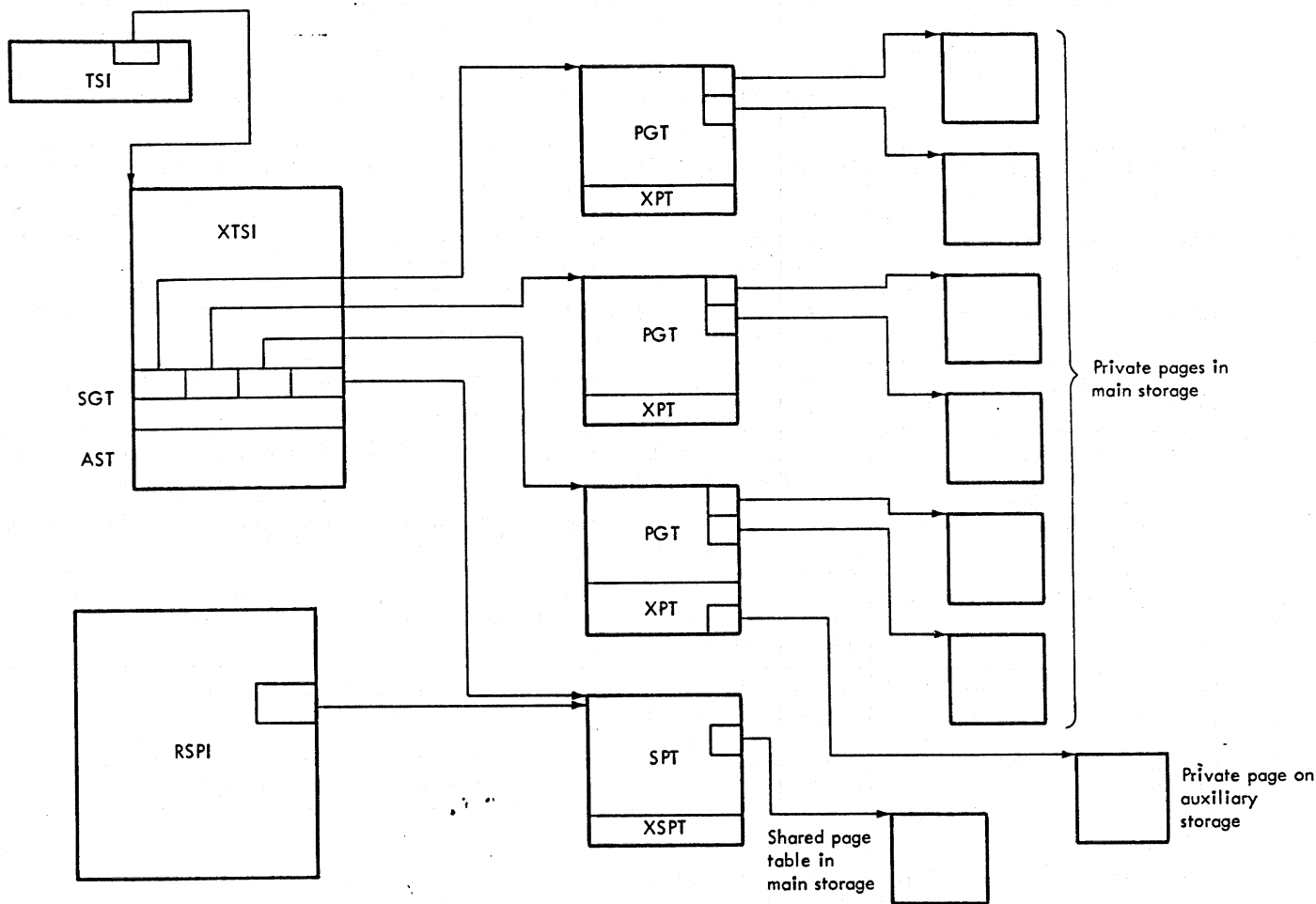


Figure 7. Page table relationship

In all other cases, the necessary steps are taken for the page to be read in: the Supervisor Core Allocation (SCA) subroutine is called to obtain 64 bytes in which to build a page control block (PCB), which is then linked to the GQE; the GQE is queued on the User Core Allocation (UCA) processor's queue; and UCA searches the core block table (CBT) for a page of main storage into which the page can be read (see "User Core Allocation"). The actual paging in will be performed as a result of an I/O interruption that will be directed to the appropriate paging queue processor (drum or disk) to bring in the page.

I/O Interruption Processing

I/O interruptions are initially separated into two main types - those requesting drum paging operations and all others. Interruptions involving drum paging are queued on the Page Drum Interrupt Processor's queue; all others are queued on

the Channel Interrupt Processor's (CIP) queue.

When control is transferred to CIP, the GQE for the interruption is examined to further determine the type of action required:

- I/O operations for the terminals of conversational tasks are passed on to the Terminal Communications Subprocessor for handling.
- The freeing of devices, channels, and/or control units is performed, when specified by the request GQE, by a call to Reverse Pathfinding.
- Synchronous and asynchronous interruptions from remote job entry (RJE) devices are transferred to the respective remote job entry processors for handling.

Table 3. Paging and storage allocation control blocks

Control Block	Function
Segment Table (SGT)	A contiguous list of entries, residing in a task's XTSI, which contain the length, origin, and availability of the task's page tables.
Auxiliary Segment Table (AST)	Immediately follows the SGT and contains information concerning page tables on auxiliary storage.
Page Table (PGT)	A contiguous list of entries containing address and availability status of task's pages in main storage.
External Page Table (XPT)	Immediately follows the PGT and contains information concerning task's pages on auxiliary storage.
Shared Page Table (SPT)	Identical to page table, the SPT contains a list of address of shared pages.
External Shared Page Table (XSPT)	Immediately follows the SPT and contains control information required for paging of shared virtual storage pages.
Resident Shared Page Index (RSPI)	Contains status and control information needed to maintain the system's currently active shared page tables.
Direct Access Interface Block (DAIB)	Contains interface data required for passing pages to or from core storage; a new DAIB is constructed for each paging operation.
Auxiliary Storage Allocation Table (ASAT)	Contains the availability status of all auxiliary storage devices, both drum and disk.
Core Block Table (CBT)	Maintains a list of main storage blocks (pages) and their status as available or unavailable for assignment.

- Preliminary processing is provided for paging interruptions from direct access devices other than drums.
- A distinction is made between initial and subsequent asynchronous interruptions so that a task can be initiated, when necessary, and affected tasks can be kept informed of the occurrence of interruptions.
- Special processing is performed when requested in the IORCB.

Pathfinding

The symbolic device address (SDA) of a device specifies the relative position of the device's queue entry on the scan table. The translation of this symbolic address into a specific hardware address is performed by the Pathfinding subroutine by finding a path to the device. A path comprises three components: the channel, the control unit, and the device. Pathfinding involves a search through the device group table, the channel table, and the control unit table, in that order, to define a path. Reverse Pathfinding is called to perform the opposite function, that is, to

translate the actual address of the device to its SDA.

The Channel Interrupt Processor calls Reverse Pathfinding to determine the SDA, and thus the entry for that device on the scan table. When the entry contains a pointer to a GQE representing a request for I/O to or from that device, it is the task of the I/O Device Queue Processor to handle that request for all devices on the system except paging drums.

Paging

The paging queue processors manage GQE requests for page movement between main storage and drum or disk storage. A paging operation can be caused by the occurrence of any of the following events:

- A page relocation exception interruption caused when a task attempts to refer to a page not currently in main storage.
- The first XTSI page, which contains the PSW and relocation tables for a task to be dispatched, is not in main storage.

- A request is made for one or more data set pages to be written out to external storage.
- Pages destined for external storage are currently residing on auxiliary storage and must be read into main storage before being written out to external storage.
- One or more buffer pages for an I/O operation are on auxiliary storage and must be brought into main storage for the duration of the I/O operation.
- A time-slice-end interruption has been received.
- The Write Shared Pages subroutine is invoked by User Core Allocation to page out all changed shared pages that have not been referred to since the last time the subroutine was invoked.

When the work associated with a GQE is a paging operation, one or more additional control blocks, called paging control blocks (PCB), are constructed and linked to the parent GQE. Each PCB can contain up to three page control block entries (PCBE), each representing a request to move one page.

Disk Paging

When the I/O Device Queue Processor is given control by the Queue Scanner, the GQE on the device queue is checked for an I/O request control block (IORCB). The absence of an IORCB is interpreted as a request for disk paging. Pathfinding is called to obtain an available channel and control unit to the device specified; then the GQE is transferred to the Page Direct Access Queue Processor (PDAQ).

PDAQ builds a Direct Access Interface Block (DAIB), which provides the interface between it and the Page Direct Access Interrupt Processor (PDAI), providing the latter routine with the channel programs and their related PCBs. PDAQ then builds the channel program and puts the necessary information into the DAIB for that interruption, and calls Start I/O to initiate I/O.

When a paging operation is completed, a device end I/O interruption occurs. PDAI receives control and inspects the DAIB for the next channel program to execute Start I/O. When all channel programs in the DAIB have been processed, control is returned to the Queue Scanner.

Drum Paging

Drum storage is used in TSS/360 whenever possible, since the drum is the fastest

auxiliary device on the system. To maximize drum throughput, a process called slot sorting is used, which depends on the following organization of the drum. The drum records are arranged in page format and stored on a pair of adjacent tracks, with 4 1/2 pages (or records) on each track. The record-overflow feature is utilized between the even and odd tracks.

Drum addresses for page storage are allocated to fill as many consecutive drum slots as possible. To each slot, one page may be assigned. There are nine slots for each two tracks on the drum. If the last slot of the first pair of tracks has been allocated, the next address allocated is from the first slot of the second pair of tracks. The availability of drum storage is reflected in a directory in the auxiliary storage allocation table (ASAT).

Drum storage is allocated in such a way that pages are assigned by slot number in cyclic order. A drum access request indicates whether the operation is read or write, and gives a slot number for the page to be accessed. The channel program is constructed so that requests are selected by slot number in cyclic order from the queue of drum paging requests. The Page Drum Queue Processor performs the slot sorting and constructs channel programs.

Each drum has associated with it two chains of nine channel programs each, one channel program for each slot on the drum. These chains are anchored in a work area of the system table called a drum interface control block (DICB). The DICB contains information describing the status of the drum as well as pointers to the PCBES, IORCBs, and associated GQEs. The DICB is accessed by both drum paging processors, the Page Drum Queue Processor (PDQP) and the Page Drum Interrupt Processor (PDIP).

When PDQP finds a PCBE for which a channel program can be built (that is, a slot is available), the channel program is built in the DICB area of the system table, using the slot number as a pointer to the proper program. When all available slots have been filled, or no work remains to be done, control is returned to the Queue Scanner. The Page Drum Interrupt Processor is activated when channel end, device end, unit check, or a program controlled interruption (PCI) is received. Processed pages are posted, storage released, and the interrupt GQE dequeued.

Main Storage Allocation

The allocation and release of pages of main storage is recorded in the core block table (CBT), which contains one entry for each page of main storage in the system. Pages that are available for assignment are

kept on an unassigned chain that is updated after each allocation and release. A set of these pages (the number will vary from installation to installation) is kept as a reserve list for satisfying resident supervisor requirements for main storage (for GQEs, TSIs, and PCBs). Since these supervisor requirements must be satisfied for operation to continue, these pages must be held available exclusively for supervisor use; all other pages are available to user tasks.

User Core Allocation

Requests for user main storage are represented by one or more PCBs chained to a GQE; they are processed by the User Core Allocation (UCA) queue processor. A request may be to reclaim a specific page that was previously assigned to the task, or for storage not previously owned by the task. A request for previously-owned storage involves a comparison of the CBT entry for that storage block with the request PCB entry to determine whether the page is still available or has been assigned to another task. If the page is available (that is, the page was written out but its main storage location was not needed in the interim), it is removed from the unassigned chain and assigned to the task, avoiding the necessity of reading a page in from drum or disk. When a new page of storage must be assigned, the first available page of unassigned main storage is allocated to it.

When no page is available for assignment, or when a check made prior to allocation indicates a low core condition, an attempt is made by the Write Shared Pages subroutine to release shared pages (write them out to auxiliary storage). If this fails to provide sufficient main storage, UCA selects a task to be forced to time-slice end and its pages released. Whenever a page is released, it is put on the unassigned chain, unless the reserve list has decreased to less than the required number of pages, in which case the page or pages needed are transferred to the reserve list.

When a task has reached the maximum number of pages in main storage allowed during one time slice (a value contained in a parameter of his schedule table entry), page stealing may be performed; this means that the task frees main storage by having some of its own pages released. Page stealing can be performed only on pages that are not: XTSI, PSW, ISA, or previously referenced pages, in transit, or in I/O or SVC hold. Before being released, the page will be written on drum if it is a changed page or no old copy exists. A certain percentage of the task's maximum pages must be retained during stealing (another STE para-

meter). The algorithm used for page stealing is explained in the description of the User Core Allocation processor in Section 3.

Supervisor Core Allocation

Allocation of main storage for use by supervisor components is processed by the Supervisor Core Allocation (SCA) subroutine; storage is obtained from the reserve list. Pages in this list must be unfragmented. Since supervisor storage is allocated in 64-byte blocks, once a block has been allocated from one of its pages, the page is removed from the reserve list, divided into 64-byte blocks, and placed in one of three chains of partially allocated pages:

- One-block chain - for filling single block requests.
- Three-block chain - used only when three contiguous blocks are needed to build a TSI.
- Miscellaneous chain - for filling all other requests.

The first block of each fragmented page contains an available block counter and a bit map indicating which 64-byte blocks within the page are available. Six pointers are maintained in SCA, two for each chain; the first pointing to the page, the second to the block in the page to be checked.

To speed up the allocation of a single block of storage (the most common type of request), and reduce the number of pages that must be fragmented, six "quick cells" are maintained that point to the most recently returned single blocks. These are searched first when a single block request is received.

When the request is for contiguous blocks, the bit maps of those pages whose available block counters indicate a good probability that the request can be satisfied are searched before the bit maps of other pages.

The reserve list is automatically replenished when necessary with pages from the unassigned chain in the CBT. Therefore, when a request for supervisor storage cannot be satisfied, an in-use page is borrowed from a task in user main storage, the assumption being that the unassigned chain in CBT is also empty.

When storage is released, single blocks are returned to quick cells, if they are not full; otherwise, they are returned to fragmented pages. When the return of a

block to a fragmented page causes it to be composed entirely of available blocks, it is returned for general system use by the User Core Release subroutine.

Auxiliary Storage Allocation

Auxiliary storage consists of the disks and drums on which a task's pages are stored when not in execution. It is confined to drum when possible, since the drum is the fastest auxiliary device on the system. The Auxiliary Storage Allocation Table (ASAT) contains a bit directory for each auxiliary device on the system, with each bit representing one page. The Auxiliary Storage Allocation Queue Processor maintains a count of auxiliary storage in use at all times for the entire system. This count is updated and checked each time auxiliary storage is allocated to a task.

Auxiliary storage is obtained when it is necessary to write a page out to disk or drum. These requests are assigned to drum except when: the request specified a drum preference and no drum storage is available; or no preference was specified and drum space has reached the system minimum. When this minimum is reached (a value contained in ASAT), a task is selected for migration; that is, some of the task's pages are moved from drum to disk, freeing drum space. The task selected for migration is the task on the inactive list (or, if necessary, the active list) with the most pages on drum in excess of its fair share. Migration can be performed on either private or shared pages. The Timer Interrupt Queue Processor selects private pages for migration; the Write Shared Pages subroutine performs this function when shared pages are involved.

When a task enters the system, its auxiliary storage requirements are compared with the available auxiliary storage count. The task is not allowed on the system if there is not sufficient auxiliary storage available. If the task should exceed its limit of auxiliary storage during execution, and available storage is less than the installation minimum, the task is first warned, if conversational, and then terminated; nonconversational tasks are terminated at once.

If there is more than one drum on the system, the drum with the largest number of available pages is used. Disk pages are allocated from the same cylinder, when possible. Once storage for a GQE has been assigned, the Auxiliary Storage Allocation Queue Processor sorts the PCBES by device type. If they have all been assigned to the same device, the GQE is queued on the queue of that device in the scan table.

When allocation has been made from different devices, a new GQE and PCB must be created for each device addressed before queuing can take place.

SVC Interruption Processing

The supervisor call (SVC) is the normal method of communication between a task and the supervisor. The interruption is caused by the execution of an SVC instruction which is usually embedded in the expansion of a macro instruction. When the interruption occurs in the problem state, the SVC code is examined to determine the type of request:

- SVC codes 0-63 - a request for problem program services.
- SVC codes 64-95 - a request for Time Sharing Support System (TSSS) services.
- SVC codes 96-127 - a request for privileged program services.
- SVC codes 128-255 - a request for resident supervisor services.

Appendix B contains a list of SVC instructions, related macro instructions, and processing routines.

The resident supervisor services only SVC codes 128 through 255. The processors responsible for servicing these requests are the SVC Queue Processor and a group of SVC subprocessors. These subprocessors can be divided into functional groups according to the general type of service they provide:

- Virtual storage processors.
- TSI/XTSI modification and extraction processors.
- Timer-maintenance and task-synchronization processors.
- System table modification and extraction processors.
- I/O and device management processors.
- Inter-task communication processor.

The individual SVC processing routines included within each of these functional groups are described in Section 3 under their respective functional headings.

The primary function of SVCs is to cause a switch from the problem state to the supervisor state, allowing the execution of privileged instructions and procedures in a non-time-sliced environment. When the SVC

interruption is received, the Interrupt Stacker invokes the SVC Queue Processor by direct linkage (the GQE is not queued on the scan table). Since not all classes of users may issue all types of SVC requests, it is one function of the SVC Queue Processor to verify that a user has the authority to ask for the specified service. Once it has been determined that the SVC was issued from a routine with the proper authority, the SVC Queue Processor uses the SVC interruption code to invoke the appropriate SVC subprocessor (see Figure 8).

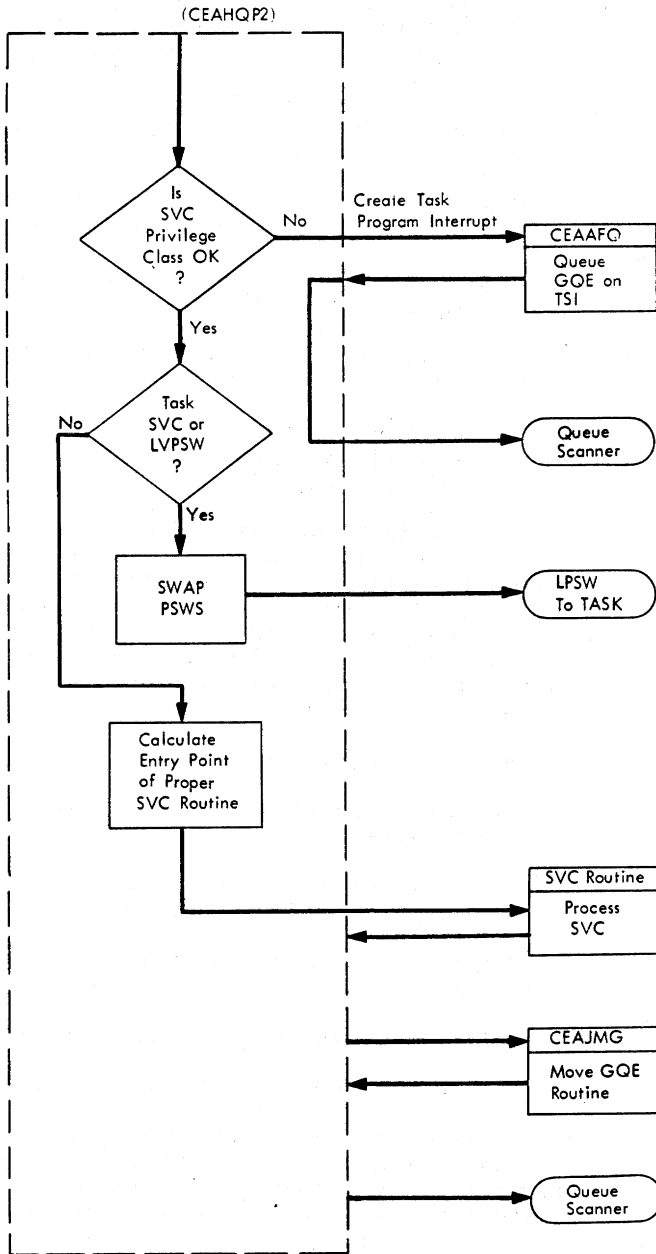


Figure 8. Activities of the SVC Queue Processor

It is possible for the Interrupt Stacker to receive an SVC interruption while the CPU is in the supervisor state (that is, the SVC instruction was executed by the resident supervisor itself). This indicates an error and causes the Interrupt Stacker to issue an ERROR SVC to invoke the System Error Processor. The processing of such an interruption is discussed in Section 3 under "Major Error Recovery Procedures."

Task Scheduling and Selection

The task scheduling and selection mechanism of the resident supervisor controls the order in which tasks are assigned CPU time, and the length of time they are allowed to execute. On entry to the system, each task is given a set of scheduling characteristics, in the form of an entry level in the schedule table (CHASTE), which determines the order in which it will be allowed to execute. These scheduling characteristics will change as the task's entry level is changed at various stages of execution.

Schedule Table (CHASTE)

The schedule table comprises a maximum of 256 entries, beginning with entry zero. Each entry contains a set of 24 parameters that become the scheduling characteristics of any task assigned to it. All tasks are assigned a schedule table entry (STE) when they enter the system. For conversational tasks, the initial levels used are 0 through 9; for nonconversational tasks, entries 10 through 19 are used after logon is completed; level 20 is used for the logon procedure, and level 21 for logoff. Operating conditions within the time-sharing environment, and their relationship to the task's service requirements, will cause a task's entry level to be changed during its life in the system. Table 4 identifies the 24 parameter fields in a schedule table entry and specifies the length of each and its meaning.

Active and Inactive Lists

All TSIs in the system are chained together on one of two lists, the active list or the inactive list. The active list is further subdivided into the eligible list and the dispatchable list.

- Eligible list - Tasks in this list are ready to execute, but have not yet been brought into main storage. They are ordered by internal priority (specified by a field in the STE), with the lowest priority number first on the list. Tasks with the same priority number are further ordered by their scheduled start time (SST). SST is a time value

computed when a task enters the eligible list; it is related to a master clock to determine whether a task is ahead of or behind schedule.

- Dispatchable list - This list consists of tasks that are in main storage attempting to compete for CPU time and, in most cases, whose SST is less than the master clock. A task whose SST is less than the master clock is said to be behind schedule. Tasks in this list are ordered according to their status as "execute bound" or "I/O bound." Those with heavy paging requirements (I/O bound) are dispatched first.
- Inactive list - These tasks are in AWAIT or TWAIT status, or have issued a TSEND SVC. Their pages are not in main storage and they are incapable of continuing execution until a particular interruption occurs.

Task Selection

When a task first enters the system, it is assigned a schedule table entry (STE) level, and placed in the eligible list on the basis of its priority (as shown in its STE) and its SST. When it is determined that a task should be moved from the eligible list to the dispatchable list, the highest priority task that is farthest behind schedule (SST less than the master clock) is selected. It is then moved to the dispatchable list only if it meets the requirements checked by the Entrance Criteria subroutine.

Tasks in the dispatchable list are ordered so that those with heavy paging requirements are dispatched first. When a task is selected from the dispatchable list to be given CPU control, the Dispatcher first checks, via the Task Interrupt Control (TIC) subroutine, to see if any enabled interruptions are pending for the task. If an interruption is pending, TIC

arranges for it to be serviced by the task monitor. The Dispatcher also checks to see if a real time interruption is to be created for the task; if so, it is handled before the task is dispatched.

Once a task is dispatched, it is allowed to run until its timer interval runs out (normal time-slice end), or until it is forced to time-slice end (see "Timer Interruption Processing").

Task Scheduling

When normal time-slice end occurs, the quanta count field in the task's TSI is decremented by one. This field is initialized to the value in the quanta count field in the task's STE. If the count has not reached zero, the task is given another quantum of CPU time; it is left on the dispatchable list, and no change is made in its STE or SST. If the count does reach zero, the Rescheduling routine is called to change the STE level to the level indicated in the TSE level field of the old STE, and to recompute the SST. If it is determined, at the end of a quantum, that a task has exceeded the maximum number of page relocations permitted per quantum, the task is moved to the top of the dispatchable list (it is "paging bound").

For TSEND, TWAIT, and AWAIT extension, tasks are removed from the active list and placed on the inactive list. A real time interruption is created for TSEND tasks forced to time-slice end, and they remain on the inactive list until the interruption occurs. At that time, Rescheduling is called to compute their SST, and they are returned to the eligible list. In AWAIT and TWAIT situations, a new STE level is assigned to the task as specified in the AWAIT and TWAIT level fields of the STE. This new set of scheduling parameters will control each task's movement through the eligible list to the dispatchable list when the AWAIT or TWAIT interruption occurs.

Table 4. Schedule table entry parameters (part 1 of 2)

Field Identification	Field Length	Meaning
Level (STELEVEL)	1 byte	The relative entry number of this entry (0-255)
Priority (STEPRIOR)	1 byte	Determines which tasks take precedence in having CPU resources allocated to them. The order of precedence is low numbers first.
Quantum Length (STETSVAL)	2 bytes	The amount of time to be used as a factor in determining how long a task will be allowed to run before time slice end (TSE). One unit=3.33 milliseconds.
Quanta Count (STEQUANT)	1 byte	Specifies the number of quanta a task is to receive when it is placed in execution before time slice end occurs.
Maximum Pages (STEMAXCR)	1 byte	Specifies the maximum number of pages allowed in main storage for a task during a complete time slice.
Maximum Page Reads (STEMAXRD)	2 bytes	Specifies the maximum number of page relocations (including XTSI and ISA pages) allowed for a task during a complete time slice.
Scan Threshold (STEST)	1 byte	When a task's pages in core exceed the limit (STEMAXCR) and the Steal request flag (STESRI), some of the task's pages will be released. This field specifies a percentage (in hex) of STEMAXCR as the number of pages to be retained for the task when stealing occurs. The task is not forced to time slice end, but its schedule table entry level is changed to the value specified in STENSL.
Pulse Level Entry (STEPULSE)	1 byte	Specifies the schedule table entry to be assigned to a task in response to the Pulse SVC.
AWAIT Extension (STEAWTEX)	2 bytes	Specifies the duration of time, in milliseconds, a task is to remain in the dispatchable list in the AWAIT state.
Delta to Run (STEDELTA)	1 byte	Specifies the real time interval at which a task is to be given a slice of CPU time.
TSE Level (STETSEND)	1 byte	Specifies the schedule table entry to be used when time slice end occurs.
Maximum Page Reads Exceeded Level (STEMPRE)	1 byte	Specifies the schedule table entry to be used when time slice end is forced because of maximum page reads being exceeded.
AWAIT Level (STEAWAIT)	1 byte	Specifies the schedule table entry to be used when a task leaves AWAIT status.
TWAIT Level (STETWAIT)	1 byte	Specifies the schedule table entry to be used when a task leaves TWAIT status.
Flag Byte (STEFLAGS)	1 byte	If the byte value is X'80' for a task being moved from the inactive list, the task's scheduled start time is recomputed to place it on schedule. Otherwise, the task remains on the same relative schedule it was on when it entered the active list.
X'80'=Recompute (STERCMPM)		When the recompute flag is off, past performance (if the task is behind schedule) is taken into account by calculating SST as the present time plus the delta-to-run less the amount behind schedule on the previous time slice.

Table 4. Schedule table entry parameters (part 2 of 2)

Field Identification	Field Length	Meaning
X'40'=Preempt (STEPRMPT)		If the byte value is X'40' for a task in the dispatchable list, and a behind schedule task of higher priority resides in the eligible list, the task in the dispatchable list can be preempted by forcing it prematurely to time slice end.
X'20'=Steal Request (STESRI)		If the byte value is X'20' for a task in the dispatchable list, whose private pages in main storage exceed the maximum limit, some of the pages will be stolen (released) from the task.
X'10'=Subtract delta to run (STESDTR)		If the byte value is X'10', STEDELTA should be subtracted from, rather than added to, the master clock in calculating the scheduled start time for the task.
Maximum Page Relocations per Quantum (STEMRQ)	1 byte	Specifies the maximum number of page relocation interruptions allowed per quantum before the task is declared paging bound.
Holding Interlock Change Level (STELCK)	1 byte	Specifies the schedule table entry to be assigned to a task at time slice end when the task is holding an interlock.
Low Core/Holding Interlock Level (STELCHL)	1 byte	Specifies the schedule table entry to be assigned to a task at time slice end when the low core condition exists and the task is holding an interlock.
Waiting on Interlock Change Level (STEWLCK)	1 byte	Specifies the schedule table entry to be assigned to a task at time slice end when the task is waiting on an interlock.
STECWO	1 byte	Specifies the schedule table entry level to be assigned to a task by the WAIT SVC processing routine when a write only operation is indicated in the terminal control table.
STELCF	1 byte	Specifies the schedule table entry to be assigned to a task by the rescheduling subroutine when a task's low core flag is on and none of the other schedule table entry level exit conditions apply. User core allocation will set the flag, TSIILCF, on when it is forcing an active task to time slice end and the low core condition exists.
STEPRJ3	1 byte	Specifies the maximum amount of time a task can be behind schedule before it will be submitted to Entrance Criteria before higher-priority tasks which have not exceeded this maximum.
STENSL	1 byte	Specifies the schedule table entry level to be assigned to a task by the User Core Allocation routine when it is determined that page stealing is to be initiated.
Drum Share (STEDSH)	2 bytes	Specifies the number of drum pages reserved for a task when more than the system calculated minimum drum space can be allocated. If the byte value is 0000, default is to the system calculated minimum.

SECTION 3: PROGRAM ORGANIZATION

INTERRUPTION CLASSIFICATION

Interrupt Stacker Module (CEAJI) Chart AA

The Interrupt Stacker comprises four subdivisions which provide the processing necessary to service all interruptions other than machine check interruptions. (Note: Machine check interruptions are serviced by the Recovery Nucleus discussed later in this section.) When an interruption occurs, the new PSW corresponding to the class of interruption is loaded and control is given to the appropriate subdivision of the interrupt stacker. In general, the function of all the subdivisions is to generate a general queue entry (GQE) and to queue it on the appropriate scan table entry or, in the case of SVC and program interruptions, to transfer control to the appropriate interrupt processor. Figure 9

presents a general flow diagram of the interrupt stacker mechanism and its relationship to other supervisor components.

Attributes: The interrupt stackers are parallel reenterable, resident, and operate in the privileged state with all interruptions except machine check disabled.

Entries: The Interrupt Stacker module has an entry point unique to each stacker, as follows:

- Program Interrupt Stacker..... CEAJIP
- SVC Interrupt Stacker..... CEAJIS
- External Interrupt Stacker..... CEAJIE
- I/O Interrupt Stacker..... CEAJII

Modules Called: Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) reserves 64 bytes of main storage for a GQE

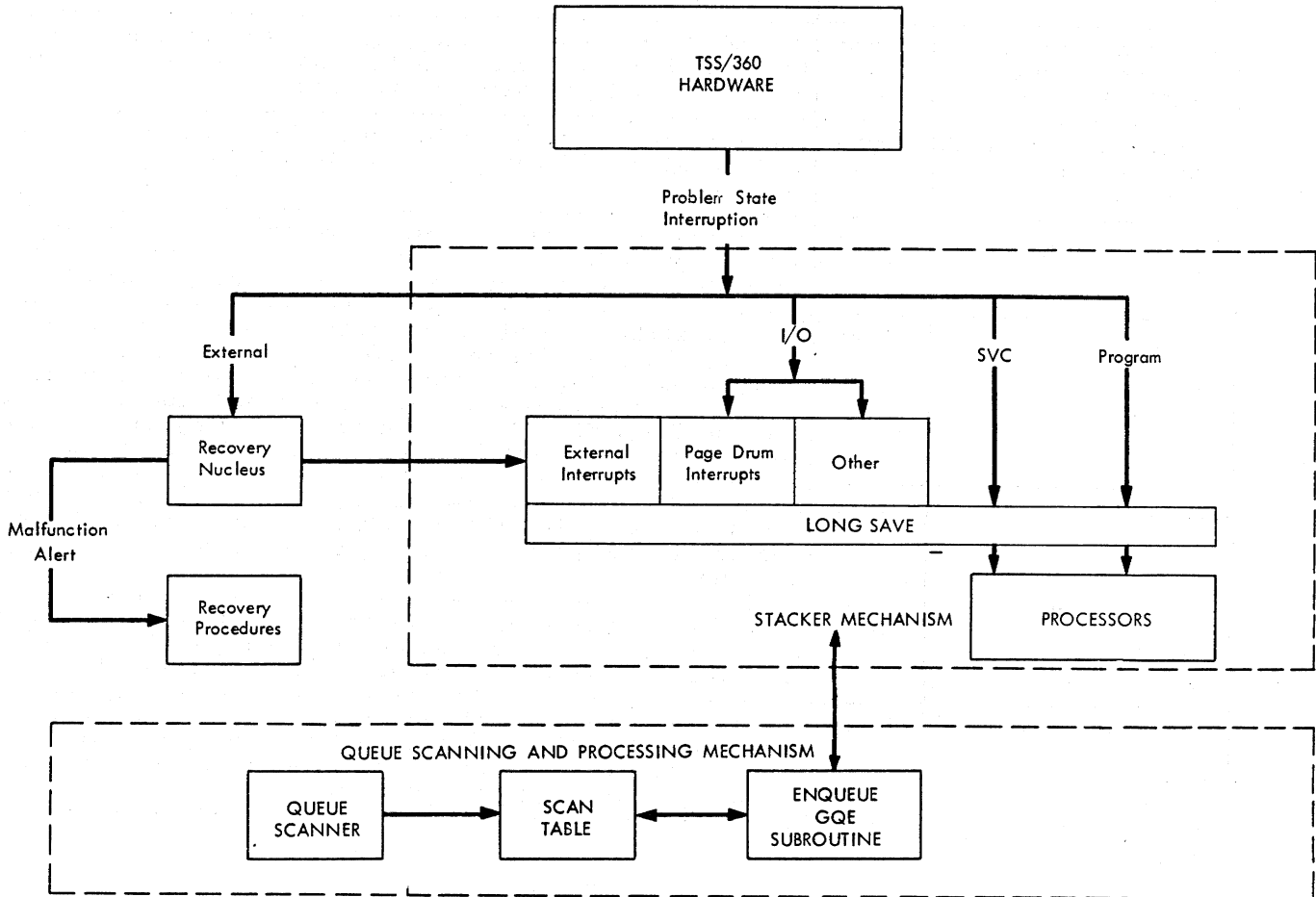


Figure 9. Interrupt stacker module overview

and 64 bytes for channel logout data when required.

Enqueue GQE subroutine (CEAJQ entered at CEAJEN) places GQEs on the appropriate processor queues.

Queue GQE on TSI (CEAAF entered at CEAAFQ) places a program interrupt GQE on a task's TSI interrupt queue when a program interrupt entry is found in the SERR auxiliary queue.

Exits: Normal - The interrupt stacker exits to the resident support system (RSS) whenever the occurring interruption is one to be handled by RSS.

Exit is to the Program Interrupt Processor (CEANA at CEANAA2) for handling of paging and segment relocation exceptions (PI codes 16 and 17).

Exit is to the SVC Queue Processor (CEAHQ at CEAHQQ) for program interrupt codes 1 through 15. This exit results in a program interrupt GQE being placed on the task's TSI (via a call to CEAAF) and exit to the Queue Scanner.

An exit to the SVC Queue Processor (at CEAHQP2) is also made to determine if the task causing an SVC interruption is authorized to issue the SVC and to call the proper SVC routine to provide the requested service.

Exit is to the Queue Scanner at CEAJQS after queuing timer and I/O interruptions on their appropriate processor queues.

Exit is made via LPSW (old I/O or old external) after queuing I/O or external interruptions occurring in the supervisor state.

Exit is to the user task via LPSW after swapping VPSWs in the ISA and PSA for task-oriented SVCs and when processing for an LVPSW SVC is completed.

Error - Exit is to the System Error Processor (CEAIS at CEAIS2) when the interruption is caused by a SYSER SVC issued by a virtual storage task.

Exit is to CEAIS1 when processing an ERROR SVC (SVC 254) for two conditions - LVPSW issued in problem state, or ERROR issued in supervisor state.

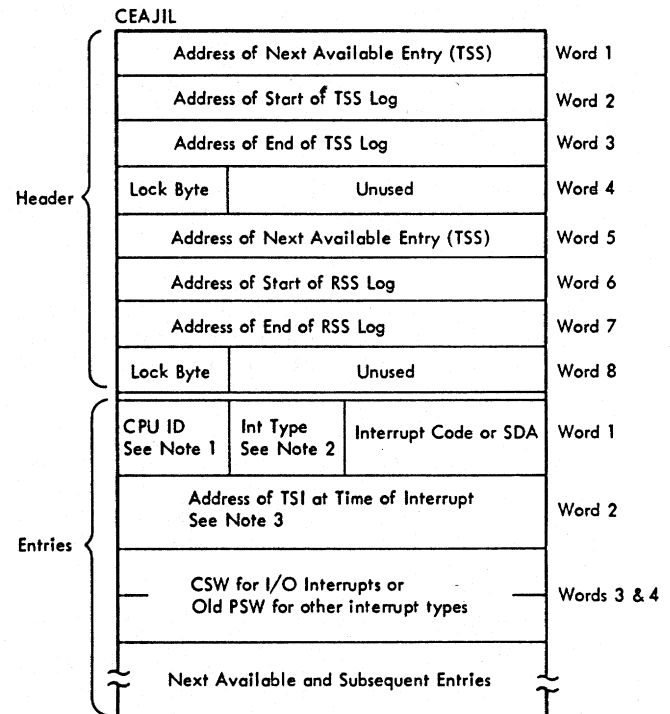
Operation: The initial processing at each of the four entry points to the Interrupt Stacker is similar. The status of the system is checked to see if the interruption occurred in problem state. If it did, the value contained in the timer (PSATIM) is saved in PSATSA. The timer is then set to

hex 0FFFFFFF to prevent a timer interruption from occurring in the supervisor state. The system elapsed time (PSAETM) is then computed and updated after setting the real time lock (SYSTMILK) in the system table.

If the system was in supervisor state at the time of interruption, none of the timer fields are changed.

After determining the system state and taking the appropriate action, the Interrupt Stacker fills in an entry in the interrupt log (CEAJIL) with data pertaining to the type of interruption just received. (For a description of the interrupt log, see Figure 10.)

The Interrupt Stacker then determines if the interrupt is one to be processed by the resident support system (RSS). If it is,



Note 1: CPU 1 = 80
CPU 2 = 40

Note 2: External = 18
SVC = 20
Program = 28
I/O = 38

Note 3: For an I/O Interrupt in Supervisor state

Word 2 = Byte 1: Byte 1 of Ext PSW
Bytes 2-4: Instruction address

For RSS Program and I/O interrupts:

Word 2 = A pointer to TSS LOG where the interrupt would have been recorded.

Figure 10. The interrupt log

it is passed to the RSS handling routine via an LPSW. Otherwise the SERR Auxiliary Queue Processing subroutine is called to check for and work off entries in the SERR auxiliary queue (CHASAQ). Entries in CHASAQ represent status information (from pending interrupt) on devices used by System Environment Recording and Retry.

The Long Save subroutine (IS01) is used in common by each of the four divisions of the Interrupt Stacker. When called, Long Save is given the address of the interrupted task's TSI in general register 2 and its XTSI address in general register 3. This subroutine stores all general purpose, floating point and control registers in the XTSI. Long Save then checks the saved time value in the PSA (PSATSA) to see if it is negative -- indicating that a timer interruption is pending along with the interruption being handled by the SVC, Program or I/O Interrupt Stacker. If PSATSA is negative, zeroes are placed in the XTSI field XTSCCTI. Otherwise the positive value in PSATSA is saved in XTSCCTI. The "in execution" bit is then turned off and the "ready" bit turned on (in the TSI). Long Save then returns to the Interrupt Stacker that called.

The activities of the individual stackers are described on the following pages.

PROGRAM INTERRUPT STACKER: On return from SAQ, the Program Interrupt Stacker checks to see if a timer interruption is pending in the hardware.

This is accomplished by testing the stored timer field in the PSA for a negative value. The purpose of this test is to take into account the possibility of a program and a timer interruption occurring simultaneously.

If a timer interruption is pending, the timer/program indicator (PSA128) is turned on in the PSA. This causes the pending timer interruption to be ignored until after the program interruption is processed. If no timer interruption is pending, the Program Interrupt Stacker calls the Supervisor Core Allocation subroutine to request storage for a GQE. When the space has been allocated, the Interrupt Stacker initializes a GQE with the following:

- The address of the current task status index (TSI) from the PSA.
- The instruction-length code (ILC) from the old program status word (PSW).

- The interrupt code from the PSA.
- The loc-on-Q (the symbolic queue number as defined by its relative location in the scan table sequence) of the queue on which this GQE will be placed.

When the GQE is initialized, all fields are first set to zero; required information is then placed in the proper fields. The stacker transfers control to the Long Save subroutine, which saves CPU status information in the interrupted program's XTSI. When the Long Save subroutine is finished, it returns control to the Interrupt Stacker.

The Program Interrupt Stacker then analyzes the interrupt code (PSAPIC) to determine which of three exit routes it should take.

If the interrupt code is less than 16, specifying a program interrupt for the task, exit is to the SVC Queue Processor at CEAHQQ. The SVC Queue Processor calls Queue GQE on TSI (CEAAF) to put the program interrupt on the task's TSI. On return from CEAAF, exit is to the Queue Scanner.

If the interrupt code is 16 or 17, specifying a segment or paging relocation exception, exit is to the Program Interrupt Processor (CEANA).

If the interrupt code is greater than 17, the ERROR macro instruction causes the System Error Processor to be invoked via an interrupt.

If the Interrupt Stacker determines that a timer interruption is pending, the timer value is set to a value such that the program can operate again when the program interruption is cleared. Thus, another timer interruption is forced to occur immediately after the program interruption has been processed by the supervisor and the task has been reactivated. The Interrupt Stacker sets the timer/program indicator in the PSA, which causes the timer interruption that is currently pending to be ignored when it occurs (that is, it will not be acted upon by the External Interrupt Stacker). When this action has been taken, the Program Interrupt Stacker initializes the GQE as described above.

SVC INTERRUPT STACKER: After updating the timer fields in the PSA, if required, and making the interrupt log entry, the SVC Interrupt Stacker again checks the status of the system.

If the system is in supervisor state, the SVC causing the interruption must be an RSS type or a system error SVC. If it is

neither of these, a system error condition exists and the SVC Interrupt Stacker issues an ERROR macro instruction. This will cause reentry into itself and eventual exit to the System Error Processor.

If it is an RSS type SVC, exit is to RSS via an LPSW. If it is a system error SVC, exit is to the System Error Processor at CEAIS1.

If the system is in problem state, the SERR Auxiliary Queue Processing subroutine is called. On return, the saved timer is checked to see if there is a simultaneous timer interrupt pending. If there is one, the timer interrupt pending flag (PSA128) is set. Then, or if no timer interruption is pending, the Long Save subroutine is called.

On return, the SVC code is checked to see which of the following types it is:

- A system error SVC (YSER) from a virtual storage task.
- An RSS type SVC.
- A task oriented SVC or an LVPSW SVC.

If it is none of these three types, it will be an SVC to be handled by the SVC Queue Processor (CEAHQ). In this case, GQE space is requested from Supervisor Core Allocation and the GQE is initialized with the following information: The TSI address and interrupt code from the PSA, the ILC from the SVC old PSW, the Loc-on-Q of the SVC queue. All other fields of the GQE are set to zero and exit is to the SVC Queue Processor at CEAHQP2.

YSER SVC Processing: Interruptions caused by YSER macro instructions issued in virtual storage tasks are processed by the System Error Processor. When one of these is detected, therefore, the SVC Interrupt Stacker exits to this processor at CEAIS2.

RSS Type SVC Processing: RSS type SVCs issued in problem state are handled by RSS. Therefore, exit is to RSS via LPSW after resetting the TSI lock.

Task Oriented SVC Processing: Certain SVCs, issued in virtual storage routines, do not require the services of the SVC Queue Processor and its related processing routines. These SVCs request transfer of control from one virtual storage routine to another. Such requests are handled in the SVC Interrupt Stacker as follows:

1. When the SVC was the object of an EXECUTE instruction, the virtual storage address of the SVC is obtained and stored in the ISA at ISACSW. This

step is skipped if it was not the object of an EXECUTE instruction.

2. The program and SVC masks are set in the TSI (TSIPMF=1).
3. The old SVC PSW is mapped from the PSA to the SVC old VPSW location in the ISA.
4. The SVC new VPSW is moved to the current VPSW location in the ISA.
5. The ILC, CC, program mask, and IC are moved from the current VPSW to the SVC old PSW location.
6. The program interrupt mask is moved from the current VPSW location to the TSI.
7. The updated PSW is then moved from the XTSI to the PSA.
8. A check is then made to see if a timer interrupt is pending. If there is one:
 - 8A. Exit is to the Queue Scanner after resetting the TSI lock and enabling interrupts.
9. When there is no timer interrupt pending, the elapsed time is computed and the elapsed timer (PSAETM) updated.
10. The timer reset value (PSATRV) is set to HEX OFFFFFFFF.
11. The 'Ready' flag is turned off and the 'Execute' flag turned on in the TSI.
12. Exit is to the virtual storage task via LPSW.

LVPSW Processing: When the interruption is caused by an LVPSW macro instruction, processing is similar to that for task-oriented SVCs. The following procedures precede those described under points 5, 6, 7, and 8A for task SVC handling:

1. If the routine that issued the LVPSW has the correct authority, the main storage address of the virtual PSW is obtained.
2. If the segment is not available and the page table is not in a page table page, an ERROR macro instruction (1455) is issued. If the segment is unavailable and in a page table page or if the page is unavailable, the instruction counter in the XTSI is backed up to cause re-execution of the LVPSW. This routine then exits to the Queue Scanner.

3. If it is on a doubleword boundary and the ISA is in main storage, the new VPSW is moved to the current VPSW location in the ISA.
4. If the ITI (inhibit task interrupts) flag is on in the ISA, it is turned off. If it is not on, the ISA lock is reset.

Two conditions are checked which can result in a program interrupt to the task issuing the LVPSW:

1. If the task is of insufficient authority to use the instruction, the program mask and the SVC mask are set in the TSI. The old SVC PSW is moved to the old SVC VPSW location in the ISA. The new SVC VPSW is moved to the current VPSW location in the ISA. Processing then continues as in steps 5, 6, 7, and 8A previously mentioned.
2. If the virtual PSW main storage address is not on a doubleword boundary, the program interrupt code in the PSW is set to indicate this condition before proceeding.

In these two cases, the task is restarted with the current VPSW set to the program new PSW.

EXTERNAL INTERRUPT STACKER: External interruptions are initially accepted by the Recovery Nucleus routine, which saves and resets the timer location in the PSA, checks for, and processes any malfunction alert interruptions, inter-CPU communication interruptions and interrupt key interruptions as described under "Major Error Recovery Procedures" in this section. If the interruption is not the result of a malfunction alert, the Recovery Nucleus transfers control to the External Interrupt Stacker. On entry to the External Interrupt Stacker, a short-save is performed, and the external indicator field in the PSA is checked to determine if any indicators are on. If none are on, an error condition exists, at which point the general registers are restored, and the System Error Processor is called via an ERROR SVC.

If a supervisor state program was interrupted, and the external interrupt code indicates that a timer interruption occurred (i.e., if the PSA location 14 is 08) the External Interrupt Stacker checks the timer/prog indicator in the PSA. If the timer/prog indicator is on, a task interruption is being processed by the Program Interrupt Stacker, and the timer interruption is to be ignored, since it will occur when the task is restarted. Therefore, the External Interrupt Stacker

turns the timer/program indicator off and checks the external interrupt code to see if an external interrupt key signal has occurred. If not, the saved registers and timer value are restored and the external old PSW loaded.

If the timer/program indicator is not on, a further check is made to determine whether the CPU was in the wait state. If not, a major system error is declared. If the CPU was in the wait state, control is transferred to the Queue Scanner.

If the interrupted program was in the problem state, the Long Save subroutine is called to save all necessary information in the interrupted task's XTSI. Control then returns to the Interrupt Stacker and a check is made for a timer interruption. If one exists, Supervisor Core Allocation is called to allocate space for a GQE. When control returns to the Interrupt Stacker, the allocated space is used to initialize a GQE as follows:

- TSI address from the PSA.
- Loc-on-Q of the timer interrupt queue.

The PCB count in the new GQE is set to zero, and the Interrupt Stacker transfers control to the Enqueue GQE subroutine, which adds the GQE to the timer interrupt queue, posts this in the scan table entry for the Timer Interrupt Queue Processor, and disables interruptions. Control returns to the Interrupt Stacker, interruptions are enabled and the Interrupt Stacker exits to the Queue Scanner.

External interrupt key signals in the problem state are delivered to RSS after a long save is performed and the TSI lock byte is reset.

In supervisor state, the interrupt key signal causes control to go directly to RSS.

I/O INTERRUPT STACKER: The I/O Interrupt Stacker saves and updates the timer fields in the PSA when the system is in either problem or wait state.

I/O interrupts for RSS devices are passed to RSS immediately via LPSW when the system is in supervisor state. If not in supervisor state, a long save is done and the TSI lock reset before exiting to RSS.

When RSS is not involved, the I/O Interrupt Stacker makes an interrupt log entry and works off any entries in the SERR Auxiliary Queue. Then, GQE space is requested from the Supervisor Core Allocation subrou-

tine. An additional 64 bytes is also requested for channel logout data when a channel or interface control check is specified.

The GQE is initialized with the CSW and interrupt code from the PSA. If the interrupt is from a paging drum, the Loc-on-Q field is set for the Page Drum Interrupt Processor (CEAA9). Otherwise the Loc-on-Q field is set for the Channel Interrupt Processor (CEAA4).

If the interruption occurred in the supervisor state, interruptions are not enabled, and the GQE is queued by transferring control to the Enqueue GQE subroutine, which returns control to the stacker. A return is then made to the interrupted program by performing the following:

- Restoring the general purpose registers and timer value.
- Turning off the wait state bit and setting the simultaneous interrupt flags (PSA128) when a timer interrupt is pending.
- Loading the I/O old PSW.

If the interruption occurred in the problem state, the I/O Interrupt Stacker obtains the current TSI address from the PSA and the XTSI address from the TSI, saves the old PSW, and transfers control to the Long Save subroutine. Long Save places the necessary status information in the interrupted task's XTSI, and returns control to the I/O Interrupt Stacker. The Interrupt Stacker then calls the Enqueue GQE subroutine to add the GQE to the appropriate queue.

If a timer interrupt is pending, the simultaneous interrupt flag (PSA128) is set. Main storage for a timer interrupt GQE is then requested from Supervisor Core Allocation. It is initialized and queued as in the External Interrupt Stacker processing. Exit is then to the Queue Scanner (CEAJQ at CEAJQS).

If no timer interrupt is pending, the TSI lock is reset and exit is also to the Queue Scanner.

QUEUE SCANNING AND PROCESSING

Queue Scanner (CEAJQ) Chart AB

The Queue Scanner functions as a centralized sequencing mechanism which determines the order in which independent processors are to be given control to perform the work specified in GQEs on the system queues.

Attributes: The Queue Scanner is parallel reentrant, resident, closed, and operates in the privileged state.

Entry: CEAJQS, by:

- The interrupt stackers.
- The queue processor's return of control.
- Task Interrupt Control.

Assumptions: It is assumed that the queue processors are maintaining the proper information in the scan-table entries using only the subroutines available for that purpose: the Enqueue GQE, Dequeue GQE, and Set Suppress Flags subroutines.

Modules Called: Timer Interrupt Queue Processor (CEAKT) takes action with respect to a task for which a GQE has been placed on the timer-interrupt queue, performing the initial step(s) appropriate to effect either the creation of a task interruption, or a task time-slice end.

Page Drum Queue Processor (CEAA8) initiates I/O on all 2301 paging drums.

Page Drum Interrupt Queue Processor (CEAA9) processes all interruptions occurring on the paging drums.

User Core Allocation Queue Processor (CEANB) allocates pages of main storage for user pages.

Auxiliary Storage Allocation Queue Processor (CEAIA) allocates and maintains storage for user pages in auxiliary drum and disk devices.

I/O Device Queue Processor (CEAA3) processes GQEs representing input or output requests to devices other than drums.

Channel Interrupt Queue Processor (CEAA4) locates GQEs which initiated I/O operations and performs these required functions: frees devices, distinguishes between the initial and subsequent asynchronous interruptions; and informs affected tasks of the occurrence of I/O interrupts.

SVC Queue Processor (CEAHQ) dequeues the GQE from the scan table, assures that the task issuing the SVC is authorized to do so and identifies and branches to the proper SVC subroutine to service the interruption.

The Internal Scheduler (CEAKI) is entered when there is no work queued or when devices required to perform queued work are not available. The Internal Scheduler sorts tasks into order on the dis-

patchable list and passes control to the Dispatcher.

SYSEERR (CEAIS) is called when software errors are encountered during processing.

Exits: To queue processor - when work is found in the scan table. If the processor is the Page Drum Queue Processor (CEAA8), the affected scan table entry lock byte (SCNF3LOK) is locked using the SETLOCK macro. This ensures exclusive use of the entry in a duplex environment. If the processor is the Page Drum Interrupt Queue Processor (CEAA9), the scan table entry lock byte (SCNF3LOK) for CEAA8 is locked. This is also done to preserve exclusive use of the entry while CEAA9 is processing interrupts for that entry. The lock will be opened using the OPENLOCK macro by the processor which was given control prior to its exiting to the Queue Scanner. To Internal Scheduler - when all GQEs have been processed or all queue processors are busy.

Operation: Upon entry, the queue scanner disables all interruptions except machine check; it then enables and immediately disables I/O and external interruptions to cause any pending interruptions to be queued. CEAJQ then checks the master count of matched facilities in the scan table master control table. If the value of the field is zero, no GQEs can be processed, and the Queue Scanner transfers control to the Internal Scheduler. If the value of the field is not zero, the Queue Scanner tests the DIG (device interaction group) counts of matched facilities in the scan table master control table to find a DIG entry for which there is work. If none is found, the Queue Scanner exits to the Internal Scheduler.

When a DIG entry is found containing a non-zero count of matched facilities, the DIG lock byte is tested. If it is off, one of the queues associated with this DIG can be processed. If it is on, the master count of matched facilities is decremented by this DIG's count of matched facilities. The search of DIG entries then continues.

When one is found, the DIG count and the master count are decremented by one, and the associated scan table entries are searched for one which satisfies the following conditions:

- The "Q" flag is on.
- No suppress flag or processor lock bytes are on (queues whose lock bytes are on are currently being manipulated and may not be entered).

When an active queue entry is found, the Queue Scanner locks the scan table entry and transfers control to its related processor. The linkage to the processor is performed by storing the following in general registers:

- The scan table queue entry (that is, the address of the GQE).
- The address of the queue processor.

If there are no Q flags on in the scan table entries associated with a DIG entry whose count is greater than 0, a system error SVC is issued. In all other cases, when no processable queue can be found, the Queue Scanner exits to the Internal Scheduler.

QUEUE-CONTROL SUBROUTINES

Four subroutines perform the general functions of controlling the queues summarized by the scan table. These are:

- The Enqueue GQE subroutine, which places a GQE on the specified queue.
- The Dequeue GQE subroutine, which removes a GQE from the proper queue.
- The Move GQE subroutine, which routes a GQE from queue to queue, and releases the GQE storage and any PCB storage associated with it when all work specified is completed.
- The Set Suppress Flag subroutine, which sets a specified suppress flag on or off in a specified scan table entry for a specified queue, and sets and resets the busy flag in a DIG entry.

These subroutines are available to all supervisor components that operate on GQEs. They are entered via subroutine linkage, with the GQE pointer in general register one. All of these subroutines are resident, reenterable, and privileged. There are other subroutines that perform special queue control functions (for example, the Dequeue I/O Requests and the Generate and Enqueue Interrupt GQE subroutines). These subroutines are described in this section under "Supervisor Subroutines."

Enqueue GQE Subroutine (CEAJQ Entered at CEAJEN)

This subroutine adds a GQE to the designated queue and updates the scan table entry for that queue to reflect the addition of the new element.

Entry: CEAJEN.

RESTRICTIONS: The subroutine will operate with all interruptions except machine check disabled.

Assumptions: The location-on-queue (loc-on-Q) field of the GQE contains a binary value equivalent to the location in the scan table of the entry on which the GQE is to be queued. The third byte of each scan table entry will contain a binary value from 1 to 255 which will identify the device interaction group (DIG) to which the entry belongs.

Exit: To caller.

Operation: On entry, Enqueue disables all interruptions except machine check, and then tests a register set by the caller to determine whether interruptions are to be enabled before control is returned to the caller. If so, Enqueue sets an indicator to specify this, locates the appropriate scan table entry (i.e., retrieves the first loc-on-Q from the GQE), and then performs one of the following:

- If there are no prior entries on the queue (that is, if the first queue entry field in the scan table entry is all zeros), the address, or pointer, to the GQE is placed in the first and last queue entry fields in the scan table entry. The forward link and reverse link fields in the GQE are set to zeros, and the Q flag in the scan table entry is set on.
- If there are other entries on the queue, the subroutine points the new GQE to the previous last GQE by placing the address currently contained in the scan table entry's last-queue-entry field into the new GQE's reverse link field. The address of the new GQE (now the last GQE) is then placed in the scan table entry's last-queue-entry field and in the forward-pointer field of the previously last GQE. The forward pointer of the new GQE is set to zero.

When Enqueue has accomplished one of the above actions, the master count of GQEs of the scan table master control table (SMC) is increased by one. If there were no other GQEs on the specified queue when the new one was added, the subroutine tests to see if any suppress flags or processor lock bytes are on. If not, the Enqueue GQE subroutine adds one to the DIG count of matched facilities for the GQE and the master count of matched facilities in the SMC. The subroutine then enables interruptions, if specified by the caller, and returns control to the caller.

Dequeue GQE Subroutine (CEAJQ Entered at CEAJDE)

This subroutine removes GQE pointers from the proper queues as specified by the callers.

Entry: CEAJDE

Assumptions: The location-on-queue (loc-on-Q) field in the GQE contains a binary value equivalent to the location in the scan table of the processor that must handle the GQE. The third byte of each scan table entry will contain a value from 1 to 255 which will identify the device interaction group (DIG) to which the entry belongs.

RESTRICTIONS: The subroutine operates with all interruptions except machine check disabled.

Exits:

Normal - To caller.

Error - To the System Error Processor if the SMC lock is locked too long or if an invalid Loc-on-Q or DIG ID is encountered.

Operation: If interruptions are to be enabled on return to the caller, Dequeue GQE sets an indicator in a general register to signify this, locates the scan table entry for the queue, and performs one of the following:

- If the GQE to be dequeued is the only one on the queue, the first and last queue entry fields in the scan table entry are set to zero, and the "Q" flag is set off.
- If there are other GQEs on the queue, forward and reverse are updated to remove the requested GQE from the chain. The scan table entry pointers are updated if it is the first or the last GQE on the queue.

When one of the above actions has been taken, the counts of matched facilities in the DIG are adjusted according to the following algorithm, unless a suppress flag is found on:

- If the Q flag has been turned off (that is, no more GQEs remain in the queue), the counts are decreased by one. In addition, if the scan table entry is found to be locked, it is unlocked, and the counts are increased by one. Dequeue GQE lowers the master count of GQEs in the scan table master control table, enables interruptions if speci-

fied, and returns control to the caller.

Move GQE Subroutine (CEAJQ Entered at CEAJMG)

This subroutine examines the sequence of queue processors required to perform the work specified by the GQE and routes the GQE from queue to queue until the last processor has finished the required processing. It then returns the storage space occupied by the GQE and any associated PCBs.

Entry: CEAJMG

Assumptions: The subroutine assumes that:

- A group of processor-sequence numbers is maintained in the queue-processor-string table so that one to three numbers can be shifted within the GQE.
- An eight-byte field in the GQE is reserved for from one to three processor sequence numbers plus an indicator signaling the end of processing for the GQE, or the location of a continuation of the processor string in the queue-processor-string table.

Modules Called: Supervisor Core Release subroutine (CEAL1 entered at CEAL01) releases main storage occupied by the GQE and any associated PCBs.

Enqueue GQE subroutine (CEAJQ entered at CEAJEN) places GQE pointers on the specified scan table queues.

Exit: To caller.

Operation: On entry, the Move GQE subroutine determines whether interruptions can be enabled, sets up the appropriate indicators to enable or disable interruptions, and adjusts a string of queue processor numbers to locate the number of the next processor required for GQE processing.

If no further work is required by the GQE, the PCB count in the GQE is checked. If this field indicates that there are PCBs attached to the GQE, Move GQE calls the Supervisor Core Release subroutine to release the storage space they occupy. When all PCBs have been released the GQE storage is released in the same manner. Move GQE then enables or disables interruptions, and exits to the caller.

If further work is required for the GQE, Move GQE determines the next required processor, calls the Enqueue GQE subroutine to move the GQE pointer to the appropriate scan table queue, and enables or disables

interruptions. On return of control, move-GQE exits to the caller.

Set Suppress Flag Subroutine (CEAJQ Entered at CEAJSF) Chart AC

This subroutine sets on or off all suppress flags in the scan table and all DIG busy flags.

Entry: CEAJSF

Assumptions: It is assumed that this subroutine will be called whenever a suppress flag must be set.

RESTRICTIONS: The subroutine runs with all interruptions except machine check disabled. The mask used to set the flags on or off must have the high-order bit set to zero. If set to one, it may cause the "Q" flag to be inadvertently turned on or off.

Modules Called: The System Error Processor (CEAIS) may be given control when an interruption indicating a system error condition occurs.

Exit: To caller.

Operation: On entry, the SSF subroutine disables all interruptions and then sets the interruption indicator as specified by the caller. SSF then checks the location-on-queue supplied by the caller against the scan table master control table. If the specified location is invalid, a system error SVC is issued. If the location is valid the subroutine checks the flag setting request to determine whether a suppress flag is to be turned on or off.

If a suppress flag is to be set on, the subroutine accomplishes this by using the OR instruction on the suppress flag and a mask specified in the caller's parameter register. When the flag is on, SSF tests the "Q" flag. If this flag is on, the scan table entry's lock byte and suppress flags are checked. If none of these is on, one is subtracted from the DIG count of matched facilities and the matched facilities. No subtraction takes place if the DIG count is already zero or if at entry, the processor lock byte is set on. At this point, or if the "Q" flag was off, a common return is performed, as follows:

- Registers are restored to their state.
- Interruptions are enabled, if specified by the caller.
- Control is returned to the caller.

The suppress flags are set off in the same manner as described for setting them on (that is, via a mask), except that the

Table 5. QUEUE Scanner operations in processing of GQE

Queue Scanner Function	Conditions	Result
Dispatch GQE to Queue Processor		Decrement Dig Count; Turn on SCNLOK
Queue 1st GQE on Scan Table Entry	No Suppress Flags on <u>and</u> SCNLOK off	Increment Dig Count
Turn Dig Busy on	SCNLOK off	Exit
	SCNLOK on; Suppress flags on	Open SCNLOK
	SCNLOK on; Suppress flags off; work on queue	Open SCNLOK; Increment DIG count
Turn Dig Busy Off	Dig Busy is now on	Increment Dig Count
Turn on 1st Suppress Flag	Non-empty queue <u>and</u> SCNLOK on	Open SCNLOK
	Non-empty queue <u>and</u> SCNLOK off	Decrement Dig Count
Turn off last Suppress Flag	Non-empty queue <u>and</u> SCNLOK off	Increment Dig Count
Dequeue a GQE	Non-empty queue remains <u>and</u> SCNLOK on <u>and</u> No Suppress Flags on	Open SCNLOK; Increment Dig Count
	Empty queue remains <u>and</u> SCNLOK on <u>and</u> No Suppress Flags on	Open SCNLOK
	Empty queue remains <u>and</u> SCNLOK off <u>and</u> No Suppress Flags on	Decrement Dig Count
All Others	No effect on Dig Counts or SCNLOK	

DIG count of the matched facilities in the scan table master control table is raised when the "Q" flag is on and the other suppress flags and the lock byte in the scan table entry are off (see Table 5).

QUEUE PROCESSORS

Timer Interrupt Queue Processor (CEAKT)
Chart AD

Each GQE on this processor's queue represents a timer interruption, a forced time slice end for a task, or a migration request. The GQE may have been created as a result of a task having reached normal time slice end, having been forced to time slice end, or having been selected to have its pages migrated from auxiliary drum storage to auxiliary disk storage.

In the time slice end situations, a user timer interruption may also be involved. If it is not, the function of this processor is to cause the task to be rescheduled according to its STE parameters and to effect the release of the space occupied by the task's pages in main storage unless the

task's status is such that it is to remain in the dispatchable list.

If a user timer interruption is involved (user timer field = 0), and it is not a forced time slice end situation, the function of this processor is to cause a task interruption to be created and placed on the TSI interruption queue. The task remains in the dispatchable list and its pages remain in main storage.

The migration function is invoked to cause a task's pages to be migrated from auxiliary drum to auxiliary disk whenever the drum space being used exceeds the limit.

The Auxiliary Storage Allocation routine calls this module for this purpose.

Entries:

CEAKT1 - by Queue Scanner.

CEAKTB - by Create Real Time Interrupt sub-routine (CEAKR).

Modules Called: Dequeue GQE subroutine (CEAJQ entered at CEAJDE) removes GQEs from the processor's queue.

Auxiliary Storage Release subroutine (CEAIA) releases storage for user pages in auxiliary drum and disk devices.

Enqueue GQE subroutine (CEAJQ entered at CEAJEN) places the GQE on the auxiliary storage allocation queue.

Move GQE subroutine (CEAJQ entered at CEAJMG) moves or releases the GQE.

Queue GQE on TSI subroutine (CEAAF) places the GQE pointer on the TSI's interruption queue.

Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) allocates main storage for a new GQE or for a PCB.

User Core Release subroutine (CEAL1 entered at CEAL04) releases main storage occupied by unchanged pages.

Supervisor Core Release subroutine (CEAL1 entered at CEAL02) releases GQE storage space.

Rescheduling subroutine (CEAKZ entered at CEAKZA) is called at normal end of time slice to recompute a task's scheduled start time (SST) and move it to the eligible list. This subroutine is also called in forced time slice end situations to add a task to the inactive list.

Write Shared Pages subroutine (CEAMW entered at CEAMWS) is called to obtain shared pages for migration when the GQE indicates that shared page migration is to be performed.

Paging routine (CEAMQ entered at CEAMQA) reads in a set of page table pages to be used for migration.

Exit: Queue Scanner.

Operation: On entry, the Timer Interrupt Queue Processor (TIP) calls the Dequeue GQE subroutine to remove the GQE from its queue and enable interrupts. On return, TIP determines whether the task's XTSI pages are in main storage. If not, the processor exits to the Queue Scanner.

If the XTSI pages are in main storage, the processor determines the cause of the interruption and performs the appropriate function:

- User Timer Interruption Processing
- Time Slice End Processing

- Page Table Scanning
- Migration

This processor's activities are illustrated in Figure 11.

USER TIMER INTERRUPTION PROCESSING: At forced or normal time slice end, the task's timers are updated. If the user timer field goes to zero, and it is not a forced time slice end, a task interruption is created and queue-GQE-on-TSI is called to put it on the task's TSI interrupt queue. The task's pages are left in main storage and the task remains in the dispatchable list.

If it is forced time slice end, the task's pages are written from main storage, and rescheduling is called to put the task on the inactive list. A task interruption is not created, but one will be when TIP subsequently gains control after the forced time slice end situation has been handled and the task again reaches time slice end normally.

TIME SLICE END PROCESSING: When a user timer interruption is not involved, and it is a forced time slice end situation, a check is made to see if it was caused by a TSEND SVC. If yes, the task is set in delay status, Rescheduling is called to put it on the inactive list, a timer interruption is set up, and the task's pages are written from main storage. The timer interruption will be fielded by a special entry point in CEAKT which will cause the task to be put in ready status, Rescheduling to be called to put it on the eligible list with its SST recomputed, and then the processor will exit to the Queue Scanner.

For forced time slice end interruptions, not TSEND, Rescheduling is called to put the task on the inactive list and recompute its SST. Then the task's pages are written from main storage.

At normal time slice end, the task is given another time slice value and then the number of page relocations caused by the task during the time slice just completed is checked. If less than the number specified by the control field in the STE, the task is moved to the bottom of the dispatchable list. In either case, the quanta count is then lowered by one. If it does not go to zero, exit is made to the Queue Scanner. If the count does go to zero, the normal time slice end flag is set in the TSI and Rescheduling is called to recompute the SST and put the task on the eligible list. The task's pages are then written from main storage. As a part of time slice end processing, this routine also updates fields in the system statistical table

(CHASST). For any occurrence of time slice end, SSTALT is increased by one. If time slice end occurs because a task reaches its quanta limit, SSTQLT is also raised.

PAGE TABLE SCAN FUNCTION: The page table scan goes through the page tables and upon encountering a page which is available and whose page hold count is zero (indicating no user I/O in progress on this page) a check is made to see if this page has been changed. If unchanged, its main storage space is released. If changed, the page must be written to auxiliary storage and the auxiliary storage space previously occupied by the page must be released. A PCB is created for each such page preparatory to writing it to the paging disk or paging drum. An effort is made to block (write to the paging device in contiguous groups) certain of the task's changed pages. Those to be blocked are the high usage pages of the task.

The procedure for selecting the pages to be blocked and the device to which they are to be written is controlled by a page usage count, maintained in the external page table (XPTPMC), and two system table fields: SYSBLK, the drum block limit; and SYSBLK2, the combined drum and disk block limit.

When CEAKT finds a changed page that has been in main storage for three consecutive time slices (XPTMC=3), a test is made to see if the number of blocked pages (TSIBLK) is less than SYSBLK. If so, a flag is set in the external page table (XPTPP) indicating that it is a blocked page and the page is written to drum. When TSIBLK is greater than or equal to SYSBLK, and the task is inactive, TSIBLK is compared to SYSBLK2. If less, XPTPP is set for the page and it is written to disk. Every time a page is blocked to the drum or disk, the count, TSIBLK, is incremented. This count is initialized at the beginning of each page table scan for a task. Also, the XPTPP flag is turned off in each page before the above investigation is made.

When the end of the page tables is reached, a check is made to see if there are any page table pages. If so, the chain of page tables is scanned, and those page table pages which do not contain virtual memory pages in page hold are written out. Then the segment table entries for the page table on a page table page that is being written out are marked "not in main storage." When the scan of page table pages is complete, a check is made to see if any pages were skipped due to user I/O. If so, a GQE is built and stored in the TSI (unless the task is identified as an MTT application. In this case, immediate exit is made to the Queue Scanner). When user

I/O processing is complete for the task, the GQE is queued on the timer queue. When control is regained, the page table scan is reentered.

If no pages were in page hold, all page table pages should have been written, and a check is made for segment table pages and auxiliary segment table pages. If present, any auxiliary storage is released, and the pages are written from main storage. Then the first XTISI page is written after release of any auxiliary storage it might have been assigned.

If this is not a migration case, or if the migration function is complete, the XTISI pages for the task are written out with the drum preference bit set in the PCB. If virtual memory pages on the page table page reside on drum, the disk preference bit is turned on instead of the drum preference bit in the PCB. Exit is then made to the Queue Scanner via CEAHQR.

MIGRATION FUNCTION:

1. A set of page table pages (the size of the set is determined by a parameter in the system table) is read in. CEAMQ is called to read in the page table pages.
2. When control returns, the page tables which correspond to these page table pages are scanned, and a GQE is set up. PCBs for all virtual memory pages to be migrated from this set are attached to the GQE. When the maximum number of PCBs that can be attached to a GQE is reached, a new GQE is built. A new GQE will also be set up when pages residing on a different drum are encountered. These GQE chains are based on the MIG work area, obtained via Supervisor Core Allocation.
3. When all page tables in this set of page table pages have been scanned, and the necessary GQE/PCBs set up, a drum read for a group of virtual memory pages is initiated. The size of this group is established at start-up time. The parameter used is SYSPCB.
4. When this read is complete, Auxiliary Storage Release (CEAIA) is called to free the drum space. The appropriate fields in the GQE/PCBs for this group of virtual memory pages are changed to indicate a write operation and the write to disk is initiated.
5. Steps 3 and 4 are followed alternately until all migration GQE/PCBs that were set up have been exhausted. The set of page table pages formed in Step 1

is written out as described under "Time Slice End Processing."

6. Steps 1 through 6 are repeated until all page table pages have been processed.
7. The first XTSI page is then processed as if it were a set of page table pages.

When a read or a write is initiated, this module exits to the Queue Scanner via CEAHQR. A returning migration GQE is recognized on entry to CEAKT by an indicator in the TSI. Reads and writes are distinguished from each other by indicators in the GQE.

SHARED PAGE MIGRATION: Migration of a task's shared pages is similar to the normal migration function except that the Write Shared Page subroutine provides the pages to migrate. The GQE for shared page migration can indicate any of three conditions:

- A GQE starting migration.
- A returning write GQE.
- A returning read GQE.

When the GQE represents a request to start shared page migration, CEAKT locks the system table lock (SYSTSKLK) and then updates SYSECB by adding the value in SYS-PCB to it. SYSTSKLK is then unlocked. A counter is set to specify the number of times that CEAKT will call Write Shared Pages to migrate; the GQE is released by a call to Move GQE; and Write Shared Pages is called to obtain pages to migrate.

When it is a returning write GQE, the GQE is released by a call to Move GQE. The counter controlling calls to Write Shared Pages is decreased by one. (If it goes to zero, the System Error Processor is invoked.) A check is then made to see if the shared pages on drum are within the limit permitting migration of more pages. If so, Write Shared Pages is recalled to obtain more pages. If not, (or if Write Shared Pages returns to CEAKT), SYSECB is lowered by the value in SYSPCB; the migration in progress flag (SYSMG) is turned off; and the Timer Interrupt Processor exits to the Queue Scanner.

A returning read GQE is handled the same as in normal migration, that is GQE/PCB fields are changed from reads to writes; the auxiliary storage is released; the XTSI write count is updated; and the drum space is released with no TSI pointer in register zero. Whenever a task's pages are migrated from drum to disk, a field (SSTMIP) in the

system statistical table is increased by the number of pages migrated.

Page Drum Queue Processor (CEAA8) Chart AE

The purpose of the Page Drum Queue Processor (PDQP) is to complete and maintain a series of channel command word programs. The CCWs are designed to handle drum paging operations to meet page control block (PCB) requirements or DRAM requests (IORCB) for dummy records between drum pages.

Entry: CEAA81. The affected scan table entry lock byte (SCNF3LOK) will be locked prior to entry to this processor in order to preserve the exclusive use of the device queue in a duplex environment.

Modules Called: Page I/O Error Recovery subroutine (CEAAM) investigates failure to obtain a device path and immediate start I/O failure.

Start I/O subroutine (CEAAG) performs the start I/O function.

Set Suppress Flag subroutine (CEAJQ entered at CEAJSF) sets the appropriate flag in the queue processor's scan table queue.

System Error processor (SYSERR) (CEAIS) may be given control when an interruption indicating a system error condition occurs.

Pathfinding subroutine (CEAA5) obtains actual path to a device.

Dequeue GQE subroutine (CEAJQ entered at CEAJDE) removes the GQE from the processor's queue.

Generate and Enqueue Interruption GQE subroutine (CEABQ) generates and queues an interruption GQE for the paging drum when a start I/O attempt returns a failure code.

Move GQE subroutine (CEAJMG) moves unposted GQEs to the appropriate queue for posting.

Queue GQE on TSI (CEAAF) queues an interrupt GQE on a TSI.

Channel Interrupt Queue Processor (CEAA4) releases pages from page hold.

Exits:

Normal - To Queue Scanner.

Error - To System Error Processor.

Operation: The Page Drum Queue processor uses information passed to it in the GQE and PCBES to maintain fields in the system table (CHASYS). The system table contains

18 seek arguments divided into two parts. The first nine seek arguments are called chain 1 and the second nine, chain 2. Following these are 18 CCW programs (one for each of the seek arguments) composed of four CCWs each:

1. SEEK
2. SEARCH ID EQUAL
3. TIC (2)
4. READ/WRITE.

To link the two chains together, a TIC command at the end of chain 1 points to the first SEEK in chain 2, and a TIC at the end of chain 2 points to the first SEEK in chain 1. These CCW programs are used by PDQP to process the GQEs and PCBES.

Processing includes the following steps:

1. The suppress flag (F5) in the scan table is set on.
2. The page drum directory is searched using the location-on-queue as the search argument. If the entry is not found, a major system error is declared.
3. When the proper drum interface control block is located, its slot mask (SYS-SLT) is set to all ones, and the 2301 device code (SYSDEV) and symbolic device address are inserted.
4. If chain 1 is not running, it is selected for operation. If it is running, chain 2 is selected. If both are running, PDQP exits to the Queue Scanner.
5. All READ/WRITE commands in the selected chain are initialized to NOPS. A TIC/NOP is included at the end of the channel programs.
6. Searching begins with the first PCBE in the string. It is terminated when all of the PCBES have been processed, or when all slots have been assigned, or when remaining PCBES are duplicate slot requests that must be assigned at a later time. When a slot request is recognized the SYSDIC slot mask is checked for availability. If it is free, the request is filled and the GQE unprocessed count is decremented. The PCBE is flagged as processed and the GQE slot mask bit is turned off. The SYSDIC slot mask bit is set to show that the slot is filled. If a duplicate slot request appears in the same GQE, the slot mask bit for the GQE is turned on again to maintain a current picture of slots yet to be assigned. This enables more than 9 PCBES to be attached to a GQE. The

next PCBE is then processed. If the paging request is for a READ, the READ/WRITE CCW is set accordingly. If it is for a WRITE, it is checked to see if it is for a WRITE/CHECK operation. If it is, and the WRITE has been performed, the write check selected flag is set on. This will cause a READ CCW, with no data transmission, to be generated and executed.

If the slot just processed is the highest or lowest, the slot number is put in the appropriate field in the system table.

The processing cycle of GQEs and their associated PCBES continues until there are no more that can be processed.

7. The lowest slot assigned is used to set up the CAW when a START I/O is required, that is, when the other chain isn't running.

Otherwise the highest slot number in the chain not being operated upon is used to set up a TIC command to the lowest slot assigned in the chain just set up. In this case, the PCI flag is set on in the SEEK CCW and the PCI pending flag set on in the system table. The system table flag byte (SYSLK) describes the state of the two chains at any given time.

The processing cycle of GQEs and their associated PCBES continues until there are no more that can be processed.

8. When START I/O is executed to get a chain running, control is transferred to 3 (above) when there is more work, or to the Queue Scanner if not. If the START I/O is unsuccessful, control is given to Paging Error Recovery.
9. For DRAM requests, PDQP tests the task to supervisor lock (SYSTKSP) to see if SERR was recording. If locked, the task to task lock (SYSTKTK) is checked to see if DRAM recording was in progress. If yes and the present request is from the task that was doing the DRAM recording, an interrupt is sent to the task to reinitiate the entire DRAM operation and SYSTKSP is unlocked. If the requesting task is not the task presently using DRAM, an interrupt to retry the operation is queued on the requester.

When SYSTKSP is not locked, SYSTKTK is checked. If locked and the requesting task is not the task presently using DRAM, an interrupt is sent to the requester to retry the operation. If this is the task using DRAM or if

neither SYSTKSP nor SYSTKTK is locked, the request is processed. If the requested path is found, and the GQE has never been inspected before, 1 is added to the total count of unprocessed operations (SYSUC). The slot number is determined from the real head and record identifiers. If the slot is not available, the appropriate slot in the GQE slot mask is set and the remaining GQEs processed. If the slot is available, the appropriate op code from the IORCB is inserted in the channel program. Then storage keys are set for all virtual memory pages associated with the request, the appropriate slot in the DICB slot mask is set off, and the GQE and IORCB addresses are inserted in the second double word of the seek argument. If there are more GQEs processing continues; if there are no GQEs control transfers to 7 (above).

The Page Drum Queue processor is capable of supporting a multidrum configuration. If the drums share a channel, however, only one drum can be in use at any one time. When a request for another drum is denied because of this situation, the chains involved are put in a hold state, that is, processing terminates until the path to the first drum is freed up. The second drum is then activated. The drum placed in hold will not necessarily be the next to execute, however. The first drum entry on the scan table (CHASCN) is completely processed before anything is done on the second drum queue.

In the duplex environment, PDQP is required to set and reset the lock bytes associated with the TSI and system table whenever the two CPUs might interfere with each other. The scan table entry lock byte (SCNF3LOK) is reset using the OPENLOCK macro instruction before this processor exits to the Queue Scanner.

Page Drum Interrupt Queue Processor (CEAA9) Chart AF

The Page Drum Interrupt Queue (PDIQ) processor processes GQEs representing interruptions caused by the following conditions:

- Return of a sense operation
- A unit check
- Channel or interface control check
- Channel data or chaining check
- Program protection or incorrect length check

- Control unit or channel end
- Channel or device end
- Program-controlled interruption

This processor also collects and stores in the system statistical table (CHASST) the number of writes and reads (private and shared) for each drum device.

Entry: CEAA91

Modules Called: Set Suppress Flag (SSF) subroutine (CEAJQ entered at CEAJSF) sets the appropriate flag in the queue processor's scan table queue.

Page I/O Error Recovery subroutine (CEAAM) investigates the failure to obtain a device path and all I/O error conditions.

Start I/O subroutine (CEAAG) performs the start I/O function.

Dequeue GQE subroutine (CEAJQ entered at CEAJDE) removes the GQE pointer from the PDIQ processor's queue.

Move GQE subroutine (CEAJQ entered at CEAJMG) determines whether further processing is specified by the GQE, and, if so, moves it to another processor's queue, or, if not, releases it.

User Core Release subroutine (CEAL1 entered on CEAL04) releases user-page storage after a write operation has been completed.

Pathfinding subroutine (CEAA5) frees the assigned device path.

Page Posting subroutine (CEAMP) updates associated page table.

Generate and Enqueue Interrupt GQE subroutine (CEABQ) generates and queues an interruption GQE for the paging drum when status is stored for the device selected on a Start I/O attempt.

Queue GQE on TSI (CEAAF) to queue an interrupt GQE on a task to indicate a completed DRAM operation.

Channel Interrupt Queue Processor (CEAA4) to release pages in page hold.

Exits:

Normal - To Queue Scanner.

Error - To System Error Processor.

Operation: The Page Drum Interrupt Queue processor is activated by the Queue Scanner. The affected scan table entry lock

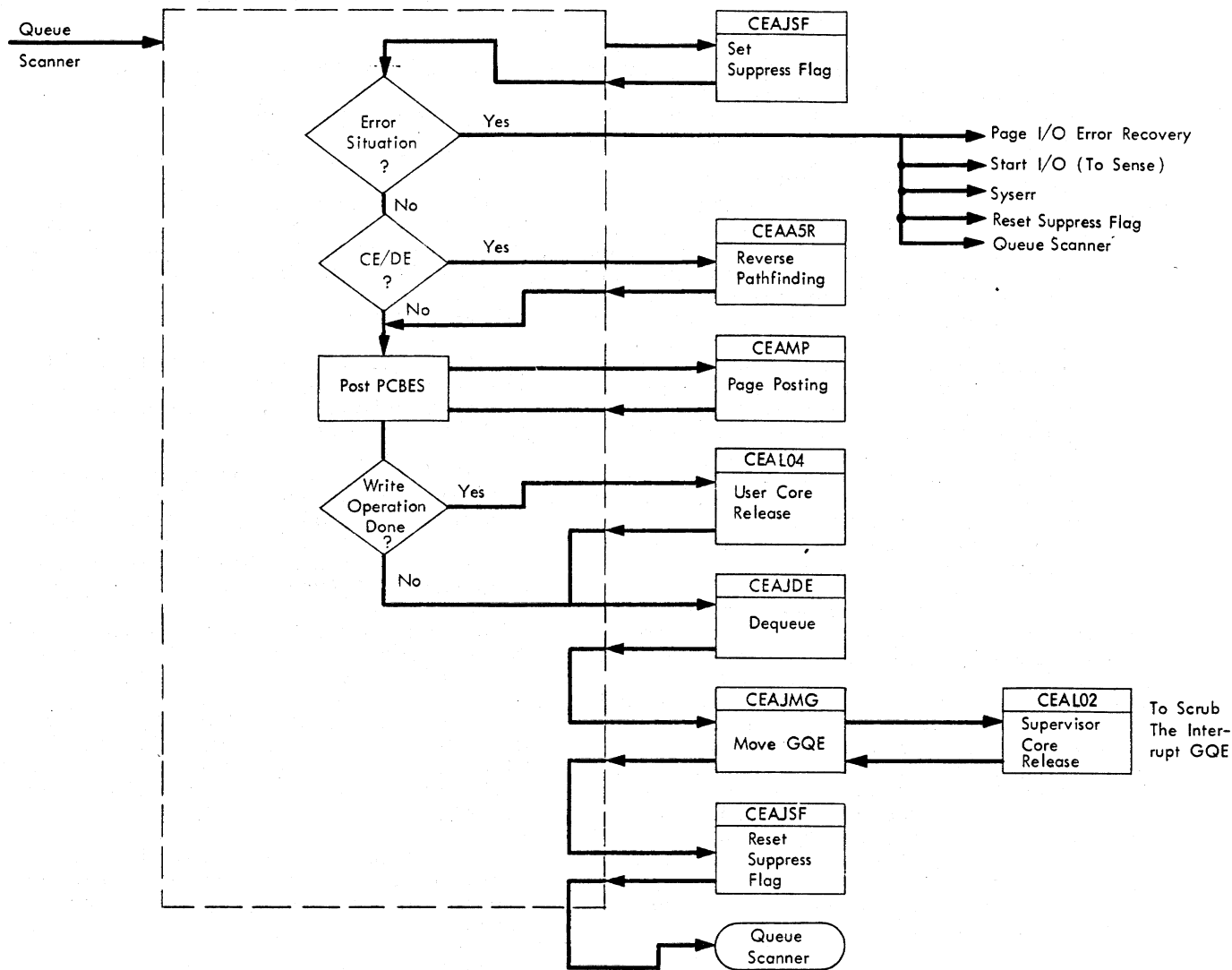


Figure 12. Page Drum Interrupt Queue Processor activities

byte (SCNF3LOK) will be locked prior to entry to this processor in order to preserve the exclusive use of the device queue in a duplex environment. Channel end, device end, and PCI interruptions trigger posting action of all PCB entry paging operations that are complete at interruption time. When all page operations are posted for any GQE, the Move GQE subroutine is called to release the GQE. Processor activities, illustrated in Figure 12, are discussed on the following pages.

On entry from the Queue Scanner, the PDIQ Processor calls the Set Suppress Flag subroutine to set the suppression flag in its scan table queue to interlock it against subsequent queue scanner entries. When control is returned to the processor, the location-on-queue value specified in the GQE is used as an argument to search

the entries of the page drum directory (PDD) for the SYSDIC address in the system table. If a PDD entry cannot be found SYSERR is called. If an entry is found, the processor determines whether the interruption resulted from a sense-operation return. If so, the Set Suppress Flag subroutine is called to unlock the processor's queue entry flag, after which the Page I/O Error Recovery subroutine is called.

If a sense operation has not been returned, the processor tests for other possible conditions which caused the interruption. These conditions and the respective actions taken are illustrated in Figure 13 and discussed on the following pages.

The scan table entry lock byte (SCNF3LOK) is reset using the OPENLOCK

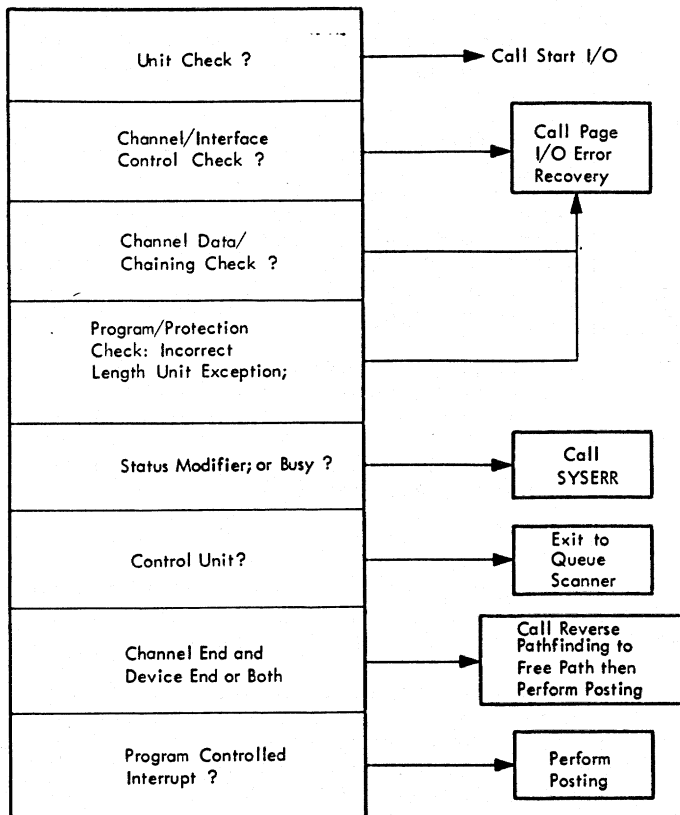


Figure 13. Page Drum Interrupt Queue Processor checking and response to conditions specified in the CSW

macro prior to any exit to the queue scanner.

UNIT CHECK: The processor initiates the sense operation by calling the Start I/O subroutine.

The Start I/O subroutine will return control indicating one of five possible conditions. These conditions and the action they precipitate are as follows:

- SIO is successful, the channel program is in execution. Exit is made to the Queue Scanner.
- The selected channel is defective. Call the Paging I/O Error Recovery Control subroutine.
- The control unit or channel is busy. Retry 17 times and, if not successful exit to the Paging I/O Error Recovery Control subroutine.
- Status was stored for the selected device. Build interruption GQE, queue it on the drum interrupt queue, and call the Queue Scanner.

- Solid start I/O failure. Call the Paging I/O Error Recovery control subroutine.

If the Start I/O is successful, either on the first attempt or after retry attempts, control returns to the PDIQ processor, after which the Dequeue GQE subroutine is called to remove the GQE pointer from the processor's queue and update relevant areas of the scan table to reflect this action. When control is returned to the processor, the Move GQE subroutine is called to determine whether further processing is specified by the GQE.

When control returns to the processor, the Set Suppress Flag subroutine is called to turn off the suppress flags in the scan table. Exit is to the Queue Scanner.

CHANNEL/INTERFACE CONTROL CHECK: If a channel or interface control check condition exists, the PDIQ processor transfers the GQE's channel log-out data, the CSW, and control to the Page I/O Error Recovery subroutine. If retry is successful, normal processing continues. If not, exit is to the Queue Scanner.

INCORRECT LENGTH, UNIT EXCEPTION, PROGRAM CHECK, PROTECTION CHECK, CHANNEL DATA CHECK OR CHAINING CHECK: If any of these conditions exists, the Page I/O Error Recovery subroutine is called.

STATUS MODIFIER OR BUSY CONDITION: If this condition is detected, the PDIQ processor declares a minor ERROR and exits to the Queue Scanner.

CONTROL UNIT END: If a control unit end condition exists, the processor exits to the Queue Scanner.

CHANNEL END/DEVICE END OR PROGRAM CONTROLLED INTERRUPTION: If a channel end or device end condition exists, the processor passes the physical device address to the Reverse Pathfinding portion of the Pathfinding subroutine. This subroutine releases the path to the addressed device, and returns control to the processor. At this point, or if a program-controlled interruption is specified the processor initiates the posting action for all PCBE paging operations that were complete when the interruption occurred. The posting action is initiated by:

1. Selecting the first channel program executed in the current chain.
2. Locating the PCBE/IORCB from the SEEK argument of the channel program in the system table. If it is not a DRAM operation processing is as follows:

- If a read operation has been completed, the Page Posting subroutine is called to update the appropriate tables. When control returns, the processor selects the next PCBE for processing.
- If a write-operation has been completed, the processor calls User Core Release subroutine to release the storage occupied by the page associated with the PCBE. The Page Posting subroutine is then called to update the associated page table and returns control to the processor. The processor selects the next PCBE to be posted and reinitiates the posting action.

When all PCBES for a GQE have been posted, the GQE is passed to Move GQE to release the main storage space for it and the PCB. Processing then continues on the remaining channel programs.

When a complete channel program chain has been posted, the other chain is checked to see if it is to be posted.

When both chains of channel programs have been posted, or if the second chain is not available, the interrupt GQE is dequeued and its main storage space released.

A test is made before exiting from PDIP to see if the current interruption is a CE/DE. If it is not, the suppress flag (F5) is reset for the Page Drum Interrupt Processor, only. Exit is then made to the Queue Scanner.

If it is a CE/DE condition, the page drum directory is checked to see if a drum is in hold state. If yes, it is freed and set to ready. This allows the path to be reassigned to either drum. If a channel is down, PDIP will go to Paging Error Recovery (CEAAM) to obtain an alternate path, if possible.

In a duplex environment, this routine sets and resets the lock bytes associated with the TSI and system table to prevent the two CPUs from interfering with each other.

DRAM OPERATIONS: If the channel program is for a DRAM operation, processing is as follows:

Control is passed to CEAA43, a subsection of the Channel Interrupt Processor, to perform completion operations on the GQE/IORCB. Upon return, IORDTSI is checked; if it is on, meaning the TSI has been deleted, the GQE is discarded via a call to CEAMG. If it is not on, the completed DRAM I/O

request is queued on the TSI as a synchronous I/O software interrupt. The paging I/O count in the TSI is decremented by one. The remaining channel programs are then processed.

Program Interrupt Queue Processor (CEANA) Chart AG

The Program Interrupt Queue (PIP) processor analyzes the interruption code in the GQE to which its scan table queue entry points, and determines the type of user-program interruption that it specifies:

- A paging-relocation interruption requiring table construction and initiation of a page-read request.
- A shared-segment exception requiring shared-page-table searches.

Entry: CEANAA1

Modules Called: Enqueue GQE subroutine (CEAJQ entered at CEAJEN) scans table queues for a timer interrupt.

Search RSPI Table subroutine (CEAMS) locates the proper resident-shared-page-index (RSPI) entry in main storage for any specified shared-page-table number, or locates the address of the next available entry in the RSPI.

Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) reserves main storage for PCBs.

Find Page subroutine (CEANC) locates segment, auxiliary segment page, and external page table entries.

Paging (CEAMQ) is called to read a page table page into main storage.

Queue GQE on TSI subroutine (CEAAF) places the GQE pointer on the TSI's interruption queue.

Add Pages (CEAHQ entered at CEAHQA) performs dynamic expansion of a variable length page table.

Add Shared Pages (CEAQ6 entered at CEAH26) performs dynamic expansion of a variable length shared page table.

Exit: To Queue Scanner.

Operation: On entry, the processor tests the interruption code from the GQE to determine the type of error that has occurred. An interruption code of 16 denotes a segment-relocation error and a code of 17, a page-relocation error. The processing for each type of error is discussed below, and illustrated in Figure 14.

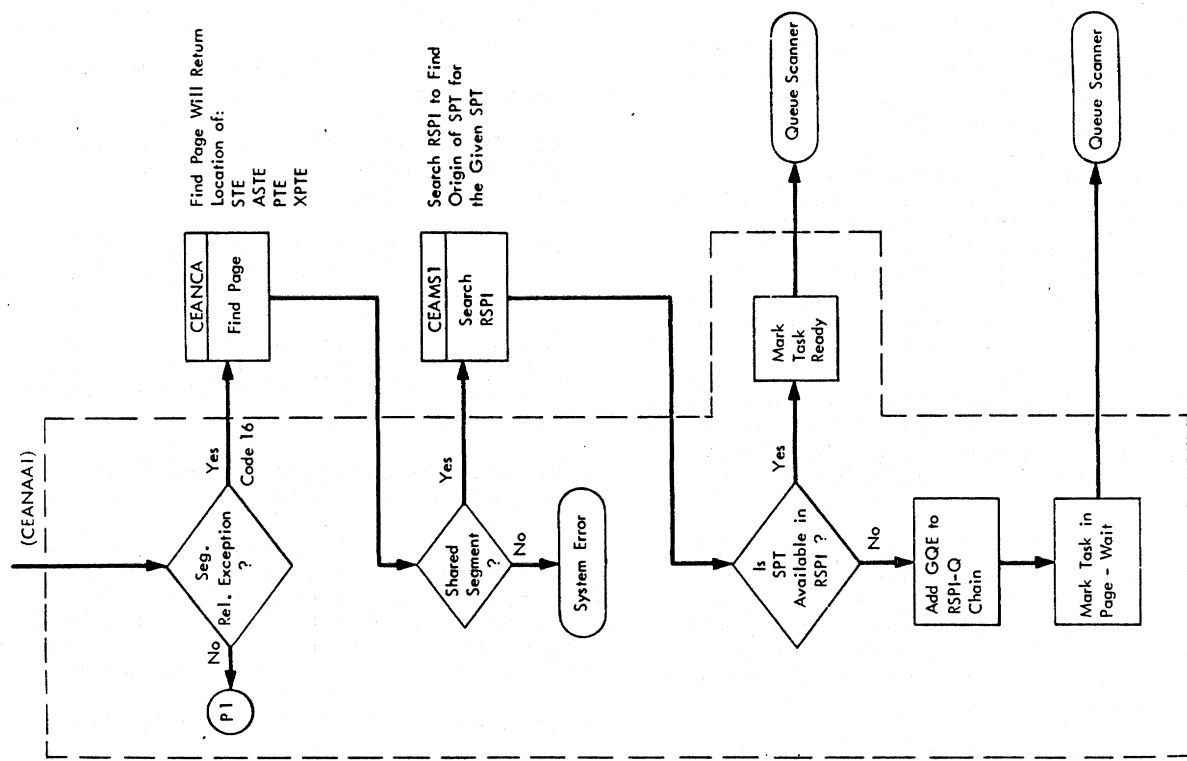
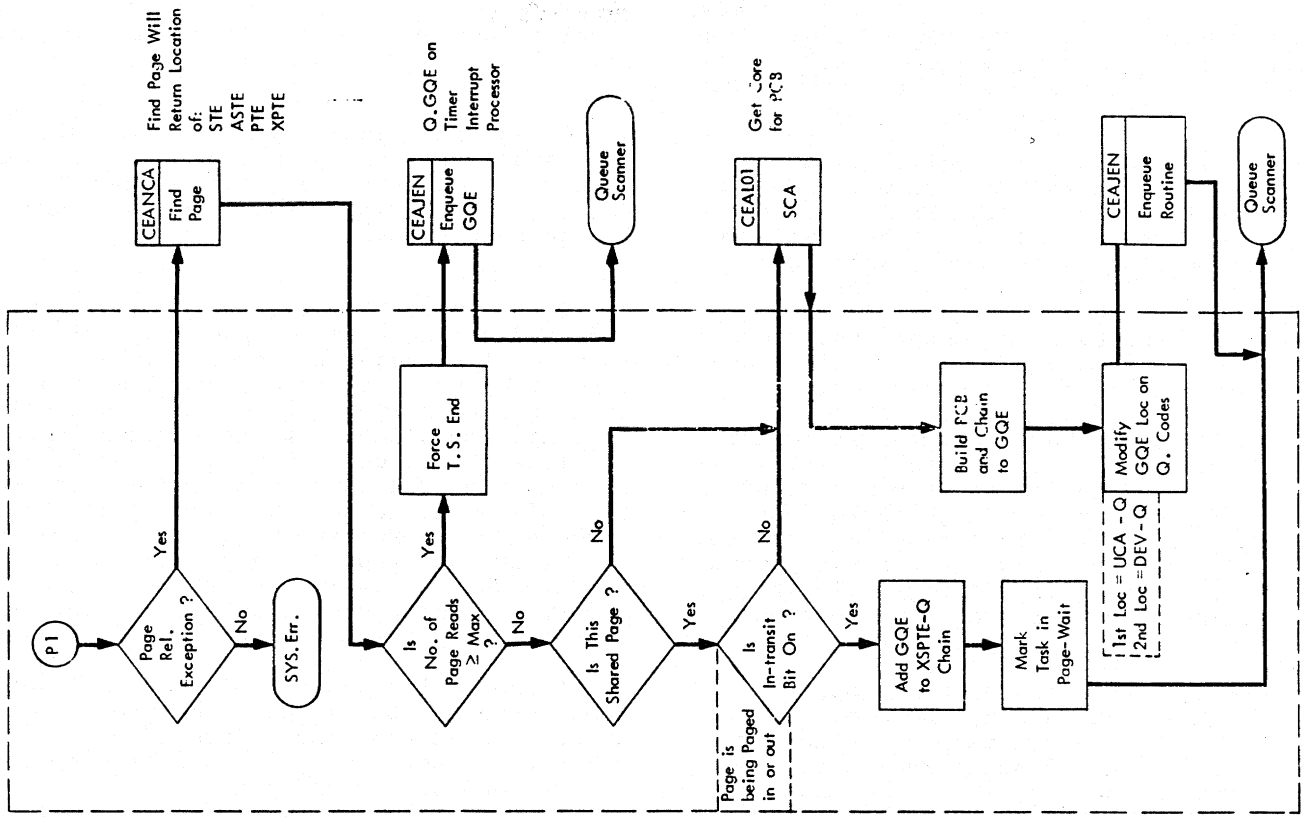


Figure 14. Activities of the Program Interrupt Queue Processor

SEGMENT RELOCATION ERROR PROCESSING: When a code of 16 indicating a segment-relocation interruption occurs, the processor performs the following:

- Loads into general registers the following: the address, from the TSI, of the first page of the XTSI; the untranslated address from the XTSI's control register save area; and the address of the Find Page subroutine.
- Calls Find Page to compute the segment-table-entry address, after which control returns to the processor.
- Tests for a shared-segment indicator in the auxiliary segment table (AST). If the shared indicator is not on, and the page table is in a page table page, CEAMQ is called to set up the read for the page table page. On return, this case is treated as a page relocation interruption (program interruption code 17).
- If the shared indicator is on, the sharing lock is set, the processor stores the shared-page-table number from AST in a general register and calls the Search RSPI subroutine to locate the RSPI entry associated with the shared segment. If Search RSPI determines the shared-page-table number is not in the RSPI, the processor calls SYSERR. Otherwise, the processor tests for page-table availability. If the table is unavailable, the interruption (code 5) is passed to the task by queuing the GQE pointer on the TSI and exiting to the Queue Scanner.

If the page table is available, its length and origin are stored in the segment table (CHASGT), the TSI's 'task-ready' indicator is set on, and control is transferred to the SVC queue processor at CEAHND, to free the GQE (that is, call the Move GQE subroutine) and exit to the Queue Scanner.

PAGE-RELOCATION ERROR PROCESSING: When an interruption code of 17 (indicating a page-relocation error) is found, the processor determines whether the associated XTSI is in main storage.

If the XTSI is in main storage, the processor retrieves the untranslated page address from the save area of the XTSI and calls the Find Page subroutine. Find Page computes the address of the page-table entry and returns it and a condition code to the processor.

If the segment is not variable length, and the condition code indicates that the page table entry is not assigned, an inter-

ruption is queued on the task's TSI (code 5).

For an unassigned page in a variable length segment, the processor sets up parameters and exits to Add Pages or Add Shared Pages to dynamically expand the variable length page table.

If the subroutine returns a condition code indicating that the page is available in virtual storage, the processor determines the number of read operations that have been performed for the task during its current time slice. If the maximum number of allowed read operations has been performed, the processor specifies a forced-time-slice-end in the GQE, and calls the Enqueue GQE subroutine to queue the GQE on the Timer Interrupt Queue processor's queue. When control returns, the field, SSTPLT, in the system statistical table is increased. PIP exits to the Queue Scanner.

If the maximum number of reads has not been performed for a task, the processor determines whether the addressed page is shared. If not, the processor sets up for a page-read operation by:

- Requesting PCB storage space from the Supervisor Core Allocation subroutine.
- Setting up a PCB with the segment and page numbers.
- Updating the GQE with information relating to the new PCB.
- Modifying Loc-on-Q in the GQE to request user storage space for the addressed page.
- If an external read is required, a GQE is set up for entry on the appropriate device queue.
- Calling the Enqueue GQE subroutine to place the new GQE on the scan table queue of the User Core Allocation Queue Processor.
- Exiting to the Queue Scanner when Enqueue returns control.

If a shared page has been requested, the processor locks the sharing lock, locates the external-shared-page-table entry and determines whether the requested page is being moved into main storage. If so, the processor queues the GQE to the XSPT by:

- Turning on the GQE's page-in flag.
- Adjusting the appropriate GQE chain fields in the XSPT.

- Updating pointer fields in the GQE.
- Updating TSI 'page I/O count' and 'page wait' fields.

When this processing is accomplished, the processor unlocks the sharing and TSI locks and exits to the Queue Scanner.

If the shared page is not being brought into main storage, the processor determines whether the requested page is being released from main storage. If so, the processor turns on the available flag in the page table, and queues the GQE on the XSPT as described previously. If the page is not being released, the processor performs the processing to initiate a page-read operation as described previously (sets up a PCB, etc.).

I/O Device Queue Processor (CEAA3) Chart AH

The function of the I/O Device Queue Processor is to process GQEs representing I/O requests. It services all of the device queues on the scan table, with the exception of paging drum queues.

Entry: CEAA31.

Modules Called: Pathfinding subroutine (CEAA5) this subroutine is entered at CEAA5P to obtain the actual path to a device. The Reverse Pathfinding portion (entry at CEAA5R) is entered to free assigned device paths.

Set Suppress Flag subroutine (CEAJQ entered at CEAJSF) sets the appropriate flag in the I/O device queue processor's scan table queue.

Dequeue GQE subroutine (CEAJQ entered at CEAJDE) removes GQE pointers from the specified queues.

Queue GQE on TSI subroutine (CEAAF) places a pointer to the specified GQE on the interruption queue in the affected task's TSI.

Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) reserves main storage for the use of the Generate and Enqueue Interrupt GQE subroutine and the Dequeue I/O Requests subroutine.

Dequeue I/O Requests subroutine (CEAAJ) removes GQEs in a device queue for a particular task and device.

Supervisor Core Release subroutine (CEAL1 entered at CEAL02) releases main storage after use.

Halt I/O subroutine (CEAAI) stops any I/O operation being performed on a specified path.

Paging I/O Error Recovery Control subroutine (CEAAM) attempts to recover and retry when no path can be found to a specified device on a paging request.

Start I/O subroutine (CEAAG) initiates an I/O operation across a specified path.

Generate and Enqueue Interruption GQE subroutine (CEABQ) generates and enqueues an interruption GQE when a condition code of 1 is returned in response to a start, halt, or test-I/O attempt and status is stored for a device other than the one addressed (that is, interruption code also stored).

Command Word Relocator subroutine (CEAAA) performs the operations required to relocate the channel-command word (CCW) addresses.

Page direct access subprocessor (CEAA6) builds a channel program to handle requests for paging operations on a direct access device.

Page Direct Access Interrupt Subprocessor (CEAA7) services interruptions occurring on direct access paging device.

Move GQE subroutine (CEAJQ entered at CEAJMG) determines whether further processing is specified by the GQE and either moves the GQE to another processor's queue or releases the space it occupies.

Exit: To Queue Scanner.

Operation: The appropriate scan table entry will have its lock byte (SCNF3LOK) set upon entry by the SETLOCK macro. If the GQE is moved or otherwise done away with, the pointer to the scan entry is saved. The SCNF3LOK is reset upon exit by way of the OPENLOCK macro.

Normally, only the symbolic device address is supplied, and any path available may be used for access to the device. If, however, the routine was reentered because of a paging error (GQEP on), or to sense the device, the path already specified is reused. The user may also specify a path in the IORCB.

In all cases, Pathfinding is called on entry to this routine, either to locate a path or to determine if the path specified is available.

If Pathfinding indicates 'path unavailable', the GQE is checked to see if it is for a paging operation. For paging opera-

tions, the Paging I/O Error Recovery routine (CEAAM) is called. On return, exit is to the Queue Scanner.

For other than paging operations, 'no path available' is set in the IORCB, and exit is to the Queue Scanner.

When Pathfinding indicates an 'available path', the I/O Device Queue Processor checks for the following conditions and performs the appropriate processing described in the following pages:

- Paging Request (GQERC off)
- Sense Request (GQEWS on)
- Halt I/O Request (IORHI on)
- User I/O Request

PAGING: If either the paging error (GQEPE) or paging interrupt (GQEIP) flag is on, the Paging Direct Access Interrupt Processor (CEAA7) is called. Otherwise, the Paging Direct Access Queue Processor (CEAA6) is called. On return, in either case, exit is to the Queue Scanner.

SENSE REQUEST: Start I/O is called to execute sense. If failure on sense is indicated by Start I/O, the operation is retried 512 times. If the failure is due to the control unit being busy, Set Suppress Flags is called to turn on the DIG busy flag, the path is freed via Reverse Pathfinding, and exit is to the Queue Scanner. For all other failures, this subroutine:

1. Suppresses the device via a call to CEAAD.
2. Dequeues all I/O for this task for this device by calling CEAAJ.
3. Exits to the Queue Scanner. If the failure on sense is corrected by the retry procedure or did not exist, Set Suppress Flags sets the F1 flag on, and exit is to the Queue Scanner.

HALT I/O REQUEST: The GQEHI flag is checked. If on, this indicates a successful Halt I/O and the IORCB has a CCW list to be started. (A user may not request a Halt I/O alone. The request must include a request for an I/O channel program to be started after the Halt I/O is completed.) Therefore, when GQEHI is on, Halt I/O processing is skipped and processing continues as described for USER I/O REQUEST, below.

When GQEHI is off, the Halt I/O subroutine is called. If the Halt I/O operation is successful, processing continues as described in USER I/O REQUEST, below.

If the Halt I/O subroutine indicates that its operation was unsuccessful, the I/O Device Queue Processor checks for the following conditions and takes the indicated actions:

1. Status Not for Addressed Device -- A CE/DE is simulated by calling the Generate and Enqueue Interrupt GQE subroutine. (The GQE is put on the Channel Interrupt Processor's queue.) Halt I/O is retried.
2. Unit Check or Unit Exception -- Halt I/O failure is indicated by turning on the IORBH flag. A sense operation is then performed.
3. Busy on Halt I/O (CC=2) -- If a multiplexor channel is involved, a minor system error is declared. On return, or if not a multiplexer channel, the await device end flag (GQEDE) and GQEHI are turned on. Set Suppress Flags is called to turn on F1. On return, exit is to the Queue Scanner.
4. Non-Operational Device on Test I/O or Halt I/O -- Halt I/O is retried.
5. Status Modifier Bit on in Stored CSW in 2nd TIO -- User I/O initiation is performed.
6. Status Modifier Only in Stored CSW on Halt I/O -- If the return code from Halt I/O indicates Test I/O was issued twice, GQEDE and GQEHI are turned on. Set Suppress Flags is called to turn on F1, and exit is to the Queue Scanner. Otherwise, user I/O initiation processing is invoked.
7. Status Modifier and Busy in Stored CSW on Halt I/O -- if the user has provided a Halt I/O retry counter (IORHF≠0) it is decremented. The entire path is freed via Reverse Pathfinding. Set Suppress Flags is called to set the DIG busy flag on and exit is to the Queue Scanner. When IORHF=0, it is set to one, indicating Halt I/O failure, and a sense operation is performed. If Halt I/O retry has been requested, it is retried.
8. Device End Only in CSW Stored by Halt I/O or Test I/O -- User I/O is initiated.
9. CE Only and Control Unit End in Stored CSW from Halt I/O or Test I/O -- Reverse Pathfinding is called to free the control unit and channel. If GQEHI is not on, GQEDE and GQEHI are turned on. Set Suppress Flags is called to turn on F1, and exit is to the Queue Scanner.

10. CE End Only in Stored CSW From Halt I/O or Test I/O -- The Test I/O is reissued. If the CSW was not stored, the channel is freed by calling Reverse Pathfinding. If GQEH1 is off, it and GQEDE are turned on. The F1 flag is set via Set Suppress Flags and exit is to the Queue Scanner.

If the reissued Test I/O results in a stored CSW with unit check or unit exception, GQEH1 is checked. If off, Halt I/O is retried. If on, the path is freed, the GQE is dequeued via CEAJDE and exit is to the Queue Scanner with the queue unsuppressed. If the CSW shows device end, a CE/DE is simulated via CEABQ. If GQEH1 is off, it and GQEDE are turned on. The F1 suppress flag is turned on, and exit is to the Queue Scanner. If the CSW shows control unit end, the channel and control unit are freed. GQEH1, if off, is turned on as is GQEDE. The F1 flag is set and exit is to the Queue Scanner.

11. All Other Errors -- Halt I/O is retried.

Note: Halt I/O Retry, where indicated above, is performed a maximum of 17 times. If still failing, the entire path is freed via Reverse Pathfinding. The GQE is dequeued via Dequeue GQE subroutine, and exit is to the Queue Scanner with the queue unsuppressed.

USER I/O REQUEST (User I/O Initiation):
The CCW list addresses are translated into real core addresses by calling the Command Word Relocator subroutine. If the return code indicates failure to translate, the entire path is freed by a call to Reverse Pathfinding. The GQE is dequeued, and exit is to the Queue Scanner with the queue unsuppressed. Otherwise, the Start I/O subroutine (CEAAG) is then used to initiate the channel program in the IORCB.

If the SIO is successful, the IORBP flag is checked to determine if the user has requested a PCI. If yes, a CE/DE/PCI is simulated by calling the Generate and Enqueue Interruption subroutine. On return, GQEH1 is set on. Set Suppress Flags is called to turn the F1 flag on and exit is to the Queue Scanner.

If IORBP is off, then the status returned from Start I/O is checked. If Start I/O is all right for disk, no further check is made. Otherwise, a CE/DE causes a simulated interrupt via calling CEABQ. The GQEH1 and GQEDE flags are turned on. The F1 suppress flag is turned on and exit is to the Queue Scanner.

Channel end only is processed as for Halt I/O with one exception: The user has provided a channel program for the disk containing a SEEK CCW as the first channel command that is not chained to the next CCW. If in this situation, software chaining is requested (IORSC on), a CE only requires a call to CEABQ to simulate CE/DE. Then, the channel only is freed; GQEH1 and GQEDE are turned on; F1 is set on, and exit is to the Queue Scanner. In all other cases, successful I/O initiation will cause the call to CEABQ and the channel freeing steps to be skipped.

Unsuccessful Start I/O situations are handled as described below:

1. Control Unit Busy -- If the device is a 2314 and the user has provided a retry count, the DIG busy flag is turned on, and exit is to the Queue Scanner.

If the device is not a 2314, the user requests a retry by setting the IORRS flag. This causes up to 512 retries of the SIO. If retry is not requested, or if unsuccessful, the IORIB flag is turned on to inform the user that SIO failed. Then the path is freed by calling Reverse Pathfinding; the GQE is dequeued; and exit is to the Queue Scanner with the queue unsuppressed.

2. Unit Exception -- The IORIB flag is turned on and the sense operation is performed.
3. All Other Cases of SIO Failure -- The IORIB flag is turned on; the path is freed; the GQE is dequeued; and exit is to the Queue Scanner with the queue unsuppressed.

Page Direct Access Interrupt Subprocessor (CEA7) Chart AK

This subprocessor answers a call resulting from the interruption of a paging operation on a direct access device. It also collects and stores in the system statistical table (CHASST) the number of reads and writes, for both private and shared pages, processed for each disk device.

Entry: CEA71

Modules Called: Supervisor Core Release subroutine (CEAL1 entered at CEAL02) releases main storage after use.

User Core Release subroutine (CEAL1 entered at CEAL04) releases a written main storage block.

Page Posting subroutine (CEAMP) updates the TSI and the XTSI pages.

Start I/O subroutine (CEAAG) issues the Start I/O command for all calling programs.

Dequeue GQE subroutine (CEAJQ entered at CEAJDE) removes a posted GQE.

Set Suppress Flags subroutine (CEAJQ entered at CEJJSF) resets the suppression flag after the successful start of I/O.

Page I/O Error Recovery subroutine (CEAAM) initiates retry procedures.

Generate and Enqueue Interrupt GQE subroutine (CEABQ) generates and queues an interruption GQE when an attempt to start I/O fails.

Pathfinding subroutine (CEAA5) frees the assigned device path.

Exits:

Normal - To Queue Scanner.

Error - To System Error Processor.

Operation: Prior to entry to this subprocessor, the affected device queue scan table entry lock byte (SCNF3LOK) is locked to preserve its integrity in a duplex environment.

On entry, the subprocessor expects to find a pointer to the original paging request GQE in a general register. The subprocessor stores the pointer, and then accesses the GQE to which it points to obtain the following:

- The related channel status word.
- The pointer to the direct access interface block (DAIB).

The subprocessor tests the GQE's 'waiting on sense' flag to determine whether the present interruption is caused by a delay in the return of a sense operation. If so, the Page I/O Error Recovery subroutine is called to retry the start-I/O operation. If the retry is successful, the Page I/O Error Recovery subroutine returns control to this subprocessor.

If the 'waiting on sense' flag is off, the subprocessor inspects the channel-status word for a unit-check indicator. When a unit-check is indicated, the GQE 'waiting on sense' flag is set in the device GQE and a sense-channel-command word is built in the DAIB to sense into the original device-GQE-sense area. The sense-command-channel-address word and physical-device address are passed to the Start I/O

subroutine where the start-I/O instruction is executed.

The Start I/O operation will return one of five codes indicating the results of its operation. These codes and the subsequent action taken by the subprocessor are as follows:

- Start I/O is successful, the channel program is in execution. The Set Suppress Flags subroutine is called to reset the suppress flag and, upon return, the subprocessor exits to the Queue Scanner.
- The selected channel is defective. The subprocessor exits to the Paging I/O Error Recovery Control subroutine in an effort to recover from the error.
- The control unit is busy. The operation is retried 256 times. If the operation is still not successful, an exit is made to the Paging I/O Error Recovery Control subroutine in an effort to recover from the error.
- Status was stored for the selected device. An interruption GQE is built and queued on the channel interruption queue in the TSI. The subprocessor then exits to the Queue Scanner.
- Solid Start I/O failure. The subprocessor exits to the Paging I/O Error Recovery Control subroutine in an effort to recover from the error.

When device-end is indicated in the channel-status word, normal posting action is also initiated. Any interrupts that are not a result of a device-end, device-end/channel-end, or unit-check condition will cause a linkage to the Paging I/O Error Recovery Control subroutine except for channel end or control unit end only conditions (or both, in combination) which cause an exit to the Queue Scanner. The following steps constitute a normal posting procedure:

- The first seek flag in the DAIB-entry header is examined to determine if this is the first interruption of the channel program. If it is on, all further posting action is bypassed and I/O is started on the next channel command word contained in the DAIB. When the interruption is the result of the first seek of the next-page operation (command chained to the previous-page operation) or the result of the last read or write of the channel program sequence, processing continues with the next step.

- The PCBE is located by using the contents of the DAIB-entry-interruption pointer, and the DAIB entry itself.
- By checking the read/write flag in the page-control block it can be determined whether or not to bypass the release of the main storage page.
- When the PCBE indicates a write operation, the PCBE pointer and control are passed to the User Core Release subroutine to release the main storage page referred to by the PCBE. The system-page-write-pending count, contained in the system table, is lowered by one. This operation is interlocked against simultaneous updating by setting the system table's lock byte. If the pending count goes to zero before it is lowered a major system error is recognized.
- When the PCBE indicates either a read or write operation the PCBE pointer is passed to the Page Posting subroutine, which updates the correct page tables.
- The 'PCB page I/O complete' flag is set to indicate the completion of a successful paging operation.
- The exit switch is checked to determine if this is the end of the channel program. If on, the following operations are initiated to purge the DAIB and paging GQE:
 - 1) The pointer to the DAIB and the size of the DAIB are retrieved from the GQE and passed to the Supervisor Core Release subroutine.
 - 2) The pointer to the GQE is passed to the Dequeue GQE subroutine to dequeue the GQE from the device queue.
 - 3) The pointer to the GQE is then passed to the Move GQE subroutine to release the main storage space occupied by the GQE and its PCB.
 - 4) The path that was used for the paging operation is released by calling the Reverse Pathfinding subroutine and, upon return, an exit is made to the Queue Scanner.
- When the exit switch is not set the DAIB entry interruption pointer is advanced to the next entry and the DAIB entry count is decremented by one. A check is made to see if all PCBEs are posted for this channel program segment. (Note: For a discussion of channel program segment, refer to the section on the Page Direct Access Queue

processor.) If not, control is returned to the normal posting procedure, described above, so that the next PCBE can be posted.

When all PCBEs are posted for a given channel program segment, processing is initiated for the next channel program segment.

The channel address word and the physical device address, which are picked up from the GQE, are passed to the Start I/O subroutine, which executes the start-I/O instruction. If the operation is initiated and the channel is proceeding with its execution, the Set Suppress Flags subroutine is called to set the F1 suppress flag, and control is returned to the Queue Scanner.

When the start-I/O instruction responds with a condition code other than zero the same action will be taken as outlined above for the initiation of the sense operation after a unit check occurs.

When the processor calls Locate Page, it first sets the task lock in the system table. If the page is shared, the sharing lock in the system table must also be set and reset.

When the page write pending count in the system table is to be updated, the processor must first set the system table lock and then reset it after the update processing.

Similarly, when Page Posting is called, the TSI lock must be set before the call and reset on return. The SCNF3LOK lock byte will be reset using the OPENLOCK macro prior to exiting to the Queue Scanner.

Page Direct Access Queue Subprocessor (CEAA6) Chart AI

The Page Direct Access Queue Processor (PDAQ) builds a channel program to handle the paging operations specified by the page control block pointed to by the first entry in the device queue.

Entry: CEAA61

Hardware Configuration Requirements: The only paging devices supported by this processor are those devices that can be attached to the IBM 2841 and 2820 storage control units.

Modules Called: Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) reserves main storage for the direct-access-interface blocks.

Start-I/O subroutine (CEAAG) issues the start-I/O instruction for the processor.

External Page Location Address Translation subroutine (CEAAE) translates a two-byte page-number field of a four-byte symbolic address into the physical I/O device address required by a seek or search channel command word.

Set Suppress Flag subroutine (CEAJQ entered at CEJJSF) sets the suppress flag in the device queue.

Paging I/O Error Recovery subroutine (CEAAM) initiates retry procedures.

Generate and Enqueue Interrupt GQE subroutine (CEABQ) generates and queues an interrupt GQE when the Start I/O subroutine returns an indication that it has failed.

Reverse Pathfinding subroutine (CEAA5R) frees the channel and the control unit after an I/O operation has been started on a disk.

Exits:

Normal - To Queue Scanner.

Error - To System Error Processor.

Operation: The affected device queue scan table entry lock byte (SCNF3LOK) has been locked prior to entry into this subprocessor to preserve its integrity in a duplex environment.

On entry, the PDAQ processor calculates the storage space required for a direct access interface block (DAIB) in which to store the necessary interface data for transferring pages between the direct-access device and main storage. The DAIB is a resident, private table which interfaces between the PDAQ processor and the Page Direct Access Interrupt subroutine and exists only during the life of a paging operation. The maximum size of a DAIB is one page in length (4,096 bytes). For each DAIB, the processor requires a 32-byte DAIB header, a 64-byte general register save area, and a 16-byte DAIBE header. Additional storage requirements are calculated by using the PCBE count field in the GQE as a factor, as follows:

- An 8-byte seek-and-search-argument table for each PCBE.
- A 40-byte area for each read or write channel program to be constructed.
- A 72-byte field for each write-with-read-back-check channel program to be constructed.

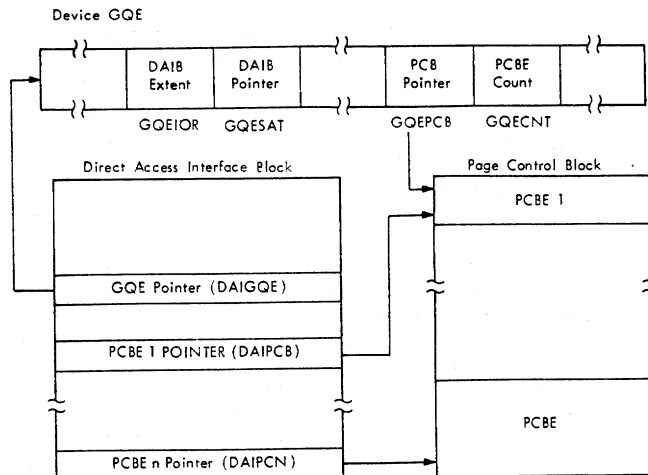


Figure 15. PDAQ Processor cross-referencing between the GQE, PCB, and the DAIB

When the space has been calculated, the processor specifies the requirement to the Supervisor Core Allocation subroutine, to which control is transferred. When storage is allocated, the PDAQ processor cross-references it in both the GQE and in the PCB as illustrated in Figure 15.

The PDAQ processor then selects the first PCBE for processing. The PCB's bypass and null flags are tested, and if either is on, the processor skips this request and initiates the processing of the next PCBE. If neither flag is on, the processor calls the External Page Location Address Translator subroutine to translate the page address to a physical address. When page control returns, the processor stores the returned address in the DAIB (that is, sets up a seek-argument table entry) and sets up the following in the first DAIBE space:

- The initial seek channel command word (nonchained), and the second seek CCW (command chained).
- The search CCW (command chained).
- The transfer in channel CCW.

The processor determines whether a read or write operation is to be performed. If the operation is a read, the processor sets up a read CCW entry for the DAIBE.

If a write operation is specified, the processor determines whether a write-check is specified. If not, the processor sets up a write-CCW for the DAIBE. If a write-check is specified, the processor sets up the same set of CCWs described above, and the following additional CCWs:

- Seek (chained immediately to the following search).
- Search.
- Transfer in channel (TIC).
- Read Skip (that is, write check).

In either of the above cases, the first seek CCW generated is not command chained to the second seek CCW. This allows the channel to be released early so that other outstanding I/O requests on different control units may be fulfilled by additional devices attached to the channel. The second seek is used to set up control unit registers to effect chaining to the next CCW (the search CCW).

In the case of the write operation with a write-check specification, the third seek CCW repositions the head when a page overflows from one track to the next. The third seek CCW is immediately command chained to the subsequent search CCW.

When the above processing has been completed, the processor places a PCBE pointer to the paging operation in the DAIB entry header so that the Page Direct Access Interrupt (PDAI) subroutine may match the page-operation-interrupt GQE against the PCBE that initiated the paging operation.

The processor constructs a DAIB entry header for each DAIB. The processor initializes the pointer to the first DAIB entry, and sets on the first-seek flag to differentiate between the first and subsequent seek operations. That is, to specify the following to the PDAI subroutine:

- No posting action is required.
- The start-I/O operation on the next CCW is to be initiated.

The channel address word (CAW) and the physical address of the device are passed to the Start-I/O subroutine together with an indicator that the call is to start I/O on a disk. The Start I/O routine issues the start-I/O instruction. A return bit-flag from Start-I/O is tested. If it is off, the control unit end condition is set in the status bits and Reverse Pathfinding is called to free the channel and control unit.

Upon return from Reverse Pathfinding or if the bit flag was on, the start I/O operation will also return one of five codes indicating the results of its operation. These codes and the subsequent action taken by the subprocessor are as follows:

- Start I/O is successful, the channel program is in execution. The Set Suppress Flags subroutine is called to set the suppress flag and, upon return, the SCNF3LOK lock byte is reset by the OPENLOCK macro prior to exiting to the Queue Scanner.
- The selected channel is defective. The subprocessor exits to the Paging I/O Error Recovery Control subroutine in an effort to recover from the error.
- The control unit is busy. The operation is retried 256 times. If the operation is still not successful, an exit is made to the Paging I/O Error Recovery Control subroutine in an effort to recover from the error.
- Status was stored for the selected device. An interruption GQE is built and queued on the channel interruption queue in SCANT. The SCNF3LOK lock byte is reset by the OPENLOCK macro prior to exiting to the Queue Scanner.
- Solid start I/O failure. The subprocessor exits to the Paging I/O Error Recovery Control subroutine in an effort to recover from the error.
- For each start I/O attempt that has failed due to a busy return, the PDAQ processor sets the appropriate lock in the system table, updates the count of start I/O failures, and resets the system table lock.

Channel Interrupt Queue Processor (CEAA4) Chart AK

The functions of the Channel Interrupt Queue Processor (CIP) are to:

- Recognize terminal I/O interruptions that are to be processed by the Terminal Communications Subprocessor (CEATC) and call that processor to handle them.
- Locate the request GQE which initiated the I/O operation, and perform the required functions of freeing devices, freeing channels, freeing the control unit, paging operations, or I/O operations; and recognize and process synchronous interruptions (that is, interruptions represented by active I/O request GQEs pointed to by the correct device queue entry).
- Recognize valid asynchronous interruptions and distinguish between the initial and subsequent asynchronous interruptions so that each is processed properly, and inform the affected tasks of the occurrence of the interruptions.

Entries:

- CEAA41 - by the Queue Scanner.
- CEAA42 - to perform the locate-page function
- CEAA43 - to decrement page hold counters.

Modules Called: Supervisor Core Release subroutine (CEAL1 entered at CEAL02) releases main storage occupied by work areas.

Terminal Communications Subprocessor (CEATC entered and CEATC1) is called to process I/O interrupts from terminal devices operating under RTAM or MTT.

Pathfinding subroutine (CEAA5R) translates the actual path address into a symbolic address and releases a path or a portion of a path.

Move GQE subroutine (CEAJQ entered at CEAJMG) determines whether further processing is specified by the GQE and either moves it to another processor's queue or releases the space it occupies as well as the space occupied by any associated PCBs.

Task Initiation subroutine (CEAMC) sets up a new TSI to initiate a new task.

Dequeue GQE subroutine (CEAJQ entered at CEAJDE) removes GQEs from the scan table.

Queue GQE on TSI subroutine (CEAAF entered at CEAAFQ) queues the specified GQE on the task's TSI interruption queue.

Generate and Enqueue Interrupt GQE subroutine (CEABQ) generates a GQE and puts it back on CIP's queue to simulate a CE/DE.

Set Suppress Flag subroutine (CEAJQ entered at CEAJSF) sets the suppress flag in the processor's scan table queue.

Dequeue I/O requests subroutine (CEAAJ) returns I/O requests for a device to the virtual storage routine.

Remote Job Entry (RJE) Asynchronous I/O Interrupt subroutine (CEABA) processes asynchronous interruptions from RJE devices.

Remote Job Entry (RJE) Synchronous I/O Error Interrupt subroutine (CEABB) processes synchronous I/O error interruptions from RJE devices.

Remove Device From Task Processor (CEAAD) removes from or suppresses a symbolic device in the task-symbolic-device list.

Locate Page subroutine (CEAML) provides the location of any page-table entry or external-page table entry.

Exits:

Normal - To Queue Scanner. Normal exit may also be made to the I/O Device Queue Processor (CEAA3) after a return from the sense operation.

For immediate sense - To I/O Device Queue Processor.

Operation: The Channel Interrupt Queue processor receives hardware I/O interruptions and performs the following:

- Provides preliminary processing for paging-interruptions from direct access paging devices other than drums.
- Queues task I/O interruptions.
- Invokes task initiation.
- Releases data paths (or parts of paths) to which ending interruptions apply.
- Automatically invokes sense procedures under device failure conditions.
- Provides special processing when requested in IORCB.

On entry, the processor establishes its base register and calls the Set Suppress Flag subroutine to set on the suppress flag in the CIP scan table entry to lock out the Queue Scanner. The processor then processes all GQE's pointed to by the entries in its queue before returning control to the Queue Scanner.

The first step in processing a GQE is to test its channel status word (CSW). If the value of the status bits in the CSW is zero, there is no work for CIP to do. Hence, the Move GQE subroutine is called to remove the GQE pointer from the Channel Interrupt Queue Processor's queue. The processor then selects the next GQE for processing.

The processor determines the interruption source by calling the Pathfinding subroutine to translate the actual path address of the device to its symbolic device address. When control returns to the Channel Interrupt Queue processor, a check is made to determine whether the actual path was successfully translated. If not, the processor reports an error to SYSERR. When control returns to the processor the Move GQE subroutine is called as described above and processing continues with the next GQE.

When the CSW value is nonzero, CIP checks to see if the device is a terminal under control of RTAM or MIT (DEVRT or DEVMT on in the device group table). In this case, a 64 byte save area is requested from Supervisor Core Allocation and its address is passed to the Terminal Communications Subprocessor (CEATC) along with pointers to the device group table, the asynchronous entry (if applicable), and the interrupt GQE.

When the actual path is successfully converted to a system symbolic device address, the return parameters, passed to the Channel Interrupt Queue processor by the Pathfinding subroutine, are saved in the interrupt GQE. The system symbolic device address is saved in GQEDEV and the device type is saved in GQEDT.

At this juncture, the processor determines whether the I/O interruption is synchronous (has an associated request GQE in a device queue) or is asynchronous (no request GQE exists). To locate the correct device queue GQE, the processor converts the system-symbolic-device address to a scan-table-entry pointer. This is accomplished by multiplying the system-symbolic device address by sixteen and then adding the beginning address of the scan table to the result. If the scan-table-entry specified by the calculated pointer has work in queue that is marked as busy, the interrupt GQE is classified as synchronous. When the processor has determined that the interrupt GQE is synchronous, the scan table entry lock byte (SCNF3LOK) is set using the SET-LOCK macro. This lock byte ensures that no other supervisor processor will operate on the associated entry in a duplex environment until the lock has been reset upon exit from this processor. An interruption is synchronous if its associated scan-table-device-queue entry has both its work-in-queue and F1-suppress flags on. If the DIG busy flag is on, the I/O Device Queue Processor (CEAA3) is involved in retrying I/O. The path to the device is freed via Reverse Pathfinding (CEAA5R), the DIG busy flag is turned off, and the interrupt GQE is discarded. When the device queue has no pending request GQE, the interrupt GQE is classified as asynchronous.

Processor activities in processing these two types of interruptions are described on the following pages.

ASYNCHRONOUS-INTERRUPTION PROCESSING: The first step in asynchronous-interruption processing is to test the CSW in the GQE for one of the following:

- An attention indicator only on.

- Channel-end/device-end indicators only on.
- Channel-end/device-end/unit-exception indicators only on.
- Device end only.

If none of these indicators are on, the processor calls SYSERR, and then calls the Move GQE subroutine to dequeue the GQE from the CIP's queue as described earlier. If the CSW contains one of these indicators, the processor checks the device group table to see if a TSI pointer exists for an interruption. If one does, it is checked to see if it equals X'00000003'. If it does, this identifies the asynchronous interruption for the master system programmer's task. CIP then enters the RSS module at the "activate RSS because of MSP attention" entry point after destroying the interrupt GQE and turning off the F4 suppress flag.

The table is checked to see if a task is to be created for the first occurrence of an asynchronous interruption on the device in question. If this is the case, the channel interrupt processor calls the Task Initiation subroutine which performs the necessary processing to set up a TSI. When control is returned, the Channel Interrupt Queue processor tests a return code register to determine whether the TSI was created. If not, the user's line must be reenabled so that he can be reconnected to the system when it is able to accept him. A hardware error could prevent reenabling the line. The GQE is deleted and the next GQE is selected for processing.

If a task was successfully initiated, or if one existed initially, the CIP's GQE is dequeued from the interruption queue and queued on the appropriate TSI's interruption queue. The dequeuing is effected by calling the Dequeue GQE subroutine. When control returns to CIP, the queue GQE on TSI subroutine is called to queue the GQE on the TSI's interruption queue. The next GQE is then selected for processing by CIP.

SYNCHRONOUS-INTERRUPTION PROCESSING: When an interruption is classified as synchronous, the processor first frees the device and then processes the paging or I/O request.

Device Freeing: CIP tests the status field of the interruption GQE to determine whether there are any devices that must be freed (that is, set to the non-busy state) in the pathfinding tables. If so, the processor calls the Pathfinding subroutine. Individual testing and processing activities are discussed below.

If the GQE status signals a device-end and/or device-failure without an associated channel-end or channel-failure, the control-unit-end flag in the GQE is tested. If it is not on, the processor requests that Pathfinding free the control unit and the I/O device. If the flag is on, a free-device-only request is made to Pathfinding.

When the GQE status signals both a device-end/device-failure and a channel-end/channel-failure, the processor requests pathfinding to free all units (channel, control unit, and device).

In each of the three cases described above, the processor checks an addressing-error indicator returned by Pathfinding. If the indicator is on, SYSERR is called, after which the processor dequeues the interrupt GQE and processes the next interruption.

If the GQE status field does not indicate a device-end or failure condition, the processor tests it for a control-unit-end indicator. If only this indicator is on, the processor requests that Pathfinding free the channel and control unit. On return from Pathfinding, the Channel Interrupt Queue Processor determines whether a device addressing error occurred. If so, SYSERR is called. When control returns to the processor, Move GQE is called to remove the GQE from the CIP's queue. When control returns to the Channel Interrupt Queue Processor, the next GQE is selected for processing.

If none of the conditions described previously exists, the processor checks for a channel-end-only indicator in the status field. If this indicator is on, and the device is a disk, the channel and control unit are freed. After they are freed, or if the device was not a disk, the processor issues a test-I/O instruction with I/O interruptions inhibited. The processor then determines whether status was stored as a result of the test-I/O operation. If not, the processor issues a free channel and control unit request to the Pathfinding subroutine. If Pathfinding returns a device addressing error indicator, the processor calls SYSERR and performs the processing described for the control-unit-end-only condition.

If status was stored in response to the test-I/O operation, the processor determines whether status was stored for a device other than the one addressed. If so, a new interruption GQE must be generated and queued on the CIP's queue. In order to accomplish this, the processor requests GQE-storage from Supervisor Core Allocation and, on return, calls the Generate and Enqueue Interrupt GQE subroutine.

The subroutine generates a new GQE and queues it at the end of the CIP's queue. When this is accomplished and control returns to the Channel Interrupt Queue Processor, the Supervisor Core Release subroutine is called to release the original GQE's space, and the next GQE is selected for processing.

When the status is stored for the addressed device, the new status is OR'ed into the CIP's GQE status field, and the Channel Interrupt Queue Processor checks the new status for a channel-end-only indicator or all zeros. If either condition is indicated, the processor calls the Pathfinding subroutine to free the channel only, and perform subsequent processing as described in the case where test-I/O operation did not store status.

When the new status contains neither a channel-end-only indicator nor all zeros, CIP ORs in the additional status and begins again at device freeing.

When the original GQE status field did not indicate a channel-end-only condition, the Channel Interrupt Queue Processor tests the status field for channel-and-control-unit-end only indicators. If only these indicators are on, the Channel Interrupt Queue Processor saves the channel-command-word address in the device queue GQE, and requests that Pathfinding free the channel and the control unit. The subsequent processing is the same as that described above for freeing a control-unit-only. If the channel and control-unit-end indicators are not the only ones and none of the other previously described combinations existed, the processor assumes that the interruption is not the result of an I/O-end condition, and performs the paging operation checking described in the following paragraphs.

Paging Operations: After freeing the required devices, if any, the Channel Interrupt Queue Processor determines whether a paging operation has been done. If not, the Channel Interrupt Queue Processor performs the I/O-operations procedure described later. If a paging operation has been done, the Channel Interrupt Queue Processor moves the CSW from its GQE to the device queue's GQE. The device GQE's await-sense flag is then tested. If await sense or an error is found, CIP calls CEAA31 directly with the GQE path error flag (GQEPE) on. Otherwise, the paging interrupt flag (GQEIP) is set in the request GQE, and the next GQE is processed.

I/O Operations: If the GQE's IORCB flag is on, one or more of the following non-paging operations is performed:

- Return from sense.
- Normal end, which involves:

- 1) Software command chaining.
- 2) Lowering page-hold counters.

The processor's first step in performing this processing is to check the GQE await-sense flag. If the flag is on, the return from sense processing, described in the following paragraphs, is performed. If the flag is not on, the normal-end processing, described later, is entered.

The return-from-sense processing begins by storing in the IORCB the sense status information, and setting the sense-condition code in the IORCB to zero. The processor then tests the failure indicators in the status field. If these indicators are on, a sense operation has failed and the processor determines whether the failing command was a sense CCW, a read-home-address CCW, or a read-record zero CCW.

This is accomplished by comparing the address for the next CCW which is stored in the channel-status word against the address of the sense command in the IORCB plus 8. The sense command in the IORCB is always located in a fixed position and therefore its address can be calculated as the sum of the IORCB pointer plus a constant. If the two addresses match, the processor assumes that it was the sense command that failed and sets on the IORCB's sense-failed flag. When the addresses do not match, the processor assumes that the read HA or read R0 CCW failed. In this event, the IORCB's read-R0-failed flag is turned on and, if unit check or unit exception, the CCW list is set to do sense only with read-in suppressed. The device queue is unsuppressed and the sense operation is performed to clear the control unit.

At this point, or, if no sense-operation failure was indicated, the Channel Interrupt Queue Processor effects the dequeuing of all GQE entries in the appropriate device queue that point to the affected task. This is accomplished by:

- Requesting work storage allocation from the Supervisor Core Allocation subroutine.
- Calling the Dequeue I/O Requests subroutine, when storage is allocated, to remove the queue entries from the interruption queue and queue them on the I/O interruption queue.

- Requesting the release of the work storage by calling Supervisor Core Release.

When control returns, CIP resets the 'scan table suppress' flag for the associated device queue processor's queue. The Move GQE subroutine is called to remove the interrupt GQE's pointer from CIP's queue and to release the storage space it occupied. When the space has been released, the SCNF3LOK lock byte is reset using the OPENLOCK macro and control is then transferred to the I/O Device Queue Processor (CEAA3) with the affected device GQE address in general register 1.

Normal-end processing is entered when the GQE await-sense flag is off. The Channel Interrupt Queue Processor moves the key, status, and count portion of the interrupt GQE's CSW to the Device Queue Processor's GQE. The Channel Interrupt Queue Processor then tests the status indicators in the CSW. If the device channel-end-only indicator is present, the processor checks for the following:

- The absence of a 'software command chain' flag and a 'length of page list' field of zero in the IORCB.
- A resetting of the GQE's dequeue flag.

If these conditions are met, the Channel Interrupt Queue Processor calls the Dequeue GQE subroutine to dequeue the GQEs from the Device Queue Processor's queue and on return from the Dequeue GQE subroutine calls the Queue GQE on TSI subroutine to queue the GQE on the TSI's interruption queue. When control returns, the Channel Interrupt Queue Processor calls the Move GQE subroutine to remove the GQE pointer from CIP's queue. When control returns to the processor, the next GQE is selected for processing.

If the 'software command chain' flag is on, the Channel Interrupt Queue Processor enters its chaining function.

If buffer pages are present in storage, the processor decrements the page-hold counters and the TSI I/O counter as each page hold counter goes to zero. Both of these functions are described later in this section.

If the GQE's 'dequeue' flag is on, all I/O requests for the task must be dequeued from the device queue processor's queue. This is effected by calling the Dequeue I/O Requests subroutine as described previously for the return-from-sense operation.

If the device-end-only indicator is on, the processor checks the await-device-end

flag. If it is off, the processor performs the normal-end processing, described above, for the device/channel-end-only condition. If the 'await device end' flag is on, it is turned off and the 'dequeue GQE' flag is checked. If the flag is on, the Channel Interrupt Queue Processor must suppress the device's symbolic address entry in the task-symbolic-device list (that is, the malfunctioning device is interlocked against any further parallel activity from the task to the device). The Channel Interrupt Queue Processor performs this function by:

- Requesting work space from the Supervisor Core Allocation subroutine.
- Calling the Remove Device From Task subroutine, which suppresses the device and returns control to the processor.

The Channel Interrupt Queue Processor then calls the Dequeue I/O Requests subroutine to remove the I/O GQEs for the affected task from the device queue and put them on the TSI interrupt queue. The work storage is then released by calling the Supervisor Core Release subroutine. When control returns to the Channel Interrupt Queue Processor, the Move GQE subroutine is called to remove the interrupt-GQE entry from the CIP queue. The Channel Interrupt Queue Processor then selects the next GQE for Processing.

If the dequeue flag is off, the processor checks for the 'software command chain condition' flag in the IORCB. If it is on, the software-command-chaining function is entered. If the flag is off, the IORCB's 'buffer pages in storage' field is checked. If it is non-zero, the processor decreases the page-hold counters and the TSI I/O counter as each page hold counter goes to zero. If the field is equal to zero after the counters have been updated, the Channel Interrupt Queue Processor calls the Set Suppress Flag subroutine to reset the device queue's suppress flag. When control returns, the Channel Interrupt Queue Processor selects the next GQE for processing. At this point, the Channel Interrupt Queue Processor calls the Move GQE subroutine to remove the old GQE from its queue, and when control returns, continues processing of the new GQE. If there are no more GQE's in its queue, the processor exits to the Queue Scanner.

If the device-end-only status indicator was not on, the processor checks for a program-controlled-interruption indicator in the CSW's status field. If the indicator is on, the Channel Interrupt Queue Processor tests for a force-device-end condition (that is, checks the forced-end flag in the IORCB). At this point, the channel

interrupt queue processor performs the await-device-end-testing and subsequent processing described above for the device-end-only condition.

If the forced-end flag is off, the Channel Interrupt Queue Processor tests the CSW/CCW address to determine whether it is within the bounds of the IORCB's CCW list. This is accomplished by calculating the last address of the CCW list in the IORCB and comparing the CSW command word address with it. If the CSW address is within the range of the beginning and ending address of the CCW list, the Channel Interrupt Queue Processor signals itself that the Device Queue Processor's scan table suppress flag must not be set off, and calls the Dequeue GQE subroutine to remove the GQE from CIP's queue. The processor then calls the Queue GQE on TSI subroutine to queue the GQE on the TSI's interruption queue.

When the CSW command word address is not within the range of the calculated length of the CCW list, CIP assumes that I/O chaining is in effect and performs the function described above for the forced-end condition but does not reset the Device Queue Processor's suppress flag.

If the status field does not contain a program-controlled-interruption indicator, the processor tests for a unit-check/exception indicator. If either indicator is on, the await-sense flag is turned on in the Device Queue Processor's GQE. The Channel Interrupt Queue Processor calls Move GQE to dequeue the interrupt GQE from its own queue. When control returns, the next GQE is selected and processing continues.

When the status indicators reflect conditions other than those discussed above, CIP assumes that there is some error and that no sense operation is required. Hence, the Channel Interrupt Queue Processor enters its return from sense procedure at the point where all the task's I/O requests for the failing device are dequeued from the Device Queue Processor's queue, and put on the TSI interrupt queue.

The software-command-chaining function is entered from normal-end processing when the IORCB's software-command-chaining flag is on. The first step in this function is to determine whether the end of the IORCB's CCW list has been reached. This is accomplished by calculating the end-address of the CCW list and comparing it with the CSW address in the GQE.

If the address of the CSW word is not equal to or greater than the calculated address, the next CCW in the list is pre-

pared for execution by adjusting the new relative starting address of the CCW list (that is, pointing it to the next CCW to be executed). The processor then updates the device queue, and its own queue as described for a successful paging operation.

If the CSW address is equal to or greater than the calculated address, the end of the CCW list has been reached, and the Channel Interrupt Queue Processor returns to the normal-end processing at the point where the length-of-page list field in the IORCB is tested.

The page-hold-counter-decrementing function is entered when the length-of-page list field contains a nonzero value. The purpose of this function is to update the page-hold counters in the external-page table to reflect the release of virtual-storage-buffer pages being held in main storage. The processor begins this function by computing the number of the first page in the IORCB's page list. The Channel Interrupt Queue Processor then passes the page address and TSI's pointer to the Locate Page subroutine. Locate Page searches the XTSI pointed to by the TSI for the storage addresses of the associated page table entries. If the page entry is located, the subroutine returns the page-table (or shared-page table) address and the external-page table (or external-shared-page table) address to the Channel Interrupt Queue Processor. At this point the Channel Interrupt Queue Processor tests the page-hold counter. If it is not equal to zero, the counter is lowered by one. If it is already a minor system error is declared.

If the page entry was not located, or if the counter is equal to zero, a SYSERR call is made. Upon return from SYSERR, or after subtracting one from the page-hold counter, CIP lowers the page-list-length field, and then tests its content. If it is nonzero, the processor repeats the steps described above. When the field is zero, all page-hold counters have been updated and the page list processing is complete. The channel interrupt queue processor now calls the Dequeue I/O Requests subroutine to decrease the page hold counter.

When CIP has processed all the GQE's queued on its queue, the following exit procedures are performed:

- The Set Suppress Flag subroutine is called to unlock CIP's queue.
- The scan table entry lock byte (SCNF3LOK) is reset using the OPENLOCK macro.

- CIP exits to the Queue Scanner.

Remote Job Entry Asynchronous I/O Interrupt Subroutine (CEABA) Chart A1

The function of this subroutine is to process all asynchronous I/O interruptions from Remote Job Entry devices which originate in main storage. This includes performing line control error recovery procedures.

Attributes: CEABA is a serially reentrant subroutine residing in main storage that operates in the supervisor state.

Assumptions and Restrictions: This subroutine is called only by the Channel Interrupt Processor to handle RJE line control operations. It is not parallel reentrant; registers are saved internally.

Entries:

- CEABA1 - entry from Channel Interrupt Processor (CEAA4).
- CEABA2 - return entry point from RJE Synchronous I/O Error subroutine (CEABB).
- CEABA3 - return entry point from Create Real Time Interrupt subroutine (CEAS7).

When entered at CEABA1, the following general registers contain significant data, as indicated:

- Register 2 - address of the device group table
- Register 3 - address of the TSI
- Register 11 - address of the device queue (SCANTE)
- Register 12 - address of the asynchronous entry in the device group table
- Register 13 - address of the interruption GQE
- Register 14 - return address
- Register 15 - entry point for this routine

Modules Called: Supervisor Core Allocation (CEAL1 entered at CEAL01) is called to get core for the asynchronous interruption GQE to be sent to the virtual memory task.

The Dequeue GQE subroutine (CEAJQ entered at CEAJDE) is called to remove the error GQE from the Channel Interrupt Processor's queue.

The Start I/O subroutine (CEAAG entered at CEAAG1) is invoked to start I/O on retry or sense operations.

The Task Communication Control subroutine (CEAAN entered at CEAAN1) is called to send error messages to the operator.

The RJE Line Control subroutine (CEABC entered at CEABC2) is called to initiate line control functions.

The Enqueue GQE subroutine (CEAJQ entered at CEAJEN) is called to put the error or retry GQE on the device queue.

The Set Suppress Flags subroutine (CEAJQ entered at CEAJSF) is called to set the F4 suppress flag to prevent interruption in the device queue during retry.

The Pathfinding subroutine (CEAA5 entered at CEAA5P) is called to assign paths for retry or sense operations.

Reverse Pathfinding subroutine (CEAA5 entered at CEAA5R) is called to release the path when CEABA is entered from the Channel Interrupt Processor.

Generate and Enqueue Interrupt GQE (CEABQ entered at CEABQ1) is called if the Start I/O subroutine returns a non-busy condition code of one. CEABQ creates an interruption GQE and places it on the Channel Interrupt Processor's queue.

The Queue GQE on TSI subroutine (CEAAF entered at CEAAFQ) is called to put an interruption GQE on the task interruption queue after an enable operation when Start I/O fails on the prime operation.

The Read Time subroutine (CEAS6 entered at CEAS6A) is called to get the current time before a real time interruption is set up.

The Set Real Time Interval subroutine (CEAS7 entered at CEAS7A) is called to set a real time interval for recycling when a busy return from the Pathfinding subroutine occurs.

Exits: Normal exit is to the Channel Interrupt Processor to one of four return points set up in general register 14. Return is made in line when CIP is to dequeue the GQE, put it on the TSI, and process the next GQE. Return is to CEAA44 when CIP is to discard the interrupt GQE and process the next. Return is to CEAA45 with a pointer to the next GQE to be processed in register 13, when no processing of other GQEs is required by CIP. Return is to CEAA46 when CIP is to have the interruption GQE put on the task's TSI and process the next GQE. An asynchronous event

code specifying that the operation was successful or the cause of failure is set in all interruption GQEs to be returned to the task.

Error Conditions: The System Error Processing routine (CEAIS entered at CEAIS1) is invoked via the ERROR SVC for the following error conditions:

- Error return from Reverse Pathfinding (7500)
- Invalid return code from RJE line control (7501)
- Invalid command code (CCW) encountered (7502)
- Channel end or device end on unexpected command (7503)
- Unexpected interruption from sense operation (7504)
- Invalid error condition (no sense bits) (7505)
- Error return from Pathfinding (7506)
- TSILCK locked more than 50 microseconds (7507)
- DEVLOCK locked more than 50 microseconds (7508)

Operation: On entry, the 'RJE disable interrupt' flag is checked in the device's asynchronous entry of the Device Group table (DEVAE). If it is on, the Reverse Pathfinding subroutine is called to free the entire path. The return code from Reverse Pathfinding is tested; and if an error is indicated, an ERROR SVC is issued. On return from the System Error Processor, exit is to the Channel Interrupt Processor at CEAA44.

Otherwise a test is made to determine if this is a Halt I/O interruption. If it is, the RJE line control subroutine is called to perform the operation indicated by the appropriate flag in DEVAE (DISABLE/ENABLE/PRIME). The return from CEABC is examined:

- If successful, the device's scan table entry (SCANTE) is checked for a GQE pointer. If none exists (indicating no previous error on this operation), return code 4 is set and control returned to CIP. If a GQE exists (indicating some previous error on this operation), the SCANTE is cleared and unlocked; the original error GQE is discarded (via CEAMJG), and control is returned to the CIP.

- If the start I/O condition code returned to CEABC was 1 and no busy conditions are present, the Generate and Enqueue Dummy Interrupt GQE subroutine (CEABQ) is called to create a simulated I/O interruption from the error status information. Processing then continues as if the RJE line control operation had been successful. (Described above.)

- For any other return from CEABC, the appropriate asynchronous event code is set. The asynchronous event codes are as follows:

- 0 - Operation successful
- 1 - Solid error on DISABLE
- 2 - Solid error on SET MODE
- 3 - Solid error on ENABLE
- 4 - Solid error on PREPARE
- 5 - Solid error on READ ENQ
- 6 - Solid error on WRITE ACK
- 7 - Failing CCW cannot be determined
- 8 - Unrecoverable software error in the supervisor
- 9 - Intervention required

The device SCANTE is then examined. Return is to CIP if there was no previous error. Otherwise, the original error GQE is discarded; the SCANTE cleared and unlocked before returning to CIP.

When a Halt I/O interruption is not indicated, a test is made to see if the interruption is due to a channel control or interface control check. If so, the operation is not retryable. In this situation the asynchronous entry flag in the device group table is tested to see if a DISABLE operation was indicated. If not, the asynchronous event code is set to 7 in the interruption GQE. The device's SCANTE is checked to see if a previous error occurred requiring the SCANTE to be cleared and the GQE to be discarded before returning to CIP.

If the DISABLE flag is set, in the device group table, it is turned off; and the asynchronous event code of 1 is put in the GQE. A message is then sent to the operator by calling task communication control (CEAAN) to inform him of the error. Return is then made to CIP after clearing and unlocking the SCANTE and disposing of

the original error GQE if there was a previous error.

If the error is not a channel control or interface control check, the CSW is checked for other error conditions. If the interruption is normal, the device's SCANTE is tested to see if there is a GQE. If no GQE is there, or if there is one, waiting on sense, the current GQE represents the successful completion of a sense operation.

If not a sense operation, the DISABLE flag in the device group table is tested. If it is on, the interruption is to be discarded. The waiting on sense flag is turned off and the SCANTE is checked for a GQE. If none exists, control is returned to CIP. If a GQE exists (indicating successful completion of a previous DISABLE error), the SCANTE is cleared and unlocked; and the GQE is moved (via CEAJMG) before returning to CIP.

If the DISABLE flag is not on, a check is made to see if the error occurred on an ENABLE command. If not, the asynchronous event code is set to zero. If this is the successful retry of a previous error, the SCANTE is cleared and unlocked and the previous error GQE discarded before returning to CIP.

If the interrupt is from an ENABLE command, the RJE line control subroutine is called to PRIME the line (bypassing Halt I/O). The return data from RJE Line Control is then examined.

- If successful, and no previous error occurred on this operation, return is to the CIP. The SCANTE is cleared and unlocked and the previous error GQE discarded before returning if a previous error occurred.
- If the return from the RJE line control subroutine indicates that it received a start I/O condition code of one with no busy conditions, an I/O error interruption is simulated by calling the Generate and Enqueue Interrupt GQE subroutine. Processing then continues as if the return from RJE Line Control indicated that it was successful.
- For any other return, the SCANTE is checked for previous error GQE. If there was none, Supervisor Core Allocation (CEAL1) is called to get main storage for a task asynchronous interruption GQE. If there was a previous error, the SCANTE is cleared and unlocked and the previous error GQE is used as the task asynchronous error GQE.

The GQE is initialized, and an asynchronous event code of zero is placed in it. Queue GQE on TSI (CEAAF) is then called to place the interruption GQE on the task's TSI interruption queue. This procedure informs the task of the successful completion of an ENABLE operation. The current GQE (the one that caused CIP to call this routine) is used to inform the task of the failure of the PREPARE operation. The asynchronous event code of 4 is set in it and return is to CIP.

If the interruption indicates the successful completion of a sense operation, processing skips to the point where first error, non-unit check errors are processed.

If the CSW status indicates an error condition, a check is made to see if a sense operation was in progress. If so, the asynchronous event code (1-6) appropriate for the condition is set in the GQE to indicate the failing CCW. The SCANTE is cleared and unlocked; the original error GQE discarded; and return is to CIP.

If the error did not occur on a sense operation, the SCANTE is checked for a previous error GQE. If there is none, the current GQE is dequeued by a call to the Dequeue GQE subroutine. Error retry counters are then set up in the GQE. The SCANTE is locked and the address of the error GQE is saved in the SCANTE. The unit check bit of the CSW is then examined to determine if a sense operation must be performed. If so, parameters to perform the sense operation are set up. Otherwise, parameters to perform the appropriate error retry procedure are set up. The common retry logic (see below) and on return a check is made to see if retry was successful. If so, the forward GQE pointer from the original error GQE is put in register 13 and return is to the Channel Interrupt Processor.

If the retry was not successful, the SCANTE is cleared and unlocked; the appropriate asynchronous event code is set in the current GQE and control is then returned to the Channel Interrupt Processor.

If a pointer to a previous error GQE is in the device's SCANTE, the unit check bit in the current GQE is examined. If a sense operation is to be performed, the sense parameters are set up and the common retry logic is entered.

If a sense operation is not to be performed, the error retry counters are checked to determine if the error has become solid. If not, the error counter is incremented and the appropriate retry para-

eters are set up. The common retry logic is then entered and the return is examined. If the retry was not started successfully, processing is the same as for solid failure (below). If the retry was successful, control is returned to CIP.

If the failure is solid, the asynchronous event code is set in the current GQE, and the DISABLE flag in the device group table is tested. If it is on it is turned off, and a return code of zero is set for CIP. Exit is then taken after clearing the SCANTE and unlocking it and discarding the old GQE.

The Common Retry Logic for this routine involves calling the Start I/O subroutine (CEAAG) to start I/O on the specified command. If start I/O is successful, control is returned to the calling location.

If the Start I/O condition code is 2, 3, or 1 with a busy indication, the Start I/O subroutine is recalled until either it is successful or the failure becomes solid. If the start I/O failure is solid, return is to the calling location.

For other conditions where the start I/O condition code is 1, the Generate and Enqueue Dummy Interrupt GQE subroutine is called to create a simulated I/O interruption GQE to be processed later. Control is then returned to the calling location.

Remote Job Entry Synchronous I/O Error Interrupt Subroutine (CEABB)

This subroutine screens all synchronous I/O error interruptions from RJE devices and perform appropriate error recovery procedures. This includes:

- Identifying the error condition from information in the CSW, the Start I/O CAW, sense fields in the GQE and IORCB.
- Maintaining error data and retry counters in the RJE IORCB.
- Determining if a retry operation is to be made and setting up to initiate it, if so.

Attributes: CEABB is a serially reentrant subroutine residing in main storage that operates in the supervisor state.

Assumptions and Restrictions: This subroutine is called only by the Channel Interrupt Processor to handle I/O retry operations after synchronous errors on the 2780 RJE terminal device. The routine is not parallel reentrant; it saves registers internally. In certain cases, it returns out-of-line to special return points in the Channel Interrupt Processor. The recovery

procedures apply only to the channel programs specified for the 2780 RJE device. Any alteration to the channel programs in MSAM will produce unpredictable results in error recovery.

Entries: CEABB1 - by Channel Interrupt Processor with these parameters:

Register 7 - address of the request GQE

Register 10 - address of the IORCB

Register 13 - address of the interruption GQE

Modules Called: Supervisor Core Allocation (CEAL1 entered at CEAL01) gets main storage for an error GQE on a PRIME failure condition.

Start I/O subroutine (CEAAG entered at CEAAG1) starts I/O on retry operations.

RJE Asynchronous Interrupt subroutine (CEABA entered at CEABA2) to process the interruption GQE when it is to be recycled.

RJE Line Control (CEABC entered at CEABC2) is called to prime the line after a unit exception error or time out.

Queue GQE on TSI (CEAAF entered at CEAAFQ) is called to put an asynchronous error interruption GQE on a task's TSI.

Pathfinding (CEAA5 entered at CEAA5P) is called to assign a path for the retry operations.

Reverse Pathfinding (CEAA5 entered at CEAA5R) is called to release a path after Start I/O failure on a retry operation.

Generate and Enqueue Interrupt GQE (CEABQ entered at CEABQ1) is called to create a dummy interruption GQE and place it on the CIP queue when Start I/O fails and returns a condition code of one.

Exits: Normal exit is to the Channel Interrupt Processor to the address specified in general register 14. Register 14 specifies the instruction following the call to CEABB, when normal processing of a synchronous error is to continue in CIP. CEAA44 is the return address following start I/O on retry to discard the interruption GQE, and CEAA47 is the return point when CIP is to reprocess an error GQE.

Exit is also made to the RJE Asynchronous Interrupt routine when Pathfinding returns a busy indication. This delays processing of the interruption in CIP because a real time interruption is created and placed on the timer interruption queue.

Error Conditions: The System Error Processing routine (CEAIS entered at CEAIS1) is invoked via the ERROR SVC when any of the following error conditions are encountered:

- Error return from Reverse Pathfinding (7600)
- Invalid return code from RJE Line Control (7601)
- Invalid command code (CCW) encountered (7602)
- Unrecoverable intermittent error sequence (7603)
- Invalid error condition (no status or sense) (7605)
- Error return from Pathfinding (7606)
- TSILCK locked more than 50 microseconds (7607)

Operation: On entry, CEABB saves the Channel Interrupt Processor's registers in an internal save area and establishes addressability for the RJE portion of the IORCB. Those IORCBs associated with remote job entry operations use the area normally occupied by the IORCB data buffer to hold RJE I/O error retry information. Individual error counters are included for each I/O error defined as retryable. There are two sets of retry counters. One set is used to accumulate the total number of errors of a particular type that occur for each IORCB. The second set records the number of errors occurring in a current intermittent I/O error retry sequence.

Next, error indicators in the GQE and IORCB are examined and the appropriate action is taken.

The only action taken before returning to the channel interrupt processor (in-line) for channel or interface control check is to increment the counter for the error type in the IORCB.

If sense failure is indicated by the IORCB sense failed flag, immediate return to the Channel Interrupt Processor is made (in-line).

For Unit Check/Lost Data errors, the current CSW is compared to the previous error CSW to determine if this is a new error. If it is, the previous error CSW and sense data are replaced by the current CSW and the retry counters are initialized to zero. Processing then continues in the "retry threshold testing" logic described below.

Unit Check/Time Out errors are processed the same as Unit Check/Lost Data until the retry threshold limit is reached. Then, the RJE Line Control routine (CEABC) is called to reprime the line.

Unit Check/Intervention Required are processed the same as Unit Check/Lost Data until the retry threshold limit is reached. Then the RJE Line Control routine is called to reenable the line.

Unit Check/Bus Out Check errors are handled the same as Unit Check/Lost Data.

Unit Check/Data Check errors are handled the same as Unit Check/Lost Data.

Unit Check/Equipment Check errors are handled the same as Unit Check/Lost Data.

Unit Check/Command Reject errors are handled the same as Unit Check/Lost Data.

Unit Exception indicates a logical termination of an I/O operation which must be communicated to virtual memory. (It is not a retryable error condition and has a retry threshold value of zero. This forces a "solid error" on the first occurrence.) Unit Exceptions are processed the same as Unit Check/Lost Data until the solid error condition is encountered. At this point, the RJE Line Control routine (CEABC) is called to reprime the line. On return, exit is to the calling program with a return code of zero.

Incorrect Length errors are processed the same as Unit Check/Data Check, except for the manipulation of the error counters. Those Incorrect Length errors that indicate Unit Check/Data Check conditions result in incrementation of the appropriate Unit Check/Data Check counters along with the current counter for Incorrect Length errors. In the case where the error results because less than the maximum number of cards were read, I/O is restarted on the next CCW. If during the printer selection sequence an ENQ (one byte) is read instead of an ACK 0 (two bytes), RJE Line Control is called to reprime the line.

Chaining Check errors are processed the same as Unit Check/Lost Data.

Program Checks are processed the same as Unit Check/Lost Data.

Protection Checks are processed the same as Unit Check/Lost Data.

Busy indications are handled the same as Unit Check/Lost Data.

Attention indications are handled the same as Unit Check/Lost Data.

Status Modifier conditions are handled the same as Unit Check/Lost Data.

Retry Threshold Testing: The appropriate current error counter (for the error being processed) is compared to the retry threshold limit. If the limit has not been reached, the proper total and current error counters are incremented and the Common Retry Logic (described below) is executed. If the limit has been reached, return code zero is set and control is returned to the calling program with the error CSW intact.

Common Retry Logic: The Start I/O subroutine (CEAAG) is called to restart the operation where specified and the returned parameters are examined:

- If the SIO was successful, a return code of four is set and control is returned to the calling program.
- If a device or control unit busy condition occurs, the SIO is retried until successful or the busy condition becomes solid. In the latter case, return parameters from SIO are stored in the IORCB, return code zero is set, and control is returned to the calling program with the original error CSW intact.
- When SIO returns a condition code of one, and the status other than device or control unit busy, the Generate and Enqueue Interrupt GQE subroutine (CEABQ) is called to simulate an I/O interruption with the stored status. A return code of four is set and control returned to the calling program.
- A SIO condition code of two or three results in a limited number of retry attempts. If unsuccessful, the SIO return parameters are stored in the IORCB, a return code of zero is set, and control returned to the calling program with the original error CSW intact.

Terminal Communications Subprocessor (CEATC) Chart AM

This subprocessor initiates and processes I/O operations for the terminals of conversational tasks. It is called by the Channel Interrupt Queue Processor when CIP finds a GQE representing an I/O interruption for a terminal on its queue. The following functions are provided by CEATC in its handling of terminal I/O operations:

- Initially issue channel programs for dial or dedicated lines to determine the type of terminal, line code, and user destination.

- Issue channel programs to MT/T users and normal TSS users.
- Post attention signaling during I/O operations or while processing.
- Post completed I/O operations for terminals.
- Detect errors or exceptions that terminate the channel program and initiate recovery action if possible.
- Provide standard translation of input and output allowing the user the option to specify no translation if he wishes.
- Allocate buffer areas for input and output operations.
- Place interruptions on the user's TSI, if required.
- Inform the user if TSS or MTT limits have been exceeded.

Entries: TCS has two main entry points, CEATC1 and CEATC2. CEATC1 is entered from the Channel Interrupt Queue Processor whenever it finds an I/O interruption GQE on its queue for a terminal device supported by the resident terminal access method (RTAM). When entered at CEATC1, six registers contain input data. The registers and significant data are as follows:

- Register 1 - the address of a 64 byte save area
- Register 2 - a pointer to the device group table
- Register 12 - a pointer to the asynchronous entry
- Register 13 - a pointer to the interrupt GQE
- Register 14 - the return address
- Register 15 - the calling address

CEATC2 may be entered by the SVC Queue Processor (CEAHQ) or the Terminal Communications subprocessor (CEATC1) itself. The conditions under which these routines enter CEATC2, and the parameters passed are as follows:

- The SVC Queue Processor calls CEATC2 when it is determined that the ATCS macro (SVC 219) has been validly issued by an MTT application task. When entered from the SVC Queue Processor, the following registers contain pertinent input data:

- Register 1 - the address of the SVC GQE (used as a TIOCB pointer)
- Register 2 - a pointer to the TSI
- Register 3 - a pointer to the XTSI
- Register 4 - either the VM address of a TCT slot, or all 'F's to indicate a FREEQ ALL operation
- Register 5 - the address of the message and its length for a Free operation. Zero if no message
- Register 6 - low order byte 'F's, if a physical disconnect is required or zero if a logical disconnect or FREE operation
- Register 15 - called address

- When the terminal communications subprocessor has been entered at CEATC1 and it encounters a pending I/O operation during its processing, a call is made to CEATC2 with the following input registers:

- Register 1 - all 'F's
- Register 3 - a pointer to the TIOCB
- Register 5 - a pointer to the TSI
- Register 6 - the real core address of the TCT slot
- Register 9 - a pointer to the MTSCB

Modules Called: Supervisor Core Allocation (CEAL1 entered at CEAL01) is called to get a 256 byte buffer area and a 64 byte terminal I/O control block to handle the initial interrupt from a user terminal.

Pathfinding (CEAA5 entered at CEAA5P) obtains the device path in initial interrupt processing.

Task Initiation (CEAMC entered at CEAMT1) is entered to set up a TSI in initial interrupt processing.

Queue GQE on TSI (CEAAF entered at CEAAFQ) puts an asynchronous interrupt GQE on the TSI as part of task initialization.

Reverse Pathfinding (CEAA5 entered at CEAA5R) releases the device path.

Supervisor Core Release (CEALI entered at CEAL02) releases unnecessary main

storage before returning to the Channel Interrupt Processor.

Start I/O (CEAAG entered at CEAAG1) is called to initiate terminal I/O operations.

Halt I/O (CEAAI entered at CEAAIH) halts terminal I/O during processing of terminal I/O requests.

Exits: When entered at CEATC1, this subprocessor exits to the Channel Interrupt Processor after setting appropriate return codes in register 15.

When entered at CEATC2, exit is to the Queue Scanner (CEAJQS) if the call was from the SVC Queue Processor. (When CEATC2 is called by CEATC1, return is to CEATC1 which then returns to the Channel Interrupt Processor.)

Operation on entry at CEATC1: There are three general classifications of terminal I/O interruptions that result in the Terminal Communications Subprocessor being called by the Channel Interrupt Processor. These are: the initial interruption from a user terminal, an interruption from a TSS user terminal, and an interruption from an MTT user terminal.

This subprocessor determines that the initial interruption from a terminal is to be processed by checking a field in the device group table (DEVTSI). This flag is always zero for the initial interruption.

To handle the initial interruption, a 64-byte area is provided by Supervisor Core Allocation (CEAL1) for a terminal I/O control block (TIOCB). A 256 byte buffer area is also obtained from Supervisor Core Allocation.

RTAM locates the appropriate entry in the terminal device table and saves its address in the TIOCB (not done for an initial interruption). For an initial interruption, a read channel program is generated for the terminal device line (according to type of terminal) and a path to the device is obtained by calling Pathfinding (CEAA5P). Return is then made to the Channel Interrupt Processor after setting a code of zero in register 15.

When the terminal responds to the read, this interruption is again passed to CEATC1 by the Channel Interrupt Processor. At this point, the input buffer is inspected to see if the terminal user has entered either the 'LOGON' or 'BEGIN' command. If it is 'LOGON' a test is made to determine whether the TSS user limit has been exceeded. If so, a message is issued informing the user of this; if not, a system terminal control table slot (TCT) and

buffer page slot are allocated to the task. Task Initiation is then called to set up a TSI for the task and an asynchronous interruption GQE is attached to the TSI interruption queue by calling Queue GQE on TSI. The operands following the LOGON command are then moved into the buffer page slot. The TCT pointer is placed in the device group table (DEVTCT) and the TCT is initialized.

To specify that the task is operating under RTAM in TSS mode, a flag is set in the device group table (DEVRT). Supervisor Core Release is called to release unneeded main storage, and the device path is then released by calling Reverse Pathfinding. Exit is made to CEAA4 with a return code of zero after a prepare command has been placed on the line.

When the command entered by the user is 'BEGIN', the application name specified by the BEGIN command is checked against the TSI chain by examining the MTSCB associated with the TSI of each MTT task. If it is there, a check is made to see if adding another user to the application task will put it over the user limit. If not, a further check is made to see if the application task has a FREEQ ALL operation pending (MTSFRE on).

If conditions are such that the request can be met, the 'number of users' count (MTSCUR) is increased by one; a prepare command is placed on the line; and a TCT slot and buffer slot are obtained. The operands following the application name are moved into the buffer and the TCT pointer is placed in the device group table (DEVTCT) and the TCT is initialized.

A flag (DEVMT) is set in the device group table to indicate that the terminal is connected to an MTT task and Reverse Pathfinding is then called to release the path.

If the TSI associated with the application task is on the inactive list, the Rescheduling subroutine is called with a code of one. On return, or if the task was on the active list, an external interrupt is placed on the TSI by calling Queue GQE on TSI with a code of two. Also, if the task is in delay status, the status is changed to ready.

A message control block (MCB) is then generated containing the following information:

1. Message code -- 255
2. Receiving task ID

3. Relative line number
4. Work byte (located at TCTWVK)
5. Line coded (located at TCTDTY)
6. Symbolic device (located at TCTSDA)
7. Message area

A return code of eight is set for the Channel Interrupt Processor.

If the application name, specified by the BEGIN command, is not active, a write '?' with response is placed on the line and a return code of zero is set for the Channel Interrupt Processor.

When the interruption that caused the Channel Interrupt Processor to call CEATC1 was not the initial interruption or one involved in the initialization phase, it is checked to see if it is for a HIO operation. If it is, CEATC2 is called to process the interruption, and on return, a return code of zero is set before exit to the Channel Interrupt Processor.

If it is for a normally completed I/O operation, the completion is posted in the TCT (the read completion is also posted when ATTENTION is received); the path is released by calling Reverse Pathfinding; and if an external interruption is not required, the task is made active before returning to the Channel Interrupt Processor with a return code of zero. If an external interruption is required, it is placed on the TSI by calling Queue GQE on TSI. The task is then made active and a return code of eight is set for the Channel Interrupt Processor. For abnormal completions, the subroutine CEA1000 is called to attempt recovery.

Entry at CEATC2 by CEATC1 is made to prepare for a terminal I/O request when a previously pending halt I/O operation has completed on a terminal line.

A CCW list is generated by CEATC2 corresponding to the I/O requested by virtual memory (that is, READ, WRITE, or Write with response). The CCW list depends on the device type (1050, 2741, TTY35, or 1052-7) and the I/O operation. The I/O operation is then started by a call to the Start I/O routine (CEAAG). Upon return, exit is made to the Channel Interrupt Processor, awaiting the completion of the I/O operation.

Entry at CEATC2 by the SVC Queue Processor occurs as a result of the ATCS (SVC 219) macro instruction having been issued during execution of a READQ, WRITEQ, CLEARQ, or FREEQ macro instruction. ATC

is also issued directly by the GATE, ABEND, LOGOFF, and RELEASE command routines.

When entered for ATCS processing (in all cases except for a FREEQ ALL), an immediate test is made to determine if the operation requested is a clear or attention. If not these, tests are made for write-type operations.

If a clear operation has been requested, CEATC2 causes the buffer slot to be released, clears TCTFL1, releases the main storage for the SVC GQE by calling Supervisor Core Release and exits to the Queue Scanner.

For a pending attention, the SVC main storage space is released and exit is taken to the Queue Scanner. For read and write operations, a HIO is done first.

When a write is specified, a check is made to determine if the data to be written is in main storage, the task ready and locked. If not, the instruction counter is decreased by eight (to point it at the ATCS instruction) and exit is to the Queue Scanner. Otherwise, a new buffer slot is obtained and the message moved into the buffer and translated, if required. If the length of the message exceeds the specified buffer length, Supervisor Core Allocation is called for a section of main storage four bytes larger than the indicated message length.

When a read is specified, the current buffer slot is released and a new buffer slot obtained. The necessary channel programs are then generated and the Start I/O subroutine (CEAAG) called to start I/O. Exit is then made to the Queue Scanner.

When a FREEQ ALL operation is specified, an application TCT slot is located. The TCTFFR flag is set and the Halt I/O routine is called. If an interruption is pending, (TCTHIO on) and it is a FREEQ ALL operation, a test is made to see if the FREEQ has been issued to all lines. If not, the above procedure is repeated. If all lines are cleared, the task is placed in page wait and exit is to the Queue Scanner.

When a Free completion is detected for a TSS user (LOGOFF or ABEND having issued ATCS), a real time interruption of two minutes is set for each line. For an ABEND due to SHUTDOWN, a physical disconnection takes place. The system TCT slot, the system buffer slot and outstanding TIOCBs are released. DEVTCT, TDELCD, and TDESTA are cleared and the DEVRT flag is reset. The task is made active if it is wait status.

For errors occurring on an initial break of a prepare command, the status is checked for unit exception or unit check. If it is a unit check, a sense is issued. When the interruption for unit check is returned, or if the original interruption were unit exception, normal processing is continued in CEATCS.

Otherwise, an attempt is made to reen-able the line.

Interruptions fall into two categories: outboard errors (unit check, unit excep-tion) and inboard errors. When a unit check is encountered, a sense is issued. Information from the sense is used to determine what action to take. There are some additional actions taken because of design and device dependent situations. A function byte, corresponding to each CCW, is located in the TIOCB. One of the fol-lowing eight functions can be specified in this byte:

1. Data Out - Data is being written.
2. Data In - Data is being received.
3. Write Addressing - Addressing charac-ters being written.
4. Write Polling - Polling characters are being written.
5. Response Polling - Response characters are being received as a result of a prior write polling.
6. Response Addressing - Response charac-ters are being received as a result of a prior write addressing.
7. Control - An operation for line or channel control.
8. TIC - Transfer in Channel.

When a line is dialed with a 1050 or 2741 terminal connected, a read is put on the line. The terminal type is unknown but marked as a 2741. This may result in the read terminating in an error. If a unit check, data in, lost data error is fielded during initialization, it indicates the device is a 2741; and an inhibit is put on the line to continue receiving data. If a unit check, data in, time out error is fielded during initialization, the terminal type will be determined; and this is noted by turning on the TDES2 flag in TDESTA. With no data in the input buffer, the device is a 1050; and this is flagged by turning off the initial read operation flag in TDESTA and switching TDEDEA to a 1050 device type. Control is then passed to the main routine to put a read on the line. This condition, but with data in the buff-

er, indicates the device is a 2741; and an inhibit is put on the line to continue receiving data.

The following actions are taken when the sense information is received after unit check:

• Data Out:

Data Check - Retry the first CCW in the TIOCB containing the error. This causes readdressing of the line.

Bus-Out Check - Action same as for data check.

Intervention Required - For 1050 with intervention required only, signal attention. For 2741 or TTY35 with above condition and data has been trans-ferred, signal attention. If identi-fied as on MTT terminal, pass attention to MTT task. If not, write "?" with response. For 1052-7 if Bell bit (MTSBEL) is on signal hard I/O Failure. If bit is not on, set bit and ring alarm by generating a CCW(0B) in a TIOCB which is chained to other out-standing TIOCBs. When an Attention is received, MISBEL should be set off and the writer operation resumed.

All others - Signal hard I/O failure.

• Data In:

Time Out - If in initialization proces-sing, handle as mentioned above. Else, process same as overrun.

Lost Data - If the residual count in the error CCW is zero, the buffer over-flowed. A flag (TCTWW7) is set in the TCT if identified as an MTT terminal, and the CCW following the error CCW is started. The CCW following the error CCW is started if not an MTT terminal. If not buffer overflow, process same as overrun.

Overrun - If identified as an MTT ter-minal, write "REENTER DATA" message. If not, write "?" with response.

Date Check - For 2741 or TTY/35 process same as overrun. For 1050, scan input buffer for X'41'. If not found process same as overrun. If found, a cancel has been indicated and the CCW preced-ing the error CCW is started.

Intervention Required - For 1050 or TTY35 if accompanied by a data check, signal attention and post length of message entered for Read Complete rou-tine (CEATC1). If not or if 2741, pro-cess same as overrun. If this condi-

tion occurs on 1052-7, the same procedure is followed as indicated under DATA OUT; however, the read operation should be resumed.

All others - Signal hard I/O failure.

- Write Addressing, Write Polling:

Data Check - Retry error CCW.

Bus-Out Check - Retry error CCW.

Intervention Required - Retry error CCW.

All others - Signal hard I/O failure.

- Response Polling:

Time Out - Retry CCW preceding error CCW.

Lost Data - Same as for time out.

Overrun - Same as for time out.

Data Check - Same as for time out.

Intervention Required - Same as for time out.

All others - Signal hard I/O failure.

- Response Addressing:

Time Out - Check CCW preceding error CCW for write addressing. If it is, retry preceding CCW. If not, it is an error in CCW string.

Lost Data - Same as for time out.

Overrun - Same as for time out.

Data Check - Same as for time out.

Intervention Required - Same as for time out.

All others - Signal hard I/O failure.

- Control:

Time Out - If operation is prepare or disable, retry error CCW. If operation is enable, retry error TIOCB at first CCW (disable). If none of the above, signal hard I/O failure.

Data Check - If operation is a break, retry first CCW in TIOCB with error CCW. If not, signal hard I/O failure.

Intervention Required - If data check also, set attention. If not, check for prepare or break. If so, retry error CCW. If not, signal hard I/O failure.

All others - Signal hard I/O failure.

- TIC:

All - Signal hard I/O failure.

The following actions are taken after unit exception:

Data Out - Retry first CCW in TIOCB with error CCW.

Data In - Turn off unit exception and return to main routine for normal processing. For 1052-7 reissue the read operation at the terminal.

Write Addressing - Retry error CCW.

Write Polling - Same as for write addressing.

Response Polling - Same as for data in.

Response Addressing - Same as for data in.

All other functions - Signal hard I/O failure.

For conditions where retry is attempted, on the third successive occurrence of the condition, retry is aborted and hard I/O failure is signaled.

For hard I/O failure, if the error is on a prepare or enable, the interruption is ignored. Otherwise, if the terminal is connected to an MTT task, the hard I/O failure flag (TCTWW5) is set and the task is informed of the condition. If not connected to MTT, the path is released, the TDE initial interruption flag is reset and a TIOCB is set up to reenable the line.

STORAGE ALLOCATION PROCESSORS

User Core Allocation Queue Processor (CEANB) Chart AN

The function of the User Core Allocation Queue Processor (UCA) is to allocate blocks of physical storage for user pages. User page allocation is requested by page control blocks (PCBs), pointed to by the GQES, which in turn are pointed to by the processor's queue entry in the scan table. PCBs may request the allocation of any block of storage for a user's page, or the allocation of a specific block of storage.

The User Core Release subroutine is logically a part of this module, and is described immediately following it.

Entry: CEANBA

Modules Called: Set Suppress Flag subroutine (CEAJQ entered at CEAJSF) sets the appropriate flag in the processor's scan table entry.

Move GQE subroutine (CEAJQ entered at CEAJMG) moves or releases the GQE.

Dequeue GQE subroutine (CEAJQ entered at CEAJDE) removes the GQE pointer from processor's queue.

Write Shared Pages subroutine (CEAMW entered at CEAMWS) scans and/or purges shared pages as directed.

Page Posting subroutine (CEAMP entered at CEAMP1) updates page tables associated with PCBES.

Set Real Time Interval subroutine (CEAST entered at CEAST7A) cancels pending real time interrupts before forcing a task to time slice end.

Enqueue GQE subroutine (CEAJQ entered at CEAJEN) places a timer interrupt GQE on the Timer Interrupt Processor queue to force a task to time slice end.

Supervisor Core Allocation (CEAL1 entered at CEAL01), supplies 64 bytes of main storage for a GQE.

Locate Page subroutine (CEAML entered at CEAML1P) locates the addresses of the page table entry and the external shared page table entry.

Exits:

Normal - To Queue Scanner.

Error - To System Error Processor.

Operation: The activities of this processor depend upon the following conditions:

- Requests for previously owned storage blocks.
- New requests for storage allocation.
- The availability of main storage.
- The existence of partially processed GQES.

The processor's response to each type of request is described below.

PREVIOUSLY OWNED STORAGE REQUEST PROCESSING:

If previously owned storage is requested, the processor determines whether the requested page can be reclaimed. This involves locking the core block table header and computing the address of the appropriate core block table (CBT) entry for the

previously used storage block and comparing it to the requested PCB entry testing for the following:

- Page not presently in use.
- TSI match.
- VM address match.

If the block requested can be reclaimed, UCA removes it from the unassigned list and updates the pointers (forward and reverse) in the list. If the CBT entry is first or last on the chain, the chain pointer is also updated. The reclaimed CBT entry is then marked "in use, user owned" and its pointers are cleared. UCA then unlocks the core block header, calls Page Posting, and selects the next PCB entry for processing.

When reclaiming fails, the page is removed from the top of the chain, and the chain is updated accordingly. The CBT entry is marked, and the internal address is put in the PCB.

When all PCB entries have been processed the UCA processor then determines if unallocated main storage is adequate, that is, it is greater than the minimum allowed. If not, an indicator is set in the system table to control the admission of new tasks to the system.

If all PCB entries have been posted or bypassed, UCA deletes the device queue string entry in the GQE. Move GQE is then called to move the GQE. On return, exit is to the Queue Scanner.

NEW REQUEST PROCESSING: When the PCB requests allocation of storage not previously owned, the processor locks the core block table header and checks the core block table (CBT) list for available storage. If the requested storage is available, it is assigned and marked "in use, user-owned". The processor then unlocks the core block table header and determines whether a read operation from an external device is required. If not, the allocated storage area is cleared to zeros, and the Page Posting subroutine is called. When control is returned to the processor, or if a read operation is required, the processor checks for more PCB entries in the GQE. If there are more entries, the processor selects the next PCB and starts processing it. If there are no more PCBs to be processed, the UCA processor compares the available number of storage blocks against its minimum reserve storage. If the available number of storage blocks is not greater than the minimum reserve storage, the 'low core' flag is set, and Write Shared Pages is called. When control returns to the processor, it exits to the

Queue Scanner. If any PCBs have not been posted, the device queue is not deleted, the move-GQE subroutine is called, after which the processor exits to the Queue Scanner.

LOW CORE CONDITIONS: Prior to processing new requests for storage, a check is made to determine if enough main storage is available. If not, an attempt is made by the processor to free main storage by calling the Write Shared Pages subroutine. In this case, the scan flag in the system table is turned off -- resulting in a purge. If this does not solve the low main storage problem, the processor will select a task to force to time slice end, preferably not the requesting task. When a task, other than the requesting task, is forced to time slice end, the processor then attempts to allocate the main storage block. If the requesting task is forced to time slice end, the processor immediately returns to the Queue Scanner. When setting or testing the 'low core' flag in the system table the TSI lock in the system table must be locked. It must then be unlocked when the function is completed.

The check of the low core threshold is made each time a page is assigned. If the threshold is not reached, the scan flag in the system table is turned off. When a task is forced to time slice end because of a low core condition, the field SSTLCT in the system statistical table is incremented.

INCOMPLETE PROCESSING: In the event that a request cannot be fully satisfied because there is not enough main storage, the processor will lock the lockout area lock and save all pertinent information in a save area and lock itself out by setting a suppress flag on. After information is stored in the lockout area, its lock is unlocked. The saved GQE will be processed before any new request for main storage. To assure this, the processor builds and queues a dummy request (GQE) to itself which will be received as soon as User Core Release causes the UCA suppress flag to be reset and control returns to UCA.

FORCED TIME SLICE END: If it becomes necessary to force a task to time slice end, UCA will lock the TSI chain and each individual TSI is then tested before selecting a task to be forced. (The requesting task will not be selected if an alternative is available.) After selecting a task to force to TSE, the TSI chain is unlocked as are the individual TSIs which were not selected. A GQE is then built and placed on the timer interrupt queue. UCA then attempts to continue. If it cannot, data is saved as in "INCOMPLETE PROCESSING" and exit is to the Queue Scanner.

Page Stealing: This portion of UCA is invoked when the steal request indicator (STESRI) in the schedule table is on, and the pages using this time slice (XTSNPG) is greater than the maximum number permitted to this task (STEMAXCR). The number of pages that can be stolen is 100 minus the percentage of the task's maximum pages that must be retained (STEST) times the number of maximum pages [(100 - STEST) * STEMAXCR]. If this percentage, divided by 100, is greater than the number of pages used, this time slice:

$$\frac{\text{STEMAXCR} * \text{STEST}}{100} > \text{XTSNPG}$$

a scan is done on all user pages, resetting the reference bits.

Before a page can be stolen, it is tested to make certain that it is not:

- An XTSI page.
- An ISA page.
- A PSW page.
- In transit (that is, in the process of being stolen).
- In I/O or SVC hold.

Only unreferenced pages may be stolen; therefore, the task's reference bits are reset each time page stealing is performed.

Auxiliary Storage Allocation Queue Processor (CEAIA) Chart AO

The function of the Auxiliary Storage Allocation Queue Processor is to allocate and maintain storage for user pages in auxiliary drum and disk devices. This processor maintains a count in the system table of auxiliary storage in use at any time for the entire system. The amount of auxiliary storage assigned to each task is maintained in the task's TSI. Also, if available drum space falls below a certain threshold, this processor will cause the migration of certain tasks from drum to disk. An additional logical function of the processor is the Auxiliary Storage Release subroutine described below.

Entry: CEAIAA

RESTRICTIONS: Auxiliary storage cannot be extended dynamically and the auxiliary storage processor can only handle the 2301, 2311, and 2314 devices.

MODULES CALLED: Dequeue GQE subroutine (CEAJQ entered at CEAJDE) removes the GQE pointer from the processor's queue.

Enqueue GQE subroutine (CEAJQ entered at CEAJEN) queues the GQE on the specified device queue.

Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) allocates main storage for a temporary save area and for GQEs and PCBES when page storage allocation is from different devices.

Queue GQE on TSI subroutine (CEAFQ entered at CEAFQ) queues a GQE on the TSI representing an interruption to cause a warning or shutdown message to be issued because of low auxiliary storage.

Supervisor Core Release subroutine (CEAL1 entered at CEAL02) releases main storage used for temporary save area.

Move GQE subroutine (CEAJQ entered at CEAJMG) causes GQEs to be queued on the proper device queue.

Exits:

Normal - To Queue Scanner.

Error - To System Error Processor.

Operation: On entry the Dequeue GQE subroutine is called to remove the GQE pointer from the auxiliary storage allocation queue entry in the scan table and to enable interrupts. When control returns to the processor, the first PCB is located and its related PCBES are examined. If at this point, there still isn't enough auxiliary storage to satisfy the PCB request, auxiliary storage management has failed, and the System Error Processor is invoked.

When drum space is to be allocated, a check is made to see if it will cause the amount of drum space available to fall below its threshold value. If it will, a task is chosen to migrate from drum to disk. The inactive list of tasks is searched looking for the task with the largest number of pages on drum in excess of its fair share (fair share is determined by the formula $ODS - BUFF - 3T/T - 2$, where ODS = original drum space, $BUFF$ = system buffer size, and T = number of tasks in the system). If no task on the inactive list exceeds its fair share of drum, the active list is searched using the same criterion. When a task has been selected for migration in this manner, a GQE/PCB is created to page in the first XTSI page. After it has been read in, and Page Posting brings in all the XTSI pages, migration of the selected task's pages will be initiated.

Under certain conditions, consideration is also given to migrating a task's shared pages from drum to disk. A field is checked in the auxiliary storage allocation

table (ASATMA) which indicates the number of drum pages available. If the number is less than the system buffer requirement, then the number of tasks already in migration is checked (SYSMC). If the value in SYSMC is less than the number of storage units in the system, minus one, and shared page migration is not already in progress, the number of shared pages on drum is compared to the maximum shared pages on drum threshold (SYSMXD). If the threshold is exceeded, CEAI A turns on the shared page migration flag (SYSMG), and a flag in the GQE (GQEMG) to indicate to the Timer Interrupt Processor that migration of shared pages is to be initiated.

When available drum space is not below the threshold value (and migration not required) a PCBE is located which requests drum storage allocation (when no bypass or suppress indicators are set). The processor then searches the drum directory slots for available pages. When available storage has been found, the processor updates the relevant auxiliary storage allocation table (ASA) fields and calculates the drum page number for the requested storage.

At this point, the processor inserts the page number and drum address in the PCBES external address field and posts the auxiliary page address in all required TSI/XTSI entries. The processor then selects the next PCBE for processing and performs auxiliary storage location and allocation as described previously.

When minimum drum availability is reached, the processor makes a check to determine if the page is a drum-preference page. If so, the processor assigns the page a drum address in the manner described previously.

The processor allocates disk storage when the following conditions exist:

- No drum storage is available.
- The requested storage is not a drum-preference page and the availability of drum space is minimal.

When these conditions exist, the processor locates a disk device with available storage, and then searches the disk directory until the available space is located. Auxiliary disk storage is then allocated in the same manner as drum storage.

When auxiliary storage has been assigned to the PCBES, the processor sorts them by assigned-device type. If page storage has all been allocated from the same device, the processor inserts the device address in the affected GQE fields, and calls the

Enqueue GQE subroutine. Enqueue GQE queues the GQE on the queue of the addressed device and enables interruptions as specified by the processor. When control is returned to the processor, an exit is made to the Queue Scanner.

If page storage has been allocated for the PCBES from different devices, the processor must set up a GQE and PCB for each addressed device as follows:

- Requesting from the Supervisor Core Allocation (SCA) subroutine a 64-byte physical storage area for each GQE. At the same time, a 64-byte physical storage area is also allocated for a PCB.
- When SCA has reserved the storage and returned control, the GQEs are set up by inserting a pointer to the PCB and the device address in the required fields.
- The PCBE is then moved to the new PCB, and the count of PCBES in the GQE is then updated. When a PCB is filled, and another PCBE is found for this device, SCA is called to provide another 64-byte storage area for another PCB.
- Setting bypass flags in the old PCBES to prevent allocation to the wrong device.
- Calling the Enqueue GQE subroutine to queue the new GQEs on the queues of the addressed devices.
- Regaining control from the Enqueue subroutine.

Whenever auxiliary storage has been allocated, the total of auxiliary storage available field, in the system table, is decremented by the amount assigned. Also, the count of auxiliary storage assigned field in the task's TSI is incremented by the amount.

If the amount of auxiliary storage assigned to a task exceeds the amount permitted for that task, a warning flag is set in the task's TSI. If the amount of auxiliary storage available to the system falls below the minimum allowed, the task which has the greatest amount of auxiliary storage assigned to it, and its warning flag on, is terminated by way of a program interruption.

When allocation is completed, this processor exits to the Queue Scanner.

Figure 16 illustrates the activities of the processor.

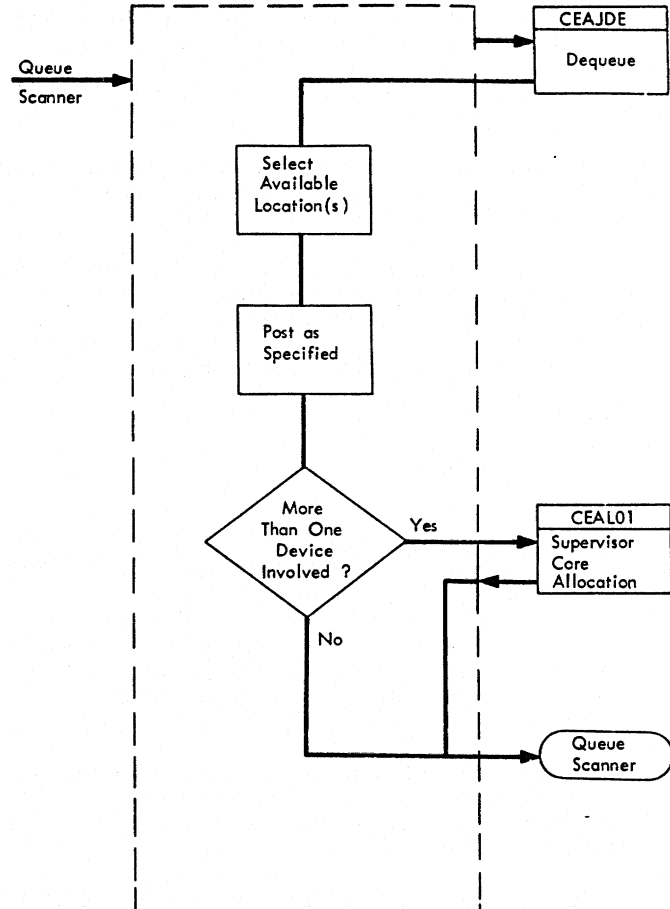


Figure 16. Activities of the Auxiliary Storage Allocation Queue Processor

Contiguous Core Allocation Queue Processor (CEANF)

This queue processor allocates contiguous main storage pages when user virtual memory expansion requires more than one segment table page. Hardware limitations require segment table pages to be contiguous.

RESTRICTIONS:

- This routine can only be called by Page Posting (CEAMP), Add Page (CEAHQ), Add Shared Page (CEAQ6), and Connect Segment to Shared Page Table (CEAQ7) subroutines.
- The GQECNT field in the GQE passed to this routine must contain the number of contiguous main storage blocks requested.
- If entry is from Page Posting, the internal main storage address of the segment table used in the previous time

slice must be contained in the PCBIA fields of the PCBs attached to the GQE.

Assumptions:

- When this routine is called, the task has at least one segment table page outside of the first XTSI page.
- Requests from ADDPG, ADSPG, and Connect Segment are for only one page of contiguous main storage at a time.
- The XTSI ID (VMA) will be provided by the caller.

Entry: CEANFA

Modules Called: Dequeue GQE subroutine (CEAJQ entered at CEAJDE) removes GQE pointers from specified queues.

Rescheduling (CEAKZ entered at CEAKZA) places a task's TSI on the inactive list when the request for contiguous main storage cannot be met immediately.

Move GQE subroutine (CEAJQ entered at CEAJMG) moves a GQE from this processor's queue to another where processing is to be done.

Page Posting (CEAMP entered at CEAMP1) posts user page status information to a task's TSI, XTSI, and shared page tables after a paging operation.

User Core Release (CEAL1 entered at CEAL04) releases storage for unused pages.

Task Communication Control (CEAAN entered at CEANN1) sends a message to the main operator.

Read Time (CEAS6 entered at CEAS6A via SETTIMER macro) gets the real time in milliseconds.

Set Real Time Interrupt (CEAS7 entered at CEAS7A via SETTIMER macro) sets real time interrupt for this task.

Queue GQE on TSI (CEAJQ entered at CEAJEN) adds a GQE to the TSI queue.

Exits:

Normal - To Queue Scanner.

Error - To System Error Processor when the following situations are encountered:

- The waiting count is exceeded while testing the lock byte.

- The task does not already have one segment table page outside the first XTSI page.

- The condition code setting after issuing the Load Real Address instruction indicates unsuccessful translation of the address of the page to be stolen.

- An invalid instruction length code is encountered.

- An error is encountered when converting the core block table entry address to a main storage page address.

Operation: If this processor is entered from the Page Posting routine, a task is starting a new time slice. In this case (determined by a check on entry), the TSI is locked and an attempt is made to reclaim pages used in the task's last time slice. If successful (all contiguous pages are obtained), pages will be reposted -- eliminating the need for any I/O operation. The GQE is then destroyed and the TSI lock reset before exiting to the Queue Scanner.

If previously used pages cannot be recovered, the core block table is locked (CHBBLK) and the core block table entry string is scanned from the top seeking the required number of contiguous main storage pages.

If found, they are allocated and the core block table entries are updated and relinked. Exit is to the Queue Scanner with the core block table lock reset.

If the request is from one of the authorized SVC queue processors (CEAMP, CEAHQ, CEAQ6, or CEAQ7), the task is active and needs contiguous main storage pages on demand. To meet the demand, the page following and the page preceding the segment table page are checked--in that order. If the trailing page is available, it is used to fill the request. If the page preceding the segment table page is to be used, all segment table page pointers in the core block table entries are updated to reflect movement of the segment table to the beginning of this page.

When neither adjacent page is available, the core block table entry string is scanned as in the procedure previously described for handling a call from Page Posting. If necessary, an available page can be created to satisfy the request for contiguous main storage by borrowing one of the pages currently occupied by the task. This is done by marking the page "unavailable" in the page table. Pages may not be stolen in this way, however, if they are

PSW, XTSI or changed pages, or if they are in I/O hold.

If they are shared pages, the sharing lock (SYSSHALK) is set, and the Inter CPU Communication subroutine is called to cause the other CPU's associative registers to be reset. After the page table has been updated, the sharing lock is reset.

When a contiguous main storage block has been allocated, its first page will be used to contain the beginning of the segment table. The address of this first page is returned to the caller.

If a request cannot be satisfied through these procedures, a retry effort is set up. First, a message is sent to the main operator to inform him of the situation; then the requesting task is deactivated and a 3-second real timer is set to control reactivation of the task. Exit is then made to the Queue Scanner. After three seconds, the task is reactivated and the request for contiguous main storage regenerated.

Supervisor Core Allocation Subroutine
(CEAL1 Entered at CEAL01) Chart AP

This subroutine allocates main storage for use by the supervisor components.

Entry: CEAL01

Modules Called: Inter-CPU Communication subroutine (CEAIC): resets associative registers for other CPUs.

Write Shared Pages subroutine (CEAMW entered at CEAMWS) to free shared pages when all other means of obtaining main storage for supervisor use have failed.

Exits: To calling routine:

CC = 0 request fulfilled
CC = 1 core unavailable.

Operation: Supervisor Core Allocation (SCA) allocates main storage in multiples of 64 bytes for any size request of a full page or less. Each physical page being used by SCA is divided into blocks, each block consisting of 64 bytes. When a request for main storage is received, the number of bytes requested is rounded up to the nearest multiple of the blocksize, and

that many contiguous blocks are made available for use by the calling component.

The first block of each SCA page is never assigned for use, but instead contains control information for the SCA subroutine. Forward and backward links chain it to other SCA pages. All SCA pages contain a unique identification word (GAIL) to ensure that any release of main storage references a valid SCA page. Also included in the control information is a count of available blocks left in the page, and a bit table representing the current status of each assignable block in the page, where a 1 bit indicates an in-use condition, a zero bit indicates that the block is available. The control information block of each SCA page is formatted in the manner shown in Figure 17.

The one exception to the use of the control block is when a full page of main storage is requested. Because of this, a full page request must be released as a whole, and never in a series of smaller blocks.

When SCA is called with an unavailable return permitted and the 'low core' flag is on, SCA exits immediately with condition code = 1. Otherwise the request is filled. When the request is for one block of main storage, the quick cell lock is set and a check is made to see if a block address is currently in one of the quick cells (words containing the address of one block of main storage that was recently released). If the quick cells are not empty, the request is filled and exit is made with condition code = 0. If the quick cells are empty, the lock is reset, and the request is filled by other means.

Requests for main storage blocks of less than a page are allocated from three chains of partially allocated pages:

- Single block chain - requests for single blocks are allocated from this chain when the quick cells are empty.
- TSI chain - blocks in this chain are allocated in groups of three and are used only when Create TSI calls for main storage for a TSI.

Forward Link	Backward Link	SCA ID (GAIL)	Unused	Flags	Count Available Blocks	Availability Bits
4 bytes	4 bytes	4 bytes	1 byte	1 byte	2 bytes	8 bytes

Figure 17. SCA control information block

- Other - all other requests are filled from this chain.

In all cases, the RSVLK is set and SCA allocates from the first two if they have storage available. When the third chain is used, each SCA page is checked before it is searched to assure a reasonable probability of meeting the request in that page. If the probability is not good, the page is skipped and the next page is checked. If all pages are eliminated because of poor probability, they are then searched in the normal manner, contingent upon the count of available blocks being at least as large as the request.

When SCA cannot find enough blocks of contiguous main storage in any of the chains, or if the request is for a full page, it goes to its list of reserved pages - a list containing the addresses of a number of pages kept in reserve to supply the immediate needs of the supervisor components. Whenever the reserved list is low, the User Core Release subroutine assigns the next user page released to the reserve list. SCA sets the protection key to "4" on all new pages it takes.

When the reserve list is empty, the core block header lock (CHBLOCK) is set and an attempt is made to find available main storage in the core block header chains. If none is available, the core block header lock is unlocked, the TSI lock in the system table is set and an attempt is made to steal pages by scanning through the various XTSI's segment and page tables.

As each task is checked, the TSI lock and CHBLOCK are set and then unlocked if no page can be stolen. Up to three passes are made on the page tables; if at the end of any pass a page or pages have been found, the pages are marked unavailable, assigned to the reserve list and normal SCA processing is resumed.

The first pass on the page tables searches for available pages that have been referenced but not changed for tasks not in the execution state. On the next pass, available pages that are not referenced and not changed for tasks not in the execution state are considered.

The final pass looks at available pages that are not changed for tasks in the execution state. When pages for tasks in the execution state are used, it is necessary to ensure that the page hasn't been changed between the time it is checked for a change and the time that it is made unavailable. This is done by checking the page for a change, and marking the page unavailable. Inter-CPU Communication is called to reset other CPU associative regi-

sters. The page is then checked again for a change.

In the event that a page cannot be stolen from another task, the TSI lock in the system table is unlocked and a special call is made to Write Shared Pages (CEAMW) to free up unchanged pages. If the first call is unsuccessful in freeing pages to place on the reserve list, a second call is made. If no main storage can be obtained, a major SYSERR is issued.

The Supervisor Core Allocation subroutine disables all I/O and external interruptions for the CPU currently executing the subroutine, in order to inhibit the interrupt stackers, so that the SCA subroutine will not be called when its pages are locked. Prior to entering SCA, the calling program indicates, as one of its calling parameters, whether these interruptions are to be enabled on exiting from SCA or whether interruptions are to remain disabled.

Supervisor Core Release Subroutine (CEAL1 Entered at CEAL02) Chart AP

This subroutine releases main storage blocks previously held for the use of supervisor components.

Entry: CEAL02

Exits: To caller - with reserve list lock (RSVLK) unlocked.

If a full page of main storage is freed by the release of blocks, exit is to User Core Release (CEAL1 entered at CEAL04) which releases full pages of main storage.

Operation: This subroutine marks available for use any returned main storage blocks. To do this, the reserve list lock is set and the availability bits in the SCA page are reset, and the count of available blocks is incremented. If the page is now composed entirely of available system blocks, the page is returned for general system usage by calling the User Core Release subroutine.

When a single block of storage is released to SCR, a check is made to see if all quick-cells are empty. If they are full, the availability for this block of storage in the SCA page is reset, as noted earlier; if the quick-cells are empty, the quick-cell lock is set, the address of the single block of storage being released is put in it, and its corresponding availability bit remains set. The quick cell lock is unlocked. The quick-cell is used to speed up the allocation of a single block of storage, which is expected to be the most common type of request.

SCA disables I/O and external interruptions for the CPU currently executing the subroutine, to prevent the interrupt stacker from entering it while SCA has certain locks locked. The calling program indicates as one of its calling parameters, whether these interruptions are to be enabled on exiting SCR, or are to remain disabled.

User Core Release Subroutine (CEAL1 Entered at CEAL04) Chart AP

This subroutine marks 4,096 bytes of main storage available for reuse in the core block table (CBT).

Assumptions: Startup will initialize the core block table.

Entry: CEAL04

Exits: To caller except when it was called by SCR, in which case exit is to the caller of SCR.

Operation: Upon entry, User Core Release (UCR) locks the reserve list (RSVLK) and core block header (CBHLOCK). UCR then determines whether the address it has received is that of a page or of a PCB containing a page address. The page address (whether sent directly or by means of a PCB), is placed into a register; from this point forward its origin no longer matters. UCR then checks to make sure that CBHPXP (the shared page chain pointer in the core block header) is not pointing to the page to be released. If it is, it must be updated to the next page on the chain, or zeroed if there are no more pages.

UCR then releases main storage to the core block table unassigned chain, unless the reserve list is low, in which case the page is put in the reserve list if the flags indicate that this is permitted. UCR then unlocks the core block header and the reserve list, and resets the low core flag, if necessary. If a check indicates that User Core Allocation is suppressed, the Set Suppress Flags subroutine is called to reset the flag, provided Write Shared Pages is not the caller.

UCR then exits to its caller unless Supervisor Core Release called it to release a full page, in which case it returns to SCR's caller.

Auxiliary Storage Release Subroutine (CEAIA) Chart A0

This subroutine releases auxiliary storage and maintains auxiliary storage control fields in the system table and task's TSIs.

Entry: CEAIA

Modules Called: Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) provides main storage for use as a register save area.

Supervisor Core Release subroutine (CEAL1 entered at CEAL02) releases main storage after use.

Set Suppress Flags subroutine (CEAJQ entered at CEAJSF) sets the suppression flag off in the Auxiliary Storage Allocation Queue Processor's scan table queue.

Exit: To caller.

Operation: The calling routine must supply three parameters:

- The TSI pointer.
- The number of pages to be released.
- The main storage location of the list of 4-byte symbolic I/O addresses (one address for each auxiliary storage page to be released).

The subroutine finds the bit for each symbolic I/O address in the appropriate bit directory, sets the bit to zero, and raises the availability count for that device by one. Each time a page is released, the count of available auxiliary storage, maintained in the system table, is raised. The TSI pointer is then checked. If it is not zero, the count of auxiliary storage assigned to the task, maintained in the TSI, is lowered. The assigned count of auxiliary storage for the task is then compared to a field which specifies the limit of auxiliary storage allocatable to this task. If the amount of auxiliary storage actually assigned is equal to or greater than the limit, the auxiliary storage requirements exceeded flag is turned on in the task's TSI. If release of a page causes the amount of storage allocated to fall below the limit for the task, the flag is turned off and the program continues.

Two conditions can cause an entry to the system error processor:

- One of the symbolic I/O addresses given does not represent an auxiliary storage page.
- One of the auxiliary storage pages was already available.

The processing of the request is complete up to the error entry.

Suppress Auxiliary Allocation Subroutine (CEAAP)

This subroutine suppresses external page allocation on the specified auxiliary device.

Entry: CEAAP1

Modules Called: Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) allocates all necessary work/save areas.

Supervisor Core Release subroutine (CEAL1 entered at CEAL02) releases the work/save area.

Exit: To caller.

Operation: On entry, processing steps are performed in the following manner:

- The Supervisor Core Allocation subroutine is called to allocate all necessary work/save areas.
- The input general registers are saved.
- A test-and-set instruction is used to test the auxiliary storage allocation table lock byte.
- If it is on, a wait loop is entered to allow a reasonable time for the lock byte to be reset. If the lock is not reset during the waiting period, a minor software SYSERR is reported, and processing continues as if the lock-byte had been found off.
- If the lock byte is off the first drum entry may be referred to via the pointer in ASAT.

The input symbolic address is compared to that of the device address field of this and succeeding drum entries until a match is found, or until the last drum entry has been examined. When a match is found, the drum suppression flag is set, the number of pages available on the drum subtracted from the total number of drum pages and control returns to the caller.

- If no matching entry can be found when the maximum number of drum entries have been tested, a test is made to determine if there are any disk entries in the ASAT.
- If so, the input symbolic device address is compared to that of the device address field of successive disk entries until a match is found or until the last disk entry has been examined.
- When a match is found, the drum suppression flag is set, the number of

pages available on the disk is subtracted from the total number of disk pages, and control is returned to the caller.

- If no match can be found when the maximum number of disk entries have been tested, a minor software SYSERR is reported.

When processing is completed, the ASAT lock byte is reset, general registers are restored, and the Supervisor Core Release subroutine is called to release the work/save areas and control is returned to the calling program.

SVC QUEUE PROCESSOR AND SERVICE ROUTINES

Supervisor Call Queue Processor (CEAHQ) Chart AQ

The function of the Supervisor Call Queue Processor is to assure that the task issuing the SVC is privileged to do so and to identify and branch to the proper SVC processor to service the interrupt.

Attributes: The SVC processor and its processor programs are reenterable, resident, open, and operate in the privileged state.

Entries: The Supervisor Call Queue Processor is entered from the SVC interrupt stacker at CEAHQP2. There are also two other entry points (CEAHQQ and CEAHQR) which allow the SVC Queue Processor to be entered as a subroutine to provide GQE routing service.

Modules Called: Queue GQE on TSI (CEAAF) queues program interrupts on the task's interruption queue in the event the task is not authorized to issue the SVC or if it has issued too many consecutive TSEND SVCs.

Move GQE subroutine (CEAJQ entered at CEAJMG) queues the GQE on the next queue when there is more work to be done or releases the GQE's storage space when there is no more work.

In addition to these, the SVC Queue Processor calls all of the SVC processing subroutines described on the following pages. These routines perform the work requested by the task and only one is entered for each supervisor call (SVC).

Exits: Upon normal completion of its work the Supervisor Call Queue Processor exits, by means of a branch, to the individual SVC subroutine or processor indicated by the interrupt code. The address of the GQE is passed, as a parameter, in register one. In addition, registers two and three will contain the addresses of the TSI and the

XTSI respectively and the contents of registers 0, 1 and 15 at the time the SVC was issued will be preserved in registers 4, 5 and 6 respectively.

If the task issuing the SVC is found to be unauthorized to do so, an exit is made to the Queue Scanner after an appropriate task interrupt has been set up. Since programs will issue SVCs as part of macro instruction expansions, an undefined interrupt code is considered a serious error and will result in a call to the System Error Processor with an indication of the nature of the error from the SVC Interrupt Stacker.

Operation: The SVC Queue Processor is initiated either by the Queue Scanner when a GQE is on the SVC queue or directly by the interrupt stacker. General register 1 contains the address of the GQE when this program is entered.

On entry, the lock byte in the TSI of the task that issued the SVC is set and the ready bit will be on meaning that until the lock byte is reset the task will not be considered for further execution.

If entered from the Queue Scanner, the SVC Queue Processor first dequeues the GQE. If entry is from the Interrupt Stacker, this function is skipped. Next, the interruption code from the GQE is used to locate an appropriate entry in the SVC flags table. This entry is one byte of information which includes the privilege status of the SVC. This privilege status is compared to that of the task (contained in the TSI). If the task's privilege does not qualify it to have issued the SVC, it is treated as a program error by generating a program error interruption. The TSI lock byte is then reset and exit is made to the Queue Scanner.

For any SVC except TSEND, the time slice end SVC count field in the TSI (TSITSC) is set to 1. If the SVC is TSEND, this count is raised by 1. It is then compared to the time slice end maximum count (SYSTSEM) in the system table. If SYSTSEM is exceeded, the task is terminated by way of a program interrupt for issuing too many consecutive TSEND SVCs.

If the task is of a sufficient priority to have issued the SVC, the SVC Queue Processor places the following in general registers:

- The GQE pointer.
- The TSI pointer.
- The XTSI pointer.

- The contents, from the XTSI's save area, of general registers 0, 1 and 15 when the SVC was issued.

This accomplished, the SVC Queue Processor uses the interruption code to find the entry in the SVC address table, which contains the actual main storage address of the SVC processor required for processing the SVC. If a corresponding entry is found, the SVC Queue Processor passes control to the processor program. If there is no corresponding entry, the queue processor branches to an error routine which calls CEAIS via the ERROR macro instruction.

Two SVC processing routines are internal subroutines within the SVC Queue Processor, the Add Page subroutine (CEAHQA) and Time Slice End subroutine (CEAHQF). They are branched to by the queue processor whenever the services they provide are requested.

Add Page Subroutine (CEAHQ entered at CEAHQ) Chart AR

This subroutine handles SVC requests for the addition of pages to the calling task's virtual storage. The Virtual Storage Allocation Service routine normally issues the SVC, specifying the virtual storage address, a protection code and the number of pages to be added. The specified virtual storage address indicates the segment and page at which the new page addition is to begin. In adding a page to a segment, the subroutine checks the segment and page tables.

Entry: CEAHQ - in response to the ADDPG macro instruction.

Modules Called: SVC Queue Processor (CEAHQ at CEAHQ) causes a GQE to be put on the calling task's interruption queue, and resets the TSI lock and exits to the Queue Scanner.

Find Page subroutine (CEANC) locates the segment table, auxiliary segment table, page table, segment page and external page table entries.

XTSI Overflow subroutine (CEAMX) is entered at CEAMXP for page-table expansion; at CEAMXS for segment-table expansion.

Paging routine (CEAMQ) is called to read in a page table page, if necessary.

Exits: Upon normal completion the Add Page subroutine exits to the SVC Queue Processor. An abnormal condition may result in an exit to either the Queue Scanner or the SYSERR routine.

Operation: The addition of pages to a segment in a user's virtual storage may involve either:

- The addition of a page to a segment that already has pages assigned to it.
- The addition of pages to an unassigned segment.

When a page is being added to a segment which already has pages assigned, the addition may be made to the end of the segment, thus expanding the page table and external-page table of the segment; or, the addition may be made within the segment to an unassigned area which has been freed by the execution of a Delete Page SVC.

In adding pages, the subroutine must consider the storage required by the XTSI. For each page added ten bytes of XTSI space are required; two bytes for the new page table entry and eight bytes for the new external-page table entry. Since the page table and/or external-page table for any segment may not overlap the XTSI page boundary, the subroutine must determine whether or not the addition of table entries would exceed the first XTSI page. If so, another page of main storage is requested in which to place the page table and external-page table for the segment involved. The number of allowable XTSI pages is a system value that is checked whenever the subroutine obtains a new page. If this maximum is exceeded, an interruption must be queued on the TSI's interruption queue. This is accomplished by calling the SVC Queue Processor.

Another consideration in adding pages is whether a shifting of tables is necessary in order to accommodate any particular addition of pages. This shifting may take one of two forms:

- A "short push", which involves moving the external-page table entries of a particular segment away from the page table entries of that same segment in order to make room for new page table entries.
- The "big push", which involves moving together the page table and external-page table of a number of segments in order to make room for the total addition of table entries to some other segment. This action is necessary only when pages are being added to a segment whose tables (page and external-page) reside within the first XTSI page and are not the last set of such tables in the first XTSI page.

On entry, the Add Page subroutine loads the virtual storage address specified by

the caller into a general register, and checks contiguous main storage requirements for segment table entries. If needed, Contiguous Core Allocation is called to get a new main storage area address. The Add Page subroutine passes this address to the XTSI Overflow subroutine (at CEAMXS) for use in segment table expansion. If there are no contiguous main storage requirements, the Add Page subroutine transfers control to the Find Page subroutine (at CEANCA). This subroutine returns to Add Page a condition code and pointers to the segment/auxiliary-segment tables, and the page/external-page table entries involved in this particular add-page operation. If the return code from Find Page indicates that the segment is unavailable, Add Page checks to see if that segment is in a page table page. If so, CEAMQ is called to read in the page table page. When CEAMQ returns, the page table will be available.

If the segment is available, the processor tests the condition code to determine whether the requested page is to be added to an internal area of a segment, to the end of a segment to which pages are already assigned, or to an unassigned segment. In the latter case, the subroutine issues an ERROR SVC. In the other two cases, the subroutine determines whether the current allocation of tables within the XTSI allows the selected page table to be expanded as much as the request requires. If not, the tables are rearranged, or the XTSI is expanded to another page. If the current size of the XTSI exceeds the current limit for the task, or if previously assigned pages are being reassigned by the SVC, the SVC Queue Processor is called to queue the GQE pointer on the TSI's interruption queue.

If Add Pages was able to allocate the requested page(s), the following occurs:

- The page-table-length field in the segment table entry is increased if the addition was made at the end of a page table.
- The page-table-availability flag in the applicable segment table entry is set to indicate that the page table for that segment is unavailable.
- The page-assigned indicator in the external-page-table entry is turned on.
- The appropriate protection keys, specified by the protection class, are set.
- The page-availability flag in the applicable page-table entry is set to indicate that the page is unavailable.

- The Add Pages subroutine exits to the Queue Scanner.

Add Shared Pages Subroutine (CEAQ6) Chart AS

This subroutine handles an SVC request for additional shared pages for a privileged program's use and tests to determine if a new shared page table must be created or if it is possible to add to the existing shared page table of the same number. The calling program specifies a virtual storage address, a number of contiguous pages, a protection class, and a shared-table number.

Entry: CEAH26 - in response to the ADSPG macro instruction.

Modules Called: SVC Queue Processor (CEAHQ at CEAHQO) causes a GQE to be put on the calling task's TSI interruption queue, resets the task lock, and exits to the Queue Scanner.

Locate Page subroutine (CEAML) provides the location of any page table entry or external-page table entry when the appropriate virtual storage and TSI addresses are given.

Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) allocates new space.

Supervisor Core Release subroutine (CEAL1 entered at CEAL02) releases old storage space.

Exit: To Queue Scanner, with following output:

Register 1 - Virtual Memory Address.

Register 15 - Protection Class and Shared page table number.

Operation: On entry, the subroutine checks contiguous main storage requirements for segment table entries. If needed, Contiguous Core Allocation is called to get a new main storage area address. The Add Shared Page subroutine passes this address to the XTSP Overflow subroutine (at CEAMXS) for use in segment table expansion. If there are no contiguous main storage requirements, the Add Shared Pages subroutine tests the shared-page table number specified by the caller. If the caller specified a shared-page table number greater than zero, the resident-shared-page index (RSPI) is searched for a matching entry. If none is found an error has occurred and the SVC Queue Processor is called. On return, the subroutine resets the TSI lock byte and exits to the Queue Scanner.

If a matching RSPI entry is found, the subroutine sets on the RSPI entry's add-shared-page bit, and determines whether there is enough available space in the indicated shared segment to add the new shared pages. If so, the subroutine determines whether the storage currently being used for page table entries contains enough space for the new page entries. If it does, the subroutine then constructs the new shared-page table entries, marking each of them as assigned and unavailable, and setting their external-page locations to zero. The protection class specified by the caller is stored in the external-shared-page table.

At this point, the length and page-table-origin fields in the RSPI are updated to reflect the new entries. The processor then searches through the entire chain of TSIs to update the segment table length field for those tasks that use this shared-page page table number. The subroutine stores the virtual storage address of the new pages and the shared-page table number in the general register save area of the XTSP, and exits to the SVC Queue Processor, which effects the release of the GQE storage space, resets the TSI lock byte, and exits to the Queue Scanner.

If a matching shared-page table number is found by the processor, or if the caller specified a shared-page table number of zero, but there is not enough space in the associated segment to add the new pages, the subroutine does two things: Assigns a new shared table number, and then prepares to construct new shared-page and external-page tables. The first step in this procedure is to request main storage from the Supervisor Core Allocation subroutine. When the storage is allocated, the processor constructs a shared-page table and an external-shared-page table. The entries in these tables are then marked as assigned and unavailable. The external-page location is set to zero, and the protection class specified by the caller is stored in the external-shared-page table.

The subroutine then determines, by searching the RSPI, whether there is room for another entry in the existing resident-shared-page index storage area. If not, the Supervisor Core Allocation subroutine is called to allocate new space, and a new RSPI is generated and chained to the previous RSPI. If a new RSPI entry may be added to the previous RSPI, a new entry is constructed and inserted in the previous RSPI storage area.

In either of the above cases, the segment and auxiliary-segment table entries for the new shared-page table number are then made in the XTSP of the calling task.

The virtual storage address and shared-page table number are then stored in the XTSI's general register save area, and the subroutine exits to the SVC Queue Processor, as described previously.

If there is not enough space in the currently used shared-page table entry to add new page entries, the subroutine calls the Supervisor Core Allocation subroutine for more space. When space is allocated, the subroutine moves the old shared-page table entries to the new storage area and releases the old space by calling the Supervisor Core Release subroutine. When control returns to the subroutine, processing continues as described previously.

Delete Page Subroutine (CEAND)

This subroutine handles an SVC request to release a number of contiguous pages from virtual storage.

Entry: CEANDA - in response to the DELPG macro instruction.

Modules Called: Find Page subroutine (CEANC) locates segment, auxiliary segment, page, and external page table entries.

SVC Queue Processor (CEAHQ) queues a GQE on the interruption queue in the TSI and resets the task lock.

Supervisor Core Release subroutine (CEAL1 entered at CEAL02) releases old storage space.

Segment Block Remover subroutine (CEANG) removes unused segment blocks from the end of the segment table.

Auxiliary Storage Release subroutine (CEAIA) releases auxiliary storage.

Search RSPI subroutine (CEAMS) locates the proper resident-shared-page index (RSPI) entry for any specific shared-page table (SPT) number, or locates the address of the next available entry in the RSPI.

Move GQE subroutine (CEAJQ entered at CEAJMG) determines whether further processing is specified by the GQE; if not, the GQE is released. If so, the GQE pointer is placed on the appropriate scan table queue.

Paging (CEAMQ) reads a page table page into main storage.

Exits:

Normal - SVC Queue Processor.

Error - Queue Scanner.

Operation: The first step in processing the request is to transfer control to CEANCA, the Find Page subroutine, which locates the segment, auxiliary segment, page, and external-page table entries and returns their addresses via general registers. If a page table entry is not assigned to the calling task, an addressing error has occurred and an interruption must be passed on to the task. This is accomplished by calling the SVC Queue Processor which queues a GQE on the interruption queue in the TSI and resets the task lock. If the page to be deleted is in IVM, exit is to the SVC Queue Processor.

If Find Page indicates that the segment is unavailable and the page table is in a page table page, the Paging routine, CEAMQ, is called at CEAMQA to read the page table page into main storage. When control is returned to Delete Page, if the page table has been read in without error, Find Page is called again to return the necessary addresses. If the page table is not in a page table page, and the page is marked unavailable or if the return from CEAMQ indicates that the page was not read in, a check is made for a shared page (see below) and processing continues for that condition.

If the subroutine finds that the segment is available and the page is assigned, it is deleted by setting the unassigned indicator in the corresponding external-page table entry. If the page to be released is currently in main storage, the page table entry is marked unavailable, and the Supervisor Core Release subroutine is entered at CEAL02. If the page is currently in auxiliary storage, the Auxiliary Storage Release subroutine is called to release the assigned auxiliary space. On return to the subroutine the auxiliary-storage flag is reset. If there are no more pages to delete, the subroutine calls the Move GQE subroutine. If there are more pages to release, the Find Page subroutine is called, and the relevant processing steps are taken.

The subroutine then determines whether an entire segment was deleted. If not, the length of the remaining page table is stored in the segment table entry. If the page table is not in the first XTSI page, the processor determines whether there are more pages to be deleted. If so, the Find Page subroutine is called to locate the pages. If not, the Move GQE subroutine is called to queue the GQE on the next processor's queue, and on return, an exit is made to the Queue Scanner.

If the Delete Page request specified that all pages in a segment were to be released, the auxiliary-segment table is

marked unassigned, the segment table is marked unavailable, and the segment indicator in the XTSI is set. If a shared page is released, the system table lock byte is set on to indicate that a resident-shared-page index (RSPI) entry is being changed. The search-RSPI subroutine is called to locate the shared page. When control returns, the subroutine uses the shared page number to delete (set to zero) the RSPI entry returned by the subroutine. The system lock byte is then set off. If the page table was in the first XTSI page, the subroutine repacks the deleted page and external-page table entry in the XTSI, allowing if necessary, for a dummy entry to keep the length of the segment even. The location fields of the affected segment entries are updated.

If the released pages were not in the first XTSI page, the subroutine calls the Supervisor Core Release subroutine to release the main storage. When control returns, the subroutine determines whether more pages are to be released. If so, Find Page is called, and the appropriate processing steps are taken. If all specified pages have been released, the Move GQE subroutine is called to move the GQE pointer to another processor's queue or to release the storage occupied by the GQE. If, as a result of this deletion, all entries in the last segment block (for example, the last 16 entries) are now unused, the Segment Block Remover subroutine is called to remove the block. When control returns, the subroutine resets the TSI lock byte and exits to the Queue Scanner.

Set External Page Table Entries Subroutine (CEAH7)

This subroutine handles an SVC request to set external-page locations within the selected external-page table entries in the XTSI to the entries in the given list. The calling task specifies the following:

- A virtual-storage address
- A parameter count
- A bit flag
- A list of external-storage addresses

RESTRICTIONS:

- The SVC must be on a word boundary.
- The SVC and all required input parameters must be in one page.
- The input parameter must not exceed 1022 (the number of external storage addresses).

Entry: CEAH07 - in response to the SETXP macro instruction.

Modules Called: SVC Queue Processor (CEAHQ at CEAHQQ) causes a task interruption to be queued on the TSI's interruption queue.

User Core Release (CEAL1 entered at CEAL04) releases pages in main storage

Find Page subroutine (CEANC) locates segment, auxiliary-segment, page, and external-page table entries.

Auxiliary Storage Release (CEAIA entered at CEAIAR) releases pages in auxiliary storage.

Paging (CEAMQ) is called to bring in page table pages.

Exits:

Normal - SVC Queue Processor.

Error - Queue Scanner or System Error Processor.

Operation: On entry, the processor attempts to locate the SVC in main storage. If either the page table or page containing the SVC is unavailable, an ERROR SVC is issued. If the SVC is located, the processor tests the count of external-storage addresses. The count must not exceed 1022. If this maximum is exceeded, or if the count is zero, the subroutine calls the SVC Queue Processor to cause a task interruption by queuing a GQE on the TSI's interruption queue. On return the subroutine TSI lock byte is reset and an exit is made to the Queue Scanner.

Other program error conditions handled in this manner are:

- SVC is not on a word boundary.
- Input specifies illegal SDA.
- Input parameters are not all in one page.
- Input specifies IVM page.
- Page to which an external location is to be added is unassigned.

If none of the above errors exists, the subroutine computes the virtual storage address of each page for which an external address is to be stored. The corresponding external-page table entry is then located by calling the Find Page subroutine. If the page table page is not in main storage, Paging (CEAMQ) is called to read in the page table page. If the page is in main storage, it is released by User Core

Release. If the page is in transit, the instruction counter is backed up and the routine exits. The subroutine releases auxiliary storage indicated in the external-page table entry, and stores the external storage address parameter for the page in the external-page table. If the bit flag is on, the unprocessed bit in the external-page table is set.

When all processing is completed, the subroutine exits to the SVC Queue Processor.

Move External Page Table Entries Subroutine (CEAP0)

This subroutine handles an SVC request to move entries from one page table to another or to change the position of an entry within a page table. The caller specifies two virtual storage page addresses and a page count.

Entry: CEAH10 - in response to the MOVXP macro instruction.

Modules Called: Locate Page subroutine (CEAML) finds the page table and external-page table entries for the old and new addresses.

SVC Queue Processor (CEAHQ entered at CEAHQQ) causes a GQE to be queued on the TSI's interruption queue. It is also entered at CEAHQR to cause the GQE to be moved or released.

Paging (CEAMQ) brings page table pages into main storage.

Exits:

Normal - SVC Queue Processor.

Error - Queue Scanner.

Operation: On entry, this subroutine checks to see if all page table pages to be used for this SVC are in main storage. If not, a list is built of the range of page table pages needed, and their addresses, and CEAMQ is called to read them into main storage. The subroutine then calls the Locate Page subroutine to find the page table and external-page table entries for the old and new addresses. When this has been done, the processor moves the page table and external-page table entries from the old to the new addresses. As pages are moved, old page table and external-page table entries are marked as assigned but unavailable and the external-page location fields are set to zero.

If the caller specifies that an entry is to be moved to a position which is marked "unassigned", an addressing error occurs,

and the processor exits to the SVC Queue Processor which calls the Queue GQE on TSI subroutine to place the GQE pointer on the TSI's interruption queue.

If no error occurs, the processor completes processing and then exits to CEAHQR to dispose of the GQE pointer or to queue it elsewhere on the scan table. When control returns, the TSI lock byte is reset and the processor exits to the Queue Scanner.

Connect Segment to Shared Page Table Subroutine (CEAQ7) Chart AT

This subroutine handles an SVC request to connect a segment to a shared-page table. The caller specifies the number of the segment to be connected, and the number of the shared-page table to which it is to be connected.

Entry: CEAH27 - in response to the CNSEG macro instruction with SYSSHALK set.

Modules Called: XTSI Overflow subroutine (CEAMX) checks the segment number for validity, and either expands the size of the task's segment table or returns an invalid condition code of nonzero.

SVC Queue Processor (CEAHQ entered at CEAHQQ) queues the GQE on the TSI's interruption queue. It is also entered at CEAHQR to move or release the GQE.

Exits: SVC Queue Processor.

Operation: On entry, the subroutine checks contiguous main storage requirements for expanding the XTSI. If needed, Contiguous Core Allocation is called to get a new main storage area address. The Connect Segment to Shared Page Table subroutine passes this address to the XTSI Overflow subroutine (at CEAMXS) for use in expanding the segment table. If contiguous main storage is not needed, the subroutine compares the segment number against the length of the calling task's segment table. If the number is greater, the processor transfers control to CEAMXS of the XTSI Overflow subroutine, which checks the segment number for validity, and either expands the size of the task's segment table or returns an invalid condition code of nonzero. If a nonzero code is returned, the processor calls the SVC Queue Processor which calls the Queue GQE on TSI subroutine to queue the GQE on the TSI's interruption queue. This results in an exit to the Queue Scanner.

If the shared-page table number did not match any number in the XTSI's entries, the segment table for the designated segment is set unavailable, the auxiliary-segment table entry is marked as assigned and

shared, and the shared-page table number and variable flag, if present, are inserted in the entry. Exit is then to the SVC Queue Processor at CEAHQR to dispose of the GQE and exit to the Queue Scanner. If a shared page table number match is found, the segment number is inserted in the XTSSI GPR1 save area and the routine exits as specified above.

Disconnect Segment From Shared Page Table Subroutine (CEAQ8)

This subroutine responds to an SVC request to remove a segment from a shared-page table. The caller specifies a shared-page table number.

RESTRICTIONS: There must be a match for the SPT number in the auxiliary-segment table.

Entry: CEAH28 - in response to the DSSEG macro instruction.

Modules Called: SVC Queue Processor (CEAHQ entered at CEAHQQ) causes a GQE to be queued on the task's TSI interruption queue and resets the task lock. It is also called at CEAHQR to dispose of the GQE and exit to the Queue Scanner.

Segment Block Remover subroutine (CEANG) removes unused segment blocks from the end of the table.

Exits:

Normal - SVC Queue Processor:

Register 1 - number of the segment disconnected from SPT

Error - Queue Scanner.

Operation: On entry, the subroutine compares the specified shared-page table number against the shared-page table number in the caller's auxiliary-segment table. If no match is found, the SVC Queue Processor is called at CEAHQQ to cause the GQE to be queued on the caller's TSI interruption queue, after which the TSI lock byte is set off and the processor exits to the Queue Scanner.

If a match is found, the corresponding segment table entry is marked not available and both words in the auxiliary-segment table are zeroed. The number of the matching segment is stored in the general register save area of the XTSSI. A check is made to determine if the removed segment table entry was at the end of the table. If it was, the Segment Block Remover subroutine is called. If not, or on return from the subroutine, the TSI lock byte is

reset, and the subroutine exits to the Queue Scanner.

Check Protection Class Subroutine (CEAQ4) Chart AU

This subroutine checks the protection classes of consecutive half pages specified by the caller. A virtual storage address and a count of contiguous half-pages are given by the caller when the SVC is issued.

RESTRICTIONS: All pages checked must be assigned and within virtual memory and the count of half-pages must not exceed 2^{20} .

Entry: CEAQ4A - in response to the CKCLS macro instruction.

Modules Called: Find Page subroutine (CEANC entered at CEANCA) provides the location of the specified page table entry or external-page table entry when the appropriate virtual storage and TSI addresses are given.

Paging (CEAMQ) reads in page table pages.

Exits: SVC Queue Processor, after placing in XTSSI, in area normally reserved for general register zero:

Normal - protection class.

Error - zero.

Operation: On entry, the processor checks to see if it is a variable request. In this case, the XTSSI segment table pointer sign bit (XTSGOS) is cleared. If it is not variable, a check is made to see if the number of half-pages is greater than 8192. If so, a program interruption of code X'6B' (extended) is queued for the task. The Find Page subroutine is then called to find the addresses of the page table entries and external page table entries. If Find Page indicates that the page table page is not in main storage, CEAMQ is called to read it in, and on return Find Page is called again.

When control returns, the processor determines whether each page is assigned, and if so, the processor checks the two protection key fields for each page and returns to the caller the protection class of the half-pages. The protection classes are:

Class A -- The half-page may be read or written.

Class B -- The half-page may be read only.

Class C -- The half-page may not be read or written by nonprivileged routines.

Thus, if any half-page has a protection class of C, the processor returns a class C indicator. If no half-page has protection class C, but at least one half page is found with a protection class of B, class B is returned. Otherwise class A is returned. Protection class indicators are assigned as follows:

Class A -- 1
Class B -- 3
Class C -- 7

When the protection classes for all half-pages have been examined, the indicator for the most restrictive class found is stored in the general register zero save area of the caller's XTSI.

If the segment searched was a shared segment, the system sharing lock (SYSSHALK) is reset. It was set by the Find Page subroutine if the segment was shared.

If the caller specifies a protection class check of an unassigned page, the subroutine stores a zero in the general register zero save area of the caller's XTSI, and exits to CEAHQR.

Create-TSI Subroutine (CEAMC)

This subroutine effects the construction of a task status index (TSI), and the placement of a task in the delay state on the inactive list when a user issues a CRTSI macro instruction.

Assumptions: The user will issue the VSEND or VSENDR macro instruction to send information to the newly created task.

Entry: CEAMC1 - in response to the CRTSI macro instruction.

Modules Called: Task Initiation subroutine (CEAMC entered at CEAMT1) performs the necessary processing to set up a TSI and initiates a new task.

Exit: SVC Queue Processor.

Operation: On entry, the subroutine establishes addressability and calls the task-initiation subroutine (CEAMT1) to generate an initialized TSI. The processor receives a task identification from the Task Initiation subroutine indicating either the creation of a TSI or that the system limit of TSI's has been reached. The processor places this identification in the save area of the XTSI, resets the lock byte of the task that issued the SVC, and exits to the SVC Queue Processor.

Special Create TSI Processor (CEAT2)

The function of this processor is to construct an initialized TSI, regardless of the system limit on the number of TSIs, and the setting of the task initiation inhibition flag, in response to the SCRTSI macro instruction.

RESTRICTIONS: Any changes on register usage in this processor will affect its communication with the Task Initiation subroutine (CEAMC).

Assumptions: The user of this SVC will use VSEND or VSENDR to send information to the newly created task.

Entries: CEATZA - in response to SCRTSI macro instruction.

R1 - Location of GQE
R2 - Location of TSI
R3 - Location of XTSI

Modules Called: Task Initiation subroutine (CEAMC) performs the necessary processing to set up a TSI and initiate a new task.

Exits: SVC Queue Processor at CEAHQR.

Operation: Special Create TSI calls the Task Initiation subroutine at a special entry point (CEAMT2). Task creation is assured because neither the task-initiation-inhibition flag nor the system limit on TSIs is tested. If the TID of the requesting TSI is 1, indicating a system operator task, the processor changes the new TSI's TID to 2. The TID of the created TSI is stored in the XTSI register save area of the requesting task.

Delete TSI Processor (CEAMD) Chart AV

This processor causes a task to be removed from the system by having the task's TSI removed from either the active or inactive scheduling lists and by having all auxiliary and main storage occupied by the task returned to the system.

Assumptions: It is assumed that the task issuing the SVC will have disconnected itself from all shared data sets and will have unloaded, from its virtual storage, all shared programs except initial VM. In addition, all accounting work will be cleared up by the LOGOFF command program.

Entries: CEAMDT - in response to DLTSI macro instruction or call from supervisor routine.

Modules Called: User Core Release Subroutine (CEAL1 entered at CEAL04) releases unshared page storage space.

Auxiliary Storage Release Subroutine (CEAIA entered at CEAIAR) releases auxiliary storage occupied by the task's pages.

Supervisor Core Release subroutine (CEAL1 entered at CEAL02) releases main storage occupied by MCBS and/or IORCBs associated with task interrupt GQE, storage occupied by the the TSI, and storage used as a work area by purge I/O.

Move GQE subroutine (CEAJQ entered at CEAJMG) determines the sequence of queue processors required to perform the work required by the GQE and routes the GQE pointer from queue to queue until the last processor has finished the required processing. It also releases GQEs and associated PCBs when no more work remains.

Locate Page subroutine (CEAML entered at CEAMLP) locates unshared pages used by the task being deleted.

Rescheduling subroutine (CEAKZ entered at CEAKZA) removes the TSI from the active list.

Purge I/O subroutine (CEAAL entered at CEAALQ) inhibits the execution of I/O requests for all devices assigned to the task and removes TSDL entries.

Supervisor Core Allocation subroutine (CEALI entered at CEAL01) reserves main storage for use as a work area.

Exits:

Normal - Queue Scanner.

Error - System Error Processor.

Operation: On entry, after establishing its base address, the processor disables any pending real time interrupt for the task. Then, if the second scan flag is not on in the GQE, the Rescheduling subroutine (CEAKZ) is called to remove the task from the scheduling lists. Any GQE left from an incomplete time-slice-end operation is disposed of by Move GQE.

The number of main storage blocks available is increased by TSIPTS. From the segment table, all available segments are searched through their page tables in order to find and release available pages (except pages in page hold) via User Core Release. Then the XTSI pages are released through UCR. Any auxiliary storage is returned to Auxiliary Storage Release. For available shared segments, neither main storage nor auxiliary storage is released. Any main storage occupied by GQEs and associated IORCBs or MCBS is released. If any pages were found in page hold, the GQE created by the SVC is placed in the TSI and exit is

made to the Queue Scanner. Otherwise the Supervisor Core Release subroutine is then called to release the main storage space occupied by the TSI. The count of TSIs in the system is decremented, Move GQE is called to dispose of the GQE, and the processor exits to the Queue Scanner.

Set up XTSI Field Subroutine (CEAS4)

This subroutine handles a request to place information in the XTSI.

The user should refer to the description of the SETXTS macro expansions in Assembler User Macro Instructions for information about what CHAXTS fields may be set up.

Entry: CEAH44 - in response to the SETXTS macro instruction.

Modules Called: SVC Queue Processor (CEAHQ at CEAHQQ) causes a program interrupt GQE to be put on the TSI's interruption queue, unlocks the TSI lock and exits to the Queue Scanner. This occurs when a request is made to set an XTSI field that may not be set.

Exit: SVC Queue Processor.

Operation: This subroutine handles an SVC request to place certain information in the XTSI. The user of the SVC may issue the SETXTS macro instruction with a mnemonic code identifying the field to be set up in the XTSI. The macro assembler substitutes code values for the set of allowable mnemonics and expands into the appropriate SVC. When the SVC is executed the code and the information to be stored in the XTSI are loaded into general registers for access by the subroutine.

On entry, the subroutine checks the legality of the code. If the code is invalid, the processor calls the SVC Queue Processor which passes the interruption on to the calling task by queuing the GQE on the TSI's interruption queue.

If the code is valid, the processor retrieves the length and displacement of the XTSI field from the table of lengths and displacements. These values are then used to store the specified information in the XTSI field. The processor then makes a normal exit via the SVC Queue Processor.

Set up TSI Field Subroutine (CEAH2)

This processor handles SVC calls for placing information in the TSI. To determine which TSI fields may be set, the user should refer to the description of the SETUP macro instruction expansions in Assembler User Macro Instructions.

Entry: CEAH02 - in response to the SETUP macro instruction.

Modules Called: SVC Queue Processor (CEAHQ at CEAHQ) causes a program interrupt GQE to be put on the TSI's interruption queue, unlocks the TSI and exits to the Queue Scanner. This is done when a request to set a TSI field, other than those allowed, is made.

Exits:

Normal - Queue Scanner.

Error - SVC Queue Processor.

Operation: On entry, the Setup TSI Field Processor establishes its base address register, and then checks the request to determine if it is legitimate. If not, an interruption code is placed in the GQE and the SVC Queue Processor is called to queue the GQE on the TSI's interruption queue. On return, the TSI is unlocked, and exit is to the Queue Scanner.

If the request is legitimate, the processor obtains the length of the field and its displacement from the beginning of the TSI, from a table of lengths and displacements and, using these values, the processor stores the specified data in the TSI; exit is then made to the SVC Queue Processor at CEAHND.

Extract TSI Field Subroutine (CEAH2)

This subroutine handles SVC requests for information from a task's TSI. To determine which TSI fields may be extracted, a user should refer to the description of the XTRCT macro expansion in Assembler User Macro Instructions.

Entry: CEAH03 - in response to the XTRCT macro instruction.

Modules Called: SVC Queue Processor (CEAHQ at CEAHQ) causes a GQE to be put on the TSI's interruption queue, unlocks the TSI and exits to the Queue Scanner.

Exits:

Normal - to SVC Queue Processor, with specified field stored in XTSI field normally reserved for saving general registers 0 and 1.

Error - To Queue Scanner, with general registers unchanged and a task interrupt pending.

Operation: This subroutine is called when the user of a service routine issues the XTRCT macro instruction with a mnemonic code such as SYSIN. The macro assembler

substitutes code values for a set of previously defined mnemonics. This code is then used by the Extract TSI Field Processor as an index to a table which contains the displacement of the indicated field from the beginning of the TSI, and the length of the field in bytes.

On entry, the subroutine tests the input code value against an internal table to determine its validity. If it is not valid, the Queue GQE on TSI subroutine is called to queue the GQE pointer on the TSI as a program interruption. When control is returned to the subroutine, the TSI lock byte is reset and the subroutine exits to the Queue Scanner.

If the input code value is valid, the start address and length of the field are retrieved from the lengths and displacements table. The subroutine then locates the requested field of information and stores it in the XTSI register save area, thus ensuring that the requested information will be loaded into the appropriate general register when control returns to the calling task, and its registers are restored from the XTSI. Exit is then made to the SVC Queue Processor at CEAHND.

Extract XTSI Field Subroutine (CEAS4)

This subroutine handles SVC requests for information to be extracted from the XTSI. The user should refer to the description of the XTRXTS macro expansion in Assembler User Macro Instructions for information about what CHAXTS fields may be extracted.

Entry: CEAH45 - in response to the XTRXTS macro instruction.

Modules Called: SVC Queue Processor (CEAHQ at CEAHQ) queues the GQE to be put on the calling task's TSI interruption queue, resets the TSI lock, and exits to Queue Scanner.

Exits:

Normal - To SVC Queue Processor, with specified field stored in XTSI field normally reserved for saving general registers 0 and 1.

Error - To SVC Queue Processor.

Operation: This subroutine handles SVC requests for information extraction from the XTSI. The user of this SVC may issue the XTRXTS macro instruction with a mnemonic code identifying the field to be extracted. The macro assembler substitutes code values for the set of allowable mnemonics and expands them into the XTRXTS SVC. When the SVC is executed the identi-

fication code will be available to the subroutine in a general register.

On entry, the subroutine tests the legality of the identification code. If the code is invalid, the SVC Queue Processor is called to queue the GQE on the calling task's TSI interruption queue.

If the code is valid, the length and displacement of the indicated XTSI field are obtained from a table of lengths and displacements, and used as the values for storing the requested information in the XTSI register save area, thus making it available to the calling task when it is restarted and its registers are restored. The processor exits to the SVC Queue Processor.

Time Slice End Subroutine (CEAHQ entered at CEAHQF)

This subroutine responds to an SVC for a service routine or a system programmer to force premature time-slice end.

Entry: CEAHQF - in response to the TSEND macro instruction.

Modules Called: Move GQE subroutine (CEAJQ entered at CEAJMG) determines whether further processing is specified by the GQE. If not, the GQE is released. If so, the GQE is queued on the appropriate scan table queue.

Exits:

Normal - SVC Queue Processor.

Error - Queue Scanner.

Operation: On entry, the subroutine sets the forced-TSE indicator in the specified GQE, and then changes the sequence string of queue processors' loc-on-queue values in the GQE so that the Timer Interrupt Queue Processor will be the next processor in line to perform work for the GQE. The Time Slice End Processor then calls the Move GQE subroutine, which examines the GQE and queues it on the Timer Interrupt Queue Processor's scan table queue. When control returns, the Time Slice End subroutine sets the TSI lock byte off and exits to the Queue Scanner.

AWAIT SVC Subroutine (CEAP7) Chart AW

This subroutine responds to a task's SVC request to test for completion of a given event. If the event is complete, it will return to the task. If it is not, the task is put in delay state and an AWAIT extension allowed before the task is forced to time slice end. It also increments two fields in the system statistical table:

SSTAWT, each time it is called for AWAIT processing; and SSTTWT, each time it is called to complete TWAIT processing.

RESTRICTION: The SVC must be the object of an EX instruction and must be located in the second half-word of a full word, which is an event control block (ECB). The delay flag must not be set and the SVC must be remotely executed.

Assumptions: If Queue GQE on TSI subroutine finds the delay bit on when queueing an interruption for the task, it will 1) reset the delay bit and turn on the ready bit in the TSI and, 2) cancel the real time interruption and turn off the AWAIT flag in the TSI.

Entries:

CEAH17 - by the SVC Queue Processor in response to the AWAIT macro instruction.

- by CEAR0 to complete processing for the TWAIT macro instruction.

Modules Called: Supervisor Core Allocation (CEAL1 entered at CEAL01) to obtain main storage for a GQE.

Real Time Interrupt subroutine (CEAS6 entered at CEAS6A) for a reading of the real time clock.

Set Real Time Interval subroutine (CEAS7 entered at CEAS7A) to set a real time interruption for the AWAIT extension period.

Move GQE subroutine (CEAJQ entered at CEAJMG) to queue up a forced time slice end on the timer interrupt processor's queue.

Exits: Upon normal completion the await-SVC subroutine exits to the SVC Queue Processor. Any abnormal condition will result in an exit to the Queue Scanner.

Operation: On entry, the subroutine determines whether the requesting SVC was remotely executed and whether the SVC instruction was on the second halfword boundary of a word. The latter is determined by examining the virtual storage address stored in the GQE. If both of these conditions are not satisfied an error has occurred, in which case the subroutine calls the Queue GQE on TSI subroutine to place the GQE pointer on the TSI's program interruption queue entry. When control returns to the processor, the TSI lock byte is turned off and an exit is made to the Queue Scanner.

If the above conditions are satisfied the subroutine computes the location of the

task's event control block (ECB) by doing an LRA from the location of the SVC specified in the GQE, and subtracting two. (A major syser results if the LRA fails.) When control returns the subroutine tests the event-completion indicator in the ECB. If the indicator specifies that the event is complete, the subroutine exits to the SVC Queue Processor.

If the event is not complete, a check is made to determine whether any interruptions are pending for the task. If so, CEAP7 exits to the SVC Queue Processor. Otherwise, the delay flag in the TSI is checked; if it is on, a major syser is issued; if not, the AWAIT flag is set and a check made to determine whether the task should be time-slice ended immediately or a timer interval set up; exit is then to the SVC Queue Processor at CEAHND.

TWAIT Subprocessor (CEAR0)

This subroutine responds to a task's SVC request to test for the completion of a given event.

RESTRICTIONS: The SVC must be the object of an EX instruction and must be located in the second half-word of a full word, which is an event control block (ECB). The delay flag must not be set.

Entries: CEAH30 - in response to the TWAIT macro instruction.

Exits: AWAIT SVC processor.

Operation: This subroutine sets TWAIT indicator flags in the TSI and GQE. It then exits to the AWAIT SVC processor.

Pulse Schedule Table Entry Processor (CEAR2)

This routine responds to a task's request to assign it the schedule table entry specified in the Pulse level field of its current STE.

Entry: CEAR2A - in response to the PULSE macro instruction.

Exit: Queue Scanner.

Operation: This routine extracts the STE field (STEPULSE) and then determines its validity by checking the following two conditions:

1. Is it within the bounds of the current schedule table?
2. Is the PULSE level zero?

If it is invalid, the task's STE level is left unchanged and a return code is set in

register 15 to indicate the condition. If the PULSE level is valid, the contents of STEPULSE are moved to the schedule table entry field in the task's TSI (TSISTE). The return code is set in register 15, and exit is taken to the Queue Scanner.

The return register (GPR 15) reflects the action taken by this routine as follows:

- Byte 0 - Set to 00 if valid change made
- Set to 01 if PULSE level out of schedule table bounds (no change made to current STE for task)
- Set to 02 if PULSE level is zero (no change made to current STE for task)
- Byte 1 - Contains old STE level for task
- Byte 2 - Unused
- Byte 3 - Contains level indicated by STEPULSE.

Change Schedule Table Entry Processor (CEAR3)

This routine responds to a privileged task's request to have a specific schedule table entry level assigned to it.

Entry: CEAR3A - in response to the CHANGE macro instruction.

Exits:

- Normal - SVC Queue Processor.
- Error - Queue Scanner.

Operation: As input, this routine receives the new schedule table entry level to be assigned in general register 6 (task register 15). Then, two conditions are checked:

1. Is the requested level change outside the bounds of the current schedule table?
2. Is the requested entry zero?

If either condition exists, the schedule table entry level is left unchanged. Otherwise, the task's STE level is changed by inserting the contents of the input register in the schedule table entry field of the task's TSI (TSISTE). The return register (15) is then set to indicate what action was taken as follows:

- Byte 0 - Set to 00 if valid change made

Set to 01 if requested entry level out of bounds (No change in level made)

Set to 02 if requested entry was zero (No change in level made)

Byte 1 - Contains old STE level of task

Byte 2 - Unused

Byte 3 - Contains requested level (If byte zero is zero, this will be the new entry level of the task.)

This routine is also entered to process requests invoked by issuing the PRESENT macro instruction. PRESENT requests a task's current schedule table entry level. Code in the macro expansion causes the request for a CHANGE outside of the limits of the table. CEAR3 processes the request for case one described above and returns a condition code of 01 in the first byte of register 15 and the STE level (old) of the task in byte two.

Set User Interval Timer Subroutine (CEA02)

This subroutine responds to a user request to limit the amount of CPU time to be utilized on a given task or on some phase of the task. The user specifies the time value, in milliseconds, that should be used to set the user interval timer.

RESTRICTIONS: The limit of the interval time is 55,364,812 milliseconds and any attempt to set the timer in excess of this value will result in the generation of a task interrupt.

Entry: CEAH22 - in response to the SETTU macro instruction.

Modules Called: Queue GQE on TSI subroutine (CEAAF) queues the GQE on the TSI's interruption queue.

Move GQE subroutine (CEAJQ entered a CEAJMG) determines whether further processing is specified by the GQE. If not, the GQE is released. If so, the GQE is queued on the appropriate scan table queue.

Exits:

Normal - SVC Queue Processor.

Error - Queue Scanner or System Error Processor.

Operation: On entry, the subroutine establishes its base register, and then determines whether the task's XTSI has been swapped out of main storage. If so, a system error SVC is issued.

If the XTSI is in main storage, the subroutine tests the specified time value. If it exceeds 55,364,812, a task program interruption is generated. This is accomplished by calling the Queue GQE on TSI subroutine to queue the GQE on the TSI's program interruption queue. When control returns, the subroutine resets the TSI lock byte and exits to the Queue Scanner.

If the value is within the acceptable limits, it is converted to an equivalent number of timer cycles (or "ticks"). The converted number is stored in the user timer value field of the task's XTSI. The subroutine then calls the Move GQE subroutine to dispose of the GQE, and when control returns, the TSI lock byte is reset and the subroutine exits to the Queue Scanner.

Set Real Time Interval Subroutine (CEAS7) Chart AX

This subroutine sets up a task timer interrupt at a specified future time requested by a task.

RESTRICTIONS:

- If the number of pending real time interruptions has reached the system limit, subsequent requests to set interrupts will be ignored, and a return code of 0 sent to a virtual memory task, or 4 to a supervisor task.
- Up to one page (4032 bytes) of supervisor main storage may be used for the queue of real-time intervals. Currently, each queue entry is 16 bytes long making the system limit 252 queue entries.
- Only one real time interrupt may be pending for any one virtual memory task at any one time.

Entries:

CEAS7A - by the SVC Queue Processor for a virtual memory request with the following input:

Register 1 - address of the GQE

Register 2 - address of the TSI

Register 3 - address of the XTSI

Registers 4&5 - a doubleword, fixed-point timer value.

- by a supervisor routine via the SETTIMER macro instruction with the following input:

Register 0 - adcon for the entry point of module to receive control when interrupt occurs

Register 1 - all zeros

Register 2 - location of TSI or any value user wants returned at interrupt time. Doubleword field in PSA - real time value.

Modules Called: Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) when more space is needed for the queue of real time intervals.

Supervisor Core Release (CEAL1 entered at CEAL02) releases the old core block when a new one has been allocated.

Exits: Normal exit after a virtual memory request has been handled is to the SVC queue processor with register 1 containing the address of the GQE, a return code in XTSG1S, and the system lock reset.

If the request was from a supervisor routine, normal exit is made to the caller after placing a return code in register 1.

Exit is to the system error routine if the real time interval lock (SYSLCK), tested at entry, remains set longer than permitted.

If a virtual memory task requests cancellation of a queue entry that cannot be found on that task's queue, a task error, program interrupt, is declared. A hexadecimal '5B' is stored in GQEINT, and Queue GQE on TSI is called to queue the interrupt on the appropriate TSI. On return, the TSI lock and the system lock are reset and control is transferred to the Queue Scanner.

Operation: When entered, this routine tests the real time interval lock (SYSLCK). If it is nonzero, a loop is entered waiting for the lock to be reset. If the lock is not reset in time, a system error results.

If SYSLCK is zero, or reset in time, this routine sets it and checks register 1 to see if the request is from a supervisor task (register 1 = 0). If it is, general registers are stored in this routine's own save area and in PSASCU and PSASCU +4. The real time value from PSASCU+8 and PSASCU+C is loaded into registers 4 and 5.

The following conditions are then checked:

- Has the number of pending real time interrupts reached the system limit?
- Is there more than one interrupt pending this task?

- Is this a request to cancel a pending real time interrupt?

If pending real time interrupts are at the system limit, return codes are: 0 for a virtual memory task, and 4 for a supervisor task.

The return code for a successful pass through this subroutine by a supervisor request is a hex 'C'. For a virtual memory request, it is an 8. A successful pass may result in one of four possible actions:

- A real time interrupt is set for a task (supervisor or virtual memory) that did not have an interrupt pending.
- A real time interrupt is canceled and not replaced (supervisor or virtual memory).
- A pending real time interrupt is canceled and a new one is set to replace it (virtual memory only).

Any request from a virtual memory task to set an interrupt when one is already pending results in the pending one being replaced by the new request. If the input time value is 0, it is assumed that the task (supervisor or virtual memory) wants to cancel a previously set interrupt without replacing it.

This subroutine creates a four-word entry to be placed in the real-time-interval-pending queue (CHARTI), in the order of increasing real time, with the most imminent time value at the top of the queue. The four-word entry includes an eight-byte time value (RTITIME), a four-byte address of the TSI, if applicable (RTITSI), a flag byte (RTIFLAG) containing the cancel interrupt flag (RTICNCL) and the ADCON present flag (RTIADP), and a three-byte address of the ADCON of the supervisor routine to receive control when the interrupt occurs. The time value in the first queue entry is compared to the system's current time at frequent intervals by the dispatcher (CEAKD). When the dispatcher finds that the time specified for interrupt has either arrived or passed, control is passed to the Create Real Time Interrupt subroutine (CEAKR) where a task timer interrupt is queued on the appropriate task.

Much of the detailed operation of this portion of the Set Real Time Interval subroutine is concerned with the maintenance of three pointers and three counters in the system table (CHASYS) which control access to the real time interval queue (CHARTI). Each entry in the queue is twelve bytes long. Main storage space for the queue is obtained from Supervisor Core Allocation.

The original main storage request is for 64 bytes. This block is large enough for four entries. When it is necessary to make a fifth entry, more main storage is requested. Since the entries in the queue are frequently reordered, chronologically, the 64-byte blocks of main storage are not chained. Instead, when more space is needed, N+64 bytes are requested, where N equals the number of bytes in the existing main storage block. When the new block is received, the existing queue is moved into it from the old block, and the old block returned to Supervisor Core Release. This procedure simplifies the necessary reordering because the queue entries remain contiguous.

The three pointers maintained by this subroutine contain: (1) the address of the first word of the current block, (2) the address of the first entry in the queue (this will differ from the address in the first pointer when an entry is deleted by the Create Real Time Interrupt subroutine), and (3) the address of the word following the last word of the last queue entry.

The three counters maintained contain, respectively, the total number of bytes in the current block (always a multiple of 64), the number of bytes being used, and the number of bytes that have been released from the top of the block by the Create Real Time Interrupt subroutine.

Restore Elapsed Time Subroutine (CEAS8)

This subroutine responds to an SVC request to reset the elapsed time for all CPUs in the system.

Entry: CEAH48 - in response to the RSTTIM macro instruction.

Exits:

Normal - SVC Queue Processor.

Error - Queue Scanner.

Operation: On entry, the processor sets to zero the elapsed timers for all CPUs in the system. The processor adds the elapsed-time value for the CPU controlled by the operative task to the time-of-day clock value in the system table. If the resulting value in the time-of-day clock exceeds 24 hours, the year-month-day field in the system table is incremented by one and a value equivalent to 24 hours is subtracted from the time-of-day clock. The processor then exits to the SVC Queue Processor.

Read-Time Subroutine (CEAS6)

This subroutine responds to a request to generate a doubleword, fixed-point number

representing: year, month, day, hour, minute, second, millisecond, and microsecond.

Entries: CEAH47 for a virtual memory request, control is transferred to this routine from the SVC Queue Processor (CEAHQ) with the following general register configuration:

Register 1 - contains the address of the GQE.

Register 2 - contains the address of the TSI.

Register 3 - holds the address of the XTSI.

Registers 4, 5 and 6 - contain what were in registers 0, 1, and 15, respectively, at the time the SVC was executed.

For a supervisor task, register 1 contains a code of 0 and register 14 the return address of the calling routine.

Exits: After normal completion of a request from virtual memory, this subroutine exits to the SVC Queue Processor.

Normal completion for a supervisor request sees control returned to the calling routine through a branch to the address specified by register 14.

Exit is to the System Error routine if the routine lock byte (tested at entry) is set too long, or if a negative number or overflow results from the sum of the contents of the system table's year-month-day cell (SYSYMD), the time-of-day clock (SYS-TOD), and the elapsed-time cell of the prefixed storage area (PSAETM).

Operation: When entered, this subroutine tests the routine lock byte (RTLCK). If it is found to be non-zero, a loop is entered until it is reset, or the waiting time is exceeded. If the lock is not reset within the waiting time, a system error results.

If RTLCK is 0, or reset within the waiting time, general registers are stored in this routine's save area. The contents of SYSYMD, SYSTOD, and PSAETM are then added together, SYSYMD is the year-month-day cell in the system table; and SYSTOD is the system table's time-of-day clock. PSAETM is a cell in the prefixed storage area of a CPU containing the elapsed time measured by that CPU's interval timer.

If a negative number, or one that exceeds 2^{-1} , results from this addition, exit is made to the System Error routine.

If a legal sum results, this subroutine determines whether the request was from a virtual memory task or a supervisor task. If general register 1 contains a nonzero code signifying a virtual memory request, the calculated system elapsed time is stored in the doubleword defined by the juxtaposition of XTSG0S and XTSG1S. The routine lock byte is reset and exit is made to the SVC Queue Processor at CEAHQR.

If the request was from a supervisor task (0 code in register 1), the calculated number is returned in general registers 0 and 1. Registers 2 through 14 are restored and the routine lock byte reset before return is made to the address of the calling task addressed by register 14.

SYSTEM TABLE MODIFICATION AND EXTRACTION PROCESSORS

Set up System Table Field Subroutine (CEAS2 Entered at CEAH42)

This subroutine responds to a request to insert certain information into the system table. The caller specifies a field identification code and the information to be inserted in the field. The user may issue the SETSYS macro with the code in mnemonic form. Valid mnemonics for designating fields in the system table are described in the System Programmer's Guide.

RESTRICTIONS: The system table (CHASYS) fields which may be set up by this SVC are intentionally omitted here because the set of allowable fields may change. These fields are assigned mnemonics by the macro assembler and a potential user should refer to the description of the SETSYS macro expansion for information about which CHASYS fields may be set up.

Entry: CEAH42 - in response to the SETSYS macro instruction.

Modules Called: SVC Queue Processor (CEAHQ) causes a GQE to be queued on the task's TSI interruption queue.

Exits:

Normal - SVC Queue Processor.

Error - Queue Scanner.

Operation: On entry, the subroutine tests the legality of the specified code. If it is illegal, the SVC Queue Processor is called to cause the GQE to be moved to the task's TSI interruption queue, resets the TSI lock byte and exits to the Queue Scanner.

If the code is legal, the subroutine sets up the field with the required information, resets the TSI lock byte and exits to the SVC Queue Processor. In order to protect the system table fields (SYSYMD and SYSTOD) when information is to be inserted in them, the system table time fields are set and interrupts disabled. If the field to be set is time of day (TOD), a call is made to Restore Elapsed Time (CEAS8) to synchronize elapsed time in all CPUs and their associated time cells in the PSA. On return from this call, the time lock is reset and interrupts enabled.

Extract System Table Field Subroutine (CEAS2 Entered at CEAH43)

This processor responds to a request to extract certain information from the system table. The caller specifies a field identification code. The caller may issue the SVC with the code in mnemonic form. The valid mnemonics for designating fields are described in the System Programmer's Guide.

The user should refer to the description of the XTRSYS macro expansion in Assembler User Macro Instructions for information about what CHASYS fields may be extracted.

Entry: CEAH43 - in response to the XTRSYS macro instruction.

Modules Called: None.

Exit: SVC Queue Processor.

Operation: On entry, the subroutine tests the legality of the identification code. If the code is illegal, the SVC Queue Processor is called to cause the GQE to be queued on the task's TSI interruption queue.

If the code is legal, the subroutine obtains the length and displacement of the indicated field from the table of lengths and displacements and uses these values to insert the information in the XTSI's general register save area. When this is accomplished, the subroutine exits to the SVC Queue Processor.

Whenever the field to be extracted is a time field, the time lock byte in the system table is set and interrupts disabled. When the operation is complete, the time lock is reset and interrupts enabled.

Extract Accumulated Time Routine (CEAT1)

This routine computes the accumulated CPU time used by a task and communicates this value to the calling routine.

Entry: CEAT1A - in response to the XTRTM macro instruction.

Exit: SVC Queue Processor.

Operation: This processor is passed the address of the XTSI in general register three. The accumulated time is computed by subtracting the current timer value from the last time slice value and adding the accumulated time value to the difference. This result is then converted from clock ticks to milliseconds and is stored in the save area for general register one in the XTSI. An exit is then taken to the SVC Queue Processor at CEAHND. When the task is reactivated, the accumulated time will be loaded into register one.

Extract from Auxiliary Storage Allocation Table (CEAT4)

This routine extracts fields from the auxiliary storage allocation table (ASA) and presents them to the user on request. The ASA fields that may be requested are: The auxiliary drum pages available (ASATMA) and the auxiliary disk pages available (ASATKA).

Entry: CEAT4A

Input

- Register 1 - address of the GQE
- Register 2 - address of the TSI
- Register 3 - address of the XTSI

Exit: Queue Scanner.

Operation: On entry, this routine establishes addressability and then extracts the drum and disk pages available counts from the ASA. ASATMA is stored into the field for general register zero of the XTSI (XTSGOS), and ASATKA is put in the field for general register one (XTSG1S). Thus, when the requesting task is reactivated, its general register fields are restored from the XTSI and the requested information will be in general registers one and two.

I/O Call Subroutine (CEAA0) Chart AY

This subroutine provides the preliminary processing for all input/output interruptions caused by the issuance of an SVC instruction embedded in the first two bytes of task's IORCB in virtual storage. I/O Call performs the initial preparation of the IORCB and queues the GQE pointer on the TSI or device queue processor's queue.

I/O Call performs the following operations when it detects an error:

- Places an interruption code in the GQE.

- Calls the Queue GQE on TSI subroutine to queue the GQE on the TSI's interruption queue.
- Unlocks the TSI.
- Returns the work area via Supervisor Core Release.
- Exits to the Queue Scanner.

RESTRICTIONS: The SVC must be in the IORCB, the task must execute the SVC, and the IORCB must be contained in one page.

Entries: The I/O Call subroutine is entered, from the SVC Queue Processor, at CEAA01.

Modules Called: Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) reserves work storage for the use of the processor and storage space for the IORCB.

Queue GQE on TSI subroutine (CEAAF) places a pointer to the specified GQE on the queue in the affected task's TSI.

Locate Page subroutine (CEAML) provides the location of a page table entry or external page table entry when the appropriate virtual storage and TSI addresses are given.

Supervisor Core Release subroutine (CEAL1 entered at CEAL02) releases main storage after use.

Command Word Relocator (CEAAA) relocates CCWs in the event of an IORCB chaining request.

Dequeue I/O Requests (CEAAJ) dequeues GQEs from a device queue in the event that IORCB chaining request is successful but command word relocations fails.

Set Suppress Flags subroutine (CEAJQ entered at CEAJSF) turns on flag F3 in the scan table entry for the device on which the GQE is queued.

Enqueue GQE subroutine (CEAJQ entered at CEAJEN) places GQE pointers on the specified scan table queues.

Paging (CEAMQ) brings in all pages necessary to perform the start-I/O operation.

Exits: Upon normal completion of its work the I/O Call subroutine exits to the Queue Scanner which will eventually invoke the device queue processor if F3 is set. Any abnormal condition results in an exit to the Queue Scanner after first queuing the GQE on the TSI interruption queue.

Operation: On entry, I/O Call requests work space from the Supervisor Core Allocation subroutine, and calls the Locate Page subroutine to find the real storage address of the IOCAL SVC. If the SVC page does not exist in main storage or in the task's virtual storage, the I/O SVC is processed as an error as described above. If the page does have a main storage address, I/O Call turns on the I/O page flag in the external page table, and checks the system-symbolic-device-address field in the IORCB against the task-symbolic device list to insure that the requested device is assigned to the task. If so, the 'reject I/O requests' flag is checked to determine whether the device is suppressed. If the device is either unassigned or suppressed, the I/O Call Processor passes the interruption on to the task. If the device is assigned and unsuppressed, I/O Call calculates and tests the IORCB's size. If the IORCB is equal to zero, or greater than 1,920 bytes, the interruption is also passed on to the task. This is also the action if the IORCB crosses a page boundary or if the pagelist-length is greater than 8.

If no IOCAL errors are detected, the Supervisor Core Allocation subroutine is called to obtain the necessary resident storage space to contain the IORCB. If no space is available, the task's time slice is terminated, the task's instruction counter is reset to the virtual storage location from which the IOCAL was issued, the GQE and work areas are returned, the TSI unlocked, the page-hold counter in the external-page table is decremented and an exit made to the Queue Scanner.

The device type field (IORDT) in the IORCB is then checked. If the device type is disk, the disk operations counter (TSICPR) in the TSI is incremented by one. This counter is then compared to the maximum disk operations allowed field (STEMAXRD) in the schedule table (CHASTE).

When TSICPR exceeds STEMAXRD, the task is forced to time slice end by creating a GQE and queuing it on the Timer Interrupt Processor's queue via Move GQE (CEAJMG). Exit is then to the Queue Scanner.

In those cases where the device type is not disk, or when TSICPR does not exceed STEMAXRD, the IORCB's 'resident origin of the buffer' field is set.

If not, the IORCB chaining flag is checked. If on, IOCAL will chain together two or more IORCBs for the same device by overlaying a transfer in channel onto the last CCW to the first CCW in another IORCB, the I/O-counter in the TSI is incremented by 1, the TSI's 'I/O wait' flag is set on, and the Enqueue GQE Subroutine is called to

place the GQE pointer on the correct device queue processor's queue. The TSI is unlocked, the work area is returned and an exit is made to the Queue Scanner.

Paging is then called. The pagelist pointer is placed in register 0, and a pointer to the GQE is placed in register 1. If Paging is unsuccessful, Start I/O will create an extended program interruption. Otherwise, when Paging returns, the requested pages will be in main storage and in I/O hold.

Note: Since the IOCAL paging operation proceeds in parallel with IOCAL preparations it is possible to bring pages into main storage before the IOCAL operation is completed. Therefore, IOCAL has to assume the responsibility for turning off the page-wait flag and turning on the ready flag in the TSI and preparing to re-issue the IOCAL if the paging counter goes to zero.

The same logic is used when the 'any page' flag in the IORCB is on. The only difference is that the 'null state' flag in the PCB entry is set on and the page-in portion of the operation is consequently ignored. Processing continues until the list is exhausted, after which IOCAL exits to the SVC Queue Processor.

Pageout Service Subroutine (CEAA1) Chart AZ

This processor provides the required logic to complete a VAM pageout I/O SVC request when the task determines that one or more of its virtual storage pages is to be returned to external storage.

RESTRICTIONS: The parameter list (which includes the SVC) must originate on a doubleword boundary and not cross a page boundary. The number of virtual storage pages returned to external storage by one pageout SVC call must not exceed 8.

Entry: CEAA11 - in response to the PGOUT macro instruction.

Modules Called: Queue GQE on TSI subroutine (CEAAF) places the GQE pointer on the calling task's TSI interruption queue.

Supervisor Core Allocation subroutine (CEAL entered at CEAL01) handles requests for allocation of working storage.

Locate Page subroutine (CEAML) provides the location of any page table entry or external-page-table entry when the appropriate virtual storage and TSI addresses are given.

Enqueue GQE subroutine (CEAJQ entered at CEAJEN) places GQEs on device queues.

Supervisor Core Release subroutine (CEAL1 entered at CEAL02) releases the working storage.

Paging (CEAMQ) reads in page table pages and virtual memory pages.

Exit: Queue Scanner.

Operation: When changed virtual storage pages are returned to external storage, they are copied from virtual storage to external storage. Such pages are not deleted from virtual storage, however, until the space they occupy is required for other user pages. The specified virtual-storage pages may be residing in main storage, auxiliary storage or both. If any of the pages reside in auxiliary storage, they must first be paged into main storage and then transferred to external storage. The processor provides the necessary control to split the pageout operation into these two logical steps when they are required. The first step in processing is to bring all the virtual storage pages into main storage. Step two effects the transfer of the pages to external storage.

Pageout is called when an SVC, contained in the first two bytes of an I/O-page-control block (IOPCB) in virtual storage, is executed. An interruption GQE is generated and queued on the SVC Queue Processor's scan table queue. When the Queue Scanner activates the SVC Queue Processor, control is transferred to Pageout.

On entry at CEAA11, Pageout requests working storage from the Supervisor Core Allocation subroutine. When space is allocated and control returned, pageout tests the protection key in the program status word (PSW). If the protection key is non-zero, a nonprivileged task issued the SVC call. Hence, Pageout calls the Queue GQE on TSI subroutine to place the GQE pointer on the calling task's TSI interruption queue. When this is done and control returns, Pageout unlocks the TSI lock byte, and calls the Supervisor Core Release subroutine to release the working storage. When control returns Pageout exits to the Queue Scanner.

If the calling task is privileged, the virtual address in the GQE is converted to an actual main-storage address. Pageout then calls the Locate Page subroutine to determine the present location of the mpageout SVC. A task program interruption occurs if any of the following conditions exist:

- The pageout SVC is not in main storage.
- The pageout SVC is not in the task's virtual storage.

- The IOPCB crosses a page boundary.

In any of these cases, Pageout calls the Queue GQE on TSI subroutine and exits as described previously.

If none of these conditions exists, pageout retrieves parameter data from the IOPCB and stores it in its work area. Pageout tests the 'number of pages to be transferred' parameter. If it is equal to zero or greater than eight, the task interruption exit procedure is entered.

If the number of pages to be transferred to external storage is less than 9 and greater than zero, Pageout checks the external address against the task-symbolic-device list to determine whether the requested devices are assigned to the task. If a requested symbolic device address is not found, a page error indicator is placed in the general register save area of the XTSI, and the task-I/O-interruption exit procedure is initiated. The same action is taken if there is no task-symbolic-device-list pointer in the calling task's TSI.

Pageout then checks to see if the page tables for the pages to be transferred are in main storage. If not, the Paging subroutine (CEAMQ) is called to read in the page table page. On return from Paging, or if the page tables were already in main storage, Pageout makes sure none of the pages to be transferred were changed since the request was made (the external addresses are the same as the ones given in the SVC). Pageout then sets up a parameter list and calls the Paging subroutine to read in the virtual memory pages and put them in I/O hold. On return, all requested pages are in main storage, and Pageout begins the second phase of its operation:

- Sets the TSI to the number of pages to be transferred to external storage.
- Scans the task-symbolic list to insure that all devices are still assigned to the task.

If a requested device is not still assigned and there are unprocessed pages remaining, the 'page ready' flag is turned on in the TSI. The Supervisor Core Release subroutine is called to release the main storage occupied by the copy GQE. An interrupt code is then placed in the GQE and the Queue GQE on TSI subroutine is called to place the GQE on the interruption queue in the affected task's TSI. The Supervisor Core Release subroutine is then called to release Pageout's work area and Pageout exits to the Queue Scanner.

If all requested devices are still assigned to the task, Pageout scans through

the list of external-storage addresses and constructs GQEs and PCB entries for each device. For each external address, Pageout constructs a copy GQE containing pointers to:

- The TSI.
- The XTSI's save area.
- The PCB.
- The paging and VAM indicators.

A PCB entry is made up containing the virtual, core block, and external storage addresses. The Pageout and write flags are set on and an indicator is set to tell the Channel Interrupt Queue Processor which entry in the list of external-storage addresses caused this PCB entry to be generated. All other PCBE fields are set to zero. The external-storage address from the original list is zeroed out and the next external-storage address, referring to the same symbolic device, is processed. At this time, the PCB entry count is added to both the system table counter field and the PCB entry count in the GQE.

At this point the Move GQE subroutine is called to place the GQE pointer on the device queue processor's scan table queue.

The above steps are repeated until all pages have been processed. When PCB entries for all pages to be transferred to the external storage have been made up, the master-GQE-pointer in the TSI is set to zero, and the storage space occupied by the GQE, PCB, and the work area is returned by calling Supervisor Core Release. When control is returned, Pageout exits to the Queue Scanner.

Remote Job Entry Line Control Subroutine (CEABC) Chart BA

The RJE Line Control subroutine performs three functions for binary synchronous communication between the data adapter unit and the RJE transmission terminal:

- Enable Line - Activate the specified line and initialize it for data transfer.
- Prime Line - Reinitialize the specified line for data transfer.
- Disable Line - Deactivate the specified line after either normal or abnormal job termination.

Entries: CEABC1 - as a result of the RJELC macro instruction being issued, with the following input:

- Register 1 - address of the SVC GQE
- Register 2 - address of the task's TSI
- Register 3 - address of the task's XTSI
- Register 4 - contents of the task's register 0:

Byte 0 - Flags:

X'80' = bypass SCANT clearing

X'40' = bypass calling HIO

Bytes 2&3 - Symbolic device address

- Register 5 - contents of task's register 1:

Function code

0 = prime line

1 = enable line

2 = disable line

- Register 15 - the calling address

CEABC2 - from supervisor routines, with the following input:

- Register 0 - same as register 4 for SVC entry

- Register 1 - input code:

0 = prime

1 = enable

2 = disable

- Register 14 - return address

- Register 15 - calling address

Modules Called: Supervisor Core Allocation (CEAL1 entered at CEAL01) is called to get main storage for a work area when this routine's work area (CEABCS) is locked (indicating in use by another CPU).

Dequeue GQE subroutine (CEAJQ entered at CEAJDE) is called to remove GQEs (left over from asynchronous error retry operations) from the scan table.

Supervisor Core Release (CEAL1 entered at CEAL02) is called to release work area space and space for those GQEs dequeued from the scan table.

Set Suppress Flags subroutine (CEAJQ entered at CEAJSF) turns off the F4 suppress flag after a scan table clearing operation.

Pathfinding (CEAA5 entered at CEAA5P) to assign a path to the device for which the line control operation is to be performed.

Halt I/O (CEAAI entered at CEAAIH) halts I/O operations on the device to be serviced.

Start I/O (CEAAG entered at CEAG1) starts I/O on the indicated CCW list for the line control function requested.

Generate and Enqueue Interrupt GQE (CEABQ entered at CEABQ1) creates a dummy I/O interruption GQE and puts it on the channel interrupt processor's queue when start I/O fails with a condition code of one and a busy condition is not indicated.

Reverse Pathfinding (CEAA5 entered at CEAA5R) is called to release the path when the disable line function is being performed. Reverse Pathfinding is also called to locate the device's asynchronous entry in the device group table after a halt I/O operation.

Exits: If invoked for SVC processing, exit is to the SVC Queue Processor at CEAHQR (to dispose of the SVC GQE before going to the Queue Scanner) with the following output for the task:

Register 0 - If Start I/O failed, bytes 0 and 1 contain return information from the Start I/O subroutine's register 0. Bytes 2 and 3 contain the physical path address of the failing device.

Register 1 - contains a return code:

- 0 = operation successful
- 4 = start I/O or halt I/O failure
- 8 = path unavailable or invalid input
- 12 = path busy

If entered from the resident supervisor, exit is to the calling routine with the same output to the task as described above.

Error Conditions: The System Error Processing routine (CEAIS entered at CEAIS1) is invoked through the ERROR SVC when any of the following error conditions are encountered:

- Impossible combination of scan table flags set (7701)
- Illegal Pathfinding request (7702)
- Invalid RJE device type code detected (7703)
- Invalid input to CEABC (7704)
- Unable to release physical path (7705)

- Work area lock not set when attempt is made to reset it (7706)
- Path translation error detected (7707)
- DEVLOCK locked more than 50 microseconds (7708)
- DEVLOCK not set when attempt made to reset it (7709)

Operation: On entry, this routine initializes itself as required for SVC or resident supervisor subroutine processing. If a work area is needed, it obtains the space through Supervisor Core Allocation.

A check of the input is then made to determine if the scan table should be cleared. If so, this operation is performed by calling the Dequeue GQE subroutine to remove the GQE from the scan table entry and then calling Supervisor Core Release to release the GQE core space.

Pathfinding is then called to obtain a physical path to the I/O device. On return the Halt I/O input parameter is checked to see whether or not a Halt I/O operation is required. Halt I/O is called, if necessary, and its return information is checked to see if an interruption is expected. If so, the device's asynchronous entry in the device group table (DEVAE) is located by a "translate only" call to Reverse Pathfinding. The flag indicating whether to enable, prime or disable the line after Halt I/O is set and a return code of zero is set in register one before exiting. When the Halt I/O interruption is processed, the RJE Asynchronous Interrupt Subprocessor (CEABA) will examine the DEVAE flags and make the appropriate call to this routine (RJE Line Control) to start I/O for the desired line function.

If there is no I/O halt I/O operation required, parameters are set up according to the line control function requested and the Start I/O subroutine is called. If the disable function is requested, the DEVAE is located as described previously for halt I/O operations, and the RJE disable flag is set. When the disable interruption occurs, the interruption will be discarded by the RJE Asynchronous Interrupt routine.

The Start I/O return parameters are checked. If the operation was successful, a return code of zero is set, the path is freed by calling reverse pathfinding, and the appropriate exit taken.

If the Start I/O operation resulted in status being stored (non-busy), the information returned by Start I/O in register zero is preserved, a return code of four is set and exit taken.

For other conditions, the Start I/O operation is retried until successful or until the retry threshold is reached. If the retry threshold is reached, the Start I/O return information in register zero is preserved, return code of four is set, the path freed, and the proper exit taken.

Reset Device Suppression Flag Subroutine (CEAAH)

This subroutine responds to an SVC request to zero the suppression flag of a specific device or of all devices in the task symbolic device list. The calling task specifies the system symbolic device address for a specific device request.

Entry: CEAAHR - in response to the RESET macro instruction.

Exits:

- Normal - SVC Queue Processor.
- Error - SVC Queue Processor

Register 0 - error indication

Operation: On entry, the subroutine obtains a pointer to the list of devices assigned to the task from the TSI. Each device-list entry in the task symbolic device list contains:

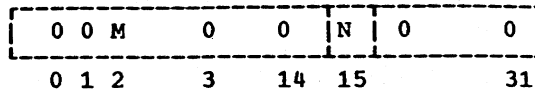
- The device address.
- In-use indicators.
- Queue-device request counts.
- Suppress indicators.

For a specific device, the subroutine searches the device list for an address which matches the input address. If none is found, an error flag is returned to the calling task via general register zero. If there is a match, the suppression indicator for the specified device entry is set to zero, and the subroutine exits to the SVC Queue Processor at CEAHND. If no TSDL exists (for suppress all devices requests) an error code is returned to the task.

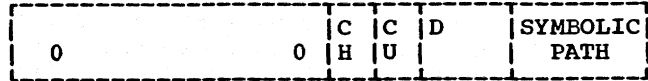
Set Path Subroutine (CEAAB)

This subroutine enables virtual storage tasks to set or reset a malfunctioning status indicator for a channel, control unit or device path. The caller specifies, via general registers, the following:

register zero



one



unit flags
16 17 18 19 31

M = set unit malfunctioning: 1 -- malfunctioning, 0 -- OK

N = must equal one, indicates path to be set is in register one.

Unit Flags = bits 16, 17, 18 indicate which components of the path, channel, control unit, device respectively, are to be set.

Path = 13 bit physical or actual address if flag in bit 5 of register zero is off; if it is on, a symbolic address is indicated. This address must be complete down to lowest unit being set.

Entry: CEAAB1 - in response to the SPATH macro instruction.

Modules Called: The Set Path portion of the Pathfinding subroutine (CEAA5S) sets the requested flags for the path.

Exits:

- Normal - SVC Queue Processor.
- Error - SVC Queue Processor with error indication stored in XTSL field normally reserved for saving general registers 0 and 1.

Operation: On entry at CEAAB1, the subroutine sets up the necessary parameters and linkage, and transfers control to CEAA5S, the Set Path section of the Pathfinding subroutine. This procedure returns the requested indicators to the Set Path routine, which stores them in the general register save area of the calling task's XTSL in the words normally reserved for saving registers 0 and 1 and exits to the SVC Queue Processor.

Queue Device on Task Subroutine (CEAAC)

This subroutine responds to a task's request for the addition of a system-symbolic-device address to its device list. The calling task specifies the device address.

Entry: CEACQ - in response to the ADDEV macro instruction.

Modules Called: Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) reserves main storage for the TSDL if needed.

Exits:

Normal - SVC Queue Processor.

Error - SVC Queue Processor with error indication stored in XTSL field normally reserved for saving general registers 0 and 1.

Operation: On entry, the processor obtains the address of the task symbolic device list (TSDL) from the TSI, and searches its entries for a device address to match the input address.

If a match of symbolic device addresses is found, the count of queue device requests for that address is checked. If it is less than fifteen, the count is incremented and the routine exits. If the count is equal to fifteen, an error flag is returned to the calling task in general register zero.

If there is no such symbolic device address in the device list, a search is made for a free area. If a free area is found, the symbolic device address is placed into it. If no entry with a free area exists (or there is no device list), a new entry is created by requesting the required number of contiguous bytes of main storage from the Supervisor Core Allocation subroutine which in turn passes back the pointer to the first byte of the allocated space. The device address is placed in the new entry, and the linkage to the old entries, if any exist, is established. All TSDL updates are interlocked using the TSI lock, which was set by the SVC queue processor. The subroutine exits to a secondary entry point (CEAHQR) in the SVC Queue Processor, which resets the TSI lock.

Remove Device From Task Subroutine (CEAAD)

This subroutine responds to a task's request to remove or suppress a symbolic device address in its device list in the task-symbolic device list (TSDL). The symbolic device address is specified by the caller.

Restrictions: The module calling CEAAD to remove devices from the task is required to set the TSI lock before making the call.

Assumptions: If Remove Device from Task is utilized as a subroutine, it is the calling routine's responsibility to save all regis-

ters before calling the subroutine.

Entries:

CEAADR - from SVC Queue Processor via RMDEV macro instruction.

- from a resident supervisor module via a branch and link.

Modules Called: Supervisor Core Release subroutine (CEAL1 entered at CEAL02) called when the device address removed is the last on the list and the area used by that list is to be released.

Dequeue I/O Requests subroutine (CEAAJ) suppresses any I/O outstanding on the device.

Exits:

Normal - entry from SVC Queue Processor - return to same at CEAHND.

- entry from resident supervisor module - address in register 14.

Error - same as normal, except that an error indication is placed in:

- Save area of XTSL for return to SVC Queue Processor.
- Register 0 for return to resident supervisor module.

Operation: On entry, the subroutine obtains the TSDL pointer from the TSI, and searches the list of assigned devices for the calling task for an address that matches the input address.

If there is no match, an error flag is returned to the caller. If there is a match, a test is made to determine the type of calling task. If the task was a privileged virtual storage program, the count of requests for this symbolic device address is checked. If it is not zero, the count is lowered and the processor exits. If the count is zero, a call is made to CEAAJ to suppress the device for that task. Upon return, if I/O is in progress for that task, the TSDL is flagged and the processor exits. Otherwise, the entry for that symbolic device address in the TSDL is cleared, as are the asynchronous entry bits in the asynchronous entry portion of the device group table (DEVI).

If the request was from a supervisor component, another test is made to determine the type of request. If the request was to remove the device, processing is the same as it is for a virtual storage program, except that the count, instead of being lowered, goes to zero. If the re-

quest was to suppress a device, the 'device-suppression' flag in the TSDL is set, and the pointer to the lock byte is saved. (Note: Only a supervisor component can request the suppression of a device for a task by this module.)

If the subroutine finds a TSDL entry in which all in-use flags are off, indicating that there are no device addresses left in the entry, the forward link of the empty entry is placed in the preceding entry, and the Supervisor Core Release subroutine is called to release the storage occupied by the unused block.

When all processing is completed, the subroutine:

- Exits to the SVC Queue Processor at CEAHND if the request was from a privileged virtual storage program.
- Returns control to the calling supervisor component.

Set Asynchronous Entry Subroutine (CEAAK)

This subroutine receives requests from any privileged program. It updates the specified line in the asynchronous-device entry in the device-group table and updates the task's task-symbolic-device list (TSDL). The caller specifies a task identification parameter and a symbolic device address.

Entry: CEAAK1 - in response to the SETAE macro instruction.

Modules Called: Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) allocates work area for the subroutine.

Supervisor Core Release subroutine (CEAL1 entered at CEAL02) releases the subroutine's work area upon completion.

Queue GQE on TSI subroutine (CEAAF) queues errors on the TSI as program interrupts.

Scan on Task ID subroutine (CEAAU) locates the TSI for the given task.

Exits:

Normal - Queue Scanner.

Error - Varies with error conditions:

- If the task device list count has reached its maximum, a return code of 1 is placed in general register zero's save area in the XTSI and control is transferred to the Queue Scanner.

- If the device was not assigned to the old task, or if the TSI does not exist, return codes of 70 and 71 respectively are passed to the Queue GQE on TSI subroutine and a program interrupt is queued on the task's TSI.

- If the device is not found in the system, an exit is made to SYSERR, passing to it a code of 7901.

- When working with TSIs, this subroutine sets and resets the TSI lock to prevent other CPUs from referring to the same TSI.

Operation: On entry, the subroutine obtains a block of main storage from the Supervisor Core Allocation subroutine for a work area and register save area. It then searches the task's TSDL for the requested device. If it is not assigned to the task, and the device is not an RTAM terminal, a program-interrupt code is placed in the GQE, the GQE is queued on the TSI interruption queue by calling the Queue GQE on TSI subroutine, the work area is returned by calling the Supervisor Core Release subroutine, and an exit is made to the Queue Scanner.

If the device is assigned to the task, the appropriate device-group table is located by searching for the correct symbolic-device address in the symbolic-to-actual-conversion table. If a match is not found, a minor system error is reported to the System Error Processor. If a match is found, the actual-device address is multiplied by eight and added to the asynchronous-device-entry pointer in the relevant device-group table. The result is a pointer to the correct asynchronous-device entry.

The subroutine then checks the task identification parameter. If it is zero, two device group table flags are tested, DEVT (indicates TSS under user control) and DEVRT (indicates TSS under RTAM control). When DEVT is on and DEVRT is off, the following fields in the device group table are cleared before removing the device from the task: DEVTSI, DEVC, DEVD, and DEVN. The device is removed from the task's TSDL by calling the Remove Device from Task subroutine. The work area is then returned and an exit is made to the Queue Scanner.

If the task identification parameter is nonzero, the TSI list is scanned using the Scan on Task ID subroutine to find the pointer to the proper TSI. When TSS is under user control and not RTAM (DEVT on, DEVRT off), the new TSI pointer is stored

in the terminal control table (TCT) at TCTTSI. In either case, DEVC, DEVD and DEVN are cleared in the device group table. If a match of ID's is not made, a program interrupt code is placed in the GQE, the GQE is queued on the TSI interruption queue, the work area returned, and an exit is made to the Queue Scanner.

If the correct TSI is found, a pointer to it is placed in the asynchronous-device entry for non-terminal or non-RTAM devices. The device is removed from the old task's TSDL by calling Remove Device from Task subroutine and the device is added to the new task's TSDL. The work area is then returned, and an exit made to the Queue Scanner.

Terminal SVC Processor (CEAR4) Chart BB

The function of this module is to perform the processing for the following resident terminal access method (RTAM) SVCs:

CONN SVC (207)
 DCON SVC (208)
 WAIT SVC (204)
 CKALOC SVC (203)
 LCD SVC (202)
 ATTACH SVC (195)
 UFLOW SVC (187)
 SETTDE SVC (205)

Entries: A unique entry point is defined for each of the eight SVC processors as follows:

CONN - CEAR41
 DCON - CEAR42
 WAIT - CEAR43
 CKALOC - CEAR44
 LCD - CEAR45
 ATTACH - CEAR46
 UFLOW - CEAR47
 SETTDE - CEAR48

Modules Called: Supervisor Core Allocation (CEAL1 entered at CEAL01) is called when the SVC processing routine requires space for a multiterminal status control block or a save area.

The Terminal Control Table Entry Allocation Subprocessor (CEATS) is called in the CONN and DCON processing to allocate and release TCT and buffer pages. Entries to CEATS, for the indicated reasons, are as follows:

CEATS1 - to allocate a TCT page.
 CEATS2 - to allocate a buffer page.
 CEATS3 - to release the first TCT page.
 CEATS4 - to release the first buffer page.

The Supervisor Core Release routine (CEAL1 entered at CEAL02) is called to release the MTSCB and save area storage.

Operation for CONN: When entered at CEAR41 by the SVC Queue Processor, five parameters are received in general registers:

R0, R1 application program name
 R2 TCT virtual memory address
 R3 Buffer virtual memory address
 R4 maximum lines, buffer length (2 half words)
 R5 number of virtual memory pages

On entry, this processor tests a flag in the TSI (TSIMTT) to determine if the task is already MTT. If it is, this processor sets a return code in register zero (all 'F's) and exits to the SVC Queue Processor at CEAHQR.

If the task is not already MTT, Supervisor Core Allocation is called to obtain 64 bytes of main storage for an MTSCB. The block is zeroed, and the parameters passed are retrieved from register storage in the XTSI and posted in the MTSCB. A pointer to the MTSCB is then set in the TSI (TSIMTSCB).

Supervisor Core Allocation is again called for save area space (64 bytes) preparatory to calling the TCTE Allocation Subprocessor at CEATS1 to allocate a TCT page. On return, from CEATS1, a call is made to CEATS2 to allocate a buffer page. The save area storage is then released by calling Supervisor Core Release.

Finally, an indicator is set in the TSI (TSIMTT) to specify that this is an MTT task, and exit is to the SVC Queue Processor at CEAHQR.

Operation for DCON: On entry, a test of the TSI flag, TSIMTT, is made to see if the task is MTT. If it is not, there is nothing for this processor to do, and immediate exit is to the SVC Queue Processor at CEAHQR.

If it is an MTT task, save area space is requested from Supervisor Core Allocation (64 bytes) and general registers are stored in it.

The TCTE Allocation Subprocessor is then called at CEATS3 to release the TCT page for the task. On return, CEATS4 is called to release the buffer page.

The registers are then restored and succeeding calls to Supervisor Core Release return the save area space and releases the MTSCB space.

The MTSCB pointer in the TSI (TSIMTSCB) is then cleared and the flag, TSIMTT, is reset before exit to the SVC Queue Processor at CEAHQR.

Operation for WAIT: On entry, this processor tests register 4 for zero. If it is not zero, it will contain a pointer to a system TCT. The main storage address of the TCT is then obtained. A flag in the TCT (TCTTSS) is then tested to see if the slot is for a TSS user task. If it is not, it will be for an MTT application task, in which case a check is made to see if there is any work pending for the task (TCTWWK on). If not, the MTT task is placed in the delay state and exit is to the SVC Queue Processor at CEAHQR. If work is pending, immediate exit is to be taken to CEAHQR.

If the TCT is for a TSS user task, and work is pending for it, exit is to CEAHQR. If there is no work indicated by TCTWWK, a test is made to see if there are any unmasked interrupts pending for the task. If there are, exit is to CEAHQR.

When there is no work nor any pending interrupts for the task, it is put in the delay state, and the 'force time slice end' flag (GQEFT) is set in the GQE. Move GQE is then called to put the GQE on the Timer Interrupt Processor's queue and exit is to the Queue Scanner.

When the parameter (register 4) contains zeroes on entry to this processor, a test is made to see if the task is MTT. If not, there is no work for the processor to do, and immediate return is made to the SVC Queue Processor at CEAHQR.

When the task is MTT, the address of its first TCT page is obtained, and a scan of the TCT pages is made to see if any work has been posted for the task since the time

that the application task issued the WAIT SVC. If any work is found, exit is made to CEAHQR, and the task will be redispached to process it. When no work can be found, the processor puts the task in the delay state and exits to CEAHQR. Any incoming work from an application user will cause the task to be reactivated.

Operation for CKALOC: On entry, this processor receives either the positive or negative symbolic device address value as a parameter in register 4 (user register 0). If the parameter is negative, it is made positive and the device group table pointer is obtained from the symbolic to actual table (CHASAC). The asynchronous interrupt list pointer (DEVAEP) is then obtained from the device group table and the symbolic device address is located for the subject device.

Then, if the device is MTT oriented, (DEVMT on) a return code of 2 is set in register zero; and if a user oriented device (DEVT on), a return code of zero is set in register zero. If it is a TSS user device, under RTAM, and the SDA on entry was negative, the 'user-oriented device' flag (DEVT) is turned off and a return code of 3 is set. If the SDA was positive, DEVT is turned on and a return code of 1 is set. Exit from this routine is then made to the SVC Queue Processor at CEAHQR.

Operation for LCD: This processor receives the symbolic device address of the subject terminal in general register four as input. It scans the terminal device table (CHATDE) for an entry corresponding to the one specified. If none is found, a return code of zero is set in register zero.

If the entry is found, the line code to be used for the SDA is returned in register zero. These codes are as follows:

- X'01' - 1050 PTTC/8
- X'02' - 2741 correspondence
- X'03' - 2741 PTTC/8
- X'04' - TTY35 ASCII
- X'05' - 1052-7

Exit is then made to the calling routine.

Operation for ATTACH: This processor searches the TCT for the entry assigned to the subject task, and returns the virtual memory address of the TCT slot in register one. A return code of zero is set in register zero.

If the TCT page is not in main storage, exit is made to the System Error Processor.

If the entry sought is not in the TCT page, register one is set to zero, and register zero is set with a return code of four. Exit is then made to the calling routine.

Operation for UFLOW: This processor is entered when a user wants to adjust or obtain the conversational task limit or MTT application user limit. An action code is received in register 1; if it is invalid, a return code of 8 is passed to the caller. A valid action code will cause the following action to be taken:

Code 1 - If the conversational task limit specified in register zero is larger than the value in MTSMAX, a return code of 4 is passed to the caller in register 15; otherwise, the MTSTLM field is updated to reflect the new limit, and a return code of zero is passed to the caller.

Code 2 - The current number and limit of users is placed in register zero, and a return code of zero is placed in register 15.

Code 3 - This code adjusts MTT user limits for different application names. On entry, register zero contains the address of an input buffer location containing application names and user limits. A scan of the active chain is performed to find application names matching those in the buffer. For each match, the proposed limit is checked against the maximum. If the new limit is greater, the buffer is marked in error; if the new limit is less, the MTSCB is updated. When a match for the application name cannot be found in either the active or the inactive list, the entry in the buffer is invalidated. When all FFS are found in the application name field of the buffer, the scan is over and a return code of zero is passed to the calling routine.

Code 4 - This code obtains the current number of MTT users, current user limit, and maximum user limit for each MTT task now active. The virtual memory address in register zero points to an output buffer location that will be updated to contain the above data for each application name. When the application name field contains all FFS, the end of the list is indicated, and a return code of zero is passed in register 15.

Operation for SETTDE: This processor will cause flags to be turned on and off in the terminal device table (CHBTDE). On entry, the TDE is scanned to locate the SDA contained in register 5. Once the entry in the table is found, the input code in

register one is checked; a code of zero will cause the device held flag (TDEST7) to be turned on; a code of four will cause the device held flag to be turned off. In both cases a return code of zero is passed to the caller. When the SDA is invalid, no processing occurs and a return code of four is passed to the caller.

Reset Drum Interlock Subroutine (CEAAZ)

This subroutine responds to a task's SVC request to reset the task/task drum interlock byte in the system table.

Entry: CEAAZ1 - in response to the RDI macro instruction.

Exit: SVC Queue Processor

Operation: Upon entry from the SVC Queue Processor, this subroutine matches the task ID of the calling task against that of the TT interlock. Any one of three conditions may prevail and the PSW condition code in the XTISI is set to indicate them as follows:

<u>Code</u>	<u>Prevailing Condition</u>
0	The drum interlock (TT) is cleared.
1	The drum interlock (TT) is not cleared because the TID of the issuing task does not match that of the TT interlock byte.
2	The drum interlock is not set and consequently cannot be reset.

Inter-Task Communication Subroutine (CEA05)

This subroutine responds to a task's request to communicate with another task. Input from the calling task is a message-control block (MCB) containing:

- The number of doublewords to be transmitted to the receiving task.
- A code field to be used by the communicating tasks.
- The VSEND SVC.
- A halfword of zeros.
- The task identification (TID) of the sending task.
- The TID of the receiving task.
- The location of the event-control block (ECB), if a reply message is expected.
- The text of the message.

RESTRICTIONS: The number of bytes in a message count may not exceed 1904. The entire message-control block (MCB) must be in one page. The virtual storage page must be in main storage. The length parameter must be less than 240.

Entry: CEAQ5V - in response to the VSEND macro instruction.

Modules Called: SVC Queue Processor (CEAHQ) causes a program interrupt GQE to be enqueued on the task's TSI if the SVC was not properly executed.

Queue GQE on TSI (CEAAF) is called to queue VSEND interruption on receiving task.

Exits: Normal - Queue Scanner with one of three return codes in XTISI save area for general registers:

Code 0 - indicates the receiving TSI specified does not exist.

Code 4 - indicates that the TSI specified is not accepting messages.

Code 8 - indicates message successfully transmitted.

Error - If the SVC is not executed, or if the message is either too long or crosses a page boundary, an exit is made to the Queue Scanner after placing an interruption code of 75 or 76 respectively in GQEINT, calling queue GQE on TSI, and setting the return code equal to zero.

Operation: On entry, the subroutine calls the Locate Page subroutine to locate the message-control block page. If the page is not in main storage, an SVC is issued to call CEAIS.

If the page is in main storage, the subroutine determines if the message exceeds 239 double words in length. If it does, the SVC Queue Processor is called to cause a program interrupt GQE to be placed on the task's TSI interrupt queue.

If the message-control block length is acceptable, the processor performs a test to determine whether the MCB resides within one virtual storage page. If not, the task interruption exit procedures described above for excessive message length are entered.

If the MCB resides within one virtual storage page, the subroutine extracts its location from the GQE and determines its main-storage address. Using the TID of the receiving task, the subroutine searches the active and inactive TSI lists. If a TSI with a matching TID is not found, the subroutine indicates this to the calling task

by placing a return code in the XTISI general register save area, and exits to the Queue Scanner.

If a TSI with the correct TID is found, but its message flag is on, indicating that the receiving task does not want to accept messages, the TID of the sending TSI is checked. If this indicates that the sender is neither the system operator nor the batch monitor, a return code of four is stored in the general register save area of the XTISI, the GQE is disposed of, the TSI lock byte is reset, and an exit to the Queue Scanner is made.

If the message bit in the receiving task's TSI is off, or if the sender is the system operator or batch monitor, main storage is obtained and the MCB is moved into it.

A GQE is then created from the input GQE with the IORCB pointer field containing a pointer to the MCB, the GQE is queued on the TSI interruption queue by calling the Queue GQE on TSI subroutine, a return code of 8 is stored in the XTISI, the TSI lock byte is reset and control passes to the Queue Scanner.

TSS Dynamic Status (CEASS)

This SVC processing routine performs either of two functions. When entered in response to SVC 194, it zeroes the system statistical table (CBSSST) fields. When entered in response to SVC 193, it moves system statistics from main storage locations to specified virtual memory locations.

Entries:

CEASS1 - in response to the ZEROSST macro instruction.

CEASS2 - with a GQE address in register one in response to the SAMPLE macro instruction.

Exit: SVC Queue Processor.

Operation: When entered to zero system statistical table fields, it determines the size of the table and loops through a series of instructions storing zeros in the appropriate fields until finished. It then moves the elapsed time from PSAETM into SSTZET and exits.

When entered to sample system statistics, the SVC, to which the routine is responding, resides in the first word of a full page to which the statistics are to be moved. The SVC is invoked by an EXECUTE instruction to guarantee that the receiving

page is in main storage when this processor gets control.

Appropriate information is moved from the auxiliary storage allocation table, the system statistical table, and the system table to the specified page. As the information is moved, the receiving page is checked to determine if it has enough room to hold the statistics to be transferred. If not, this routine exits before completing its processing. Otherwise, it exits when all requested information has been moved.

SUPERVISOR SUBROUTINES

Several subroutines are used by the queue processors and other system components that provide these major components with commonly required services. These subroutines may be logically divided into five subgroups, based on the type of service they perform:

The first of these five groups is the page handling group which provides for the location and manipulation of pages within main storage or between main storage and virtual storage. With the exception of Real Core Error Recording and Real Core Statistical Recording, which are non-reentrant, these subroutines are resident in main storage, reentrant and privileged.

The second group of subroutines deals with errors which occur during paging operations. They provide the means by which errors are classified and retry is attempted. As with the paging subroutines, these subroutines are resident in main storage, reentrant, and privileged.

The I/O services subroutines handle the queuing of I/O requests, the issuance of SIO and HIO commands, error handling and recording and, in general, all phases involved in the completion of an I/O request. They share the common attributes of being resident in main storage, reentrant, non-recursive and privileged.

The special task service subroutines provide services required by changes in a user task and include task initiation, task activation and deactivation and the location of a specified TSI. These subroutines are also resident, reenterable, non-recursive and privileged.

Finally, there exists a group of subroutines which handle inter-CPU communication, hardware device configuration, I/O operation changes, and deletion of work from all or a portion of a CPU prior to partitioning. These subroutines are nonreentrant, nonrecursive, resident and privileged.

PAGE-HANDLING SUBROUTINES

Find Page Subroutine (CEANC)

This subroutine locates segment, auxiliary segment, page and external page table entries.

Entry: CEANCA

Modules Called: Search RSPI Table subroutine (CEAMS) locates the proper resident-shared-page index (RSPI) entry in main storage for any specific shared-page table (SPT) number, or locates the address of the next available entry in the RSPI.

Exits:

Normal - To caller.

Error - To System Error Processor.

Operation: On entry, Find Page expects to find the task's XTSI address and the virtual-storage address of the requested page in general registers. The subroutine then computes the segment-table-entry address. Find Page compares the segment portion of the virtual storage address with the segment table length. If the segment portion of the address is greater than the length of the table, an addressing error has occurred. Find Page specifies the error via general register and returns control to the calling program.

If the specified segment is found in the segment table, the Find Page subroutine tests its availability indicator. If the indicator specifies that the segment is unavailable, Find Page returns an addressing error via general register, and returns control to the caller.

If the segment table entry is available, but the segment is shared, the sharing lock is set and the RSPI-entry corresponding to this shared segment is interrogated. This is done by extracting the shared-page-table number from the auxiliary-segment-table entry, placing it in a general register and calling the Search RSPI Table subroutine. If there is no RSPI entry for this SPT number a system error SVC is issued. If the Search RSPI Table subroutine returns the address of an RSPI entry for this shared-page table number, Find Page tests the RSPI-entry lock byte. If it is on, a delay occurs. If the lock byte is not turned off within a specified time interval, a call is made to CEAIS. It is then locked and the content of the segment table entry in the RSPI is used to compute the page table and external-page table entries for the specified page. If the segment was not shared, Find Page will have previously

placed the segment-table entry from the XTSI in the proper register for processing.

If the page portion of the virtual storage address requested is not greater than the length of the page table, condition code 1 is set and control is returned to the calling program. If the length is exceeded by 1, condition code 0 is set before returning control. Otherwise a condition code of 3 is set, a general register is set to one to indicate the specific error and return is made to the calling program. The calling routine is responsible for resetting the sharing lock if the page was a shared page. The TSI lock is also set by the calling routine.

Locate Page Subroutine (CEAML)

This subroutine provides the location of any page table entry or external-page table entry when the appropriate virtual storage and TSI addresses are given.

Entry: CEAMLP

Modules Called: Search RSPI Table subroutine (CEAMS) locates the proper resident-shared-page index (RSPI) entry in main storage for any specific shared-page table (SPT) number or locates the address of the next available entry in the RSPI.

Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) provides 64 bytes of main storage for use as a register save area when the save area in the Locate Page subroutine is in use.

Supervisor Core Release subroutine (CEAL1 entered at CEAL02) releases the register save area after use.

Exits:

Normal - To caller.

Error - To System Error Processor.

Operation: Upon entry, Locate Page searches for main-storage addresses associated with any virtual storage address in a task's XTSI. The caller specifies virtual storage and TSI addresses. Locate Page tests the segment and page portions of the specified virtual storage address for validity, in order to prevent the modification of unassigned main storage within the task's XTSI.

The TSI address is used to locate the XTSI, which contains control information associated with the task's virtual storage. If the XTSI has been "swapped out" of main storage, a SYSERR will occur. Next the segment portion of the virtual address is assigned to the task. If a violation

occurs, condition code 1 will exist when control is returned to the user.

If the segment is shared (a bit in the auxiliary segment table is assigned for this purpose) the system table sharing lock is set.

The shared-page-table number is extracted from the auxiliary-segment-table-entry and placed in a general register for linkage to the search-RSPI subroutine.

If the segment is unavailable, the segment table and auxiliary segment table addresses are placed in registers 0 and 1 respectively and a condition code of 3 is set. Control is returned to the calling routine.

If the segment is valid, the page portion of the virtual storage address is checked for length validity as above. A violation causes condition code 2 to be placed in the PSW. If both segment and page are correct, the condition code is set to zero.

The segment entry contains the pointer to the page table. Using this as a base, the Locate Page subroutine calculates the location of the page table entry. The location of the external-page table entry for this page is also calculated. The page table (or shared page table) and external-page table (or shared external page table) addresses are placed in general registers for return to the calling program, to which control is given. The sharing lock remains on to protect the validity of the pointers returned. The sharing lock is reset by the calling program.

Page Posting Subroutine (CEAMP) Chart BC

This subroutine posts information to a task's TSI, XTSI and shared-page tables to indicate the status of user pages after a paging operation, and handles delayed posting of write checks on XTSI pages and virtual memory pages.

Assumptions: It is assumed that a skeleton XTSI exists according to the TSI and XTSI formats defined and that the segment and auxiliary-segment table will occupy the first XTSI page.

Entry: CEAMP1

Modules Called: Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) provides 64 bytes of main storage for use in a register save area when the save area of the Page Posting routine is in use. It also obtains main storage for a GQE/PCB.

Supervisor Core Release subroutine (CEAL1 entered at CEAL02) releases the register save area after use, if the work area was assigned by Supervisor Core Allocation subroutine.

Enqueue subroutine (CEAJQ entered at CEAJEN) queues the master pageout pointer or any READ/WRITE built by this routine.

Locate Page subroutine (CEAML) locates the proper page for posting.

User Core Release subroutine (CEAL1 entered at CEAL04) releases the main storage block.

Search RSPI subroutine (CEAMS entered at CEAMS1) is used to obtain the shared page table address.

Queue GQE on TSI (CEAAF entered at CEAAFQ) is called to queue a GQE for a page to be processed by the dynamic loader on the TSI.

Exits:

Normal - To caller.

Error - To System Error Processor.

Operation: On entry, the block page count (TSIBLK) is checked for zero before initiating the scan of page tables. If the count is zero, Page Posting exits. Otherwise, Page Posting checks for two input parameters in general registers: the address of the PCB entry associated with the completed paging operation; and the GQE pointer.

Three types of page posting operations are performed by this subroutine: virtual storage page posting, XTSI page posting, and page table page posting.

VIRTUAL STORAGE PAGE POSTING: When Page Posting finds a virtual storage page indicator in the PCB entry, the read indicator in the GQE is examined to determine whether the virtual storage page has been read into a storage area or written out of storage.

If a read operation has taken place, the TSI is locked via the SETLOCK macro, Page Posting places the TSI address and the virtual storage address in general registers, and calls the Locate Page subroutine. Locate Page returns to Page Posting the page table entry and external-page table entry addresses associated with the VS page. If Locate Page returns an invalid address, Page Posting issues an ERROR SVC.

If the virtual storage address is valid, Page Posting moves the main-storage block address from the PCB entry to the page

table entry, and marks the page available. Page Posting then obtains from the external-page table the required information for setting up the pages' storage keys for each half-page.

A four-bit field in the XPT entry is used to determine the protection class, 2 bits for each half-page. Each 2 bit field value is used as a table look-up to extract the byte content to place in a general register before issuing the set-storage-key instruction for each half Page. Page Posting inserts the proper storage-protect keys at this time. If the page being posted is the ISA and there is a block paging count, processing continues as for a block page read. The XPT entry is then examined to determine if the page just read in is marked by the dynamic loader. If it is, Page Posting obtains 64-bytes of storage from the Supervisor Core Allocation subroutine (CEAL1), copies the contents of the GQE into it, zeroes out the PCB count field in the new GQE, turns off the unprocessed bit, and inserts the VSA found in the PCB in the CSW field of the new GQE. The new GQE address and a program-interruption indicator are placed in general registers, and Page Posting calls the Queue GQE on TSI subroutine to queue the GQE pointer to the task's interruption queue, thus informing the task monitor that the Dynamic Loader routine must be called to process the page. When control returns, Page Posting turns off the unprocessed bit.

Page Posting next examines the shared-page indicator in the XPT entry. If the page is not shared, the page-I/O-count is now decreased by one, and the TSI lock is turned off via the OPENLOCK macro. If the count does not go to zero, no further action is taken, registers are restored and return is made to the calling program. If the count goes to zero, the GQE is examined to determine if this is part of a pageout function. If the master bit in the GQE is on, pageout is in progress. Under this condition the 'page-wait' bit remains on in the TSI, the GQE master-pageout pointer in the TSI is placed in a general register and the Enqueue GQE subroutine is called to queue the GQE on the SVC Queue Processor's queue. Page Posting then restores all registers and returns to the calling program. If a pageout function was not in process the page-wait bit in the TSI is turned off, the ready bit turned on, registers restored, and return made to the calling program.

If the page is shared, the count of estimated main storage blocks available (SYSECB) is lowered, after setting the system table lock (SYSTSKLK), which protects the SYSECB count. The lock is reset after the count is returned, and the 'in-transit

in' bit in the shared-XPT entry is turned off. While this page was being read in, there may have been other requests for this page from those sharing it. The 'GQE chain indicator' bit in the shared-XPT entry reflects this condition. If the bit is off no chain exists and Page Posting goes to decrement the page-I/O count, reset the sharing lock (SYSSHALK) which was set by Locate Page and returns to the calling program as for private pages. If a GQE chain does exist, Page Posting frees each task's GQE found in the chain in the following manner:

- Sets the TSI lock.
- Decrements the task's page-I/O count by one, tests the new count and performs as described above.
- Resets the TSI lock.
- Frees the GQE on the chain by calling the Supervisor Core Release (CEAL1 entered at CEAL02) subroutine.

Page Posting processes other chained GQEs in the XSPT entry in a similar manner as described above. When the last GQE has been processed, the GQE chain indicator in the XSPT entry is turned off. Page Posting then proceeds as described previously for the page that the calling program provided as input.

When shared chains are examined, Page Posting locks the sharing lock (SYSSHALK) to ensure that no other CPU may perform the same function of working off or adding to shared GQE chains at the same time.

When Page Posting is called to perform a posting operation for a VS page that has been written out of storage, a test is made to determine whether the page is shared. If not, Page Posting is only required to test the TSI lock byte, and when off, to set it on, and to lower the page-I/O count field.

If the count did not go to zero, the lock byte is turned off, all registers restored, and return is made to the calling program. If the count goes to zero, the 'page-wait' bit in the TSI is turned off and the 'time-slice end' indicator is turned on in the TSI. If this was a pageout operation, the TSI lock byte is turned off, all registers restored and return is made to the calling program.

Page Posting recognizes a write for a shared page by testing a bit indicator in the GQE. Using the VMA in the PCB, and the TSI address in the GQE as input parameters, Locate Page is called. The number of pages written for this shared page is then decre-

mented by one. If the GQE-chain indicator is off, there have been no further requests for this page. If the write count is now zero, and this is a pageout operation the storage block containing this page is not released, and Page Posting goes to decrease the page-I/O count as discussed previously. If the operation was not pageout and the write count is zero, the storage block is released, since a supervisor component (the Write Shared Page subroutine) is performing the write of a shared page. Page Posting in this case restores all registers and returns to the calling program. If the write count did not go to zero, the page-I/O count is decreased as described earlier.

If a GQE chain exists for this page, the Enqueue GQE subroutine is called to start the write request processing.

The GQE on the chain will have previously been initialized correctly for this procedure. If a read request is found, the TSI associated with the chained GQE is handled in the same manner as virtual memory incoming pages. After the GQE chain has been worked off, the shared-page table entry is marked available.

XTSI PAGE POSTING: When the PCB entry contains an XTSI-page indicator, Page Posting checks if it is for a read or a write operation. If a write operation is being processed, the TSI lock is set and the page is checked to see if it is the first XTSI page. If it is, the external address of the XTSI page is updated in the TSI. If not, it is checked to see if it is a page table page, a segment table page, or an auxiliary segment table page. If it is none of these, a system error is detected. If yes, the I/O pending count is checked. If 0, a system error is detected. Otherwise the count is lowered by one and checked again for 0. If not 0, control is returned to the calling program.

If it is 0, the task is in page wait. The task is then made ready and the TSI lock byte turned off. If there is a pageout GQE on the TSI, it is queued, the master pageout pointer cleared, and the task left in page wait.

If the posting operation follows a read operation, the TSI lock is set and Page Posting must also determine if the first XTSI page is involved. A check then is made to see if the task has been selected for migration. If it has, a GQE is built and queued on the timer-interrupt processor's queue to trigger migration. If it is the first XTSI page but doesn't involve migration, the following operations are performed:

- If the segment table and auxiliary segment table are both in the page, the internal location of the first XTSI page is inserted in the TSI. If the TSI pointer in the XTSI has not been inserted, it is inserted. Pointers to the segment table (control register zero) and to the auxiliary segment table (AST) are updated. The number of pages this time slice is set to one. The delayed posting queue is then scanned for page table page entries. If any are found, the GQE on the delayed posting queue is changed to indicate a read; and the GQE/PCB is queued on a device queue to read in a page.

The page count is increased, the entry is dequeued, and the 'processed by time-slice end' bit in the AST turned off for that page. This bit indicates that the AST entry corresponds to the first segment in the PTP. At this point, the 'XTSI swapped out' bit is turned off.

After the delayed posting queue is scanned, the segment table is scanned for an ST entry with the availability flag off. The location of the page table is then checked. If it is in the first XTSI page, the PTP block address is inserted into the ST entry. If the page table is in other than the first XTSI page, and the 'time-slice end' bit is on, a check is made to determine if the entire XTSI is to be brought into main storage. If not, only those page table pages required to process the pre-page set are readied for reading into main storage. If all page table pages are to be read into main storage a read is set up for the page and the I/O count raised. The read is then queued. If the 'time-slice end' bit is off, a read has already been built for that PTP.

At this point the delayed posting queue is scanned for any virtual memory entries. If any are found, the entry is dequeued, a locate page is issued, and the external location in the XPT is updated.

After the VM entry scan, or if the I/O count was 1, GQE/PCBs are created to read in the ISA page and all blocked pages (those with XPTPP on). Then the I/O count is lowered.

- If the segment table is in the first XTSI page, but the AST is on another page or pages, the delayed posting queue is scanned for entries of AST pages. If found, the external location

of the page in the XTSI is updated and the entry dequeued.

When the delayed posting queue has been scanned, reads are set up for the AST pages, the I/O count is raised, and the reads are queued.

At this point, the internal location of the first XTSI page, the TSI pointer, and the PTP chain are updated. Control register 0 is updated with a new block address, the number of pages is set to 1, the number of ST and AST pages read in is set to 0, and the I/O count is lowered.

- If neither the ST or the AST is in the first XTSI page, the delayed posting queue is scanned for entries of AST pages or ST pages. If any are found, the external location of the page is updated in the XTSI, and the entry is dequeued.

Contiguous main storage requirements for the ST are then checked. If there is only one ST page, a GQE is built, the page I/O count raised, and the GQE is queued on the User Core Allocation queue entry in the scan table. The AST pages are then read in.

If contiguous main storage for the ST is required, a GQE is built specifying the number of contiguous main storage pages needed. It is queued on the Contiguous Core Allocation queue entry in the scan table after the page I/O count is increased. The internal location of the first XTSI page, the TSI pointer, the PTP chain, the number of ST and AST pages read in, and the number of pages this time slice end are initialized. The I/O count is then lowered.

If the GQE is for an ST page, the number of pages this time slice end and the number of ST pages read in are raised. If there is only one ST page, control register 0 is updated. If all AST pages are in main storage, the 'XTSI swapped out' bit is turned off and the delayed posting queue is scanned for any PTP entries. If any are found, a GQE is set up to read in the page, the 'processed by time-slice end' bit is turned off, and the page I/O count is raised. The ST and AST are then scanned to read in any other page table pages. The I/O count is lowered. If there is more than one ST page, a test is made to see if this is the first. If it is, control register 0 is updated. If all ST pages are not in main storage, the I/O count is lowered. If all ST pages have been read in, reads are set up and the I/O count raised for each AST page. The I/O count is then lowered.

If the GQE is for an AST page, the number of pages in this time slice end, and the number of AST pages read in are increased. The internal address of the AST page is updated in the XTSI. If all the AST pages have not been read in, the I/O count is lowered. If all AST pages have been read in and all ST pages are in main storage, the 'XTSI swapped out' bit is turned off and the delayed posting queue is scanned for PTP entries. If any are found, they are read in, the I/O count raised, and the 'processed by time-slice end' bit is turned off. The PTPs are then read in. If all AST pages have been read in, but not all ST pages are in core, the I/O count is lowered.

If the GQE is for a PTP, the PTP chain in the XTSI is updated, and the number of pages this time slice end is raised. If the suppress posting bit in the page control block entry is on, a write check has occurred on the page. The PTP is then scanned, and the internal and external addresses for every segment in that PTP are updated (internal is in the ST; external in the AST). If the 'suppress posting' bit is not on, only the internal address in the ST is updated. If at this time there are no outstanding page table page reads, the VM delayed posting and the read of the ISA page are set up. The I/O count is then lowered.

Write Shared Pages Subroutine (CEAMW)

This subroutine returns storage blocks occupied by shared pages to the supervisor, provided the pages have not been referred to within a specified time period. If storage is released for any changed pages, the pages are written out of storage before their storage space is released.

Entry: CEAMWS

Modules Called: Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) provides main storage for work areas required.

Supervisor Core Release subroutine (CEAL1 entered at CEAL02) releases the work areas when no longer needed.

Enqueue GQE subroutine (CEAJQ entered at CEAJEN) queues requests to assign auxiliary storage areas on the scan table queue.

User Core Release subroutine (CEAL1 entered at CEAL04) releases main storage occupied by user pages which are to be written out.

Auxiliary Storage Release subroutine (CEAIA) releases auxiliary storage previously occupied by user pages which are to be written out.

Exit: If called to perform the shared page migration function, exit is to the Queue Scanner after queuing a GQE on the Program Interrupt Processor's (CEANB) queue. Otherwise, exit is to the calling routine.

Operation: On entry, Write Shared Pages tests a lock byte within its own coding. If the lock byte is on, control is returned to the calling program immediately, since another CPU is performing the write shared pages function for another program. If the lock byte is off, it is turned on, the calling program's registers are saved, and Write Shared Pages begins processing the request.

The sharing lock is set to protect the SPT information. If the RSPI count is zero in the system table, the lock byte is turned off, and return is to the calling program with all input registers restored, and the scan flag in the system table turned on.

If there are RSPI entries, the Inter-CPU Communication subroutine is called to notify each CPU in the system to reset its associative registers. This is necessary to guarantee that no other CPU's are presently using a shared page whose reference bit may have been turned off earlier by Write Shared Pages. When this has been accomplished and control returns the subroutine routine proceeds by checking to see if the request was from Supervisor Core Allocation.

If this is the case, no main storage is available for SCA's reserve list and requests for main storage cannot be met until some pages become available. In this situation, Write Shared Pages sets a switch to indicate that it has been called by SCA; and, therefore, it cannot call SCA for GQE/PCB space. To meet this special request, a dummy PCB is passed to User Core Release requesting shared pages which are both unreferenced and unchanged. The scan of the tables is the same as in normal processing.

Next, a check is made to see if the call to Write Shared Pages is from the Timer Interrupt Processor to obtain pages for shared page migration. If so, the internal flag (SCASW-same as switch used for calls from SCA) is turned on. The scan flag is then tested to see if the call was just to reset the reference bits, in which case they are reset by obtaining the main storage address from the chain of shared page core block table entries in the core block header. Write Shared Pages then returns to the calling routine.

Otherwise, Write Shared Pages calls Supervisor Core Allocation to get 128 bytes

of main storage for a GQE/PCB combination. The main storage obtained is zeroed and the PCB address is placed in the GQE. Then, if it is a migration case, three flags are turned on in the GQE: GQEPIM, GQEPPM, and GQESMGM. The scan of RSPI entries is then started after restoring the registers which point to where the search ended, the last time Write Shared Pages was called. If this is the initial call, the registers are set with one pointing to the RSPI entry and the other with the RSPI entry count.

If the RSPI entry contains a shared-page-table number other than zero, and if the RSPI entry lock byte is off, it is turned on and the shared-page-table scan is performed. If the SPT number is zero, or if the lock byte is on, the next RSPI entry, if any, is examined as above. After all entries have been examined, Write Shared Pages turns off the sharing lock byte and its own lock byte, and calls the Supervisor Core Release subroutine to release the last GQE/PCB obtained which was not used. When processing is complete, Write Shared Pages restores registers and returns control to the calling program.

If an active RSPI entry was found, a scan of the shared-page table (SPT) is started. If an SPT entry is found unavailable, the next SPT entry, if any, is examined for availability until all entries have been examined. If the PCB count in the GQE is zero, when all SPT entries have been examined for a particular RSPI entry, and the RSPI entry lock byte is turned off, Write Shared Pages returns to the RSPI table scan. If any PCBs were built up, the GQE is initialized as follows before the GQE is moved to perform the write operation.

- The loc-on-Q fields are filled with the Auxiliary Storage Allocation Queue Processor's scan table queue locations, followed by four Fs to ensure disposal of the GQE after processing is completed.
- The PCB count is placed in the GQE count field of the loc-on-Q field in the GQE. This is required so that the RSPI entry lock byte can be turned off at the completion of all writes associated with this GQE. The count field is lowered by the Page Posting subroutine.
- The RSPI entry address associated with these writes is stored in the shared page table field of the GQE so that the Page Posting subroutine can determine which RSPI entry lock byte to turn off when the write count in the GQE's loc-on-Q field goes to zero.

- The write bit in the GQE is turned on.
- The shared page bit in the shared-page table is turned on.

The count of the number of writes pending in the system is then incremented by the number of PCB entries. This count is located in the system table. A lock-byte is employed when the count is being raised. At this point the Enqueue GQE subroutine is called to queue the GQE on the scan table queue of the Auxiliary Storage Allocation Queue Processor. When control returns to the Write Shared Pages subroutine, the Supervisor Core Allocation subroutine is called to allocate another GQE/PCB storage block. When the space is allocated, the Write Shared Pages subroutine begins scanning the RSPI entries again.

When an SPT entry is found available (meaning a shared page is in main storage), Write Shared Pages marks the SPT entry unavailable by setting on a bit in the page table. This ensures that if any other CPU refers to this page a page-relocation interruption will be queued on the scan table on the Program Interrupt Queue Processor's queue. The Write Shared Pages subroutine, however, may reset the SPT entry to available if the page is found referenced, as discussed below. The Program Interrupt Queue Processor will recognize this fact when dealing with page-relocation interrupts for shared pages.

Having found the page available and marked it unavailable, the subroutine uses the ISK instruction for each half-page to gain access to the reference, change, and hardware indicators. If any reference indicator is found on, it is turned off. The SPT entry is then marked available, the shared-page lock byte in the system table is turned off, and the subroutine continues scanning the SPT for this RSPI entry.

If the page is unchanged, the storage block associated with this page is returned to the system by calling the User Core Release subroutine.

If the page was found changed, the following steps are taken:

- If the page previously resided in auxiliary storage, the old auxiliary location is released.
- The write count in the XSPT entry is raised.
- If a PCB entry is available, the PCB count field in the GQE is increased.

- Space is obtained for a PCB entry by calling the Supervisor Core Allocation subroutine.
- The PCB count in the GQE is raised.
- The SXPT-entry address is placed in PCB.
- The storage block address is placed in PCB.
- PCB flags are turned on to indicate I/O and drum-storage respectively.

When the number of shared pages to be written out of main storage plus those released from main storage reaches a value indicated in the system table, no more purging of main storage takes place. The write GQE, if one has been set up, is queued and return is to the calling routine as above.

When Write Shared Pages is called by Supervisor Core Allocation, a switch is set to prevent recursive calls to CEAL01 and CEAL02 and to inhibit the setting up of writes. In this case, only unreferenced, unchanged pages are released. Otherwise processing proceeds as described above.

External Page Location Address Translator Subroutine (CEAAE)

This subroutine translates a two-byte page-number field of a four-byte symbolic address into the physical I/O device address required by a seek or search channel command. The symbolic address, the device type, and a work area pointer are supplied by the caller in general registers.

Entry: CEAAE1

Exits:
Normal - To caller.

Error - To System Error Processor.

Operation: The subroutine sets up the device type as a search argument for a device routine to use in searching a device-type table in the format shown in Figure 18.

If a particular device type cannot be found in the device-type table, a major-system-error call is issued to SYSERR. Each device routine checks to determine if a symbolic address is in the range of the disk device with the following ranges applied:

- 2301 : 0 -- 899 1
- 2311 : 0 -- 1623 1
- 2314 : 0 -- 6496 1

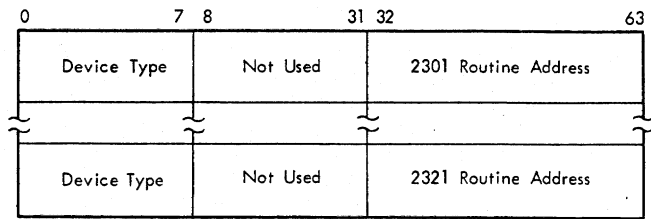


Figure 18. Device type table format

If a symbolic address is not in an acceptable range, a major, system-error call is issued to SYSERR. Each device routine divides the symbolic-page address by the number of pages that can be placed on a cylinder, or in the case of the 2301, the number of pages that can be placed on a track. The quotient becomes the cylinder or track address. The remainder of the division is used as a search argument to locate a track and record ID or a record ID in the case of the 2301. A separate track-and-record-ID table exists for each type of device. The format of the table is shown in Figure 19. If a remainder cannot be found a major-system-error call is issued to SYSERR.

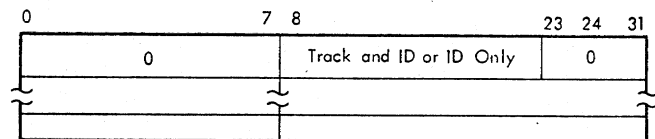


Figure 19. Track and record ID format

Translated symbolic addresses are returned to the calling program via general registers, in the format shown in Figure 20.

When processing is completed, the subroutine restores the general registers, and exits to the calling program.

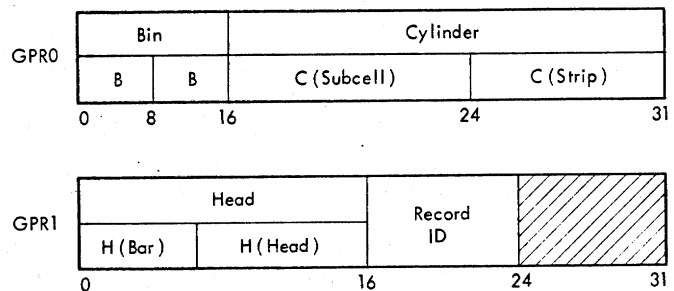


Figure 20. Format of translated symbolic addresses

Search-RSPI-Table Subroutine (CEAMS)

This subroutine locates a resident-shared-page-index (RSPI) entry in main storage for any specified shared-page-table (SPT) number, or locates the address of the next available entry in the RSPI.

Entry: CEAMS1

Exits:

Normal - To caller.

Error - To System Error Processor.

Operation: On entry, the subroutine saves the calling routine's registers, and compares the specified SPT number with the RSPI-entry count in the system table. If the count is zero and the input SPT number is not zero, a system-error-SVC is issued to CEAIS. The subroutine informs the calling program that the specified address was that of a chain rather than an RSPI entry.

If the RSPI-entry is nonzero, a search is made through the chained RSPI blocks, examining each successive entry and stepping from one block to another as necessary. If a chain address of zero is discovered before the count goes to zero, a system-error-SVC is issued to call CEAIS.

If a matching SPT number is found in some RSPI entry, the entry address is placed in a general register, the calling program's general registers are restored, and the subroutine returns control to the calling program.

If, after searching all RSPI entries, no match is found a zero is returned to the calling program.

The significance of SPT number zero is to enable users to locate the first currently unused RSPI entry if one exists, since all RSPI table entries are set to zero when unused. In the case where no available entry exists for SPT number zero, the correct chain address location is returned as discussed above. The calling program is then responsible for acquiring storage space from the Supervisor Core Allocation subroutine, storing the address in the chain field and then constructing an RSPI entry.

When an RSPI entry is being created, tested, or deleted, the user is responsible for setting the sharing lock (SYSSHALK) before calling this subroutine and resetting it upon return.

Segment Block Remover Subroutine (CEANG) Chart BD

This subroutine removes unused blocks from the end of the segment table.

Entry: CEANG1

Register 2 - Address of the TSI

Register 3 - Address of the XTISI

Register 7 - Base address of the calling routine

Register 14 - Return Address

Modules Called: Auxiliary Storage Release (CEAIA entered at CEAIAR) releases auxiliary storage when ST and AST pages are no longer needed.

User Core Release (CEAL1 entered at CEAL04) releases storage for ST and AST pages when no longer needed.

Exits: To caller - with registers 2, 3, 7, and 14 unchanged. All others are changed.

Operation: Segment table blocks are scanned from the end of the ST seeking an assigned ST entry. If an assigned ST entry is found in the last segment block at the end of the ST, the Segment Block Remover returns to the caller because there is no work.

When an unused ST block is found, the count of ST blocks is adjusted. The corresponding AST block is checked to see if it is the only one on the AST page. If it is, User Core Release is called to release the AST page; and the AST page count is adjusted. Any auxiliary storage assigned to the AST page is also released by a call to auxiliary-storage-release. Then the ST block is checked to see if it is the only one on the ST page. If it is, User Core Release is called to release the ST page; and the ST page count is adjusted. Any auxiliary storage assigned to the ST page is also released by a call to Auxiliary Storage Release.

This procedure continues until an ST block containing an assigned entry is found. The locations of the ST and AST are then checked. If these tables are not in the first XTISI page, this routine returns to the caller. If they are, the AST is moved to the end of the ST, and page tables in the first XTISI page are moved to the end of the AST. ST pointers to page tables in the first XTISI are adjusted, and the bytes-available count for the first XTISI page is updated before returning to the caller.

XTSI Page Packing Subroutine (CEAMY)

This subroutine optimizes space usage in page table pages.

Entry: CEAMY

R1 - Address of first XTSI page.

Modules Called: Auxiliary Storage Release (CEAIA entered at CEAIAR) releases auxiliary storage when page table pages are no longer needed.

User Core Release (CEALI entered at CEALID) releases storage for a page table page when no longer needed.

Exit: To caller.

Operation: When called, this subroutine scans the first PTP seeking holes in it caused by page table deletion. Any holes found are eliminated by pushing the PT/XPT entries to the top of the page. ST entries, page table headers, and PTP headers are adjusted to reflect resulting changes.

When there are no holes in the page (either because none existed, or because they have been eliminated by the 'pushing' procedure), the space available at the bottom of the PTP is calculated. PT/XPT entries in the other PTPs are then examined to find one that will fit in the available space. When one is found, it is moved, and relevant pointers and headers are updated accordingly.

If space is still available, the scanning/moving process continues until no more PT/XPT can be found that will fit. If this activity empties an XTSI page, it is deleted from the XTSI chain by adjusting appropriate forward and backward links in the page table headers. Then User Core Release and Auxiliary Storage Release are called to release the unused space.

After the first PTP is packed, the process continues on any other PTPs until all have been consolidated before returning to the caller.

Real Core Statistical Data Recording Subroutine (CEAI6) Chart BE

This subroutine accumulates error statistics on the I/O outboard failures of the direct access paging drum or disk devices. This subroutine is different from other page-handling subroutines in that it is nonreentrant.

RESTRICTIONS: Entry is made from the Page I/O Recovery program or SVC Queue Processor (SVC 223).

Entries: CEAI61, CEAI62

Modules Called: Real Core Error Recording subroutine (CEAI7) formats an error record that is output on drum by the preservation recording portion of SERR.

Exit: To caller.

Operation: Upon entry to CEAI61, the subroutine performs the following:

- Identifies the failing paging-I/O device and locates its table entry.
- For each sense bit which is set to 1 in the summary-sense-data, raises the associated statistical data recording (SDR) bucket by 1 in the table entry, setting the overflow marker if the SDR bucket overflows in the process.
- Updates the 'last path', 'total error count', and 'error time stamp' fields of the table.
- Checks to see if the call is typed as a solid outboard failure. If so, the Real Core Error Recording subroutine is called.
- Checks the overflow marker to see if an SDR bucket overflowed. If not, control is returned to the Paging Failure Recovery subroutine.
- Generates the call parameters and calls the Real Core Error Recording subroutine to format and output the SDR data on the drum device.
- Returns control to the caller after resetting the table entry and interlocks.

Upon entry to CEAI62, the subroutine either turns on (if SETIR) or turns off (if RESETIR) the PSDIR field in CHAPSD (Direct Access Paging Statistical Data Record).

Real Core Error Recording Subroutine (CEAI7) Chart BF

This subroutine formats an error record that is output on drum by the preservation recording portion of SERR to preserve records of I/O errors occurring on direct access paging devices in the system. This subroutine is different from other page-handling subroutines in that it is non-reentrant.

Entries:

CEAI61 - on a solid outboard failure.

CEAI62 - on an inboard failure.

Modules Called: Inter-CPU Communication subroutine (CEAIC) transmits commands from one CPU to another in the TSS domain and emits external signals over the extended direct control in order to have the receiving CPU perform the functions required for the coordination of CPUs in a multiprocessing system.

SERR Bootstrap (CMASA1) provides an interface between the Real Core Error Recording subroutine and the preservation recording portion of SERR.

Exit: To caller.

Operation: This subroutine performs the following functions:

- On entry from the Real Core Statistical Data Recording subroutine it prepares the outboard error record in the record-buffer area.
- On entry from the Paging Failure Recovery subroutine it prepares the inboard error record in the record-buffer area.
- Halts other CPUs, if any, in the TSS domain, via the Inter-CPU Communication subroutine.
- Generates the necessary call parameters and transfers control to SERR Bootstrap which pages in the SERR Preservation Recording program to write the contents of the record buffer on the drum device.
- Upon return from the recovery nucleus, signals other CPUs, if any, in the TSS domain to resume operation, via the Inter-CPU Communication subroutine and returns control to the caller.

PAGING ERROR RECOVERY ROUTINES

Paging Failure Recovery Subroutine (CEAAQ) Chart BG

This subroutine performs three functions:

- Takes remedial action when a path analysis indicates that no actual I/O path is available to service a paging request.
- Takes remedial action when a paging failure occurs because of a dynamically defective medium.
- Takes remedial action when a paging failure is caused by a dynamically defective device.

Entry: CEAAQ1

Modules Called: Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) allocates the necessary work/save area(s).

Supervisor Core Release subroutine (CEAL1 entered at CEAL02) releases work/save areas after use.

Queue GQE on TSI subroutine (CEAAF) queues the specified GQE on the affected task's TSI.

Task Communication Control subroutine (CEAN) constructs a message control block (MCB) for a calling task's message and attaches it to the called task's TSI.

Enqueue GQE subroutine (CEAJQ entered at CEAJEN) places a pointer to a GQE on the specified scan table queue.

Dequeue GQE subroutine (CEAJQ entered at CEAJDE) dequeues the GQE.

Move GQE subroutine (CEAJQ entered at CEAJMG) moves a previously dequeued GQE to the next specified queue or releases its storage if no work remains.

Suppress Auxiliary Allocation subroutine (CEAAP) suppresses external-page-allocation of the specified auxiliary device.

Rescheduling subroutine (CEAKZ) moves task from the dispatchable list to the eligible or inactive list and moves tasks from the inactive list to the eligible list.

Exits:

Normal - To caller.

Error - To System Error Processor.

Operation: On entry, the subroutine performs the following:

- Calls the Supervisor Core Allocation subroutine to allocate the necessary work/save area(s).
- Saves input general registers.
- Tests an input register flag to determine the type of table whose address is specified by the caller. If it is:
 - 1) The SYSARG entry corresponding to the failing channel program is located by the pointer in paging error control block (PECB), whose pointer is in the system table. The PCB entry associated with the failing channel program is then accessed

from the pointer in the SYSARG entry.

- 2) Direct access interface block (DAIB), the device-GQE is accessed from the DAIGOF field of the input DAIB and is dequeued by calling the dequeue-GQE subroutine. The PCB entry associated with the failing channel program is then accessed from the pointer in the PECB, whose pointer is in the input DAIB.
- 3) General queue entry (GQE), the PECB is accessed from the GQESAT field of the input GQE and the PCB entry is accessed by the pointer in GQEPCB.
- 4) If the GQE/PCB are contained within the SYSARG field of the DAICB, the failing CCW is computed.
- 5) When this subroutine accesses the TSI or auxiliary storage allocation table, it sets and resets the appropriate lock bytes.

At this point, the read/write flag in the PCB entry is tested and if it is off, other PCB entry flags are tested to determine the type of read operation and the appropriate action to take. These flags specify the following operations:

- Relocation read
- Dispatcher read
- Page Posting read
- TWAIT read
- Supervisor paging read

If the PCB's 'relocation read operation' flag is on, the caller's request indicator is tested to determine if the defective-medium-recovery function is requested. If so, the 'relocation page in on defective volume' program interruption code X'93' is stored.

If the function was not requested, the 'demountable volume' flag in the direct access paging statistical data recording table is tested. If the flag is on, the failing device is demountable, in which case the 'relocation page-in on moveable volume' program interruption code X'92' is stored. If the flag is off, the 'relocation page-in on permanent volume' program interruption code X'91' is stored.

At this point, or if a failure occurred during another READ operation or after a SYSERR, the subroutine calls the Supervisor Core Allocation subroutine to allocate storage for a task program interruption

GQE. When storage is returned, the Paging Failure Recovery subroutine places a program interruption code and the TSI address in the new GQE area, and calls the Queue GQE on TSI subroutine to place the GQE pointer on the TSI's program interruption queue.

The paging count in the TSI is decremented by the paging count in the GQE. If the result is zero, the task is taken out of page wait status and placed in ready status. The subroutine then restores the general registers of the calling program, releases the save areas via the Supervisor Core Release subroutine, and returns control to the caller.

If the PCB specifies a dispatcher-read or page-posting-read operation, the task ID from the TSI pointed to by the device GQE is examined. If it is that of an operator task, the Purge subroutine is called to purge any outstanding I/O request. The Reinitialize Operator Task subroutine is called to create a new operator task. The Task Communication Control subroutine is then called to notify the operator of his new status, after which a program interruption GQE (code X'97') is generated and placed on the task's TSI interruption queue, as described previously.

If the task ID is not that of an operator, a time-slice-end is forced on the task by generating a time-slice-end GQE and calling Enqueue GQE to place it on the Timer Interrupt Queue Processor's scan table queue.

When this is accomplished, the Task Communication Control subroutine is called to notify the operator of the task's condition, the operation failed-return code is set, and control returns to the caller.

If the PCB TWAIT-read operation flag is on, the operation-failed return code is saved, and the TSI paging count is adjusted as above.

If the PCB supervisor paging read operation flag is on, the caller's request indicator is tested to determine if the defective-medium-recovery-function is requested. If so, a program interruption of X'96', specifying a 'supervisor paging page-in on defective device' error is saved. If not, and if the device is demountable, a program interruption code of X'95', specifying a 'supervisor paging page-in on moveable volume' error, is saved. If the device is not demountable, a program interruption code of X'94', specifying a 'supervisor paging page-in on permanent volume' error, is saved.

The subroutine calls the Queue GQE on TSI subroutine to pass the interruption on to the task, and processing continues as described previously.

If none of the above six PCB flags is on, a minor software SYSERR is reported. A temporary program interrupt code of X'92', is set in the temporary code area, and the Queue GQE on TSI subroutine is called as described previously.

If the PCB write flag is on Paging Failure Recovery checks the PCB's 'VAM/SYSTEM paging' flag. If it indicates VAM paging, the caller's input request indicator is checked to determine whether the defective-medium-recovery function is requested. If so, an error code of X'B' is returned to the task. If not, the 'dismountable volume' flag is tested. If the device is dismountable, an error code of X'9', indicating a 'pagout on movable volume' error, is returned. If the device is not dismountable, an error code of X'A', indicating a 'pagout on permanent volume' error, is returned to the task.

If the 'VAM/SYSTEM paging' flag indicates system paging, the 'defective medium recovery requested' indicator is checked. If it is on, the 'auxiliary allocation bad page' counter is incremented by one. If it is full, or if the above indicator is off, the Suppress Auxiliary Allocation subroutine is called for this paging device and the Task Communication Control subroutine is called to send a message to the operator. The PCB chain is scanned for PCBES with the 'page I/O complete' flag set. When such a PCBES is found, its 'bypass' flag is set. In all other cases the 'suppress posting' flag is set.

The Enqueue GQE subroutine is then called to queue the original device GQE pointer on the Auxiliary Storage Allocation Queue Processor's queue.

The subroutine restores all registers and returns control to the calling program.

Paging I/O Error Recovery Routine (CEAAM)

This routine constructs the paging error control block; it then initiates retry procedures on the first occurrence of a paging I/O error and subsequent occurrences of the same error.

Entry: This routine is entered at CEAAM1 by any one of five processors:

- The Page Drum Queue Processor when a desired paging device is unavailable or when a start I/O operation is unsuccessful.

- The Device Queue Processor when a desired paging device is unavailable.
- The Page Direct Access Queue subroutine when a start I/O operation is unsuccessful.
- The Page Drum Interrupt or the Page Direct Access Interrupt Processor when the CSW indicates that the operation failed.
- The Page Direct Access Interrupt subroutine for the same reason as above.

Exit: To caller.

Modules Called: Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) provides 192 bytes of main storage for the construction of a PECB.

Supervisor Core Release subroutine (CEAL1 entered at CEAL02) releases various control block areas depending on the degree of retry success and the type of error.

Queue GQE on TSI (CEAAF entered at CEAAFQ) queues an interruption GQE on the interruption queue in the TSI.

Alternate Path Retry subroutine (CEAAS) attempts to start an I/O operation on an alternate path when an I/O error has been attributed to a faulty component along the original path.

Same Path Retry subroutine (CEAAV) attempts to start an I/O operation along the original path when no faulty component is detected.

Start Retry Operation subroutine (CEAAX) initiates a write operation after a read operation has been successfully retried.

Paging Failure Recovery subroutine (CEAAQ) performs defective medium recovery when an I/O operation has been successfully retried on a standard area.

Move GQE subroutine (CEAJQ entered at CEAJMG) moves the original device GQE after a retry operation has failed.

Real Core Statistical Data Recording subroutine (CEAI6) records error data after a successful retry.

SYSERR (CEAIS) performs error recovery functions after the occurrence of a minor software error.

Operation: For an I/O error, or SIO failure, the address of the failing channel program is computed from the CSW and is placed in PECCPS. The number of CCWS is computed and is placed in PECNCW. The

relative number of the failing CCW is computed and is placed in PECRCW, and the pointer to the PCBE/DIBE for the CCW specified in PECCPS is found and is placed in PECEPT.

If the I/O error was a channel or interface control check, the command address portion of the CSW is unpredictable. The last SIO address is therefore put into PECCPS, number of CCWs is computed and is placed in PECNCW, zero is placed in PECRCW, and the pointer to the PCBE/DIBE for the CCW specified in PECCPS is placed in PECEPT.

For unavailable original path errors, a pointer to the first PCBE is put in PECEPT and PECCPS, PECNCW are cleared.

If the seek CCW is not found after searching through the CCWs, a major software system error (code 6602) is reported.

The next step in processing is to find the displacement to the appropriate device entry in the direct access paging statistical data recording table (CHAPSD). The table is referred to, its header information is saved, and the first table entry is accessed. After determining the type of device (drum or direct access) on which the error occurred, the device GQE and subsequently, the symbolic device address are retrieved. This address is then matched against the symbolic device addresses in the first and succeeding entries in the PSD until a match is found or until all entries have been tested. When a matching entry is found, its displacement from the beginning of the table is computed and entered in the PECB. If no match is found, a minor SYSERR (code 6603) is invoked and processing continues as in the case of no matching seek CCW described above.

The final step in processing an initial entry to this routine is to test the "SIO failure" indicator in general register zero. If it is on, a component failure is indicated along the original path, the alternate path retry flag in the PECB is turned on. The malfunctioning element flag in general register zero is tested and, depending on its setting, the malfunctioning device or the malfunctioning channel flag in the PECB is set on. The Alternate Path Retry subroutine is called to retry the operation and when control returns, it is immediately given over to the end processing portion of this routine.

If the 'SIO failure' flag is off, no device or channel failure is indicated. The 'same path retry' flag in the PECB is turned on and the Same Path Retry subroutine is called to retry the operation. When control returns, it is immediately

transferred to the end processing portion of this routine.

If the test for initial entry, described in the first paragraph, indicates that this is not the first entry, the input general registers are saved in the PECB and the 'error sense' flag in the PECB is tested. If the PECB indicates that Start Retry Operation has issued a Sense instruction, Alternate Path Retry is called to allow checking of the sense data. If the 'error sense' flag is off, the 'same path retry' flag is tested. If the 'same path retry' flag is off a test is made for intervention required in the PECB. If intervention is required, a call is made to Alternate Path Retry.

Note: The 'same path retry' flag and other pertinent data are available on subsequent entries since they were provided at the time of initial entry. If the flag is on, the PECB corresponding to the failing operation is retrieved.

If the PECB indicates that the paging operation was not a read the 'previous retry successful' flag in general register zero is tested. If the retry was successful and the I/O error was outboard, a call is made to the Real Core Statistical Data Recording subroutine and, when control is returned, this routine enters an operation successful code in general register zero and branches to its own end processing portion. If the retry was unsuccessful, the sense data applying to the previous retry operation is Ored into the sense data field in the PECB and the same path master error counter in the PECB is incremented by one. (If this value reaches 255, each of the error counters in the PECB is filled with X'FF'.) The Same Path Retry procedure is then called and upon return, control is immediately transferred to the end processing portion of this routine.

If the paging operation is found to have been read and the I/O error was a data check, one of six courses may be taken. This is because on the successful retry of a data check paging read operation, the data must be written out to reinforce the magnetic image and then re-read. Success of the paging operation is determined by the success of this re-read operation. A series of tests is then conducted. The possible results and the actions they precipitate are as follows:

If the 'previous retry successful' and the 'rewrite on successful read' and 'reread on successful read' flags are both off, the re-write/re-read procedure must be started. The rewrite on successful read flag is set, a write channel program is constructed, and the Start Retry subroutine

dynamic occurrence of a defect in the surface of the recording medium; while an unsuccessful Standard Area Retry indicates the dynamic occurrence of a defect in the device.

On entry, the subroutine calls the Supervisor Core Allocation subroutine to allocate the necessary work/save areas. When the space is allocated, the subroutine saves all input registers, and then determines what type of table is specified in the calling program's input register. If the table is a:

- Drum interface control block, the failing segment of the drum channel program is located as follows: the paging error control block (PECB) is accessed from the pointer in system table. The channel program segment field of the PECB contains a pointer to the channel program segment. This segment (4 command words) is then copied into the PECB starting at the channel program word field. The doubleword SEEK argument, located by the pointer in PECEPT is moved into PECSAR. The data address field in each PECB CCW is updated to reflect the change of channel-program location and SEEK argument location. The drum standard area head number (196) is then inserted into PECSAR+4.
- Direct access interface block, the 'write check option' flag in the DIAB is tested. If it is off, the 4-command-word-channel program segment located by the pointer in the PECB channel program segment field is copied into the PECB's channel-command word field. If the flag is on, the 9-command-word channel program segment located by the pointer in PECB is copied into the channel-command word field. The doubleword SEEK argument, located by the 3-byte pointer in PECCW+1 thru PECCW+3, is moved into PECSAR. The data-address field in each PECB CCW is updated to reflect the change of channel program and SEEK argument locations. The 2311/2841, 2314 standard area cylinder number (199) is inserted into PECSAR4.

In either of the above cases, the Start Retry Operation subroutine is called, the calling program's registers are restored, the main storage is released via the Supervisor Core Release subroutine, and control is returned to the calling program.

Alternate Path Retry Subroutine (CEAAS) Chart BH

This subroutine causes the malfunctioning components along the path where a paging error occurred to be set logically

unavailable and then performs one of the following:

- Retries along an alternate path if one can be found, or
- Enables the system and (as far as possible) the task to recover from the paging error, if an alternate path cannot be found.

Entry: CEAAS1

Modules Called: Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) allocates space for work and save areas.

Reverse Pathfinding (CEAA5 entered at CEAA5R) is called to release paths to defective channels, control units, or devices.

Set Path (CEAA5 entered at CEAA5S) is called to mark paths unavailable to defective channels, control units, or devices.

Pathfinding (CEAA5 entered at CEAA5P) is called to obtain alternate paths for retry operations.

Paging Failure Recovery subroutine (CEAAQ) performs defective device recovery when the I/O device is the cause of I/O failure.

Start Retry Operation subroutine (CEAAX) starts a specific paging I/O operation and informs the caller of the status of the start I/O operation.

Standard Area Retry subroutine (CEAAT) retries a paging operation on the standard area of the recording medium of the specified device.

Supervisor Core Release subroutine (CEAL1 entered at CEAL02) releases main storage after use.

Exit: To caller.

Operation: On entry, the subroutine performs the following:

- Calls the Supervisor Core Allocation subroutine to allocate space for work and save areas.
- Saves registers and establishes addressability.
- Locates the paging error control block and tests its 'previous retry successful' and 'control under standard area retry' flags in order to determine whether this is the result of an unsuccessful standard area retry. If so, the entry is processed as a device

failure as described below. If not, the 'alternate path busy' flag is tested to determine if this entry is the result of an I/O path having been freed. If so, the appropriate 'alternate path busy' flags are reset and the Pathfinding subroutine is called as described below. If not, the 'malfunctioning channel detected' flag is tested to determine if the channel failed. If so, the Reverse Pathfinding portion of the Pathfinding subroutine is called to release the entire input path.

When control returns to Alternate Path Retry, the Pathfinding subroutine is called to mark down the failing path and find an alternate path to the paging device, if one exists. Pathfinding's return code is tested to determine which of the three following conditions exists, and the appropriate actions to take:

- **Path-Unavailable.** The Paging Failure Recovery subroutine is called and its unavailable-alternate path-recovery function is requested. The Alternate Path Retry subroutine then returns control to the calling program.
- **Path-Busy.** The calling program's input register is checked for a pointer to a SYSDIC. If one is present, the PECB's 'alternate path busy' flags are set on in the PECB and the SYSDIC, and the drum involved is put in a hold state by turning on a flag in the page drum directory. If no SYSDIC pointer is given, the 'path busy' flags in the PECB and the device GQE are set on. The DIG busy flag in the scan-table entry for the device is set on (via Set Suppress Flags), and the subroutine returns control to the calling program as described above.
- **Path-Available.** The Start Retry Operation subroutine is called and the returned code is tested to determine if the start was successful. If so, the PECB's 'alternate retry' flag is set off, the 'control under same path retry' flag is set on, the PECAPM field is zeroed out, and control is returned to the caller. If the start was not successful, the PECAPM field is tested to see if this is the first restart failure. If so, the field is increased by one and the subroutine goes back to the path-available procedure. If not, the start-retry-operation return code is tested for a defective channel indicator. If one is present, the processing described previously is performed. If one is not present, a malfunctioning-device check is per-

formed and the subsequent processing steps are followed.

If the malfunctioning-channel flag is not on, the malfunctioning-DCU-detected flag in the PECB is tested. If a device-control-unit-failure is indicated, the subroutine performs the following:

- Calls the Reverse Pathfinding portion of the Pathfinding subroutine to release the entire input path.
- Calls the Pathfinding subroutine and proceeds as described above.

If a DCU failure was not indicated, the PECBs 'malfunctioning device detected' flag is tested. If a device failure is indicated, the subroutine performs the following:

- Calls the Reverse Pathfinding portion of the Pathfinding subroutine to release the entire input path.
- Calls the Pathfinding subroutine to mark down the failing path and to find an alternate path to the device. Processing continues as described above.

If no malfunctioning device condition existed indicating that, the defect may be either the I/O device or the recording medium, the subroutine calls the Standard Area Retry subroutine to initiate the paging operation on the standard area thereby determining whether the device or the medium is defective, and the return code is tested to determine if the retry was started successfully. If so, the PECB's 'alternate-retry' flag is set off, the 'control under standard area retry' flag is set on, a retry-in-progress return code is set, registers are restored and control is returned to the caller.

If the retry was not successful, the 'alternate path retry master error' counter in the PECB is tested. If it is zero, indicating the first restart failure, it is incremented by one and the Standard Area Retry subroutine is recalled. If the counter is not zero, the return code is checked for a defective-channel indicator. If one is present, the defective-channel procedure described previously is entered. If no indicator is present, the malfunctioning-device check is performed.

If the original test of the PECB's 'previous retry successful' and 'control under standard area retry' flags indicate that this entry is the result of an unsuccessful standard area retry, the latter flag is set off, the 'control under alternative path

is called to start the write operation. If the restart is unsuccessful, the failing element is noted and the Alternate Path Retry subroutine is called.

If the 'previous retry successful' and the 'rewrite on successful read' flags are on, and the 'reread on successful read' flag is off, the write-back operation succeeded and the re-read operation must be performed. A re-read channel program is constructed and initiated as described above.

If the 'previous retry successful' flag is on and the re-read and rewrite on successful read flags are on the original I/O error was outboard, the retry is considered successful and a call is made to the Real Core Statistical Data Recording subroutine.

If the previous retry was unsuccessful, the 'rewrite on successful read' flags are on and the 're-read on successful read' flag is off, the rewrite operation failed and the device is considered defective. A call is therefore made to the Alternate Path Retry subroutine as described above.

If the previous retry was unsuccessful, and both the re-write and reread on successful read flags are on, the reread after a successful re-write failed. The paging operation is deemed a failure; all PCB error counters are filled and the same path retry subroutine is called as described above.

Note: If the 'previous retry successful' flag is off, and the 'rewrite on successful read' and 'reread on successful read' flags are off, a normal unsuccessful retry is indicated, and a call is made to the Same Path Retry subroutine.

The preceding paragraphs describe processing when the 'same path retry' flag in the PCB is on after restoring the registers which point to where the search ended the last time Write Shared Pages was called. If this is the initial call, the registers are set with one pointing to the RSPI entry and the other with the RSPI entry count. If this flag is off and the 'standard area retry' flag is on, the 'previous retry successful' flag is tested. If it is on, the Paging Failure Recovery subroutine is called to perform its defective medium recovery function. When control returns, this routine enters its end processing portion.

If standard area retry is indicated but the 'previous retry successful' flag is off, the 'malfunctioning device' and the 'alternate path retry' flags are set on and the 'same path retry' flag is set off. A call is made to the Alternate Path Retry

subroutine and, when it returns control, this routine enters its end processing portion.

If neither same path retry nor standard area retry are indicated in the PCB the 'alternate path retry' flag is tested. If it is on, the Alternate Path Retry subroutine is called and, when it returns, this routine enters its end processing portion. If the alternate path retry flag is also off, a minor software SYSERR (code 6601) is issued.

The end processing portion of this routine is entered with a return code indicating one of four conditions in general register zero. These codes are generated by one of the routines called or by the Paging I/O Recovery routine itself and reflect the status of retry operations. The conditions which may exist and the processing they require are as follows:

Operation successful - all paging error retry flags in the DAIB are reset to zero, the address of the PCB is placed in general register one, general registers 2-15 are restored.

Operation reinitiated - if the error occurred on a drum, SYSDIC is set to zero, and the Supervisor Core Release subroutine is called to release the main storage occupied on a drum. If the error occurred on a direct access device, the Supervisor Core Release subroutine is called to release the main storage occupied by the DAIB. In either of the above cases the address of the PCB is entered in general register one and processing proceeds as for operation successful above.

Retry in progress - general registers 2-15 are restored from the PCB and control returns to the calling routine.

Operation failed - processing is the same as for operation reinitiated above except that prior to placing the address of the PCB in general register one, the Move GQE subroutine is called to move the original device GQE.

If an invalid return code is found in general register one, a minor software SYSERR (code 6604) is issued, a call is made to the Supervisor Core Allocation subroutine and processing continues as for seek CCW not found described above.

Start Retry Operation Subroutine (CEAAX)

This subroutine starts a specific paging I/O operation, and informs the caller of the status of the Start I/O operation.

Entry: CEAX1

Modules Called: Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) allocates main storage work/save areas.

Start I/O subroutine (CEAAG) issues a start I/O instruction for all calling programs.

Supervisor Core Release subroutine (CEAL1 entered at CEAL02) releases main storage after use.

Set Suppress Flag subroutine (CEAJQ entered at CEJJSF) sets the specified flag in the specified scan table entry.

Exit: To caller.

Operation: On entry, the subroutine calls the Supervisor Core Allocation subroutine to allocate main storage work-save areas. When this has been done, the subroutine saves the calling program's input registers. The input parameter register is tested to determine whether the channel program to be started is located in the paging-error-control block (PECB). If so, the address of the PECB channel-command word (CCW) is placed in the command address field of the channel-address word (CAW) to be used to start the I/O operation. The Start Retry Operation subroutine determines, from the input register, whether a drum-interface control (DIC) is specified. If so, the physical device address to be passed to the Start I/O subroutine is obtained from the system table.

If no DIC is specified, the physical-device address is obtained from the symbolic-device address field of the device queue. The 'error sense' flag in the PECB is tested to determine if Start Retry Operation had previously issued a Sense command. If so, the call to Start I/O is skipped and the sense data are tested. If intervention is required, a return code of 12 is set (otherwise the return code is 4) and control is returned to the calling routine. If there was no previous Sense command, the channel-address word and the device address are loaded into general registers and the Start I/O subroutine is called.

Start I/O returns the results of its attempts in a general register. The Start Retry Operation subroutine tests the returned-result indicators and sets up return codes for the calling program. If Start I/O indicated a successful restart, the subroutine specifies a return code of 0 to the caller. If Start I/O was unsuccessful, one of the following codes is returned:

- If a defective channel caused the failure, the return code is 8.
- If a defective device caused the failure, the return code is 4.

These codes are loaded into the return register, all other registers are restored from the save area, the storage is released by calling the Supervisor Core Release subroutine, and control is returned to the calling program.

If channel status is zero and 'unit check' is set, a Sense command is executed by calling the Start I/O routine. On return, if Start I/O was successful, the 'error sense' flag is set in the PECB and a 'retry in progress' indication is returned to the calling routine.

If the channel program to be started was not located in the paging error control block, Start Retry Operation determines whether a DIC pointer is specified in the input register. If not, the subroutine places a pointer to the original, failing channel program segment field (obtained from the paging error control block) in the command-address field of the CAW. The physical-device address is obtained as described previously and Start I/O is called.

If there is a DIC pointer specified, processing is the same as above, except that the physical device address is obtained as described previously, under no DIC specified and Start I/O is called.

Processing continues as described above.

Standard Area Retry Subroutine (CEAAT)

This subroutine retries a paging operation on the standard area of the recording medium of the specified device.

Entry: CEAT1

Modules Called: Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) allocates the necessary work/save areas.

Start Retry Operation subroutine (CEAAX) starts a specific paging I/O operation and informs the caller of the status of the start-I/O operation.

Supervisor Core Release subroutine (CEAL1 entered at CEAL02) releases main storage after use.

Exit: To caller.

Operation: A successful Standard Area Retry, when used as a part of the Paging Error Recovery subroutine, indicates the

retry' flag is set on, and the Standard Area Retry subroutine is called as described above. If not, and the 'alternate path busy' flag indicated that the entry was a result of an I/O path having been freed, the 'paging error busy path' and 'alternate path busy' flags are set off, and the Pathfinding subroutine is called as described earlier.

In all cases, the Alternate Path Retry subroutine performs the following exit:

- Indicates a return condition code in a general register.
- Restores the calling program's registers.
- Calls the Supervisor Core Release subroutine to release the work/save area.
- Returns control to the calling program.

Same Path Retry Subroutine (CEAAV) Chart BI

Entry: CEAAV1

Modules Called: Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) for the use of the Generate and Enqueue Interrupt GQE and Dequeue I/O Requests Subroutines.

Supervisor Core Release subroutine (CEAL1 entered at CEAL02) releases the work and save areas after processing has been completed.

Alternate Path Retry subroutine (CEAAS) attempts retry along an alternate path when it cannot be accomplished along the same path.

Task Communication Control subroutine (CEAAN) attaches any required messages to the operator task.

Start Retry Operation subroutine (CEAAX) restarts all I/O retry operations whether along the same or alternate paths.

Paging Failure Recovery subroutine (CEAAQ) performs clean up procedures when an alternate path is not available or when a defective medium condition exists.

Statistical Data Recording subroutine (CEAI6) saves sense and status data for later diagnostic use.

Real Core Error Recording (CEAI7) records inboard errors.

Move GQE subroutine (CEAJQ entered at CEAJMG) removes a GQE from one queue and adds it to another.

Exits:

Normal - Paging I/O Error Recovery (CEAAM)
Error - No major software system error is issued.

Operation: On entry at CEAAV1, the following action is taken:

- The Supervisor Core Allocation subroutine is called to obtain all necessary work/save area(s) space.
- The input general registers are saved.
- The SYSDIC flag in the input register is tested to determine if a drum interface control block pointer was specified and therefore the failing paging device was a drum.
 - 1) If so, the "2301/2820 recovery procedure" is invoked.
 - 2) If not, the "2311/2841, 2311 recovery procedure" is invoked.
- Recovery procedures are invoked according to error type.
- General registers 2 through 14 are restored. Supervisor Core Release is called to release the save area, and control is returned to the caller.

2301/2820 ERROR RECOVERY PROCEDURE: Error recovery procedures for drum when status data indicates error follow.

Channel Control Check:

- Retry threshold not reached - the original operation is retried as described in "Restart" (a).
- Retry threshold reached - Real Core Error Recording is called. On return, the System Error Processor is called to report the error. Alternate Path Retry is then called.

Interface Control Check: Procedure is the same as for Channel Control Check.

Channel Data Check:

- Retry threshold not reached - the original operation is retried as described in "Restart" (a).
- Retry threshold reached - The System Error Processor is called to output operator message 12. A minor software system error (Code 7412) is reported. Real Core Error Recording is called. Paging Failure Recovery is called for

its "Unavailable Original Path Recovery" function.

Unit Check: The 'outboard error' flag (PECOE) is set on. The 'await from sense' flag (SYSSN) is set off. The sense data is placed into the PECB and tested. The sense data testing is covered in "Error Recovery Procedures for Drum When Sense Data Indicates Error."

Chaining Check:

- Retry threshold not reached - the original operation is retried as described in "Restart" (a).
- Retry threshold reached - The System Error Processor is called to output operator message 12. A minor software system error (Code 7412) is reported. Real Core Error Recording is called. Paging Failure Recovery is called for its "Unavailable Original Path Recovery" function.

Program Check: The System Error Processor is called to output operator message 13. A minor software SYSERR (Code 7413) is reported. Real Core Error Recording is called. Paging Failure Recovery is called for its "Unavailable Original Path Recovery" function.

Protection Check: The System Error Processor is called to output operator message 14. A minor software SYSERR (Code 7417) is reported. Real Core Error Recording is called. Paging Failure Recovery is called for its "Unavailable Original Path Recovery" function.

Unit Exception: The System Error Processor is called to output operator message 15. A minor software SYSERR (Code 7415) is reported. Real Core Error Recording is called. Paging Failure Recovery is called for its "Defective Medium Recovery" function.

Incorrect Length: The System Error Processor is called to output operator message 16. A minor software SYSERR (Code 7416) is reported. Real Core Error Recording is called. Paging Failure Recovery is called for its "Defective Medium Recovery" function.

Attention:

- Retry threshold not reached - the original operation is retried as described in "Restart" (a).
- Retry threshold reached - The System Error Processor is called to output operator message 28. (See "Flag Setting" (b).) Real Core Error Recording

is called. Alternate Path Retry is called.

No Status Data Present:

- Sense data present - processing continues as described in "Unit Check."
- Sense data not present:
 - 1) Retry threshold not reached - the original operation is retried as described in "Restart" (a).
 - 2) Retry threshold reached - The System Error Processor is called to output operator message 1. A minor software SYSERR (Code 7401) is reported. (See "Flag Settings" (a).) Real Core Error Recording is called. Alternate Path Retry is called.

Error recovery procedures for drum when sense data indicates error follow.

Equipment Check:

- Retry threshold not reached - the original operation is retried as described in "Restart" (a).
- Retry threshold reached - The System Error Processor is called to output operator message 19. (See "Flag Settings" (d).) Real Core Statistical Data Recording is called. Alternate Path Retry is called.

No Record Found:

- Retry threshold not reached - the original operation is retried as described in "Restart" (a).
- Retry threshold reached - The System Error Processor is called to output operator message 3. A minor software SYSERR (Code 7403) is reported. (See "Flag Settings" (c).) Real Core Statistical Data Recording is called. Alternate Path Retry is called.

Invalid Address: The System Error Processor is called to output operator message 6. A minor software SYSERR (Code 7406) is reported. (See "Flag Settings" (d).) Real Core Statistical Data Recording is called. Alternate Path Retry is called.

Intervention Required:

- Retry threshold not reached - the original operation is retried as described in "Restart" (a).
- The System Error Processor is called to output operator message 20. Real Core

Statistical Data Recording is called. Paging Failure Recovery is called for its "Defective Device Recovery."

Bus Out Check:

- Retry threshold not reached - the original operation is retried as described in "Restart" (a).
- Retry threshold reached - The System Error Processor is called to output operator message 21. (See "Flag Settings" (b).) Real Core Statistical Data Recording is called. Alternate Path Retry is called.

Data Check:

- Retry threshold not reached - the original operation is retried as described in "Restart" (a).
- The System Error Processor is called to output operator message 22. (See "Flag Settings" (c).) Real Core Statistical Recording is called. Alternate Path Retry is called.

Overrun:

- Retry threshold not reached - the original operation is retried as described in "Restart" (a).
- The System Error Processor is called to output operator message 7. A minor software SYSERR (Code 7407) is reported. Real Core Statistical Data Recording is called. Paging Failure Recovery is called for its "Unavailable Original Path Recovery" function.

Command Reject: The System Error Processor is called to output operator Message 8. A minor software SYSERR (Code 7408) is reported. Real Core Statistical Data Recording is called. Paging Failure Recovery is called for its "Unavailable Original Path Recovery" function.

Byte 0, Bit 6: (See "Flag Settings" (b).) Real Core Statistical Data Recording is called. Alternate Path Retry is called.

Track Overrun: The System Error Processor called to output operator message 9. A minor software SYSERR (Code 7409) is reported. Real Core Statistical Data Recording is called. Paging Failure Recovery is called for its "Unavailable Original Path Recovery" function.

End of Cylinder: The System Error Processor is called to output operator message 10. A minor software SYSERR (Code 7410) is reported. Real Core Statistical Data Recording is called. Paging Failure Recovery

is called for its "Unavailable Original Path Recovery" function.

File Protect: The System Error Processor is called to output operator message 11. A minor software SYSERR (Code 7411) is reported. Real Core Statistical Data Recording is called. Paging Failure Recovery is called for its "Unavailable Original Path Recovery" function.

Overflow Incomplete: The System Error Processor is called to output operator message 17. A minor software SYSERR (code 7417) is reported. Real Core Statistical Data Recording is called. Paging Failure Recovery is called for its "Unavailable Original Path Recovery" function.

No Sense Data Present:

- Retry threshold not reached - the original operation is retried as described in "Restart" (a).
- Retry threshold reached - The System Error Processor is called to output operator message 2. A minor SYSERR (Code 7402) is reported. (See "Flag Settings" (b).) Real Core Statistical Data Recording is called. Alternate Path Retry is called.

2311/2841, 2314 ERROR RECOVERY PROCEDURE: The recovery procedures for disk when status indicator error are the same as those for drum, excepting Unit Check.

Unit Check: The 'outboard error' flag (PECOE) is set on. The 'AWAIT from sense' flag (SYSSN) is set off. The sense data is placed into the PCB and tested. The sense data testing is covered in the next section.

Error recovery procedures for disk when sense error data indicated error follow.

Equipment Check:

- Retry threshold not reached - the original operation is retried as described in "Restart" (a).
- Retry threshold reached - a message requesting that the disk pack be moved is sent to the operator.

No Record Found:

- No missing address marker - A read home address channel program is constructed in the PCB. Following the read home address, a check is made to see if this is the correct track.

- 1) Correct Track - The System Error Processor is called to output operator message 24. A minor software SYSERR (Code 7424) is reported. (See "Flag Settings" (c).) Real Core Statistical Data Recording is called. Alternate Path Retry is called.
 - 2) Incorrect Track - Retry threshold not reached - the original operation is retried as described in "Restart" (b).
 - 3) Incorrect Track - Retry threshold reached - The System Error Processor is called to output operator message 4. (See "Flag Settings" (d).) Real Core Statistical Data Recording is called. Alternate Path Retry is called.
- Missing address marker - Retry threshold not reached - the original operation is retried as described in "Restart" (b). A read home address channel program is constructed to do a read home address on a different track.
 - 1) Successful - processing continues as described under "Missing Address Marker."
 - 2) Unsuccessful - operation is retried if the retry threshold is not reached. If the retry threshold is reached, the System Error Processor is called to output operator message 27. (See "Flag Settings" (d).) Real Core Statistical Data Recording is called and Alternate Path Retry is called.

Seek Check: A check is made to see if there was also a command reject. If yes, the System Error Processor is called to output operator message 25. A minor software SYSERR (Code 7425) is reported. Paging Failure Recovery is called for its "Unavailable Original Path Recovery" function.

If no, and retry threshold is not reached, the original operation is retried as described in "Restart" (b).

If retry threshold is reached, the System Error Processor is called to output operator message 5. (See "Flag Settings" (d).) A message requesting that the disk pack be moved is sent to the operator. Real Core Statistical Data Recording is called. Alternate Path Retry is called.

Intervention Required:

- Retry threshold not reached - the original operation is retried as described

in "Restart" (a).

- Retry threshold reached - The System Error Processor is called to output operator message 20. Real Core Statistical Data Recording is called. Paging Failure Recovery is called for its "Defective Device Recovery."

Bus Out Check:

- Retry threshold not reached - the original operation is retried as described in "Restart" (a).
- Retry threshold reached - The System Error Processor is called to output operator message 21. (See "Flag Settings" (b).) Real Core Statistical Data Recording is called. Alternate Path Retry is called.

Data Check: If the retry threshold is not reached, the original operation is retried as described in "Restart" (a). Otherwise, reset counter to zero and check the recalibrate counter:

- Retry threshold not reached - the original operation is retried as described in "Restart" (b).
- Retry threshold reached - A read home address is constructed in the PECB and executed.

The System Error Processor is called to output operator message 22 (see "Flag Settings" (c)) for Standard Area Retry. Real Core Statistical Data Recording is called. Alternate Path Retry is called.

Overrun:

- Retry threshold not reached - the original operation is retried as described in "Restart" (a).
- Retry threshold reached - The System Error Processor is called to output operator message 7. A minor software SYSERR (Code 7407) is reported. Real Core Statistical Data Recording is called. Paging Failure Recovery is called for its "Unavailable Original Path Recovery" function.

Missing Address Marker:

- Retry threshold not reached - the original operation is retried as described in "Restart" (a).
- Retry threshold reached - A RHA channel program is constructed in the PECB and executed. The System Error Processor is called to output operator message 26 (see "Flag Settings" (c)) for Standard

Area Retry. Real Core Statistical Data Recording is called.

Command Reject: Same as for the 2301.

Track Condition Check: The System Error Processor is called to output operator message 18. A minor software SYSERR (Code 7418) is reported. Real Core Statistical Data Recording is called. Paging Failure Recovery is called for its "Unavailable Original Path Recovery" function.

Track Overrun: Same as for the 2301.

End Of Cylinder: Same as for the 2301.

File Protect: Same as for the 2301.

No Sense Data Present:

- Retry threshold not reached - the original operation is retried as described in "Restart" (a).
- Retry threshold reached - The System Error Processor is called to output operator message 2. A minor SYSERR (Code 7402) is reported. (See "Flag Settings" (b).) Real Core Statistical Data Recording is called. Alternate Path Retry is called.

RESTART SUBROUTINE: Start Retry Operation is called to start the specific retry operation.

- (a) The original operation is retried.
- (b) A recalibrate command for disk is issued followed by a seek to the original address and transfer in channel to the failing program segment.

The return code is tested. If restarted, the "retry in progress" return code is tested to see if:

- (1) Malfunction Channel. (See "Flag Settings" (a).) Real Core Statistical Data Recording is called. Alternate Path Retry is called.
- (2) Malfunctioning Device. (See "Flag Settings" (d).) Real Core Statistical Data Recording is called. Alternate Path Retry is called.

FLAG SETTINGS:

- (a) Malfunctioning Channel: The 'same path retry' flag (PECSR) is set off. The 'malfunctioning device detected' flag (PECDV) is set off. The 'malfunctioning control unit detected' flag (PECCU) is set off. The 'malfunctioning channel detected' flag

(PECCH) is set on. The 'alternate path retry' flag (PECAR) is set on.

- (b) Malfunctioning Control Unit: The 'same path retry' flag (PECSR) is set off. The 'malfunctioning device detected' flag (PECDV) is set off. The 'malfunctioning channel detected' flag (PECCU) is set on. The 'alternate path retry' flag (PECAR) is set on.
- (c) Standard Area Retry: The 'same path retry' flag (PECSR) is set off. The 'malfunctioning device detected' (PECDV) flag is set off. The 'malfunctioning control unit detected' flag (PECCU) is set off. The 'malfunctioning channel detected' (PECCH) flag is set off. The 'alternate path retry' flag (PECAR) is set on.
- (d) Malfunctioning Device: The 'same path retry' flag (PECSR) is set off. The 'malfunctioning control unit detected' flag (PECCU) is set off. The 'malfunctioning channel detected' flag (PECCH) is set off. The 'malfunctioning device detected' flag (PECDV) is set on. The 'alternate path retry' flag (PECAR) is set on.

I/O SERVICE SUBROUTINES

Pathfinding Subroutine (CEAA5) Chart BJ

The Pathfinding subroutine determines which of one or more data paths is available to an I/O device. A section of the subroutine is responsible for reverse-pathfinding and partitioning.

The SETLOCK macro is used to insure that only one CPU is accessing the pathfinder tables at a time.

Hardware Configuration Requirements: Each selector subchannel on a multiplexor channel is allowed a maximum of one control unit. Actual addressing for each section (channel, control unit, device) must be numbered consecutively from zero.

Entries: The Pathfinding subroutine has one distinct entry point for each of the three functions it performs:

1. Pathfinding (CEAA5P).
2. Reverse Pathfinding (CEAA5R).
3. Set Path (CEAA5S).

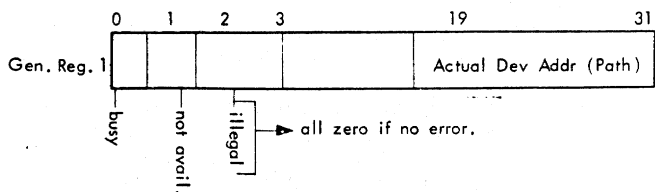


Figure 21. Format of path availability check results

Modules Called: Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) allocates main storage for register save areas and work areas, if needed.

Supervisor Core Release subroutine (CEAL1 entered at CEAL02) frees the main storage allocated by SCA.

Set Suppress Flag subroutine (CEAJSF) sets DIG busy flag.

Exit: To caller.

Operation: The operation of this subroutine depends on the entry point used.

Pathfinding (CEAA5P): When Pathfinding is called, the channel table lock (CHLLOCK) is set and the path availability is checked by the subroutine, and the results are returned to the calling program in the format shown in Figure 21.

A specific path to a device may be requested by setting an input flag to the Pathfinding subroutine. The caller may also specify that a specific path be disabled and an alternate path assigned.

When a specific path to an I/O device is not requested by the caller, the Pathfinding subroutine searches for the first available path to that device. A path comprises three components: the channel, control unit, and the device.

Pathfinding expects to find the system symbolic-device-address in the caller's input register. This address is converted to an actual-path-address and a list of all possible paths is then scanned. If the system-symbolic-device-address passed to Pathfinding does not match a system device address in the pathfinding tables, the 'illegal' flag is set on in the return general register. If all units of a path are available, the 'busy' flags in the appropriate table entries are set on, and the complete device address is sent to the caller via a general register. Another general register contains the device type as specified by the device-group table.

In order to have an available path, each unit along a given path must be free and not be partitioned, malfunctioning, in sense-hold, or reserved. Pathfinding checks this by first scanning the device-group table for the specific device. The device group table is protected in duplex operations by setting and resetting the device group table lock (DEVLOCK). If this specific device is found to be unavailable (for example, Partitioned), Pathfinding sets the 'busy' or 'not available' error flag in the return general register and returns control to the caller.

If the device is available, the device entry in the device table is marked as being busy. A check is made to see if a channel is available. If no channel is available, the 'device-busy state' flag is reset and Pathfinding returns to the caller with the 'busy' flag set on in the return register.

If the channel is available the availability of the associated control unit is checked. If the control unit is available, its entry in the control-unit table (CHACUT) is set busy, the symbolic device address is entered into the control unit table, and Pathfinding returns to the calling program with the actual-path-assigned contained in the return register.

If the control unit is not available, the 'channel busy' state is reset and another complete path is tested as outlined above. If no path is available, the device busy state is also reset and the appropriate error bit in the return general register is set on. If all available paths to the device are currently busy, the 'busy' flag is set on. If all existing paths to the device have units malfunctioning persistently, in sense-hold, reserved, or are partitioned out of the system the 'not available' flag is set on. After setting and resetting the appropriate flags, Pathfinding returns to the calling program.

If the requested symbolic-device address is not recognized as having an equivalent actual-device address, the illegal bit in the return general register is set on.

Reverse Pathfinding (CEAA5R): On entry for Reverse Pathfinding, the channel table lock is set, and an indicator bit is checked in the caller's input register. If the bit is on (signifying translate-only) the symbolic-device address is returned via general register, and the busy flags are not changed. If the bit is off the appropriate busy flags are reset. If an addressed channel or control unit is found not to exist (for example, if a channel with three associated control units 0, 1 and 2 is specified by the caller as having

a control unit 4) or is partitioned, the request is considered illegal, and an appropriate bit in a return register is set on to identify this error for the caller. Depending upon the type of interruption, the symbolic-device address may or may not be placed in a register. If the actual address is illegal (does not exist in the pathfinder tables), an indicator in the return general register is set on. The pointer to the associated asynchronous interruption entry is placed in a return general register, and if the device is capable of receiving asynchronous interruptions, a 'valid asynchronous interruption entry' flag is set in the return register.

The 'DIG busy' flag may or may not be turned on, depending on an input parameter.

The device group table is protected in duplex operations by the device group table lock (DEVLOCK).

Set Path (CEAA5S): When partitioning is needed, entrance is through Set Path (CEAA5S). A test is made to determine whether the path to be set is in register one; if it is, the path may be actual or symbolic, depending on the setting of the SDA flag. If it is not, an input register points to the first-partioned-device address (actual path). The number of partitioned addresses and the appropriate flag settings are assumed to be in an input general register. This will result in updating the availability flags of the sections involved (for example, setting on a tape drive's 'partitioned' or 'unit down' flag). For any specific unit only one flag may be set on at one time.

In addition to setting the status flags (partitioned or down) for each device, it is also possible to inhibit or restore an actual path at the control unit level. This action permits or inhibits the usage of a specific path without reactivating or deactivating any units. Such a request is only valid if the path to be restored or inhibited is one that intersects a control unit which has the two-tail-channel switch feature. In this case, the specified path is made available or non-available to the TSS/360 system, but the status of the other path through the control unit is unaffected. Use of this capability for paths that do not intersect a control unit with the two-tail-channel switch feature results in the setting off or on of the partitioning flag for that control unit.

If the two paths through a control unit with a two-tail-channel switch feature are inhibited independently rather than by partitioning out the control unit, then they must be restored independently (that is, when the two paths through a control unit

are inhibited, they cannot be restored by issuing a cancel partitioning request to the appropriate control unit on one path.

Note: The channel table lock (CHLLOCK) is reset upon exit from any of these routines.

Start I/O Subroutine (CEAAG) Chart BK

The Start I/O subroutine issues a start-I/O instruction for all calling programs. It investigates start-I/O failures and delivers error information to the caller. The caller program specifies the physical device address, symbolic device address protection key, and a CCW list pointer, and an input bit in register one which, if on, enhances disk I/O operations.

Entry: CEAAG1

Modules Called: Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) provides main storage for the construction of a GQE.

Enqueue GQE subroutine (CEAJQ entered at CEAJEN) places a pointer to the GQE on the channel interrupt processor's scan table queue.

Supervisor Core Release subroutine (CEAL1 entered at CEAL02) frees the main storage after use.

Exit: To caller.

Operation: On entry the subroutine creates a CAW (channel address word) for the specified CCW (channel command word) list and places it in the PSA. The I/O PSW interruption code is zeroed and then its high order bit is turned on. If after a start-I/O operation or a test-I/O operation, this bit is found to be off, it will be assumed that status was stored for a device other than the one addressed. The start-I/O instruction is then issued.

When the resultant condition code is zero, indicating the successful initiation of the start I/O instruction, the input bit for enhanced disk operations is tested. If it is on, a test I/O loop is entered until status is stored or the device is available.

If the device becomes available, bit 26 of register zero is set to one and processing continues. If the status is stored and only channel end has occurred, no bit setting takes place and processing continues. If status is stored and both channel end and device end have occurred, bit 26 of register zero is set to one, the CCW address portion of the CAW is increased by eight, and the start I/O instruction is reissued. If status is stored and the

device is busy, bit 26 of register zero is set to one and processing continues.

If status other than the above is stored, an error condition exists and the interruption is recreated. This is done by setting the interruption code field of the PSA to the device address; setting the problem program bit in the I/O old PSW to zero; setting the system mask, protection key, and program mask to the appropriate settings; and branching to the I/O portion of the interrupt stacker.

After the above processing is complete or if the input bit for enhanced disk operations was not on, control is returned to the calling program with a successful start-I/O indicated by a condition code of zero. A condition code of 1 with only the channel-end bit on in the status or with the status all zero is also treated as a successful start-I/O procedure.

Error flags are used to provide the calling program with information related to the unsuccessful initiation of the start-I/O instruction.

When the resultant condition code from the start-I/O instruction is 1, 2, or 3, additional investigation is undertaken to determine the cause of the start-I/O failure by the use of test-channel (TCH) and test-I/O (TIO) instructions. Resultant condition codes from SIO, TIO, and TCH are delivered at exit time to the calling program. Tests made on the SIO status yield the following conclusions, which are delivered to the calling program:

- Start-I/O condition code is 1 and the modifier bit only is on, or only the modifier and busy bits are on in the status indicates that the control unit is busy.
- Start-I/O condition code is 1; test-I/O condition code is 1; and status was stored for a device other than the one addressed and the I/O retry count is greater than 17: indicates that the channel is down, and too many interruption codes are stored for the device other than the one addressed, respectively.

If SIO or TIO return a condition code of three, a check is made to determine if a channel control check or interface control check has occurred. If either has, sense information in the PSA is saved for later diagnosis.

If the investigation shows the device to be busy, which is an impossible situation because of the existence of pathfinding, the start-I/O operation is attempted 17

times before declaring that a device is in the down condition.

If status was stored for another device (not the one addressed), a GQE will be queued on the appropriate interrupt queue processor's queue before SIO is retried.

These errors are by no means a complete list of all error combinations that could occur. They are only those error conditions which when analyzed in the light of the retry procedures attempted, permit the Start I/O subroutine to make a reasonable determination of the cause of the I/O failure.

Any time the content of the return general register is other than zero on return from the Start I/O subroutine, the SIO step has failed, except when the second SIO condition code is 1 and only the channel end bit is on in the status (indicating an immediate operation), or the SIO condition code is 1 and the status is all zero. This applies to failures that Start I/O makes no attempt to analyze (for example, condition code 1 from SIO followed by a condition code 3 from TCH). This type of error is returned to the calling program by using separate bits of the return general register to contain the condition codes for the SIO, TCH, and TIO instructions.

Halt I/O Subroutine (CEAAI)

This subroutine terminates input/output operations on specified line adapters. The only devices supported by this module are start/stop line adapters in the 2702 transmission control unit, type II synchronous line adapters in the 2701 data adapter unit start/stop line adapter in the 2703, and type II synchronous line adapters in the 2703. If I/O cannot be halted, status information regarding the I/O operation is returned to the caller.

Entry Point: CEAAIH, with the following input:

GR0

Flags:

bits 16-23 (X'20', leave I/O interrupts disabled on return)

device type code (bits 24-31)

GR1

Physical path address (bits 16-31)

GR14

Return address

Exit: To caller. When unable to halt I/O, output information is passed to the calling routine in registers 0 and 1 as follows:

GR0

- Bits 2-3: TIO condition code
- Bits 4-5: TCH condition code
- Bits 6-7: HIO condition code
- Bits 8-15: I/O select instructions issued
- Bits 16-23: Flags
 - X'80' - Information is stored for 2nd TIO
 - X'40' - CEAAI called for nonsupported device
 - X'20' - I/O interrupts are disabled

GR1

- Bits 0-15: CSW status from TIO condition code 1
- Bits 16-31: CSW status from HIO condition code 1

Operation: This subroutine determines the condition of an addressed device path. If any unit (channel, subchannel, or device) of a path is busy, a halt-I/O instruction is issued to terminate the I/O operation. The calling program passes the device type code and physical path address to halt I/O.

I/O interruptions are masked off at entry to Halt I/O and unmasked at exit, unless the caller requests that they be left disabled. The channel status word (CSW) is cleared before the device path is tested. The interruption code in the program status word (PSW) is cleared and its high-order bit is set on. This setting of the interruption code does not change if the test-I/O instruction produces a response for the addressed device. The code changes if the response pertained to other than the addressed device.

A test-I/O (TIO) instruction is issued to check the condition of the addressed device path. The CSW and condition code are saved. The setting of the condition code determines whether a halt-I/O is issued or not.

The condition code setting of zero indicates that all units of the device path are available. The subroutine then transfers control back to the calling program.

For a condition code setting of one, the status bits are tested to further define the hardware response to test-I/O. When

the CSW status bits indicate a channel interface control check, return condition codes are set to two and one for TIO and HIO instructions respectively when the device is a 2702 control unit. Otherwise, return condition codes are set to one for TIO and zero for HIO instructions. CEAAI then returns to the calling program with interrupts enabled or disabled as requested by the caller. If the channel and subchannel are available and the device and/or the control unit is busy, the status is saved and the subroutine issues a halt-I/O. For all other indications when the condition code is one, the routine saves the status and returns to the calling program.

An indication of channel or subchannel busy, via a condition code setting of two, means the subroutine must issue a halt-I/O instruction.

A test-channel (TCH) instruction is issued if the condition code is set to three (meaning a unit on the path is not operational) to ascertain if the unit is the channel or not. The condition code setting resulting from the test-channel is saved, and the subroutine exits to the calling program.

When a halt-I/O (HIO) is to be issued, the status save area is cleared to prepare for possible new status bit settings. After issuing halt-I/O, the condition code is saved and checked.

For a condition code equal to three, the TCH instruction is issued as detailed in a preceding paragraph. For a condition code equal to one, if the status is not zero, it is saved and the subroutine exits to the calling program. If the status is zero, a second TIO instruction is issued, the condition code and status are saved, and the subroutine exits to the calling program.

If a channel control check or interface control check is detected during processing, PSA sense information is saved for later diagnosis.

For all other condition code settings, the subroutine exits to the calling program.

The outputs for this subroutine are condition codes for TIO, TCH, HIO; status bits for TIO and HIO and, interruption code for TIO. These outputs are returned via general register as shown in Figure 22. The TIO status bits are returned in the GQE's CSW field and the interruption code is returned in the GQESAT field of the GQE.

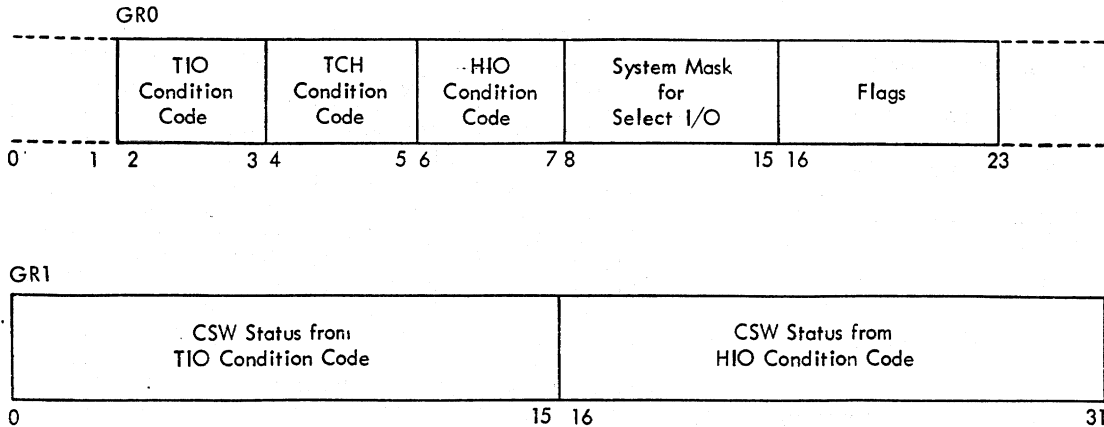


Figure 22. Contents of Halt I/O return registers

Dequeue I/O Requests Subroutine (CEAAJ)
Chart BL

This subroutine suppresses or removes pointers to GQEs in a device queue for a particular task. Many GQE pointers may appear in a device queue. These pointers may be associated with many tasks. The Dequeue I/O Requests subroutine removes or suppresses only those associated with the specified task.

Assumptions: The scan table entry lock byte (SCNF3LOK) for the device queue in question is locked by the calling routine using the SETLOCK macro.

Note: It is the responsibility of the calling routine to reset the lock using the OPENLOCK macro after exit except in the case of drums. CEAAJ will open the lock prior to exiting for drum operations.) A work area of 32 words must be supplied by the calling routine.

Entry: CEAAJD

Modules Called: Set Suppress Flags subroutine (CEAJQ entered at CEJJSF) resets I/O active flag when I/O is halted.

Move GQE subroutine (CEAJQ entered at CEAJMG) releases the space occupied by the GQE when processing is complete.

Locate Page subroutine (CEAML entered at CEAMLPL) provides the location of page hold flags which are to be reset.

Supervisor Core Release subroutine (CEAL1 entered at CEAL02) releases the main storage occupied by IORCBs which are no longer required.

Dequeue GQE subroutine (CEAJQ entered at CEAJDE) dequeues the first GQE from the device queue on which it appears.

Queue GQE on TSI (CEAAF entered at CEAAFQ) causes a GQE to be put on the TSI interruption queue and exit to the Queue Scanner.

Reverse Pathfinding subroutine (CEAA5 entered at CEAA5SR) determines the symbolic device address for an I/O interruption associated with a halt I/O instruction.

Halt I/O subroutine (CEAAI entered at CEAAIH) halts an active I/O operation so that its associated GQE can be removed on a 2702 terminal device.

Generate and Enqueue Interrupt GQE subroutine (CEABQ) creates an interruption GQE for a halt I/O operation on another device.

Pathfinding (CEAA5 entered at CEAA5P) determines if the device is a 2702 terminal by checking the device group table.

Search RSPI (CEAMS) uses the shared page number passed to it to determine the shared page table address.

Exit: To caller.

Operation: Dequeue I/O Requests is called by resident supervisor modules with the address of a 128-byte work area in Register 1 and the TSI pointer for the task involved in Register 0. The first word of the work areas contains the Symbolic Device Address of the device involved and the first byte of the following word contains a flag requesting one of the following functions:

Purge: Halt all possible I/O on this device

Suppress: Halt no I/O but allow no new I/O

The caller may also set a flag in this byte identifying the request as "IOS Request".

On entry the work area is zeroed leaving only the required input, the registers are saved in the work area, and the scan table entry for the device involved is located.

The first GQE in the scan table for this device, if any, is located and the first GQE flag in the work area is turned on to prevent the removal of a channel program in progress. It is turned off when the next GQE not software chained to the first GQE in the scan table is found. (Software chaining will cause the resident supervisor to link the channel programs in the IORCB's involved by means of a TIC command.)

If the function is purge, the GQE is flagged as such. If the symbolic device address is less than that of User Core Allocation the device is assumed to be a drum and the special drum processing described below is used. Otherwise, if the F1 suppress flag is on in the scan table and the first GQE flag is on, the I/O must be in progress on this GQE. A call to Pathfinding (CEAA5P) is made to convert the symbolic device address into a device type. If the device is not a terminal and the GQE is not a paging GQE, the count of outstanding I/O is raised and the next GQE processed. If the device is a terminal, Halt I/O (CEAAIH) is called to halt the device. If the GQE is a paging GQE, the I/O outstanding count is not incremented in the work area, and the next GQE is processed.

Upon return from Halt I/O, Reverse Pathfinding (CEAA5R) is called to locate the DEV entry for this device. If the halt I/O condition code was two, the halt I/O was successful and the F1 suppress flag is turned off in the Scan Table via a call to Set Suppress Flag (CEAJSF). If the condition code was not 2, the 'ignore device end' flag is set in the DEV to prevent a subsequent interrupt, if any, from being processed and the path is cleared via a call to Reverse Pathfinding (CEAA5R). The F1 suppress flag is then reset on the scan table via a call to Set Suppress Flag (CEAJSF).

The GQE for the terminal can now be discarded. Subsequent processing is also done if either the first GQE flag or the F1 suppress flag is off and the function is either Suppress or Purge.

If there are any pages in page hold, the page hold counts are decremented. If any private page has come out of page hold, the TSI is checked for a second scan time slice end GQE. When one is present, it is queued

on Time Slice End via a call to Move GQE (CEAMG). The main storage for the IORCB or DAIB is released unless the request is flagged as from IOS. If the 'skip dequeue' flag is not on, the GQE is dequeued; and, if the request is flagged as from IOS, is sent to the task via a call to Queue GQE on TSI (CEAAF). If the request is not flagged as from IOS, the dequeued GQE is released via Move GQE (CEAMG).

The remaining GQE's are then processed. If the original request had been 'suppress', the processing would have been identical except that a different flag (SKIP Flag) would have been set in each GQE, no call to Halt I/O would have been considered even for a terminal, and the 'skip dequeue' flag would have been set in the work area.

In all functions, it is assumed that if the symbolic device address is less than that of User Core Allocation, the request is to suppress the drum GQE's outstanding. Drum records on the drum are provided for the purpose of aiding the Customer Engineer in maintaining the hardware at an installation. For this reason, it is assumed that there is no case in which it is advisable to abort the writing of a drum record, even if I/O hasn't been started on it.

The first GQE found on the scan table for this TSI causes a check for a drum symbolic device address. If the device is a drum, the address of the SYSDIC for this drum is found in the paging drum directory in the system table and SYS89 is locked in the SYSDIC to prevent duplex problems. A flag is set in the work area so that CEAAJ will lock SYS89 only on the first GQE.

If the GQE does not have an associated IORCB, it is ignored. Each GQE in the scan table for this drum and each GQE in the SYSDIC is flagged so that the GQE will be discarded when the drum I/O is complete.

Finally SYS89 is unlocked, SCNF3LOK is unlocked, and return is to the caller.

Generate and Enqueue Interrupt-GQE Subroutine (CEABQ)

This subroutine sets up a GQE containing the information specified by the caller and effects its placement on the appropriate processor's scan table queue.

Entry: CEABQ1

Modules Called: Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) reserves main storage for constructing the GQE.

Enqueue GQE subroutine (CEAJQ entered at CEAJEN) places pointers to the various GQEs on the specified scan table queues.

Exits: To caller.

Operation: On entry, the subroutine performs the following:

- Saves the input registers of the calling program in a work area provided by the calling program.
- Establishes addressability for itself and all referenced tables.
- Calls the Supervisor Core Allocation subroutine to allocate a 64-byte storage space for the new GQE.
- Using a parameter list supplied by the caller, sets up the appropriate fields of the interruption GQE with the following:

Channel log-out data (if supplied)
 Location on queue value
 The CSW
 Interruption code

- Calls the Enqueue GQE subroutine to place the interruption GQE pointer on the appropriate interrupt processor's scan table queue.
- Restores the calling program's registers and returns control.

Note: It must be remembered that the contents of the hardware location of the channel log-out data, CSW, and interruption code are continuously subject to change due to incoming interruptions, and therefore these locations should not be used to pass information to this routine.

Command Word Relocator Subroutine (CEAAA)

This subroutine performs the operations required to relocate the channel command word (CCW) addresses from a virtual storage location to a main storage location.

Assumption: General registers 0, 3, 11, and 12 can be used to perform the channel command word relocation.

Entry: CEAAAR

Exit: To calling routine.

Operation: On entry, a test is made for a CCW list length of zero. If the result is positive, an error exit is taken, with a return code of 4 in general register 15.

The virtual storage CCWs and the related page lists are structured as shown in Figure 23.

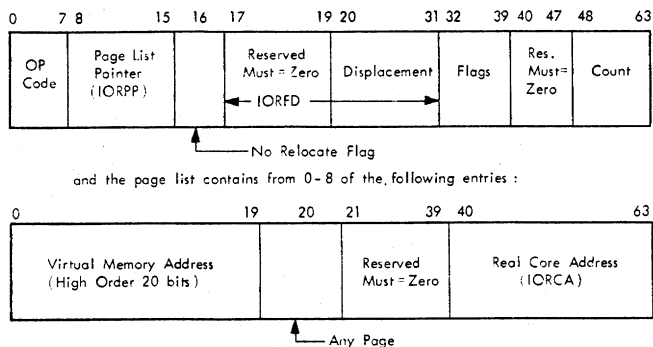


Figure 23. CCW - page list structure

The object of the address relocation operation is to replace the information contained in bits 8 thru 19 of each channel command word with the main storage address of the page. The first step in the relocation process is to examine the page-list pointer field in the IORCB. A nonzero value indicates that this channel command word references a user's virtual-storage-buffer page. If the value is greater than the IORCB's page-list-length parameter, a return code of 4 is placed in the return general register, the IORCB CCW 'specification error' flag is set on and the Command Word Relocator returns to the calling program. This error exit procedure is also invoked if the IORCB CCW list-length parameter is equal to zero.

An acceptable nonzero value is used to denote which page entry in the page list is to be used to relocate the channel-command word's address. For example, a value of 1 specifies the first entry in the page list, 2 the second entry, and so on. One is subtracted from the value (to adjust for the fact that the list entries begin relative to zero rather than one and as a consequence entry 1 is in list position zero, etc.) and the value is adjusted to its equivalent byte count value.

This adjusted value is added to the page list beginning address to get the base address of the required page list entry. Once the entry is located, its main storage address in the IORCB put into the channel command word without destroying the existing displacement value.

The location of the page list is calculated by taking the IORCB's 'relative-origin of page list' field, shifting it left 3 positions to develop 3 low-order

zeros, and then adding the beginning storage address of the IORCB to it.

If the IORCB's 'page list pointer' field is zero, then the channel-command-word address is relocated to an address within the IORCB's data buffer. The only exceptions to this rule are TIC commands which are relocated relative to the origin of the channel command word list rather than the data buffer. To relocate all channel-command words except TIC, the beginning main storage address of the IORCB's data buffer is added to the content of the IORCB's CCW displacement field and the resultant sum replaces the entire contents of the page list (IORPP) and displacement fields in the IORCB. For TIC commands, the processing is exactly the same with one minor exception: the beginning main storage address of the CCW list is used in place of the beginning main storage address of the data buffer.

The channel-command word contains a no-relocation flag. This is to prevent the address relocation of control commands in those cases where address relocation would destroy the command itself. If the flag is set on, the address is not relocated regardless of the type of command being processed.

The beginning main storage address of the data buffer is calculated by taking the contents of the IORCB's 'relative origin of data buffer' field, shifting it left 3 places to develop 3 low-order zeros, and then adding the beginning main storage address of the IORCB to it. The beginning main storage address of the CCW list is calculated in the same manner except that the 'relative origin of the CCW list' (IORCS) is used in place of that of the data buffer.

If the 'page list pointer' (IORPP) in the channel-command word is greater than IORGL, or if the CCW list length (IORCL) value is equal to zero, an error return code of 4 is placed in the error-return general register and the Command Word Relocator subroutine exits to the calling program.

Purge Subroutine (CEAAL) Chart BM

This subroutine inhibits the execution of an I/O request for one or all tasks. If the I/O request is in execution it may either be stopped immediately or allowed to quiesce. If it is stopped immediately the task symbolic device list (TSDL) entries can be removed either with the same call for all other purge activity or with a separate call. The calling routine specifies which action is to be taken.

Entries:

CEAALQ - as a subroutine.

CEAALP - as an SVC processor.

Modules Called: Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) reserves storage for use as a work area and for the use of Dequeue I/O Requests subroutines; if Purge is entered as an SVC subprocessor.

Supervisor Core Release subroutine (CEAL1 entered at CEAL01) - releases storage used as a work area if Purge is entered as an SVC subprocessor.

Remove Device from Task Processor (CEAAD entered at CEAAD2) either suppresses or removes a specific device from the TSDL for a specified task.

Dequeue I/O Requests subroutine (CEAAJ entered at CEAAJD) suppresses or removes pointers to GQEs in a device queue for a particular task and maintains a master I/O outstanding count.

Set Suppress Flag subroutine (CEAJQ entered at CEAJSF) sets the appropriate flag in the flag byte of the I/O Device Queue Processor's scan table entry.

Enqueue GQE subroutine (CEAJQ entered at CEAJEN)-queues GQE on Scan Table.

Pathfinding (CEAA5 entered at CEAA5R) obtains device type code from the device group table's asynchronous entry pointer.

Exits:

Subroutine - To caller.

SVC routine - To Queue Scanner or SVC Queue Processor.

Operation: When entered as a subroutine, a pointer to a 256-byte work area must be supplied by the caller, and the return is local.

When entered by the SVC Queue Processor, Purge obtains the work area space by calling the Supervisor Core Allocation subroutine. If space is not allocated, Purge resets the task to reissue the purge call by calling the Enqueue GQE routine to place a force-time-slice-end GQE pointer on the Timer Interrupt Queue processor's scan table queue. When control returns to Purge, the task's instruction counter, located in the PSW save area of the XTSL, is backed up by two times the instruction length count to enable the SVC to be issued. Return is then made to the Queue Scanner.

On entry the Purge subroutine expects to find the following parameters in the work area:

- A GQE pointer.
- One of the following action codes:
 - AR - Purge all devices immediately, removing the TSDL entries.
 - AL - Purge all devices immediately, leaving the TSDL alone.
 - AD - For all devices, remove the TSDL only.
 - AS - Purge all devices, letting active devices quiesce.
 - SL - Purge single device immediately, leaving TSDL alone.
 - SD - Remove TSDL for single device.
 - SR - Purge device specified in parameter 2 immediately, removing the TSDL entry.
 - SS - Purge device specified in parameter 2 letting the device quiesce if active.
- A symbolic device address (if the request is to purge one device).
- One of two type-of-action codes:
 - AT - Purge for all tasks
 - ST - Purge for one task
- A task identification if the request is to purge I/O requests for one task.

Purging of I/O requests may be for one of four combinations:

- One device for one task.
- All devices for one task.
- One device for all tasks.
- All devices for all tasks.

The entry parameter code, type and task ID are checked for valid entries. A system error with code 6101 is called for an invalid code. If the request affects any tasks (either singly or collectively) other than the calling task, the calling task must be the operator. Otherwise, system error with a code of 6107 is called. The same error code is used for an invalid type code.

Before processing the request, purge locates the necessary information about the task and associated devices.

The TSI is located either via the parameter list (for purging the calling task) or the TSI list of both active and inactive tasks. Before accessing the TSI list, the TSI chain lock byte (SYSTSIAD) is set using the SETLOCK macro. If the lock byte is not free within a specified amount of time, SYSERR is called with a code of 6103, and an error exit is made. For an all-task request each TSI is used, but for a single task request the list is scanned until the appropriate TSI is found. If no TSI is found SYSERR is called, with a code 6105, the work area released and purge returns to the calling routine or to the SVC Queue Processor.

Once the TSI is located the appropriate symbolic-device address is accessed either in the entry parameter list (for a single request) or the task's task-symbolic-device list (TSDL) (for an all-device request), and the TSI lock is set.

At this point unless the request code was AL, Purge calls the Remove Device from Task subroutine at CEAADR to either suppress or remove a specified device from the TSDL for the specified task, or for all tasks. Removing an entry consists of clearing the device entry, while suppressing consists of clearing the device entry lock byte and the device entry suppress flag in the TSDL. If the request is to suppress, the TSDL entry remains locked and will be unlocked by Purge after the I/O-request is dequeued.

Upon a successful return from the Remove Device from Task subroutine, the Dequeue I/O Request subroutine is called. Before calling the Dequeue I/O Request subroutine, the affected scan table entry lock byte (SCNF3LOK) is locked using the SETLOCK macro. If the lock byte is not free, SYSERR is invoked with a RC minor code 6131 and an error exit is made. This lock byte is used to preserve exclusive use of the queue entry in a duplex environment (that is, to prevent any other supervisor component from processing the queue). The Dequeue I/O Requests subroutine suppresses or removes all GQE's and IORCB's associated with the specified task in the locked queue. Upon return, the scan table entry lock byte (SCNF3LOK) is reset using OPEN-LOCK to again free the entry.

After a device has been removed or suppressed, Purge determines if more than one device was specified. If so, the basic procedure is repeated for each entry in the TSDL. Next Purge determines whether more than one task was specified. If so, the next TSI in the list is accessed in order to purge the associated device(s), and the TSI lock is set for the next task (after unlocking the preceding one). Once the

entire TSI list has been scanned the list lock byte (SYSTSIAD) is turned off.

Before exiting, Purge checks to see whether any devices were purged for any tasks. If none were, an error indication is set in a return register. Registers are restored and the work area is released for purge SVC processing. Exit is made to the calling program, or the SVC Queue Processor at CEAHQQ.

Terminal Control Table Entry Slot Allocation Subroutine (CEATS) Chart BN

This module allocates and releases terminal control table (TCT) slots and buffers for CEATC, acquiring and releasing pages of main storage as required to fill the request.

Entries:

CEATS1 - TCT slot allocation
CEATS2 - Buffer allocation
CEATS3 - TCT slot release
CEATS4 - Buffer release.

Input

R3 - one word save area
R5 - TSI pointer
R9 - MTSCB pointer
R14 - return address.

Input for CEATS3 is TCT address to be released in register 6.

Input for CEATS4 is buffer address to be released in register 13.

Modules Called:

Supervisor Core Allocation (CEAL01)
Supervisor Core Release (CEAL04)
Locate Page (CEMLP)

Output for CEATS1 is TCT address in register 6.

Output for CEATS2 is buffer address in register 13.

Exit: Return to the calling routine.

Operation:

CEATS1 - This subroutine allocates TCT entries for TSS users and MT/T users. If all slots are in use within a TCT page, an additional page is obtained and placed in a chain, in the CBT chain and entered in the page tables. The ECB count, if required, is updated and other counts in the TSI and XTSI for MT/T are updated to reflect the added page.

CEATS2 - This subroutine allocates buffer slots for TSS users and MT/T users. The size of the buffer is contained in the MTSCB in field MTSBLH. If all slots are in use within a buffer page, an additional page is obtained and placed in a chain, in the CBT chain and entered in the page tables. The ECB count, if required, is updated and other counts in the TSI and XTSI for MT/T are updated to reflect the added page.

CEATS3 - This subroutine will release TCT entries for TSS and MT/T users. If none of the entries are being used within a TCT page except for the first page, the page is released and removed from the TCT page chain, the CBT chain and the page table. The ECB count, if required, is updated and other counts in the TSI and XTSI for MT/T are modified to reflect the released page.

The first page for MT/T tasks is released only when specified by CEATS3 being called with Register 6 containing all FF's. The first page for TSS is never released.

CEATS4 - This subroutine will release buffer slots for MT/T and TSS users. If none of the entries are being used within a buffer page except for the first page, the page is released and removed from the Buffer page chain, the CBT chain and the page table. The ECB count, if required, is updated and other counts in the TSI and XTSI for MT/T are modified to reflect the released page.

The first page for MT/T tasks is released only when specified by CEATS4 being called with register 13 containing all FF's. The first page for TSS is never released.

SPECIAL TASK SERVICE SUBROUTINES

This group of subroutines provides such services as task initiation, activation and deactivation of TSIs, task ID scanning, XTSI-page expansion, task interruption queuing, and communication control via message block construction. The activities and functions of individual task service subroutines are described on the following pages. Attributes of the Special Task Service subroutines, except where otherwise noted, are reentrant, nonrecursive, resident, and operate in the privileged state.

Task Initiation Subroutine (CEAMC) Chart BO

This subroutine sets up an initialized TSI and assigns a unique task ID (TID) number to the TSI, provided the number of TSIs presently in the system is less than the maximum number allowed.

Entries:

CEAMT1

CEAMT2 - a special entry point, used by the Special Create TSI subroutine to insure that a TSI is created. This entry point bypasses the logic that might give an "unavailable" return.

Modules Called: Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) reserves main storage for the use of the Generate and Enqueue Interrupt GQE and Dequeue I/O Requests subroutines.

Rescheduling (CEAKZ entered at CEAKZA) is called with the address of the created TSI in general register 1 and a code of 8 in general register 4 in order to add the TSI to the inactive list.

Exit: To caller.

Operation: On entry, the subroutine interrogates the TSI indicator bit in the system table to make sure that tasks can be initiated into the system at this time. If task initiation is inhibited, a task ID number of zero is placed in the return register and control is returned to the calling program.

If task-initiation is not inhibited, the subroutine examines the TSI count in the system table to determine if any more TSIs can be created at this time. If the count has reached its maximum limit, no TSI is created, a task-ID number of zero is placed in the return register, and control is returned to the calling program.

If the count is less than the maximum it is incremented by one and the new count is stored in the system table. The count lock byte is then reset to allow another CPU to enter the same logic if required. Linkage is then made to the Supervisor Core Allocation subroutine to obtain 128 bytes of storage for a TSI and 64 bytes for temporary work storage to save the calling program registers. The storage obtained for the TSI is initialized in the following manner.

- All 128 bytes are set to zero.
- The count of the number of XTISI pages for this TSI is set to 1.

- The location of the skeleton XTISI is obtained from the system table and placed in the TSI.
- All task interruption bits in the TSI are enabled.
- The 'conversational' bit is set on.
- The 'XTISI swapped out' bit is set on.
- The 'delay' bit is set on.
- The 'inactive status' bit is set on.
- The task identification (TID) number in the system table is incremented by one. If no overflow occurs, this new number is placed in the TSI as the TID as well as in the system table. If overflow occurs, the TSI and the system table are assigned a TID number of 17, since a TID of zero indicates an error and TID numbers 1-16 are reserved for special system tasks.
- The initial task level from the system table is placed in the TSI schedule table index (TSISTE).

Rescheduling is then called to add the TSI to the inactive list. The 'pages used last time-slice' field (TSIPT5) is set to 15 (hex F). The subroutine then places the assigned TSI address and TID number in the return registers, restores all other registers, and returns to the calling program.

XTISI Overflow Subroutine (CEAMX)

This subroutine expands the number of XTISI pages needed for a task because of either page-table overflow or a segment-table overflow. An XTISI-overflow can occur because of a page-table expansion beyond the capacity of the first XTISI page or because of a segment table expansion within the first XTISI page.

Assumptions: Not more than 512 new segments will be added for each call to this routine.

Entries:

CEAMXP - for page-table expansion.

CEAMXS - for segment-table expansion.

Modules Called: Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) provides main storage for use as a register save area for page tables and auxiliary page tables.

Supervisor Core Release (CEAL1 entered at CEAL02) releases main storage after use.

XTSI Page Packing (CEAMY) consolidates page table pages that have become fragmented because of page table deletion.

Paging (CEAMQ) reads a page table into main storage.

Exit: Upon completion the XTSI Overflow subroutine exits to the calling routine by means of an unconditional branch to the address provided in register 14.

Error Checks: Segment number validity is verified in CEAMXS.

Operation: When entered at CEAMXP, this routine will perform the operation described under "Page Table Expansion;" its operation at CEAMX is described under "Segment Table Expansion."

PAGE TABLE EXPANSION: This portion of the XTSI Overflow subroutine allows the calling program to find room outside the first XTSI page for a page table associated with a specific segment within a task's virtual storage. The caller specifies the following parameters to this portion of the subroutine:

- The segment number.
- The number of pages required, where the specified number is equal to the number of entries presently assigned to the segment, plus the number of entries being added to this number.
- A pointer to the task's XTSI.

On entry, Expand Page first determines if the segment presently has a page table in a page table page (PTP). This is determined by examining the ASTP bit in the auxiliary-segment table entry for this segment. If the bit is zero the page table does not reside in a page table page and one must be found to handle the required page table expansion. Two fields within the XTSI are used to describe the PTP chain: XTSPTF and XTSPTL, designating, respectively, the first and last page table pages.

A check is made to see if there is any room in any page table page, in or out of main storage. Page table pages in main storage are checked first. If they are filled, page table pages not in main storage will be read in by the Paging routine (CEAMQ) and checked for unused space by XTSI Page Packing. This check is repeated until room is found.

If no PTP chain exists, a call is made to Paging (CEAMQ) to obtain a page from User Core Allocation for a page table. The PTP chain in the XTSI is then updated to

reflect the inclusion of this page. The header for the new PTP is then initialized with the following information:

- The forward and backward PTP pointers in the header are set to zero since there is only 1 page in the PTP chain.
- The number of segment fields in this PTP is set to one.
- The number of bytes in the PTP header, plus the number of bytes required for the expanded-page table and its header with room for ten (10) spare table entries, is deducted from the page size to determine the number of bytes available in the PTP. This field is then filled in.
- The location of the next available byte is inserted.
- The pointer to the first segment in this PTP is set to the byte following the PTP header field.

Following the initialization of the PTP header, the segment header for the page table to be placed in this PTP is initialized as follows:

- The segment number is inserted.
- The pointer to the segment table entry is inserted.
- The block size (header + PT + spares) is set.
- The number of bytes available in this block is set.
- The segment availability bit is set to 1.

Having completed the initialization of the PTP, this portion of the subroutine sets up the new page table origin in a return register. The calling program's registers are restored and control is returned.

If a PTP chain existed, it would be scanned to locate a PTP with adequate space to handle the expansion. If no room is found in any of the PTPs, and the XTSI page limit would be exceeded if a new PTP were added, XTSI Page Packing (CEAMY) is called to clean up the PTPs. On return from CEAMY, the PTPs are rescanned. If no room is found on the second scan, the XTSI page limit is checked. If at the limit, return is made with a nonzero return code. If the XTSI page limit will not be exceeded, a new page of main storage is requested from Supervisor Core Allocation. Forward and backward pointers are set up between this

new PTP and the existing PTP chain. Otherwise, the PTP header and segment header information are handled as though no other PTP chain existed.

If room is found in one of the PTPs, the PTP-header information is updated to reflect the new-page expansion to PTP. The segment header for the page table is initialized as described earlier. The new PT origin is placed in the return register as before, all registers are restored, work areas are released, and control is returned to the calling program.

The last condition involves the expansion of a page table which already occupies a PTP. In this case, the table's present location is expanded to determine if there are enough bytes available in the storage block assigned to this segment to handle the request. If there are, the expansion is made, the segment header is updated, the output parameter set up and the normal return is made. If there is not enough room in the present block, but the segment in question is the last segment in this PTP, the number of bytes remaining in the page is examined to determine if the page expansion will fit with 10 or fewer spares. If it will, the expansion is made within this PTP, the PTP header and segment header fields are updated as required, and control is returned to the caller.

If this segment must be moved to another PTP, the PTP and segment headers must be updated to reflect the deletion of this segment from its present PTP. The fields are updated and then the PTP chain in the XTISI is scanned in an attempt to find room as before. The location of the new PT origin is set up as before, registers restored and return made to the calling program.

SEGMENT TABLE EXPANSION: There are three states in which the XTISI can exist: '00', '01', and '02'. The state of an XTISI page is indicated in the flag byte XTISF2. Figure 24 illustrates these.

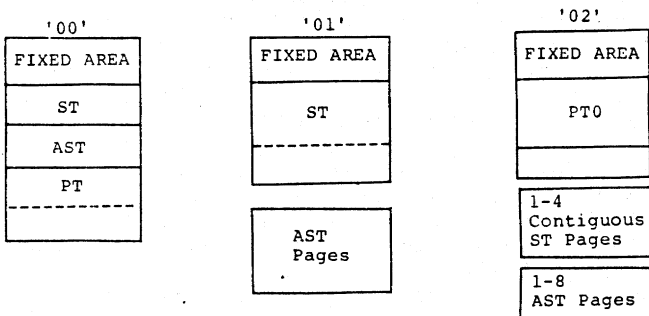


Figure 24. XTISI states

The skeleton XTISI initialized by system startup contains only 16 segment-table and auxiliary-segment-table entries. Any expansion beyond this number is accomplished by the segment table expansion portion of the XTISI Overflow subroutine.

On entry, this portion of the subroutine expects to find, in general registers, the following input parameters:

- The segment number.
- The address of the task's XTISI.

Expand Segment first establishes the validity of the segment number (that is, greater than 15). If the segment number is invalid, the condition code is set to non-zero and the routine returns to the calling program.

For XTISI state '00' the number of bytes required for expansion is calculated and compared to the number of bytes available in the first XTISI page. If room is available the following steps are taken:

- All page tables within the first XTISI page are 'pushed down' by the amount required for the segment table and auxiliary-segment table expansion.
- The segment-table entries of all 'pushed down' page tables are then updated to reflect their new page table origins.
- The present auxiliary segment table area is pushed down by the size of the segment table expansion.
- The new segment-table entries are initialized to indicate that no page table is available. The ADDPG routine will add the page table pointers as required.
- The auxiliary-segment-table entries are zeroed out, except for the ASTA bit, to signify that the segment is assigned. The ADDPG routine will fill in the AST entries as required.
- The save area in the XTISI for control register zero is then updated to reflect the new length of the segment table.
- The number of bytes available for XTISI expansion is reduced by the amount required for the segment expansion.
- Control is then given back to the calling routine.

If the number of bytes available in the first XTISI is insufficient to handle the

segment expansion, the existing segment table is scanned to locate the last page in the first XTSI page. The input parameters needed by Expand Page (CEAMP) are then set up and CEAMP is called to move the page table out of the first XTSI page. Upon regaining control, Expand Segment updates the entry whose page table was just moved out to reflect its new page table location.

This process continues until the amount of main storage reclaimed, when added to the number of bytes available, is sufficient to handle the segment expansion. At this time, Expand Segment begins the 'push-down' operation described previously.

If moving all PTs out of the first XTSI page fails to provide enough space for segment table expansion, a new XTSI page is requested from Supervisor Core Allocation and the auxiliary segment table moved to it. The XTSI page limit count governs this action, and the same procedures used in page table expansion are followed; that is, page-packing tries to clean up the XTSI pages to provide needed space.

Moving the AST out of the first XTSI page puts the XTSI in state '01' and requires updating the first AST page pointer and the XTSI state flag (located in the fixed area of the first XTSI page). The available byte indicator in the XTSI page count is also updated and the ST/AST entries initialized.

If segment table expansion in XTSI state '01' would cause the ST to overflow the first XTSI page, the same procedure is followed to locate space for it as in relocating the PT and AST; that is, Page Packing tries to clean up XTSI pages to provide needed space. When the segment table is moved out of the first XTSI page, the page table for segment 0 is moved in from its PTP, and the appropriate pointers and flags adjusted accordingly. This provides for more efficient operation of the system. Moving the ST out of the first XTSI page puts the XTSI in state '02'.

If the segment table requires more than one page, these pages must be contiguous to meet hardware requirements. The routines that call Expand Segment (ADDPG and ADSPG) anticipate and meet this requirement. Contiguous main storage is obtained by queuing a GQE for the request and queuing the Add Page request again. On return, this causes ADDPG to be re-entered with the address of the contiguous main storage in the GQE. This address is passed in register 0 to Expand Segment. Subject to the XTSI page count limit, Contiguous Core Allocation is called to get a new page. It will try to allocate the page following the existing ST pages. If it cannot, another group of con-

tiguous pages is allocated and Expand Segment moves the ST to the new ST pages, releases the old ST pages, updates the ST pointer, and initializes the new ST/AST entries. If the XTSI page limit would be exceeded, the new ST pages are returned to User Core Allocation before Expand Segment returns.

Note: Pages obtained for additional auxiliary segment tables and page tables need not be contiguous. They are obtained, as needed, from Supervisor Core Allocation and marked as user main storage with the appropriate pointers inserted.

Queue GQE on TSI Subroutine (CEAF) Chart BP

This subroutine queues a GQE, representing a specified interruption, on a queue off the appropriate task status index (TSI).

RESTRICTIONS: No information may be saved in bytes 28 to 35 of the GQE by the calling routine since this subroutine saves registers 2 and 3 in those bytes. The GQE must be dequeued from the scan table by the calling routine.

Entries: CEAFQ is used by TSS routines; CEAF2 is used by VSS routines to force the GQE to be queued directly on the regular TSI queue.

Exits:

Normal - To caller.

Error - To System Error Processor.

When an alternate TSI is to be constructed, exit is to the resident support system (RSS) via LPSW.

Modules Called: Rescheduling subroutine (CEAKZ entered at CEAKZA) puts the TSI in the active list.

Supervisor Core Release (CEAL1 entered at CEAL02) releases GQE core before exit to RSS.

Operation: On entry at CEAFQ, the subroutine expects the following parameters to be specified by the caller in general registers:

- The GQE pointer.
- One of the following interruption type codes:

Binary Code	Interruption Type
0	Program interruption pending
1	SVC interruption pending
2	External interruption pending (Task)
3	Asynchronous I/O interruption pending
4	Timer interruption pending (Task)
5	Synchronous I/O interruption pending
6	VSS activate interruption pending
7	Invalid
and up	

The subroutine inserts the binary interruption code into the task interruption code field (GQETIC) of the GQE. These codes establish a priority for queuing the GQE. GQEs with codes zero and 1 are chained onto the top of the queue; those with codes 2 through 5 are chained onto the bottom.

Interruption code 6 is always queued at the top of the queue. It is the interruption that occurs when a task system programmer connects to a task and activates the virtual support system (VSS). When this interruption is processed, the VSS active flag in the TSI is set on. Two other flags in the TSI may also be set indicating that a task system programmer is

connected and two terminals are in use for the task. When the TSI has the VSS active flag on, a pointer in that TSI points to an alternate TSI for the task. Subsequent interruptions are then handled according to their type and the various combinations of the TSI flag settings. (See Figure 25.)

Alphabetic characters in Figure 25 specify the action to be taken and correspond to the letters below.

- A. The interruption GQE is queued on the regular TSI interruption queue (TSITIP); and appropriate counts, flags, and interruption code are set in the GQE.
- B. If the external interruption code is zero, the GQE is queued on the alternate TSI queue. Otherwise, it is queued on the regular TSI queue.
- C. If the IORCB VSS flag is on, the GQE is queued on the regular TSI queue. Otherwise, it is queued on the alternate queue.
- D. These interruption GQEs are all queued on the alternate TSI queue.
- E. The PSW is loaded with the pointer to the "RSS activate VSS because of TSP attention" entry point. The interruption GQE and TSI pointers are left intact in their registers.

Flags								
VSS ACTIVE	yes	yes	yes	yes	no	no	no	no
TSP CONNECTED	yes	yes	no	no	no	no	yes	yes
TWO TERMINALS	yes	no	yes	no	no	yes	yes	no
////////////////////////////////////								
Interruption Type								
VSS Activate (6)	A	A	A	A	A	A	A	A
Program (0)	A	A	A	A	A	A	A	A
SVC (1)	A	A	A	A	A	A	A	A
External Task (2)	B	B	B	A	A	A	A	A
Asynchronous I/O (3)	D	E	D	D	A	A	F	E
Timer Task (4)	D	D	D	D	A	A	A	A
Synchronous I/O (5)	C	C	C	C	A	A	A	A

Figure 25. Action Matrix for TSI Flag Settings

F. If the TSI's SYSIN terminal and the terminal causing the interruption are the same, the interruption is queued on the regular TSI interruption queue. If they are not the same, the action described in E (above) is taken.

This subroutine checks for: more than one program interrupt type 3; more than one SVC; and the interrupt counter for type of interrupt overflow. On finding any of these, a program interrupt is queued on the task and a resident supervisor minor syserr is declared.

If an interruption-type code greater than 6 is encountered, an interruption code is placed in the GQE, the error flag is set, and the GQE is queued on the TSI.

If the interruption-type code is valid, the corresponding interrupt-counter field in the TSI is checked. If it is at its maximum value, the queue-error flag field in the GQE is checked. If off, it is turned on, the proper interruption code is stored in the GQE, and the GQE is queued on the TSI. If it is on, an exit is made to SYSERR.

If the counter value is correct, the counter is incremented by 1 and the appropriate 'software interrupt pending' flag in the TSI is turned on. The 'queue error' flag in the GQE is then checked to see if an error was encountered. If so, the SYSERR exit is taken.

If a pending interruption is enabled, the TSI is checked to see if it is on the active list. If it is, the TSI is checked to see if it is in AWAIT. If the TSI is in AWAIT, the AWAIT and real time clock are cancelled before setting the task to ready and returning to the calling routine.

If the TSI is on the inactive list, it is checked to see if it is in migration. If so, a bit is turned on in the TSI (TSIST) which will cause the Time Slice End Subroutine to determine whether or not migration will continue and when the task will be activated, and a return is made to the calling routine. If the migration flag is not on and the interruption is not a program controlled interruption (instead it is from an IORCB other than the last chained IORCB), Rescheduling is called to put the TSI on the active list, the task is placed in ready state, and a return is made to the calling routine.

Task Communication Control Subroutine (CEAN)

This subroutine constructs a message control block (MCB) for a calling task's

message and attaches it to the called task's TSI.

Entry: CEAN1

Modules Called: Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) reserves main storage for the use of the Generate and Enqueue Interrupt GQE and Dequeue I/O Requests subroutines.

Scan on Task ID subroutine (CEAAU) searches the active and inactive lists of TSIs for a task identification number that matches a number specified by the calling program.

Queue GQE on TSI (CEAAF) places a pointer to the specified GQE on the interruption queue in the affected task's TSI.

Supervisor Core Release subroutine (CEAL1 entered at CEAL02) releases main storage after use.

Exit: To caller.

Operation: On entry, the subroutine performs in the following manner:

- The Supervisor Core Allocation subroutine is called to allocate storage for all necessary work/save areas.
- The caller's input general registers are saved.
- The Scan on Task ID subroutine is called to locate the addressed task's TSI. If no TSI can be found, a minor software SYSERR is invoked and an appropriate call code is set. If the TSI is found, the size of the MCB is computed by:
 - a. Testing the specified message size and if it exceeds 238, truncating it to 238.
 - b. Converting the message size from double words to bytes.
 - c. Adding 16, the length of the MCB header, to the message size.
 - d. If the message is being sent to the operator a 16 byte operator header is also generated following the MCB header and preceding the text.
 - e. Rounding off the message size to the nearest multiple of 64.
- The Supervisor Core Allocation subroutine is called to allocate enough main storage to contain the MCB and the GQE.

- Inserts the following in the MCB as well as the operator header (OPH) when necessary.
 - a. The message length
 - b. The message byte conversion
 - c. The message text
- The GQE fields are established as follows:
 - a. The TSI pointer returned from the Scan on Task ID subroutine is inserted.
 - b. The MCB pointer is placed in the GQE.
 - c. Various OPH fields are filled in if the operator is to receive the message.
- The TSI lock byte is set on to prevent the task from being activated until the queuing process is complete. The Queue GQE on TSI subroutine is called to place the MCB GQE on the TSI-external-interruption queue, after which the TSI lock byte is set off.
- The calling program's general registers are restored, the Supervisor Core Release subroutine is called to release the work/save areas, and control is returned to the calling program.

2. Upon entry into the subject intercom, the intercom lock byte is set to prevent any other CPU from placing another message in the drop area until the current message has been removed by the subject CPU.

Entries:

- CEAIC1 - for the object Intercom.
- CEAIC2 - for the subject Intercom.

Exit: To caller.

Operation: On entry, Intercom performs the following:

- Transmits communication data from an object CPU's drop area (12 bytes of text and 2 bytes of control information) into the receiving (subject) CPU's drop area.
- Emits an external signal (external-start or external-interruption) over the extended direct control to the specified subject CPU.

There are two subroutines that comprise Intercom: the Object Intercom is a subroutine that can be called by a resident program operating in the supervisor state. The Subject Intercom is triggered into action solely by the external signal emitted across the extended direct control by the object CPU that is executing the Object Intercom. A functional description of each of these subroutines is presented on the following pages.

Object Intercom: This subroutine expects the following input parameters:

- The text of the Intercom message.
- One of the following Intercom message codes (IMC):

<u>IMC</u>	<u>Message</u>	<u>Operation Performed by Subject CPU</u>
00	Invalid	
01	External Start	Switch Prefix and/or LPSW from Words 1 and 2 of text (byte of Word 3 used as data byte)
02	Halt	Go into the Wait state (text ignored)
03	Resume	Resume at the Point of Halt (text ignored)
04	Halt & Transfer	Save CPU status and transfer to the address given in Word 2 of Text

GENERAL SERVICE SUBROUTINES

Inter-CPU Communication Subroutine (CEAIC) Chart BQ

This subroutine transmits commands from one CPU to another in the TSS domain and emits external signals over the extended direct control in order to have the receiving CPU perform the functions required for the coordination of CPUs in a multiprocessing system. Intercom is resident in the PSA and operates in supervisor state.

RESTRICTIONS:

1. The intercom text transmitted from one CPU to another via the drop area is limited to 12 bytes in length.
2. Entry into the subject intercom is solely by the external signals across the extended direct control.

Assumptions:

1. Since Intercom and the drop area are resident in the PSA, each CPU has a unique copy.

05 Reset Reset Associative
 Register
 06
 to Invalid
 FF

- The identity (ID) of the subject CPU, as follows:

If IMC is 01, ID is:

08 for CPU1
 04 for CPU2

If IMC is 02 to 05, ID is:

80 for CPU1
 40 for CPU2

- Return address
- Base address

Upon entry, Object Intercom performs the following:

- Checks the IMC in the input parameter to see if it falls within the valid range of values. If not, the return-code 1 (invalid IMC) is generated to reject the call, and control is returned to the caller.
- Identifies the subject CPU by scanning the ID byte of the input parameter (starting with CPU1) and when an associated bit of 1 is found, the CPU status table is checked to determine whether or not the subject CPU is currently active in the TSS domain. If not, control is returned to the caller, with return-code 2 or 4 depending upon whether the specified CPU is non-existent in the installation or currently partitioned out of the TSS domain.

If the CPU is currently active, its prefixed storage area is located. The intercom lock byte of the subject CPU's drop area is tested, and a wait loop is entered if the lock is on. When, and if, the lock is turned off, it is turned on. The text and IMC of the input parameters and the object CPU's ID are then placed in the subject CPU's drop area.

At this point the IMC is checked. If it is not one, the subroutine scans the ID byte for the next CPU and repeats the processing described above. If the IMC is one, the subject CPU's new prefix (whether primary or alternate as specified in the input parameters) is determined. The prefix fields of the status tables in every active PSA are updated, if necessary, and the IPL PSW of the subject CPU is set up

with the Subject Intercom subroutine's entry point (CEAIC2) in the instruction address. The subroutine then scans the input ID byte for the next CPU and repeats the processing described previously.

When an ID bit of one is not found associated with a CPU, or when all processing is complete the subroutine issues the Write Direct instruction to generate the signal specified by the IMC, (external start, or external interruption) and returns a return code of 0 and control to the caller.

Subject Intercom: Entry into the Subject Intercom subroutine at CEAIC2 is made either by an external-start, issued by the object CPU, or from the recovery nucleus on an external-interruption caused by the execution of a Write Direct instruction.

When the subroutine is entered, the subject CPU, with external interruptions disabled, transfers the communication data from its drop-area into working storage and releases the intercom lock in order that another CPU may place messages in the drop area. The following functions are then performed as specified by the intercom code given by the object CPU.

IMC 1
 External Start: Switch to the extended-PSW mode, and place the first two words of the text into the PSW.

IMC 2
 Halt: Check the external-old-PSW in the PSA to see if the external-interruption occurred while in the wait state. If not, save the extended-old-PSW and timer, and go into the wait state. If the CPU was in the wait state, update the elapsed timer field in the PSA and go back to the wait state.

IMC 3
 Resume: Update the elapsed timer, restore the timer from the timer value saved by the halt, and load the saved external-old-PSW into the PSW to resume at the point of the halt.

Note: A Resume is required to continue operation in a HALTed CPU. Halt commands received while the subject CPU is in the wait state are effectively ignored in order that the subject CPU may always resume its operations at the point of the first halt. A Resume command does not put the subject CPU into the wait state regardless of the number of intervening halt commands it may have received.

IMC 4
 Halt and Transfer: Save the general

and floating point registers in the Subject Intercom's save area, and transfer control to the address given in the second work area of the text. If control is returned, update the elapsed timer, restore the general and floating point registers and timer, and resume at the point of the external interruption.

IMC 5

Reset: Store the extended control register 0 into a temporary working area in main storage and load it back into the extended control register to cause the reset of its associative registers.

Create Real Time Interrupt Subroutine (CEAKR) Chart BR

This subroutine sets up a task timer interruption when the dispatcher finds that the system's elapsed time matches or exceeds the time interval value in the first entry of the real-time-interval-pending queue.

Entry: CEAKRT

Modules Called: Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) provides main storage for the purpose of constructing a GQE.

Queue GQE on TSI subroutine (CEAAF entered at CEAAFQ) queues the GQE on the TSI.

Supervisor Core Release subroutine (CEAL1 entered at CEAL02) releases 64 byte blocks of storage when a sufficient number of entries have been worked off the top of the queue.

Exit: Queue Scanner.

Operation: On entry, the subroutine checks the cancel flag (RTICNCL) in the RTI flag byte. This flag is set by the Delete TSI subroutine (CEAMD). If it is on, the RTI entry is ignored. If the RTIADM flag is on, it is a supervisor request. If not, the subroutine requests a 64-byte storage area for a GQE from the Supervisor Core Allocation subroutine. This storage block is initialized as a GQE which is used to create a task timer interruption. The TSI for the affected task is addressed by the third word of the first queue entry in the real-time-interval-pending queue. The address of the TSI and an interruption code are stored in appropriate fields in the new GQE and a call is made to Queue GQE on TSI subroutine to place the GQE pointer on the interruption queue in the TSI.

If the request is from the supervisor, the address constant of the calling routine must be in the RTI ADDRESS. After this routine creates the interrupt, it branches to the address specified by the ADCON.

The remainder of this subroutine's functions are concerned with the maintenance of the configurations of both the system table pointers and counts, and the maintenance of the real-time-interval-pending queue configuration. Each time the subroutine is activated, the pointer-to-first-queue-entry in the system table is incremented, as is the count of the number of bytes released. When this count reaches 64 the subroutine returns the first 64 bytes in the existing main storage block by calling the Supervisor Core Release subroutine and updates the appropriate pointers and counts. The exit from this subroutine involves resetting the lock and branching to the Queue Scanner.

TASK SELECTION AND SCHEDULING ROUTINES

The supervisor routines which comprise the task selection and scheduling mechanism are: the Internal Scheduler which receives control from the Queue Scanner when there is no work to do on the scan table and which maintains the order of tasks on the dispatchable portion of the active list; the Entrance Criteria subroutine which determines whether a task can be moved from the eligible list to the dispatchable list; Rescheduling which moves tasks from the dispatchable to the eligible or inactive list and from the inactive list to the eligible list and computes the task SST; the Dispatcher which selects a task to be given CPU time and calls Task Interrupt Control to check for pending task interrupts and arrange for them to be serviced by the task monitor before dispatching the task. A detailed description of these routines can be found in the following pages.

Internal Scheduler (CEAKI) Chart BS

The Internal Scheduler serves as an interface between the queue scanning function of the resident supervisor and the Dispatcher. It selects and moves tasks from the eligible list to the dispatchable list.

Entry: The Internal Scheduler is entered at CEAKIA by the Queue Scanner when there is no processable work on the scan table.

Modules Called: Supervisor Core Allocation (CEAL1 entered at CEAL01) to obtain main storage for a GQE to force time slice end or for a GQE/PCB.

Enqueue GQE subroutine (CEAJQ entered at CEAJEN) to put a GQE on the User Core Allocation queue and the Timer Interrupt Processor queue.

Entrance Criteria subroutine (CEAKE entered at CEAKEA) to determine if a task may be moved from the eligible to the dispatchable list.

Exits: Exit is to the Queue Scanner when a task is to be forced to time-slice end or when the first XTSI page must be read in for a task moved to the dispatchable list.

Exit is to the Dispatcher when no work has been placed on the scan table.

Operation: The Queue Scanner enters the Internal Scheduler with interrupts disabled when there is no processable work on the

scan table. The first function performed by this routine is to test the flag CEAKIS (Sleep flag). This flag is set by RSS to signal that hardware partitioning is to take place. When the flag is on, the Internal Scheduler puts the system into a wait state via an LPSW instruction. Otherwise, the Internal Scheduler sets the dispatching algorithm lock. It then checks the system table field SYSCTP. If not zero, it will contain a task ID in the first halfword and a schedule table entry level in the second halfword. The active list is scanned looking for the task specified by the TID. If it is found, and it is on the dispatchable list, its STE level is changed in the TSI (TSISTE) to that specified in the second halfword of SYSCTP. SYSCTP is then set to zero and control passes to Step 1. If the task is on the eligible list, its STE level is changed, and the Entrance Criteria subroutine is called to determine if the task can be added to the dispatchable list. If not, control passed to Step 2 to assure that this task will be the first moved to the dispatchable list. If accepted, SYSCTP is set to zero and control passes to Step 4.

If SYSCTP is zero, processing is as follows:

Step 1

The 'internal scheduler no work' flag (SYSNWK) is checked. If off, the master clock (MC) is compared to the lowest ahead of schedule SST (SYSLSST) found in the last pass through the Internal Scheduler. If this task is presently behind schedule, SYSLSST is set to zero and the eligible list is scanned from the top. Otherwise scanning begins at the next task to check as specified by SYSNTSI. During this scan to the end of the eligible list, SYSLSST is updated as necessary.

When a behind schedule task is found a check is made to see if it is in migration (TSIMG=1). If not, it is compared with the Schedule Table "maximum behind schedule" field. If it is above its maximum and ready, it is submitted to Entrance Criteria. If not, the search continues. If no task is above maximum, the first task behind schedule any amount is submitted to Entrance Criteria. If it is rejected, SYSNWK is set on, SYSNTSI is set to point to the rejected TSI and control passes to Step 3. If accepted by Entrance Criteria, control passes to Step 5.

If no behind schedule task is found during the scan, the first task in the eligible list that is not in migration (TSIMG=0) is submitted to the Entrance

Criteria subroutine. If rejected, or if all eligible tasks are in migration, SYSNWK is set on and control passes to Step 3. If accepted, control passes to Step 5.

Step 2

If Entrance Criteria rejects an eligible task specified by SYSCTP or a behind schedule task, the dispatchable list is searched to find an unlocked task in delay or ready status with the preempt flag on in its STE. If one is found with a lower priority than the rejected task, time slice end is forced, SYSTSILK is unlocked, interrupts are enabled, and exit is to the Queue Scanner. If no preemptable task is found, control passes to Step 4.

Step 3

Exit is to the Dispatcher with interrupts disabled and SYSTSILK locked.

Step 4

This portion of the Internal Scheduler is entered when the Entrance Criteria subroutine determines that the TSI submitted to it for evaluation can be added to the dispatchable list. First, parameters in the affected entries in the eligible list are adjusted; and, if necessary so are the system pointers (SYSFW and SYSLT). The TSI is then added to the dispatchable list ahead of all execute-bound tasks and behind tasks in page wait by modifying dispatchable list entry pointers and the system pointer (SYSPC). The count of eligible tasks (SYSELG) is increased by one.

The TSI just added to the dispatchable list is then modified by setting the TSI quantum count (TSIQCT) to the value specified in the schedule table for this TSI's STE level. If the scheduled start time (SST) is not zero, indicating that the task is behind schedule, the master clock value is subtracted from the SST and the result stored in TSISST. This negative value will be used by Rescheduling to compute the task's new SST and bring it back on schedule. If the SST is zero it is not modified.

A check is then made to see if the first XTSI page is in main storage. If it is, processing in the Internal Scheduler continues at Step 2.

If the first XTSI page is not in main storage, Supervisor Core Allocation is called for GQE/PCB space. The GQE and PCB are initialized and Enqueue GQE is called to bring the first XTSI into core. Interrupts are then enabled, TSILOCK unlocked and exit is to the Queue Scanner.

The Dispatcher (CEAKD) Chart BT

This routine selects a task to be given CPU control when no work has been created for the Queue Scanner by the Internal Scheduler.

Entry: CEAKD1 - with SYSTSILK locked and interrupts disabled.

Modules Called: Task Interrupt Control (CEAA2 entered at CEAA20) to allow the task being put in execution to receive interrupts.

Exits: Exit is to the task by loading the task's PSW.

If the Dispatcher finds that the system's elapsed time equals or exceeds the time interval value specified in the first entry of the real-time-interval-pending queue, exit is to the Create Real Time Interrupt subroutine with SYSTSILK unlocked, SYSTMILK locked and interrupts enabled.

The Dispatcher can also enter a wait state (by loading a PSW) when there are no tasks that can be dispatched. This, in effect, is an exit to the Queue Scanner.

Operation: On entry, the Dispatcher checks for real time interrupts due. If there is one, the Dispatcher exits to Create Real Time Interrupt.

If none is due, the dispatchable list is scanned from SYSPEC looking for an unlocked task in ready status with no paging requests pending (TSICP=0). If found, the user timer (XTSUTI) is compared to the current timer (XTSCTI). If the user timer is the lesser, the task's accumulated time is updated, and the current timer value set to the user timer value. Otherwise, the current timer remains unchanged. The task is then set in execution status and unlocked. SYSTSILK is reset and Task Interrupt Control is called to process any task interrupts. On return, the system elapsed time is updated, the task's registers are loaded

from the XTSI and control is given to the task by loading the task's current PSW.

If the Dispatcher cannot find a task to put in execution, the CPU will be put in the wait state. The timer value for the wait is determined by calculating the difference between the present time and the first entry in the real time interrupt pending queue (SYSRT2). If this value is greater than the idle timer setting (SYSIDL), or if the real time interrupt pending queue is empty, SYSIDL is used. System elapsed time is updated, SYSTSILK is reset, and the CPU is put in the wait state via LPSW.

Task Interrupt Control Subroutine (CEAA2) Chart BU

This subroutine checks for pending interruptions to a ready task prior to giving the task CPU control. If no interruption is found pending, TIC returns to the dispatcher. If an interruption is found pending, it is serviced and control is returned to the Dispatcher.

Entry: CEAA20

Modules Called: Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) provides space for the GQE and the PCB, when the ISA must be brought into main storage.

Supervisor Core Release subroutine (CEAL1 entered at CEAL02) releases the space occupied by the IORCB at the time a solicited I/O interrupt was recognized by the MCB for a VSEND, and releases the interrupt GQE space.

Enqueue GQE subroutine (CEAJQ entered at CEAJEN) queues the GQE on the User Core Allocation Queue Processor's queue.

Exits: The Task Interrupt Control subroutine exits to the Dispatcher. If an asynchronous interruption is pending in VSS mode, exit is to the resident support system (RSS) via LPSW.

When the ISA or first page table is not in main storage, exit is to CEAMQ, the Paging subroutine, at CEAMQA.

Operation: The Task Interrupt Control (TIC) subroutine provides the software required to relay a selected set of hardware interruptions, received by the resident supervisor, to a predefined task interruption processor. TIC is a re-entrant, resident, closed subroutine operating in the privileged state with I/O interruptions masked.

TIC provides user tasks with a software-interruption mechanism similar in many

aspects to the hardware interruption system. The major difference between the two types of interruptions is that a hardware interruption conveys a change in status of the entire system to the supervisor, while a software interruption conveys a change in status of only that portion of the system currently allotted to a given task.

There are eight types of hardware interruptions that can have a direct influence on the processing flow of a task. These, in the order in which they are processed by TIC, are:

1. VSS Activate interruption resulting when a task system programmer (TSP) connects to a task and activates the virtual support system (VSS).
2. Program interruptions resulting from task program and supervisor-detected task errors.
3. Recoverable data set paging interruptions (a special set of program interruptions detected by the I/O routines and processed by recovery modules in virtual memory).
4. SVC interruptions resulting from an SVC interruption originated by the erroneous execution of an SVC instruction by a task.
5. External interruptions resulting from inter-task communication.
6. Asynchronous I/O interruptions resulting from asynchronous signals usually associated with terminal devices.
7. Timer interruptions resulting from the expiration of a task-specified time interval.
8. I/O interruptions resulting from the termination of a task-oriented I/O operation.

The basic software interruption mechanism consists of an interruption queue in the task's TSI; an interrupt storage area (ISA) in the task's virtual storage which is defined as segment 0, page 0; eight interruption pending bits in the TSI, one for each type of interruption; and the Task Interrupt Control subroutine.

The logical sequence which terminates in a task interruption commences when a queue processor determines that an interruption condition must be brought to the attention of the task. At this point a general queue entry (GQE) pertinent to the condition is referenced by a pointer in the appropriate queue processor's queue. The Queue GQE on TSI subroutine is called to attach the GQE

pointer to the TSI's task-interruption queue, update the queue pointers and count, and set the queue's interruption-pending bit to one.

The interruption queue contains a chain of zeroes or GQEs. New GQEs are added to the chain by priority identical to that of the interruption type descriptions. Each GQE pointed to by the task-interruption-queue entries contain the following information:

- Interrupt code.
- Instruction length code (ILC).
- Sense data for I/O interruptions with unit check or unit exception only.
- Channel status word (CSW) for I/O or asynchronous interruptions only.

The interrupt storage area (ISA) contains preassigned areas for old-virtual-program-status-words (OLD VPSWs), several register save areas, and NEW VPSWs.

At task selection time, the Dispatcher loads the TSI and the XTSI pointers into general registers, locks the TSI and transfers control to TIC.

On entry at CEAA20, TIC tests to see if the ISA page is in main storage. If it is not, Task Interrupt Control exits to the Paging subroutine (CEAMQ at CEAMQA) to read it (and its page tables, if necessary) into main storage. TIC matches the flags against the interrupt mask in the task's TSI. The VSS activate, program, and SVC interruption mask flags are always on (these interruptions can never be masked out). This is a left-to-right match; therefore, the priority for concurrently unmasked pending interruptions is determined by the order of the interruption flags (VSS activate, program, SVC, external, asynchronous, timer, I/O). For those cases where the software interruption pending flags are a complement of the interruption mask (all pending software interruptions are masked), TIC returns to the Dispatcher.

If there is an unmasked software interruption pending, TIC determines whether the VSS flag is on in the TSI. If yes, the GQEs to be serviced will come from the VSS queue instead of from the normal queue of task interruptions (the 'regular' TSI queue, instead of the 'alternate'). A check is then made to determine if it is an SVC or external interruption. If it is and a message control block (MCB) exists associated with it, the MCB size is checked. A size of zero is treated as 64 and anything over the maximum is treated as the maximum,

or 1920 bytes. The size is then saved for later reference. If the pending interruption is not an SVC or external interruption, TIC checks to see whether it is an I/O interruption. If it is and an IORCB exists associated with it, the IORCB size is checked and treated in the same manner as described above for the MCB. If the interruption is neither external nor I/O, or if an MCB or IORCB does not exist as tested above, or if the size check is complete, processing continues.

The ISA page flag is set on, and the ISA lock byte is tested. If it is nonzero, all interruptions are masked off except program and SVC. If neither of these interruptions are pending, the page-hold flag is turned off and TIC exits to the dispatcher. If there are interruptions pending, or the ISA lock byte is equal to zero, the XTSI's PSW data for the interrupt is saved in the correct interruption-old-VPSW location of segment 0, page 0; the current VPSW is set equal to the old VPSW; the interruption mask from the new VPSW is moved to the TSI, and the XTSI's PSW is set to the new VPSW. The sense data and channel status word (CSW) are transferred from the GQE to segment 0, page 0; the TSI is updated, and a flag setting in the GQE is checked to determine if the GQE is to be returned in the IORCB. This is done, if indicated, (flag off) or else the processed GQE's storage space is returned by calling the Supervisor Core Release subroutine. At this time, the IORCB or MCB, if one exists, is moved into the interrupt storage area and the space it occupied is returned by calling the Supervisor Core Release subroutine. TIC then returns to the dispatcher after turning off the page-hold flag in the task's XTSI.

Entrance Criteria Subroutine (CEAKE) Chart BV

This subroutine determines if the task submitted to it by the Internal Scheduler meets the necessary conditions to allow it to be moved from the eligible list to the dispatchable list, and passes notice of the task's acceptance or rejection back to the Internal Scheduler.

Assumption: SYSFW will be zero if no TSIs are in the eligible list.

Entry: CEAKEA

R10 - address of TSI

R14 - return address

Exit: To Internal Scheduler

R15 - Code 0 = task accepted
Code 1 = task rejected.

Operation: On entry, this subroutine first checks the page pending count of the task whose TSI is passed to it in register 10. If the page pending count is not zero, the task cannot be added to the dispatchable list. A code of one is loaded into register 15 and immediate return is made to the Internal Scheduler.

If the page pending count is zero, a test is made to see if the system is in a low core state (SYSLCM=1). If it is, the task will not be accepted -- unless there are fewer than the minimum number of tasks in the dispatchable list.

If the system is not in a low core state, the number of pages used by the task during its previous time slice (TSIPTS), plus a buffer value, is compared to the count of estimated main storage blocks available (SYSECB). If TSIPTS is the lesser, indicating sufficient main storage available, register 15 is loaded with a code of zero to indicate that the task is acceptable. Return is then made to the Internal Scheduler.

If there is not enough main storage, the number of tasks currently on the dispatchable and eligible lists is determined. If the count is less than the minimum permitted, the task is accepted anyway. If it is at the limit, the task is rejected.

Whenever a task meets the conditions for admission to the dispatchable list, the count of estimated main storage blocks available is decremented by the task's count of pages used last time slice, and the time slice end flag (TSITSE) is turned off before returning to the Internal Scheduler via register 14.

Rescheduling Subroutine (CEAKZ) Chart BW

This subroutine is called when a task reaches time slice end (forced or normal) to change the task's STE level, if necessary, and to determine whether the task is to be placed on the eligible list, the inactive list, left in the dispatchable list, or deleted from all of the scheduling lists. This subroutine also moves tasks from the inactive list to the eligible list. Whenever a task is moved from one scheduling list to another (or if it remains in the dispatchable list), rescheduling recomputes the task's schedule start time (SST). System table fields are also maintained to reflect the movement of a task from one list to another. SYSELG is incremented each time a task enters the eligible list; SYSINA is lowered when a task is moved from the inactive list and raised when a task enters the inactive list.

Entry: CEAKZ1

R1 - address of TSI

R4 -

Code 1 - move TSI from inactive to eligible list. eligible list.

Code 2 - call from Timer Interrupt Queue Processor.

Code 8 - put TSI on the inactive list

Code 16 - remove TSI from the dispatchable list (LOGOFF).

Exit: To caller.

R15 -

Code 0 - task remains in the dispatchable list.

Code 1 - task put in eligible list.

Code 2 - task put in inactive list.

If register 4, on entry, does not contain a code of 1, 2, 8, or 16, exit is taken to the system error processor (SYSERR).

Operation: On entry, a save area is obtained and general registers are saved. The entry parameter is then checked for validity. If invalid, exit is to SYSERR.

If the parameter in register 4 is a 1, the task is checked to see if it is in TWAIT, then AWAIT. If in TWAIT, the task is assigned the STE specified by the STETWAIT level in its current STE. If AWAIT, the STEAWAIT level is used. The appropriate level is put in the task's TSI (TSISTE).

If the task is in neither TWAIT nor AWAIT, the TSEND level (STETSEND) is used.

The task's scheduled start time (SST) is then computed. If the delta to run (DTR) value in the STE is zero, the SST is set to zero. If DTR is positive, the new SST is set to equal: $DTR + MC + (1 - R)SST$. If DTR is negative, STEDELTA is subtracted from the master clock to calculate the new SST.

The task is then placed on the eligible list according to its internal priority level and SST. The 'no work for internal scheduler' flag (SYSNWK) in the system table is set to zero. The 'lowest ahead of schedule SST' field (SYSLSST) is reset to zero, and Rescheduling sets a return code of 1 in general register 15 and exits.

If general register 4 contains a code of 2 on entry, the call is from the Timer Interrupt Processor and is not the result of a TSEND SVC. If the task is in delay status, it is placed at the head of the inactive list, and the return code set to 2 in general register 15.

If the task is not in delay status, a subroutine within Rescheduling is called to check the following:

- Was the task holding an interlock and forced to TSE because of low main storage?
- Was the task holding an interlock (not low main storage)?
- Was the task waiting on an interlock?

If any of these situations are verified, as checked in the order given, the task is assigned the appropriate STE level. Otherwise the task is checked for forced time slice end due to maximum page reads exceeded. If this condition exists, the STE specified in STEMPRE is assigned to the task.

The task's new SST is then computed to be equal to MC+DTR unless the old SST was negative. If it was negative, then the new SST becomes MC+DTR+the old SST.

The task is then put on the eligible list in its proper place.

If the code in register 4 is an 8, the SST is set to zero, the task is placed at the head of the inactive list, and the return code in register 15 set to 2.

If the code in register 4 is a 16, the task is removed from the dispatchable list and not added to any of the scheduling lists. No return code is set, and exit is to the calling routine.

MAJOR ERROR RECOVERY PROCEDURES

An important function of the TSS/360 resident supervisor is to process hardware interruptions that affect total system operation, and software interruptions that occur in the supervisor state. The processing capability for performing this function is provided by three groups of procedures:

- The recovery nucleus and two related procedures: the Reconfiguration routine; and the External Machine Check Interrupt Processor. The Recovery Nucleus routine aids in a nondisruptive degradation of the system in the event of a CPU or storage malfunction, thus

attempting to prevent the system from coming to a halt or going into an unending loop because of a single-machine-error situation. The Reconfiguration routine provides recovery capability after an internal machine-check interruption. The External Machine Check Interrupt Processor performs the function of making a hardware-detected inboard error involving a channel control unit transparent to the system.

- The System Environment Recording and Retry (SERR) programs, which provide error recording capabilities, and minimize the effect of hardware malfunctions on system performance.
- The System Error Processor, which provides the capability for handling conditions caused by system software errors and hardware errors detected by the supervisor components and privileged service routines.

These procedures are discussed individually on the following pages.

Recovery Nucleus-67 (CEAIR) Chart BX

The Recovery Nucleus routine aids in a nondisruptive degradation of the system in the event of a CPU or storage element malfunction. The primary function of the Recovery Nucleus is to prevent the system from coming to a halt or going into an unending loop because of a single-machine-error situation.

Attributes: The Recovery Nucleus is privileged, read only and reentrant. It resides permanently in main storage.

Entries:

- CEAIR1 - for duplex mode operation, via an external interruption. All maskable interruptions, except machine check, are disabled by means of the external interrupt new PSW.
- CEAIR3 - for simplex mode operation. All maskable interruptions are disabled by means of the machine check new PSW.
- CEAIR2 - by System Error Processor, when a program-detected error occurs in a CPU or storage element, or when a major software error occurs in a TSS/360 program.

RESTRICTIONS: No CPU in the TSS/360 domain operates with its prefix set to zero or deactivated in a multiprocessing configuration. This enables every CPU to have access to any other CPU's prefixed storage

area. Each CPU has its own copy of the error recovery control table (ERC) in the PSA. This table contains a work area for the Recovery Nucleus. Thus unique work areas are provided for each CPU. This is critical to error recovery in a multi-processing environment.

Assumptions: The recovery strategy is based on the assumption that only a single machine error can occur at any one time. For instance, a machine check is due to either a storage failure or a CPU failure, but never both, and a second machine error does not occur until the first one is completely processed. However, this recovery strategy can handle double indications from the same error.

A 5120-byte block of main storage is reserved for the SERR operating area and remains unassigned at all times. This provides operating space for the SERR modules (overlaid) and space to save the system environment.

Modules Called: SERR Bootstrap program (CMASA) loads the modules needed to record errors, analyze retry possibilities, and load the Reconfiguration routine (CGCMA).

External Machine Check Interrupt processor (CEABE) records information pertaining to an external machine check interruption, identifies its source, and sets up for Instruction Retry or Reconfiguration.

Inter-CPU Communication subroutine (CEAIC) restarts a specified CPU from the controlling CPU. It also restarts the controlling CPU's clock and updates the elapsed time when called at CEAIC2.

Reconfiguration routine (CGCMA) removes a malfunctioning unit from the system and continues or restarts the remaining system.

Exit: Successful completion of recovery results in an exit to the interrupted program at the point of interruption. An exit may be taken to the Reconfiguration routine when a unit must be removed from the system.

Abnormal conditions result in a TSS/360 abort.

Operation: The Recovery Nucleus is a body of code which is universally valid for all TSS/360 configurations up to duplex. In the simplex mode, recovery and retry are attempted by the failing CPU. In duplex mode, recovery and retry analysis are attempted by a CPU other than the failing CPU. When a CPU executing recovery is subsequently interrupted by a hardware failure, recovery is attempted by the CPU which was originally interrupted. In the event

that two CPUs receive simultaneous interruptions, the higher priority CPU executes recovery for the lower priority CPU and, if successful, signals the lower priority CPU to execute recovery for it.

This module is valid for any initial or subsequent configuration (perhaps a configuration reduced in complexity as a result of previous machine checks and reconfigurations).

Entry to the Recovery Nucleus is made by means of the machine check new PSW. In simplex mode, this interface is direct (that is, the address portion of the machine check new PSW transfers control directly to the Recovery Nucleus at CEAIR3). In duplex mode the interface is indirect. The machine check new PSW, when loaded, places the affected CPU in the wait state. A malfunction alert (a hardware generated machine check-out signal) is sent to a second CPU, which receives it as an external interruption. The loading of the external new PSW results in a transfer of control to the Recovery Nucleus at CEAIR1.

Note: When reconfiguration degrades a system from duplex to simplex mode, the Reconfiguration routine replaces the duplex mode, machine check new PSW with the simplex version.

The Recovery Nucleus is divided into three major sections which service hardware malfunctions in the simplex mode, hardware malfunctions in the duplex mode, and SYSERR calls resulting from major software or program detected hardware errors. Each of these major sections can be logically divided into two phases. The first phase is concerned with error classification, while the second phase handles the actual processing of analysis, recovery, and retry.

Simplex Mode: -- In simplex mode, three types of errors may occur, all of which result in an entry to the recovery nucleus at CEAIR3. These errors are a simple machine check, an external machine check, and a dual-accept failure. This latter error involves a malfunction in a storage switching unit, and is processed as a simple machine check.

On entry at CEAIR3, the registers are saved and the timer is reset and saved. The saved timer is adjusted to prevent a timer interruption and the CPU's active prefix is located. When the prefix is located, it is saved in an area known as "TEMP." An area is then cleared and a damage report is set up in it. The SERR save area is located and the logout data is moved to it. If the prefix cannot be found, a message is printed at the console,

an audio alarm is sounded, and the system is placed in a manual state loop. In short, TSS/360 is aborted.

If the error is an external machine check, the External Machine Check Interrupt Processor is called and, upon return, a call code of X'29' is placed in the parameter register. If the error is a simple machine check or a dual-accept failure, a call code of X'01' is placed in the parameter register and the CPU's ID is placed in another parameter register.

An operating area and a register save area are provided for the SERR Bootstrap program, which is then called and passed the above parameters. If SERR Bootstrap encounters errors in the arrangement of SERR modules on the drum, the Recovery Nucleus treats it as a "retry not possible" condition, discussed below. Any paging errors encountered by SERR Bootstrap, will result in a TSS/360 abort as described above.

- **Retry Possible** -- When SERR indicates that retry is possible, the machine check new PSW and the logout areas are cleared, the timer is reset, all registers are restored, the test bytes are cleared and execution continues by loading the PSW returned by SERR in "DMPMOP."
- **Retry Not Possible** -- If SERR indicates that retry is not possible, the machine check old PSW, the logout area, and the test bytes are cleared. A work area and a register save area are provided and another call is made to SERR Bootstrap passing it a code of X'80'. If SERR Bootstrap encounters a paging error, TSS/360 will be aborted as described above. Otherwise, the reconfiguration routine will be called for corrective action as indicated.

Duplex Mode: -- In duplex mode, four types of errors, in addition to those discussed above, may occur. These four errors are unique to a multi-CPU environment and the TSS/360 Recovery Nucleus is designed to cope with these errors in any system comprising two CPUs. These errors are:

- **Machine check in recovery execution.** CPU2, while executing recovery for CPU1, receives a machine check interruption. (The error is assumed to be common to the error first found in CPU1). Control is passed back to CPU1, which begins execution of recovery for itself. If retry is possible CPU1 is retried and CPU2 is restarted from the point at which it received the malfunction alert from CPU1.

- **Dual-accept failure in two CPUs.** CPU1 and CPU2 receive machine check interruptions as a result of malfunctioning storage switching units. The higher priority CPU is determined and is given control. If it successfully executes recovery, the lower priority CPU is given control and it begins execution of recovery.
- **Double machine check.** CPU1 and CPU2 receive simultaneous machine check interruptions in the same cycle of instruction execution. Recovery is executed on a priority basis as described in dual accept failure above.
- **Machine check during execution of recovery from a double machine check.** CPU1 and CPU2 receive machine check interruptions as described in double machine check above. While in execution of recovery and prior to determining which is the higher priority CPU, CPU1 receives another machine check. CPU2 gains control and begins execution of recovery nucleus again; no retry will be attempted and control will pass to the Reconfiguration routine for system restart after SERR has operated.

Note: Instruction retry is not possible in this case since the second interruption will have destroyed the machine check old PSW.

Initial Entry to the Recovery Nucleus: The occurrence of a machine check interruption in the duplex mode results in the affected CPU being placed in the wait state and in a hardware generated malfunction alert being sent to another CPU. This malfunction alert, which is an external interruption, causes the receiving CPU to enter recovery nucleus at CEAIR1.

Since all external interruptions follow this path, the recovery nucleus sorts out normal interruptions. If an external interruption, other than a malfunction alert, is received by a CPU in which no previous machine check has occurred, the timer is reset and saved and either the Subject Intercom routine or the External Interrupt Stacker is given control for normal processing. If a machine check has occurred in this CPU, the external interruption is effectively ignored by reloading the machine check new PSW, which returns the CPU to the wait state.

If the interruption is the result of a malfunction alert, the timer is reset and saved, the registers are saved, and the test bytes are cleared. If this is the third malfunction alert in this CPU, TSS/360 is aborted. This is done to prevent a solid failure from resulting in an endless

series of machine check and external interruptions.

The first malfunction alert, received by a CPU, indicates a simple machine check or an external machine check in another CPU and the saved timer is adjusted so there will be no timer interruption while recovery is being attempted.

If a second malfunction alert is received and no machine check interruption has occurred in the receiving CPU, TSS/360 is aborted. If a machine check has previously occurred in the receiving CPU, a machine check during execution of recovery from a double machine check has occurred and a flag is set to indicate this.

In both of the above cases, the external old PSW is saved, the address portion of the LPSW instruction is adjusted to reflect the save address, and the CPU status table (CST) entry for the failing CPU is located.

If the CST entry is not found, TSS/360 is aborted.

If the CST entry is found, the prefix of the active CPU is located, the failing CPU's prefix and the active prefix are saved.

When no previous machine check has occurred in the CPU receiving the malfunction alert, (indicating that the error is a simple machine check or a simple external machine check) machine checks are disabled, the timer, in the failing CPU, is reset and saved, a damage report is set up, and the logout data is moved to the SERR save area. The machine check new PSW is saved and, after issuing the DIAGNOSE instruction to clear pending machine check interruptions, it is restored. Processing then continues as described under "Error Type and Retry Analysis" below.

If, after locating the CST, the CPU determines that a previous machine check has occurred, it enables external interruptions and disables machine check interruptions. The other CPU's timer is reset and saved, if necessary, and an indicator is set. A delay then occurs to allow the other CPU to set this CPU's indicator. These bytes, when set in both CPUs, indicate that both CPUs are simultaneously executing recovery for each other or, if not set in both CPUs, serve to identify machine check errors encountered in execution of recovery from single and double machine checks.

Following the delay, this CPU tests its indicator. If it is not on the other CPU's indicator is cleared; if it is on, this CPU sets its own "PING" byte to X'02' and

clears its indicator. In both cases the active CPU temporarily marks itself as the failing CPU.

If the "PING" byte is X'02', recovery from a double machine check is indicated and the priority of CPUs is determined. The lower priority CPU sets its priority byte, saves its saved timer, resets its saved timer, and enters the wait state by loading its machine check new PSW. The higher priority CPU continues to execute recovery and, upon completion, will restart the lower priority CPU and the lower priority CPU continues recovery execution.

Machine check and external interruptions are enabled, the damage report is prepared, and the logout data is moved to the SERR save area. Processing then continues as follows:

External machine check interruptions are turned over to the External Machine Check Interrupt Processor and, upon return, are assigned a call code of X'29'. Dual-accept errors and errors occurring in execution of recovery are given a call code of X'09' and the ID of the active CPU is placed in a parameter register. Simple machine check interruptions are assigned a call code of X'01' and the ID of the failing CPU is placed in the parameter register. After setting up work and save areas, the SERR Bootstrap program is called and is passed the above parameters. SERR modules out of order on the drum and paging errors are handled as in previous calls; otherwise, SERR returns an indication of retry possible or not possible.

If retry is not possible, double machine check errors and errors occurring in execution of recovery are given codes of X'80' (this will force Reconfiguration to call for a system restart) and the prefix of the failing CPU is retrieved. For all errors which cannot be retried, the machine check old PSW and the logout data of both CPUs are cleared, call code of X'80' is placed in the parameter register and the test bytes are cleared. Work and save areas are setup and SERR Bootstrap is called. Paging errors encountered by SERR result in an abort; otherwise, the Reconfiguration routine is called to perform degradation as discussed under "Retry Not Possible" for simplex mode.

If retry is possible, and the error was a machine check in execution of recovery from other than a double machine check, the prefix of CPU2 is retrieved, CPU2's external old PSW is saved. CPU1 then saves its own retry machine check old PSWs and the logout areas are cleared in both CPUs. The clock is adjusted in CPU2, to preclude a timer interruption, and is then restored.

The ETM is calculated and the clock is resaved. CPU1 then sets up for and calls the Intercom routine to start CPU2 at a point at which it will restore its timer and its registers, clear its test bytes, and resume execution at the point of the original interruption by loading a PSW. Upon return from Intercom, CPU1 resets and restores its own timer, restores its registers, clears its test bytes, and attempts to retry the failing instruction by loading a PSW.

If a retry possible indication is returned from SERR but the error is a machine check during execution of recovery from a double machine check, it is treated as a retry not possible condition discussed above.

If retry is possible and the error did not occur during execution of recovery, the machine check old PSW and the logout areas are cleared in both CPUs and the prefix of the failing CPU is retrieved. If the error was a double machine check and this is the priority CPU, the retry machine check old PSW, returned by SERR, is moved and the original changed to continue recovery.

If a double machine check has occurred and this is not the priority CPU, the machine check old PSW is saved, and the original set to continue the other CPU recovery. The saved clock is moved and the "PING" byte is reset. At this point, or if the error was not a double machine check, the other CPU's clock is adjusted to prevent a timer interruption and it is restored. The ETM is computed and the timer is resaved.

The Intercom routine is called to restart the other CPU and to restore its clock. If this is the priority CPU, it places itself in the wait state by loading its machine check new PSW. When the lower priority CPU restarts this CPU, this CPU will restore its own registers, clear its test bytes, and resume execution, at the retry point by loading a PSW which now contains the retry machine check old PSW returned by SERR. If this is the lower priority CPU issuing the restart, or if priority was not a relevant factor in processing, the CPU resets and restores its timer, restores its registers and clear its test bytes. It then resumes execution, at the point of the original interruption or retry by loading a PSW.

SYERR CALLS: The Recovery Nucleus is entered at CEAIR2 when SYERR detects a major software error or on a program detected hardware error. The registers are saved, the test bytes are cleared, and the active prefix is located and saved. If the error is a major software error, a code of

X'80' is placed in a parameter register and the Recovery Nucleus exits to the Reconfiguration routine for system restart.

If the error is a hardware malfunction detected by the software, a code of X'41' is placed in a parameter register and this CPU's ID is placed in a second parameter register. SERR Bootstrap is called and, upon return, the code is tested. If the code is zero, the registers are restored and control is returned to SYSERR. If the code is nonzero, a value of X'80' is set, and the Recovery Nucleus exits to the Reconfiguration routine for system restart.

Reconfiguration Routine (CGCMA)

The reconfiguration routine performs recovery functions after an internal machine check. It results in either, (1) notification to a user that his task has been affected, (2) a system restart on a degraded system, or (3) a system interruption due to insufficient hardware.

Attributes: The Reconfiguration routine is privileged, resident, nonreenterable and operates with all interruptions, except machine check disabled. Time sharing operations are suspended during execution of the Reconfiguration routine.

Assumptions: Only one machine check may occur at any one time. The SERR Bootstrap routine will be capable of reading in the Reconfiguration routine. The maximum system configuration is duplex.

Entry: CGCMAC - by Recovery Nucleus via SERR Bootstrap.

Modules Called: Supervisor Core Allocation subroutine (CEAL1 entered at CEAL01) reserves main storage for a task program interruption GQE.

Queue GQE on TSI subroutine (CEAAF) places a pointer to the specified GQE on the affected task's TSI interruption queue.

Exit: The Reconfiguration routine exits to one of the following:

- The point of interruption by loading the external interrupt old PSW and pointing other CPU to the Dispatcher.
- Startup, if an automatic restart is necessary.
- The wait state, if insufficient hardware is available to permit system operation.

Operation: On entry to Reconfiguration, the failure classification code in the damage report is examined and a message is

sent to the operator specifying the nature of the malfunction. Since time sharing is not being performed during the execution of Reconfiguration, the facilities of the command language system are not available. Reconfiguration contains the code necessary to perform all input/output functions required, such as communication with the operator.

Prior to calling Reconfiguration, SERR has diagnosed some failures as requiring a system restart. These failures have classification codes beginning with X'80'. When a system restart is necessary, Reconfiguration stores, in the PSA of the operating CPU, the word RESTART followed by the damage report and the latest effective settings of the configuration console switches. Reconfiguration then reads the startup prelude and transfers control to it.

If the damage report indicates a storage element has more than 3 solid failing pages the S.E. is marked out of the effective configuration console switches and a system restart is performed as described above. It is assumed that the loss of an entire storage element implies the loss of supervisor information.

If the DAMAGE REPORT indicates more than 3 failing pages or a page of supervisor information, a system restart is performed as described above.

If the damage report indicates fewer than 4 solid failing nonsupervisor pages they are marked "not operational", "unavailable", and "reserved".

If the damage report indicates fewer than 4 intermittent failing nonsupervisor pages, the tasks to which they belong are assigned a program interruption as described below. The other active CPU (if any) is sent to the Dispatcher and the controlling CPU continues operation at the point of original M.A. However, if the system is an effective simplex it will go to the Dispatcher itself.

If a CPU has failed, the machine-check-old PSW is examined to determine in which state the failure took place. If it occurred in the supervisor state, Reconfiguration initiates an automatic restart.

If the only CPU failed, it will drop into the wait state.

If the CPU failure occurred in the problem state, the Reconfiguration routine locates the task status index of the affected task. This is accomplished by using the TSI pointer from the prefix area of the CPU. The Supervisor Core Allocation

subroutine is then called to reserve storage for a GQE. When the storage has been reserved, Reconfiguration constructs a program interruption GQE and places the TSI pointer and a program interruption code in it. This interruption code will subsequently be interpreted by the task program interrupt processor, operating in virtual storage, as a notification that a hardware failure occurred during the execution of the user's task. The Queue GQE on TSI subroutine is then called to place a pointer to the GQE on the task's TSI-program-interruption queue. The TSI lock byte is set to zero, to permit the interruption to be processed, the ready bit is set on and the in-execution bit is set off. When this has been accomplished, Reconfiguration updates the CPU status tables and system configuration console switch words (SYSCCS) to indicate that the CPU is no longer in use. The machine-check-new PSW is altered to reflect the correct entry point for simplex mode. Time sharing operations are resumed in the current CPU by restoring general registers and loading the external-interrupt-old PSW.

If it is determined that the machine check was caused by a page of main storage belonging to the supervisor, automatic restart is initiated. Should a solid failing page in an effective 1/2 duplex contain the active PSA, the prefix is switched and startup is read into the new PSA. Therefore when possible restart is performed from a good prefix area, if for any reason the alternate PSA is not available (for example, if it is partitioned out), the CPU is treated as though it were failing.

If the damage report specifies a failing CCU the corresponding items are marked out of the configuration console switch words that are placed in the restart drop area and startup is read into the system and receives control.

External Machine Check Interrupt Processor (CEABE) Chart BY

The External Machine Check Interrupt Processor is a routine called by the Recovery Nucleus (CEAIR) when it has detected an external machine check interrupt. The processor records the circumstances of the error in the system environment recording and retry table (CHAERE) for later use by the Recovery Nucleus. It then attempts to make the error transparent to the system by resetting error-old-PSWs to retry the I/O instruction. EMCIP also maintains failure count information on failing channels and CCUs so that a failing CCU can be identified and reconfigured out of the system.

Entry: CEABEM.

Modules Called: System Error Processor (CEAIS) given control when an interruption indicating a system error condition occurs.

Exit: To Recovery Nucleus.

Operation: If the entry is recursive, the routine will return to the calling module after setting up the recursive entry return parameters. Otherwise, the routine initializes itself by saving the input registers and establishing the base registers.

The routine then fills into the system environment recording and retry data recording table information common to all types of EMCIs. This includes:

- Inserting table length.
- Inserting S/360 model number.
- Inserting the call type code.
- Setting the 'record entry complete' flag.
- Resetting the 'retry successful' flag.
- Inserting the ID of the offending CPU.
- Inserting the address of the primary or secondary PSA, as appropriate.
- Recording the PSW.
- Inserting the PSAMOP interrupt code.
- Recording the ID of the current user.
- Inserting the time.

EMCIP then determines the error type by checking the EMCI code. The following error types are recognized:

1. Multiple CPU Recognition
2. Multiple Interrupt Recognition
3. Multiple CCU
4. Channel Address-Bus-Out
5. Channel Address-Bus-Out and Unit Address-Bus-Out
6. Unit Address Bus-Out
7. Multiple Channel Recognition
8. Storage Interface Time Out
9. CSW Store
10. CPU Prefix ID - Parity Check

11. Multi-Storage Element Selection
12. Channel Interface Time Out
13. Storage Address Bus-Parity Check

Once the type of error is determined, EMCIP completes the necessary processing and fills in the rest of the table (CHAERE) accordingly. In each case (except CSW Store and CPU Pre-fix ID errors), the routine checks to see if the error occurred on an SIO instruction. If yes, flags are set for CSW present, channel logout data present, SIO failure, and actual device address present. The old PSW is changed to indicate a retry SIO. If no (this includes CSW store and CPU Pre-fix ID), the routine marks the instruction as not retryable.

EMCIP also maintains a failure count for each channel connected to a CCU. If the failure count for a channel reaches the limit, the instruction is marked as not retryable. If the failure count on all channels connected to a CCU reaches the limit, the CCU itself is then identified as the faulty unit. Exit to the Recovery Nucleus then includes information telling it to call Reconfiguration to eliminate the bad CCU from the system.

Before exiting to the Recovery Nucleus, EMCIP fills in the channel activity map which records the channel types, and flags a channel which was active at the time of the error.

SYSTEM ENVIRONMENT RECORDING AND RETRY PROGRAMS

The System Environment Recording and Retry (SERR) programs are provided to perform error recording and to minimize the effect of hardware malfunctions on system performance. The heart of SERR is a comprehensive error analysis capability which supports the following functions:

- Instruction retry: Whenever possible, the failing instruction will be retried under controlled conditions. If the retry attempt is successful, processing may continue with no loss in performance.
- Isolation of the failing element: If the retry cannot be attempted or is unsuccessful, information is made available to the resident supervisor and to the operator which facilitates graceful degradation and reconfiguration of the system. This information includes a description of the failure (solid or intermittent) and identification of the failing CPU or storage element.

- **Environment recording:** SERR maintains a continuous error history on the paging drum for subsequent use by the customer engineer. This data includes the complete hardware environment at time of failure. An edit and print program allows the customer engineer to retrieve this data selectively.

SERR processes information relating to the following types of malfunction:

- **Internal machine check (CPU/memory):** error recording, analysis (including C/M checkout), instruction retry and element isolation.
- **External machine check (failure detected by channel controller) and paging I/O errors:** error recording only.
- **System error:** error recording and C/M checkout.

SERR consists of ten basic programs, nine of which reside in modular form on the paging drum(s). These programs are:

SERR Bootstrap (SERRB): A small main storage resident executive routine which provides the interface between SERR and the Recovery Nucleus. It calls in the various SERR modules as required from the paging drum.

Environment Recording (ER): This program performs preservation recording on all failures. It collects and formats the error environment (logout, register contents, etc.) and other data, and writes an error record on the paging drum for subsequent edit and print.

Immediate Print (IP): This program provides for immediate console messages to the operator or customer engineer whenever required by SERR.

Checker (CHKER): This program saves the system environment (logout local store, etc.) and checks this data for parity and a valid log. At the conclusion of SERR operation, CHKER is again called to restore the environment.

Pointer (PNTR): This program determines the address of the instruction that was interrupted by a machine check.

Restore and Validate (R/V): This program indicates whether or not the retry threshold has been exceeded and restores general purpose registers if needed for retry.

Instruction Retry Execution (IRE): This program determines (from reports of the

other SERR modules) whether or not retry should be attempted, and combines the SERR reports into a damage report to recovery nucleus.

CPU/Memory Checkout 1 (CM1): This program provides a cursory check of the CPU hardware in each processor by performing an instruction test.

CPU/Memory Checkout 2 (CM2): This program checks the data paths from each CPU to each storage element and also scans all of storage, testing for bad parity.

CPU/Memory Checkout 3 (CM3): This program provides additional hardware testing by performing a check-sum of the ROS words and exercising local store.

The paging drums provide residence for the SERR routines and are also the media for environment recording. The paging drums are formatted into 4K byte records (pages), separated by unused spaces of approximately 246 bytes. These spaces are used as the error recording area by ER.

SERR is broken down into 4K modules. Startup moves the SERR modules from systems residence to the paging drum using a drum address available to the Recovery Nucleus.

SERR operates in either a multiprocessing or single-CPU environment. In a multiprocessing environment, SERR operates in a good CPU and storage element. Other CPU's, including the malfunctioning one, are placed in the WAIT state, enabled only for external interruptions. The malfunctioning CPU is activated occasionally (under controlled conditions via the write-direct instruction) to test itself or to store its own registers.

SERR is entered from the Recovery Nucleus which provides a 1 page overlay (SERR operating area) for the SERR routines and, for a multiprocessing system, sets up SERR in a good CPU and storage unit. The Recovery Nucleus will branch to SERR Bootstrap with a call-type code describing the type of error. After clearing a path to the paging drum and saving the hardware environment, SERR Bootstrap will call the first required SERR module from the paging drum into the SERR operating area. Flow through the various SERR modules, illustrated in Figure 26, is then controlled by the modules themselves. Most modules, upon their return to SERR Bootstrap, specify the next module to be called.

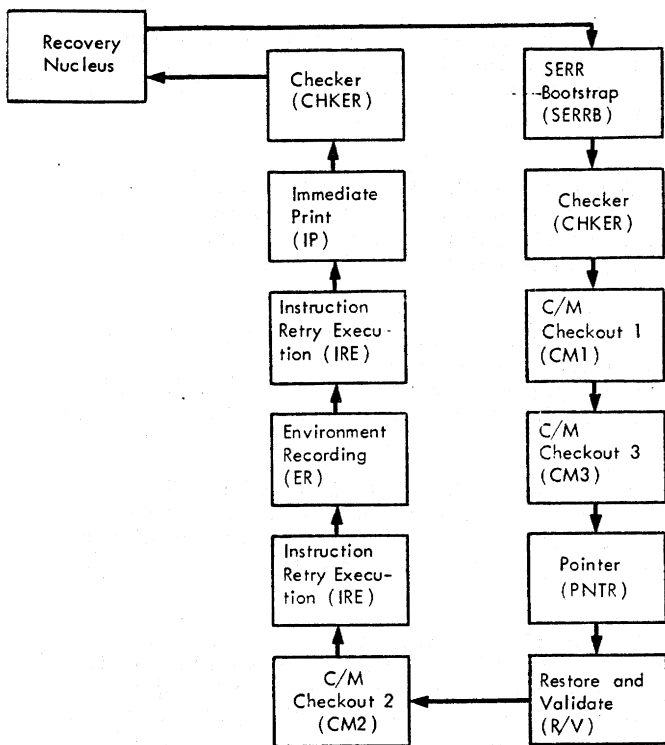


Figure 26. General flow through SERR after a machine-check interruption

The following is a simplified account of a typical SERR operation following a machine check interruption:

Upon receiving control from the Recovery Nucleus following a malfunction alert, SERRB calls CHKER from the paging device to save key portions of the system environment. CM1 and CM3 then check out each CPU, after which PNTR is called to analyze the logout of the failing CPU to determine the address of the instruction being executed at the time of the failure. R/V is called next, to find out if the instruction has passed a threshold beyond which retry cannot be performed. CM2 then scans each storage element to insure that all storage data is valid.

The next module called is IRE which combines the reports of the various modules into a damage report, and indicates in this report whether or not retry can be attempted. IRE is recalled to determine if the two sets of failure data indicate a solid failure.

Following the second IRE run, IP is called to send an error summary message to the operator, and CHKER is called to restore the system environment. SERRB then returns control to the Recovery Nucleus. If a retry is possible, the Recovery Nuc-

leus then causes the failing CPU to be restarted at the address determined by PNTR. If retry is not possible, the Recovery Nucleus must take whatever action is appropriate to recover from the error.

Various tables are used by the SERR programs, as follows:

SERR SAVE AREA: This table contains a record of the hardware environment at the time of failure, including contents of registers, prefix storage area and logout area of both the failing CPU and the controlling CPU. This table is set up by SERR Bootstrap.

SERR AUXILIARY QUEUE: Contains status information (from pending interruptions) on the devices used by SERR (the paging drum and operator's console). This table is updated by SERR Bootstrap.

DAMAGE REPORT: This table contains a summary of the results of the analysis including identification of failing element. It is updated by various SERR modules.

CPU STATUS TABLE: This table allows SERR to determine the environment in which it is operating.

SYSTEM TABLE: This table provides SERR with access to system information.

The SERR programs are described individually on the following pages.

SERR Bootstrap (CMASA) Chart BZ

The function of SERR bootstrap is to selectively call the SERR/reconfiguration modules into main storage from the paging drum(s) for the purpose of one or more of the following:

- Error recording.
- Instruction retry analysis.
- CPU/memory checkout.
- Reconfiguration.

Attributes: SERR Bootstrap is reentrant and read only. It resides permanently in main storage and runs with interruptions masked or controlled to prevent recursive entry.

Restrictions: There is only one copy of CMASA within the TSS/360 system. Each CPU must, however, have an error recovery control table in its PSA. This table contains the SERR common pool and SERR Bootstrap work area. The table provides CMASA with a unique work area for each processor in a

multi-processing environment. Each CPU's PSA must be available to any other CPU. Therefore, no CPU prefix can be set to zero.

Assumptions: In case of a CPU failure in a multiprocessing configuration, the Recovery Nucleus gives control to SERR Bootstrap in an operative CPU. In case of a storage element failure, the Recovery Nucleus designates the SERR/reconfiguration operating (overlay) area and SERR save area in an operative storage element, if available. Control unit/device reservation (if available for paging drum operations) will not be utilized.

Entries: CMAS1 - by the Recovery Nucleus (CEAIR) as a result of the following:

- Internal machine check
- External machine check
- System error
- External interruption machine checkout signal (malfunction alert)

This entry point is also used by Real Core Error Recording (CEAI7) to load the Environment Recording routine (CMASB). Paging I/O error records are recorded by Environment Recording for the Real Core Error Recording routine.

Modules Called: All other SERR programs, as required.

Exits: The SERR bootstrap exits to the recovery nucleus or the SERR/reconfiguration subroutine. For paging I/O error recording calls, control is returned to CEAI7.

Operation: On entry from the Recovery Nucleus, SERR Bootstrap assumes that the Recovery Nucleus has saved the controlling CPU's general registers, and the machine-check-logout data, if any, in the SERR save area.

SERR Bootstrap saves the controlling CPU's PSA-hardware area and then performs the following:

- Obtains an I/O path to the paging drum (or drums) via a drum-path-table pointer established in the system table.
- Clears the path by means of the test-I/O instruction, which causes a channel status word (CSW) to be stored (if an interruption was pending) thus freeing the path for subsequent use.

- If a unit-check indicator is present in the CSW, executes a sense command to obtain sense data.
- The CSW and sense data (if any) are placed in the auxiliary queue for later processing.

Once a path is cleared to the paging drum(s), SERR Bootstrap examines the Recovery Nucleus call type to determine the type of error and consequently which program module to load into main storage.

When the desired SERR or reconfiguration program is loaded, SERR Bootstrap turns control over to that program with general register 15 containing the object program's base address.

Upon entry from SERR programs, SERR Bootstrap either:

- Calls another SERR program if requested by the current SERR module.
- Restores the system environment to its original state at the time of entry from the Recovery Nucleus, and exits to the Recovery Nucleus.

SERR bootstrap maintains a common pool of address constants via the use of a DSECT (CHAERC), defining areas of interest to the SERR/reconfiguration program complex. These address constants are resolved at startup time and are positioned in the SERR Common Pool area in the error recovery control table (CHBER) beginning at hexadecimal location C00 in the PSA.

SERRB Symbolic Address

<u>Address</u>	<u>Address Constant Definitions</u>
ERCSYS	Pointer to System Table
ERCPLS	SERR page locations (BBCCHHR)
ERCPLR	Reconfiguration page locations (BBCCHHR)
ERCSAQ	Auxiliary Queue Pointer
ERCDPP	Paging Drum Adr. TBL. Pointer
ERCDPL	Paging Drum Adr. TBL. Length
ERCODP	Operator Dev. Path TBL. Pointer
ERCODL	Operator Dev. Path TBL. Length
ERCPDA	Drum Path
ERCODA	Operator Dev. Path

The following address pointers and/or constants are maintained by SERR Bootstrap for SERR program use only:

<u>SERRB Adr/Bytes</u>	<u>Definitions</u>
ERCSSA	Pointer to SERR save area
ERCSTR	SERRB Return Adr.
ERCSP1	Special Control Reg

ERCR14	RN Return Address
ERCR00	Call Type
ERCR01	SE/CPU ID's
ERCR02	Failing CPU Prefix
ERCR03	SERR Operating Area Addr.
ERCR04	CTL CPU GPR Save Addr.

Environment Recording Program (CMASB)

The Environment Recording program records machine check and paging I/O error records on the drum. These error records are placed in chronological order on the interpage gaps of the paging drum. This information can be retrieved at a later time by a customer engineer via VMEREP or stand-alone EREP which will format and print it. Thus, error incident information is accumulated for the customer engineer until he requests this information for analysis.

Attributes: The program is resident in auxiliary storage, nonreentrant, and operates in the privileged state.

Entry: CMASB1

RESTRICTIONS: Approximately 192,000 bytes are available for information recording in the unused dummy slots on a paging drum. When this area is filled, "drum-overflow" occurs and no more information will be recorded until the recorded information is printed.

Assumptions: A paging drum will be available upon which information may be recorded. The device address of the paging drum(s) will not be changed without notifying the SERR program.

Exit: To SERR Bootstrap.

Operation: On entry, the Environment Recording program collects data on error incidents and writes this information onto paging drum slots unused by the system. Approximately 500 incidents can be recorded on a paging drum. The different incident reports contain some or all of the following:

- Identification of the CPU that received a machine check.
- Time, date, and program-user ID.
- Channel information (CSW, channel map, channel activity, channel log).
- CPU environment (log, contents of local store).
- Analysis and retry information (counts paths, instructions, storage).

The Environment Recording program records the information on the paging drum itself since it has complete control over the system when it is operating. There is, therefore, no danger of interference with normal monitor usage of the paging drum.

The format used for these reports is determined by the call-type received from the Recovery Nucleus, as illustrated in Figures 27 through 31 on the following pages.

Immediate Print Program (CMASC)

The Immediate Print program is called upon to record failure data on the system console typewriter. This data is intended to alert the customer engineer to machine errors and is condensed in order to have a minimum effect on TSS operations.

Attributes: The program is resident in auxiliary storage, nonreentrant, and privileged.

Entry: CMASC1

Restriction: No message will be typed if an operator's device is not available.

Exit: To SERR Bootstrap.

Operation: The immediate print program types one line on the operator's device whenever the Recovery Nucleus calls SERR because of a machine-check interruption or a system error. The information presented is:

- Time of day.
- Cause of error.
- CPU in error (if applicable).
- Storage element in error (if applicable).
- E-Reg. contents (if applicable).
- ROS address at time of failure.
- Failure classification code ID of error record (for retrieval by VMEREP or IPL-EREP).

The program is also used to type unexpected errors within SERR. In the event of a machine-check or program-check interruption, the program is called to notify the operator. The line types in the following format:

- "SERR MESSAGE".
- Error code and ID of SERR module that detected the error.

	0	1	2	3	4	5	6	7
0	CPU ID	SE ID	Byte Count		Mod NR	Call Type	Spare	
8	Date and Time							
16	Program ID							
24	Machine Check Old PSW							
32	Int. Code		Prefix		Spare			
40	Spare							
48	General Purpose Regs							
112	Floating Point Regs							
	Extended Control Regs							
208	CPU Logout							
384	SERR Module Reports							

Figure 27. Record format for an internal machine check (call types 01 and 09 from the recovery nucleus)

	0	1	2	3	4	5	6	7
0	CPU ID	SP.	Byte Count		Mod NR	Call Type	Flags	
8	Date and Time							
16	Program ID							
24	Machine Check Old PSW							
32	Prefix				CCU-Chan, Activity			
40	Channel Map							
48	CUA		SDA		Interruption Code			
56	CSW							
64	Channel Log							
88								

Figure 28. Record format for an external machine check (call type 29)

	0	1	2	3	4	5	6	7
0	Spare		Byte Count	SP.	Call Type	Spare		
8	SYMB. DVC. ADDR.		Alt. Path			Spare		
16	Last Seek Address							
24	Last Path		Tot. Error. Cnt.	Tot. Retry Cont.		Spare		
32	Retry Thresholds							
					Time of Error N-2			
48	Time of Error N-1				Time of Error N			
56	SDR Buckets							
88	No. of CCWs		Spare		Pointer to Falling CCW			
96	CCW List (Up to 10 CCWs)							

} Call Type 28 Only

Figure 29. Record format for a paging device SDR overflow or a solid paging I/O outboard error (call types 27, 28, and 2E)

	0	1	2	3	4	5	6	7
0	CPU ID	SP.	Byte Count		SP.	Call Type	Spare	
8	Date and Time							
16	Used ID							
24	Prefix		Spare					
32	SERR Module Reports (11 Double Words)							
112								

Figure 30. Record format for a system error (call type 41)

	0	1	2	3	4	5	6	7
0	Spare		Byte Count		SP.	Call Type	Spare	
8	SYMB. DVC. ADDR		Alt. Path		Last Path Used		Spare	
16	Last Seek Address							
24	Date and Time							
32	Program ID							
40	CSW							
48	Channel Log							
72	Channel Map							
80	No. of CCUs		Spare		Pointer to Failing CCW			
88	CCW List (Max. 9 Double Words)							

Figure 31. Record format for a paging I/O inboard failure (call type 2D)

- A message explaining the error (32 characters maximum).
- A program check, machine check or SVC old PSW, as applicable.

The SERR operation is aborted following the timeout and control is returned to the Recovery Nucleus via SERR Bootstrap.

Checker Program (CMASD) Chart CA

This program performs three functions:

- Saves the system environment before a SERR operation.
- Check the environment for a good logout and good parity log and registers.
- Restores the environment before control is returned to the Recovery Nucleus.

Attributes: The program is resident in auxiliary storage, nonreenterable, and operates in the privileged state.

Entry: CMASD1

RESTRICTIONS: The ability of the Checker program to detect incorrect data in a register is restricted to determining whether parity is good or bad. Errors that

cause an even number of bits to be picked or dropped will therefore not be detected.

Exit: To SERR Bootstrap.

Operation: The Checker program is called by Bootstrap itself, to save the machine environment. It is, then, the first program called each time SERR is operated because of a machine check. At the conclusion of the SERR operation, the program is again called, this time to restore the environment.

The environment saved by the program includes the general, floating point, and extended control registers of each CPU; the logout from the failing CPU; and the first 128 bytes of each CPU's PSA. Registers are saved by causing each CPU, in turn, to go through a register save routine in the checker program. This is initiated by means of external starts.

The checker program then verifies the validity of the data being saved. In the case of the logout, this is accomplished by checking for the existence of certain indicators which should be set, and also by counting the number of 'one' bits in each register logged and comparing this count with the logged parity. Register parity for all the registers is verified by bringing the saved register from memory to the

CPU and then performing a diagnose instruction. The diagnose is used to provide a log-on-count function which makes the error indicators available to the program. These are then checked to insure that no error indicator is set. Existence of a storage data check would indicate that the saved register had incorrect parity.

Errors found by the Checker program are recorded in its report area for use by the other SERR programs. The existence of bad parity data in a 'key' register, for example, is grounds in many cases for indicating 'no-retry'.

Pointer Program (CMASE)

This program locates the address of an instruction which has been interrupted by a machine check.

Attributes: The program is resident in auxiliary storage, is non-reenterable, and operates in the privileged state.

Entry: CMASE1

Assumptions: The operation of the program is contingent upon a Checker program report indicating the existence of a valid log.

RESTRICTIONS: Because of the CPU hardware characteristics, an identification of the address of the interrupted instruction is not always possible. The program indicates a no-retry in this case.

Exit: To SERR Bootstrap.

Operation: The primary task for the program is to examine the saved environment of the failing CPU and determine the address of the instruction in operation at the time of the failure. This address is used by the Restore and Validate program to determine the type of instruction (and therefore choose the appropriate analysis routine), and by the Instruction Retry Execution program to make up the damage report. The address found, therefore, is the address at which the failing CPU is restarted, if it is found that retry is possible.

Since neither the machine-check-old PSW nor the logged IC necessarily reflect the true location of the failing instruction, it is necessary for the Pointer program to make an analysis of the log in order to obtain the desired address.

The Pointer program relies for the most part on the IC, E-register and current ROS address, all from the saved log. If the "IC in LSWR" latch was set, the program uses the address in the local-store-working register as the IC value. The Pointer program also uses a diagnose instruction to

extract the current ROS word. The ROS word is examined and the results of the various tests are used to obtain a value which must be added to, or subtracted from, the IC in order to determine the exact location of the failing instruction.

The Pointer program also determines if the error that occurred was a storage data error during a store operation. If so, the field stored is cleared so that the CPU/Memory Checkout 2 program will not indicate no-retry. If the instruction is otherwise retryable, this will allow the instruction to be retried. Items considered during the analysis include:

- Was an 'End-OP' cycle in progress?
- Was a 'byte-update' cycle in progress?
- Was an execute instruction in progress?
- Was an active branch instruction in progress?
- Was an interruption in progress?

Each item above will cause a different routine to be executed in order to determine the instruction address.

Restore and Validate Program (CMASF) Chart CB

This program determines whether a retry threshold exists for the interrupted instruction, and indicates "no-retry" if such a threshold has been passed. The program also restores registers if necessary.

Attributes: The program is resident in auxiliary storage, nonreenterable, and operates in the privileged state.

Entry: CMASF1

Assumptions: The program is to some extent micro-program dependent. Changes to the ROS commands of any instruction may require corresponding changes to the program.

Exit: To SERR Bootstrap.

Operation: This program receives control from the SERR Bootstrap program, following the operation of the pointer program. If the no-retry trigger (in the saved logout) is off, restoration is not needed. If the trigger is on, the program extracts the Op. code and the R1 and R2 fields of the interrupted instruction from the pointer program's damage report. The Op. code is then used to index into a subroutine which does analysis and restoration appropriate to the particular instruction. If restoration is indicated, the original contents of the R1 register are retrieved from a saved hard-

ware register (A reg1, S reg., etc.) in the logout. This data is then placed in the correct slot in the saved general register area. When the general registers are restored at the end of SERR operation, the R1 register will have been restored to the value it contained prior to executing the interrupted instruction.

In addition to analysis and restoration, this program also tests the Checker program's report to determine if any key registers had bad parity at the time of the interruption. A bad parity indication for a critical register will force a no-retry condition.

Instruction Retry Execution Program (CMASG) Chart CC

This program determines if retry is possible (following a machine check interruption), forms a damage report, and increases a statistical element count.

Attributes: The program is resident in auxiliary storage, is nonreenterable, and operates in the privileged state.

Entry: CMASG1

Exit: To SERR Bootstrap.

Operation: This program performs a number of bookkeeping functions necessary to the completion of a SERR operation. The prime function is the generation of a damage report, used by the Recovery Nucleus to determine what action should be taken after a malfunction alert. The damage report is made up following an analysis of the reports of the various SERR programs. (Retry is considered possible unless a program reports a no-retry condition).

This program contributes to the retry decision by means of its report. The program that effects retry is the test for a recurring error. On its second pass, (following environment recording), the program matches its report on the current error with its report on the preceding error. If the failure indications and the time duration between the two failures are such as to indicate the probability of a solid error condition, the no-retry flag is set in the damage report.

Whenever SERR operates without a malfunction being detected by a CPU/Memory Checkout program, this program updates statistical element counters. These counters (one for each CPU and storage element) are incremented by the program and later stored on the paging drum by the environment-recording program. The customer engineer may retrieve them with either VMEREP or IPL-EREP. The update is performed by

adding a one to the counter for the malfunctioning CPU and for each storage element involved in the instruction.

This program also saves the operand addresses and the first eight bytes of each operand used by the failing instruction. This data is also recorded on the paging drum and may be retrieved by either EREP program.

CPU/Memory Checkout 1 Program (CMASH) Chart CD

This program performs a cursory check of one or more CPUs. The objective of the program is to uncover machine malfunctions while performing a controlled check of the hardware. In this way, hardware malfunctions that are not detected by error circuitry can be uncovered and the element can be isolated.

Attributes: The program is resident in auxiliary storage, non-reenterable, and operates in the privileged state.

Entry: CMASH1

RESTRICTIONS: Testing is restricted to a cursory test. This program should not be considered a substitute for the normal diagnostic package.

Exit: To SERR Bootstrap.

Operation: The CPU/Memory Checkout 1 program receives control from the SERR Bootstrap program. Its job is to exercise CPU(s) while enabled for machine checks. The program is limited in its exercise since certain instructions cannot be executed. It can, however, use and verify the operation of all but a few special instructions.

Any error checks uncovered by validity or machine check interruptions are recorded, along with the CPU that is being exercised. If an error is found, the recorded information is used when the damage report is formed. This program, along with the other two checkout programs of SERR, may be thought of as having two logical parts. The first, a control section, runs only in the CPU that has gained control following a malfunction alert. The second section contains the testing facilities and is run (from the same point in memory) by all CPU's. It is a function of the control section to initiate the execution of the test section by each CPU (via an external start) to monitor its operation, and to report on errors discovered, if any.

The test section of this program consists of a CPU instruction test which is similar in operation to most standard CPU

diagnostics. Each instruction to be tested is executed and the resulting data is analyzed to determine whether or not the instruction was executed correctly. Each instruction is tested, with the exception of I/O instructions and certain set status switching instructions.

While a particular CPU is exercised by the program, the controlled CPU is standing by in a time-out loop waiting to hear from the CPU doing checkout. Appropriate action is taken if the CPU doing checkout is not heard from in an allotted time.

CPU/Memory Checkout 2 Program (CMASI) Chart CE

The function of this program is to check each SSU and to scan each storage element from each CPU, looking for possible bad parity data. Bad parity locations are exercised and the results of the tests are left in a report area for analysis by the Instruction Retry Execution program.

Attributes: The program is resident in auxiliary storage, nonreenterable, and operates in the privileged state.

Entry: CMASI1

Exit: To SERR Bootstrap.

Operation: This program runs in each CPU in the system whenever called during a SERR operation. When called, it performs a cursory SSU test by saving 16 bytes in each storage element and exercising the 16 byte test area with various patterns. Assuming the SSU test is successful, each test area is restored with its original contents.

Following the SSU test, all storage elements are scanned in a test which looks for bad parity data. In the event that a bad parity word is found, that particular area is also exercised to determine if the failure was solid or intermittent. In either case the word is cleared (to good parity if possible) at the conclusion of the exercise.

The results of the tests are left in the program's report area for analysis by the instruction-retry-execution program and inclusion in the damage report.

Scanning storage is accomplished by successive 256-byte CLC instructions. Following each 'compare', a DIAG instruction is executed which causes a logout. The error triggers in the log are then examined to determine if a storage failure has been detected.

CPU/Memory Checkout 3 Program (CMASJ) Chart CF

The function of this program is to check ROS and local store circuitry in each CPU.

Attributes: The program is resident in auxiliary storage, non-reenterable, and operates in the privileged state.

Entry: CMASJ1

Exit: To SERR Bootstrap.

Operation: This program consists of a control section and a test section as described for the CPU/Memory Checkout 1 program.

The test portion of the program is designed to check ROS and local store circuitry. The ROS test consists of a ROS check-sum routine which uses a diagnose instruction to logout each ROS word. The 100 bits in a particular ROS word are combined to form a check sum which is temporarily stored. The test procedure is performed three times and the three results are compared. A non-comparison results in an error report. This procedure is repeated for all ROS words which can safely be tested in this fashion. (Certain ROS words, such as those involved in a power-on-reset, cause undesirable functions to be initiated.)

The local-store test loads each of the general registers with patterns which are then compared with the original data words. Again, non-comparison results in an error report.

While a particular CPU is performing the local-store test, the controlling CPU is standing by in a time-out loop waiting to hear from the CPU under test. Appropriate action is taken if the CPU being tested does not respond in an allotted time.

System Error Processor (CEAIS) Chart CG

The function of the System Error Processor (SYSERR) is to provide a central routine for handling the conditions caused by system software errors and hardware errors detected by the supervisor modules and privileged service routines. It is called to output an error message, or handle any of five types of errors:

- Main storage minor software errors.
- Main storage major software errors.
- Virtual memory minor software errors.
- Virtual memory major software errors.

- Hardware failure for resident or virtual memory.

Restrictions: Entry into SYSERR is made solely from the interrupt stacker area when an SVC interruption is invoked by either supervisor modules or privileged service routines. These SVC interruptions are not queued on the system queues.

Entries:

CEAS1 - resident supervisor call

CEAS2 - virtual memory privileged service routine call. Input parameters from either source include an error severity code, a dump operation code, a module code, and an error condition code.

Modules Called: RSS Inter-CPU Communication (CEHCC entered at CEHCCA) calls TSS Inter-CPU Communication subroutine (CEAIC) to halt and transfer other active CPUs when TSS/360 is operating with more than one active CPU in the system.

Supervisor Core Allocation (CEALI entered at CEAL01) gets main storage for a task interrupt GQE.

Interrupt Stacker (CEAJI entered at CEAJIL) stacks pending interrupts from devices used by SYSERR.

Queue GQE on TSI (CEAAF entered at CEAAFQ) queues the task interrupt GQE.

Recovery Nucleus (CEAIR entered at CEAIR2) initiates system restart when a major resident system error or hardware failure is encountered.

Task Communication Control subroutine (CEAAN) builds a message control block (MCB) and queues it onto the operator task when overlapping of SYSERR messages is allowed.

Exit: After handling a minor resident error, control is returned to the caller.

After writing a message to the operator's console, control is returned to the calling routine.

After handling a major or minor virtual memory error, SYSERR exits to the Queue Scanner (CEAJQ at CEAJQS). In the case of a major VM error, the task interrupt GQE is processed causing the task to be deleted from the system.

A task which has issued more than two consecutive minor VM errors in a period of two minutes will also be deleted from the system.

Operation: When entered, SYSERR first saves the registers it needs in order to operate, then immediately checks, via the SETLOCK macro, to see if it is handling a previous call by the same CPU. (Is the SYSERR lock-byte on?) If yes, SYSERR then checks to see whether this is a recursive call by the CPU which initiated the previous SYSERR. If a recursive call has occurred, SYSERR builds and transmits a message to the operator's console declaring the situation. The wait state is then entered to allow the operator to call the support system. Recovery from this type of error is impossible.

If not a recursive entry, the SYSERR lock-byte is set and the system status is saved. SYSERR then checks to see if other CPUs are active in the system. If yes, the Inter-CPU Communication subroutine is called to halt and transfer other CPUs. The transfer address points to a SETLOCK macro instruction loop. Since the lock byte being tested is on, any other CPU that may have gotten through the Interrupt Stacker (which disables external interrupts), before the halt and transfer took effect, will loop on the SYSERR lock byte. After SYSERR finishes processing an error, this lock byte is turned off via the OPENLOCK macro instruction to allow pending SYSERR SVCs through.

On return from CEAIC, or if no other active CPUs are involved, SYSERR builds a message from information received in the input parameters and contained in the TSI, system table, and ISA. The type of error is then checked. For message output only, the message is transmitted to the operator's console for immediate printing. Control is then returned to the calling routine. For resident SYSERS, the 'real core wait' flag (CEAISR1) is tested and the message transmitted to the operator's console via immediate print. For virtual memory SYSERS, the appropriate wait flag and the 'YSER overlap print' flag are tested for '00'. When both conditions are met, a check is made for two or more errors in the last two minutes by this task.

When there have not been two or more errors, an MCB is built by Task Communication and queued onto the operator task, and the message printed from SYSLOG as just another message to the operator. If there have been two or more errors in two minutes from the same TID, the flag CEAISV3 is set to 'FF' and the SYSERR message is printed via immediate print. A message is also sent to the operator informing him that the task has exceeded two SYSERS; he then has the option of forcing the task or allowing it to continue. However, he must use the support system to set CEAISV3 = '00' in order to again use the message overlap fea-

ture. The message is transmitted to the operator's console in the following format:

<ER> ,type,error-number,severity,user ID,
task ID, conv/non-conv,symbolic device
address, HH:MM, [VPSW wd. 1, VPSW wd. 2]

Where:

type

RC for a real core SYSERR

VM for a virtual memory SYSERR

error-number

a four digit number identifying the error number for resident SYSERRs.

a nine digit number identifying the error number for virtual memory SYSERRs.

severity

MIN for minor software code 1

MAJ for major software code 2

HAR for hardware code 3

user ID

the 8 character user ID taken from the TSI.

task ID

the 4 character task ID taken from the TSI.

CONV/NON-CONV

C for conversational

N for nonconversational

symbolic device address

the 4 character symbolic device address for sysin taken from the TSI.

HH:MM

hours and minutes where $0 \leq HH \leq 23$
and $0 \leq MM \leq 59$.

VPSW word 1 and VPSW word 2

the first and second words of the virtual program old PSW from the tasks ISA. This information will only be printed if a 081302502 or 081302508 SYSERR is declared.

The following flags are external symbols and are initially set to '00':

CEAISR1 - For minor resident errors.

CEAISV1 - For minor virtual memory errors.

CEAISV2 - For major virtual memory errors.

CEAISV3 - For SYSERR message overlap print.

With the exception of CEAISV3, if the flag tested is set to 'FF', SYSERR enters the WAIT state allowing the operator to call the support system. If the operator then responds with a 'RUN CEAIS3', processing continues in SYSERR. If they are set to '00', processing continues as follows:

- Minor resident - resume system operation at the point of interrupt by returning to the caller.
- Major resident - No flag is tested for a major RC SYSERR and SYSERR always goes to a wait. If the operator responds with 'Run CEAIS3', the following message is printed: 'CEAIS002 REAL CORE MAJOR SOFTWARE ERROR - RESTART CANNOT BE ACCOMPLISHED - RE-IPL'.
- Major virtual memory - A call is made to Supervisor Core Allocation to get main storage for a task interrupt GQE. The GQE is created with a program interrupt code of 202, a call is made to Queue GQE on TSI to queue it on the affected task, and exit is to the Queue Scanner.
- Minor virtual memory - If the VM minor ABEND flag is set, a call is made to Supervisor Core Allocation to get main storage for a task interrupt GQE. The GQE is created with a program interrupt code of 201, a call is made to Queue GQE-on-TSI to queue it onto the affected task and an exit is made to the Queue Scanner.
- Hardware failure - (main storage) call the recovery nucleus. On return, if recovery is successful, the SYSERR is treated as a minor error and return is made to the caller. If VM, it is treated as a minor VM error and exit is to the Queue Scanner.

SECTION 4: FLOWCHARTS

The Flowcharts in this manual have been produced by an IBM program, using ANSI symbols. The symbols are defined in the left column below, and examples of their use are shown at the right.

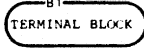
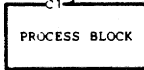

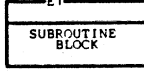
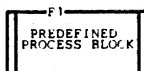
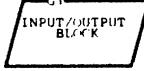
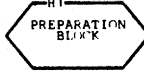


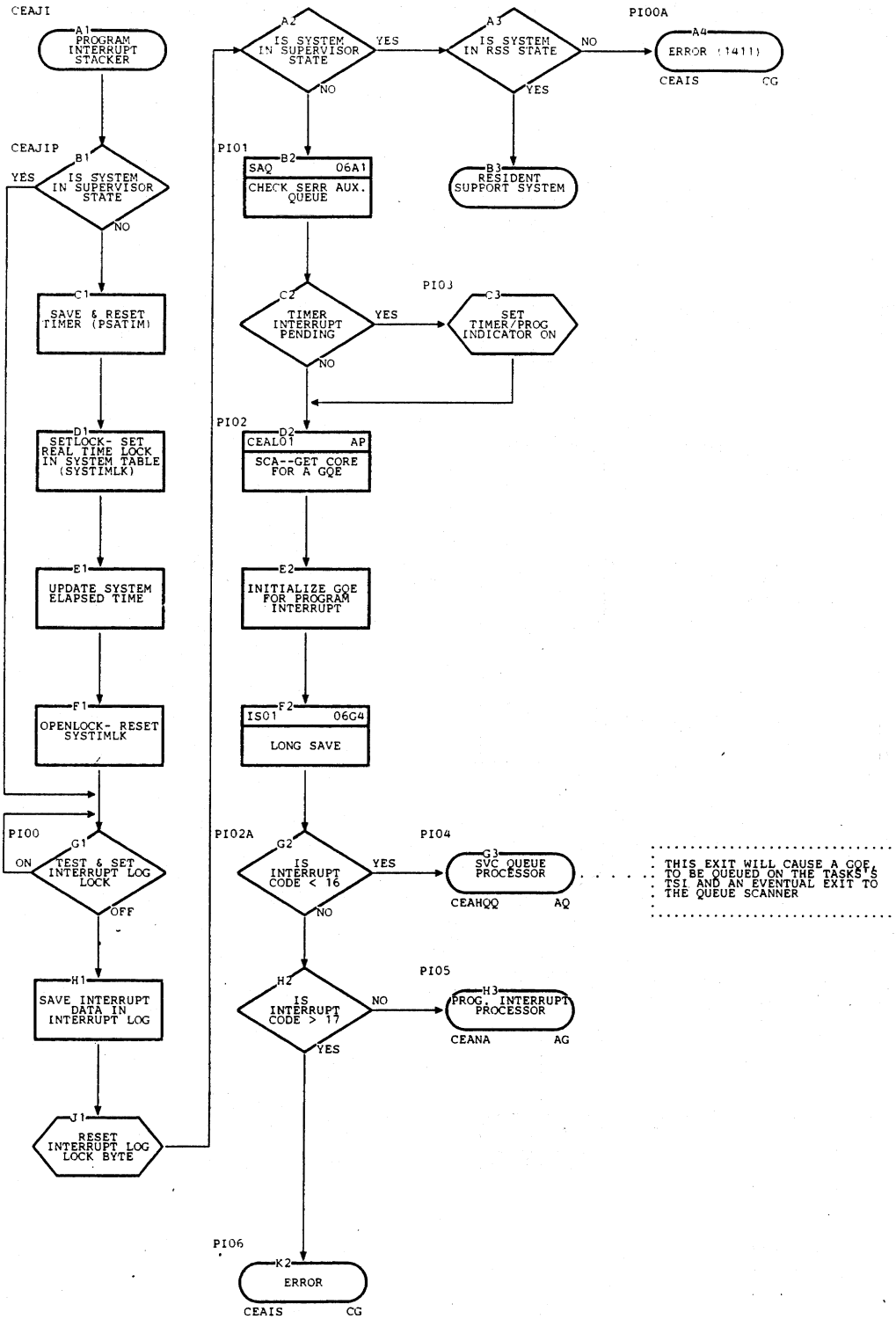
SYMBOL	DEFINITION	EXAMPLE	COMMENTS
	INDICATES AN ENTRY OR TERMINAL POINT IN A FLOWCHART; SHOWS START, STOP, HALT, DELAY OR INTERRUPTION. MAY ALSO INDICATE RETURN TO THE CALLING PROGRAM.	MODNAME B3 COMNAME	B3: MODNAME IS THE LOAD MODULE OR LIBRARY NAME OF THE ROUTINE DESCRIBED BY THIS FLOWCHART. COMNAME IS THE COMMON NAME OF THE ROUTINE.
	INDICATES A PROCESSING FUNCTION OR A DEFINED OPERATION CAUSING CHANGE IN VALUE, FORM OR LOCATION OF INFORMATION.	C3 CSECT LABEL1	C3: CSECT IS THE CSECT NAME OR OTHER ENTRY POINT AT WHICH PROCESSING BEGINS. LABEL1 IS THE LABEL OF THE FIRST INSTRUCTION.
	INDICATES A DECISION OR SWITCHING-TYPE OPERATION THAT DETERMINES WHICH OF A NUMBER OF ALTERNATE PATHS SHOULD BE FOLLOWED.	D3	D3: PROGRAM EXECUTION CONTINUES WITH BLOCK H3 WHEN THE DECISION IS NO. OR WITH BLOCK E3 WHEN THE DECISION IS YES.
	INDICATES A SUBROUTINE OR MODULE THAT IS DESCRIBED IN THIS MANUAL.	E3 SUBRTN	E3: LABEL2 IS THE LABEL OF THE SECTION OF CODE IN THIS ROUTINE FROM WHICH CONTROL IS PASSED TO THE SUBROUTINE. CONTROL RETURNS TO THE NEXT INSTRUCTION FOLLOWING THE SUBROUTINE CALL. ENTRYPT IS THE ENTRY POINT. SUBRTN IS THE COMMON NAME OF THE SUBROUTINE IN FLOWCHART AG.
	INDICATES A SUBROUTINE OR MODULE THAT IS INCLUDED IN THE FLOWCHARTS OF ANOTHER MANUAL.	F3 -PDPNM-	F3: LABEL3 IS THE LABEL OF THE SECTION OF CODE FROM WHICH CONTROL IS PASSED TO THE PREDEFINED PROCESS PDPNM, WHICH IS DOCUMENTED IN ANOTHER PUBLICATION (-PDPNM- MAY ALSO BE USED IN A PROCESSING BLOCK).
	INDICATES GENERAL I/O FUNCTIONS, SUCH AS GET, PUT, READ, WRITE, SIO, AND DEVICE-CONTROL MACRO INSTRUCTIONS.	G3	G3: EXECUTION CONTINUES WITH BLOCK H3 WHEN THE DECISION IS YES. OR WITH BLOCK A1 ON PAGE 2 OF THIS SET OF FLOWCHARTS WHEN THE DECISION IS NO.
	INDICATES A PROCESS THAT CHANGES SYSTEM OPERATION FOR EXAMPLE SETS A SWITCH, MODIFIES AN INDEX REGISTER, OR INITIALIZES A ROUTINE.	H3	H3: LABEL4 IS THE LABEL OF A SECTION OF CODE OF THIS ROUTINE THAT INITIATES I/O.
	INDICATES ENTRY TO OR EXIT FROM ANOTHER BLOCK ON THE SAME FLOWCHART PAGE.	J3 NEXTRTN	J3: NEXTRTN IS THE COMMON NAME OF THE ROUTINE THAT EXECUTES AFTER THIS ROUTINE. ENTRYPT IS THE ENTRY POINT OF NEXTRTN, WHICH IS DESCRIBED IN CHART AC. VIA: PASSMECH INDICATES HOW CONTROL PASSES FROM COMNAME TO NEXTRTN.
	INDICATES ENTRY TO OR EXIT FROM A BLOCK ON ANOTHER PAGE OF THE SAME SET OF FLOWCHARTS.	EP-ENTRYPT CHART AC VIA: PASSMECH	

Chart AA. Interrupt Stacker (CEAJI) (page 1 of 6)



Program Logic Manual

GY28-2012-5

Resident Supervisor

Flowcharts on pages 169-312 were not scanned.

APPENDIX A: MODULE IDS AND NAMES

<u>Module ID</u>	<u>Entry Point(s)</u>	<u>Module Name</u>
CEAAA	CEAAAR	Command Word Relocator Subroutine
CEAAB	CEAAB1	Set Path Subroutine
CEAAC	CEAACQ	Queue Device on Task Subroutine
CEAAD	CEAADR	Remove Device from Task Subroutine
CEAAE	CEAAE1	External Page Location Address Translator Subroutine
CEAAF	CEAAFQ	Queue GQE on TSI Subroutine
CEAAG	CEAAG1	Start I/O Subroutine
CEAAH	CEAAHR	Reset Device Suppression Flag Subroutine
CEAAI	CEAAIH	Halt I/O Subroutine
CEAAJ	CEAAJD	Dequeue I/O Requests Subroutine
CEAAK	CEAAK1	Set Asynchronous Entry Subroutine
CEAAL	CEAALQ; CEAALP	Purge Subroutine and SVC Processor
CEAAM	CEAAM1	Paging I/O Error Recovery Routine
CEAAN	CEAAN1	Task Communication Control Subroutine
CEAAP	CEAAP1	Suppress Auxiliary Allocation Subroutine
CEAAQ	CEAAQ1	Paging Failure Recovery Subroutine
CEAAS	CEAAS1	Alternate Path Retry Subroutine
CEAAT	CEAAT1	Standard Area Retry Subroutine
CEAAV	CEAAV1	Same Path Retry Subroutine
CEAAX	CEAAX1	Start Retry Operation Subroutine
CEAAZ	CEAAZ1	Reset Drum Interlock Subroutine
CEAA0	CEAA01	I/O Call Subroutine
CEAA1	CEAA11	Pageout Processor
CEAA2	CEAA20	Task Interrupt Control Subroutine
CEAA3	CEAA31	I/O Device Queue Processor
CEAA4	CEAA41	Channel Interrupt Queue Processor
CEAA5	CEAA5P; CEAASR; CEAA5S	Pathfinding Subroutine
CEAA6	CEAA61	Page Direct Access Queue Subprocessor
CEAA7	CEAA71	Page Direct Access Interrupt Subprocessor

<u>Module ID</u>	<u>Entry Point(s)</u>	<u>Module Name</u>
CEAA8	CEAA81; CEAA50; CEAA82	Page Drum Queue Processor
CEAA9	CEAA91	Page Drum Interrupt Queue Processor
CEABA	CEABA1; CEABA2; CEABA3	RJE Asynchronous I/O Interrupt Subroutine
CEABB	CEABB1	RJE Synchronous I/O Interrupt Subroutine
CEABC	CEABC1; CEABC2	RJE Line Control Subroutine and SVC Processor
CEABE	CEABEM	External Machine Check Interrupt Processor
CEABQ	CEABQ1	Generate and Enqueue Interrupt GQE Subroutine
CEAHQ	CEAHQA	Add Page Subroutine
CEAHQ	CEAHQP	Supervisor Call Queue Processor
CEAHQ	CEAHQF	Time Slice End Subroutine
CEAHQ	CEAHQG	Categorize Paging Requirements Subroutine
CEAH2	CEAH02	Setup TSI Field Subroutine
CEAH2	CEAH03	Extract TSI Field Subroutine
CEAH7	CEAH07	Set External Page Table Entries Subroutine
CEAIA	CEAIAA	Auxiliary Storage Allocation Queue Processor
CEAIA	CEAIAR	Auxiliary Storage Release Subroutine
CEAIC	CEAIC1; CEAIC2	Inter-CPU Communication Subroutine
CEAIR	CEAIR1; CEAIR2; CEAIR3	Recovery Nucleus
CEAIS	CEAIS1; CEAIS2	System Error Processor
CEAI6	CEAI61; CEAI62	Real Core Statistical Data Recording Subroutine
CEAI7	CEAIP1; CEAIP2; CEAIP3	Real Core Error Recording Subroutine
CEAJI	CEAJIP; CEAJIS; CEAJIE; CEAJII	Interrupt Stacker Module
CEAJQ	CEAJDE	Dequeue GQE Subroutine
CEAJQ	CEAJEN	Enqueue GQE Subroutine
CEAJQ	CEAJMG	Move GQE Subroutine
CEAJQ	CEAJQS	Queue Scanner
CEAJQ	CEAJSF	Set Suppress Flag Subroutine
CEAKD	CEAKD1	The Dispatcher
CEAKE	CEAKEA	Entrance Criteria Subroutine
CEAKI	CEAKIA; CEAKIB	Internal Scheduler
CEAKR	CEAKRT	Create Real Time Interrupt Subroutine

<u>Module ID</u>	<u>Entry Point(s)</u>	<u>Module Name</u>
CEAKT	CEAKT1	Timer Interrupt Queue (TIQ) Processor
CEAKZ	CEAKZA	Rescheduling Subroutine
CEAL1	CEAL01	Supervisor Core Allocation Subroutine
CEAL1	CEAL02	Supervisor Core Release Subroutine
CEAL1	CEAL04	User Core Release Subroutine
CEAMC	CEAMC1	Create TSI Subroutine
CEAMC	CEAMT1	Task Initiation Subroutine
CEAMD	CEAMDT	Delete TSI Processor
CEAML	CEAMLP	Locate Page Subroutine
CEAMP	CEAMP1	Page Posting Subroutine
CEAMS	CEAMS1	Search RSPI Table Subroutine
CEAMW	CEAMWS	Write Shared Pages Subroutine
CEAMX	CEAMXP	XTSI Overflow Subroutine
CEAMY	CEAMY	XTSI Page Packing Subroutine
CEANA	CEANAA	Program Interrupt Queue Processor
CEANB	CEANBA	User Core Allocation Queue Processor
CEANC	CEANCA	Find Page Subroutine
CEAND	CEANDA	Delete Page Subroutine
CEANF	CEANFA	Contiguous Core Allocation Queue Processor
CEANG	CEANG1	Segment Block Remover Subroutine
CEAP0	CEAH10	Move External Page Table Entries Subroutine
CEAP7	CEAH17	AWAIT SVC Subroutine
CEAQ2	CEAH22	Set User Interval Timer Subroutine
CEAQ4	CEAQ4A	Check Protection Class Subroutine
CEAQ5	CEAH25	Inter-Task Communication Subroutine
CEAQ6	CEAH26	Add Shared Pages Subroutine
CEAQ7	CEAH27	Connect Segment to Shared Page Table Subroutine
CEAQ8	CEAH28	Disconnect Segment from Shared Page Table Subroutine
CEAR0	CEAH30	TWAIT Subprocessor
CEAR1	CEAR1A	Present Schedule Table Entry Processor
CEAR2	CEAR2A	Pulse Schedule Table Entry Processor
CEAR3	CEAR3A	Change Schedule Table Entry Processor
CEAR4	CEAR41	Terminal SVC Processor (CONN)

<u>Module ID</u>	<u>Entry Point(s)</u>	<u>Module Name</u>
CEAR4	CEAR42	Terminal SVC Processor (DCON)
CEAR4	CEAR43	Terminal SVC Processor (WAIT)
CEAR4	CEAR44	Terminal SVC Processor (CKALOC)
CEAR4	CEAR45	Terminal SVC Processor (LCD)
CEAR4	CEAR46	Terminal SVC Processor (ATTACH)
CEAR4	CEAR47	Terminal SVC Processor (UFLOW)
CEAR4	CEAR48	Terminal SVC Processor (SETTDE)
CEASS	CEASS1; CEASS2	TSS Dynamic Status SVC Processor
CEAS2	CEAH42	Setup System Table Field Subroutine
CEAS2	CEAH43	Extract System Table Field Subroutine
CEAS4	CEAH44	Setup XTSI Field Subroutine
CEAS4	CEAH45	Extract XTSI Field Subroutine
CEAS6	CEAH46	Read Time Subroutine
CEAS7	CEAH47	Set Real Time Interval Subroutine
CEAS8	CEAH48	Restore Elapsed Time Subroutine
CEATC	CEATC1, CEATC2	Terminal Communication Subprocessor, ATCS Processor
CEATS	CEATS1, CEATS2 CEATS3, CEATS4	Terminal Control Table Entry Allocation Subprocessor
CEAT1	CEAT2A	Extract Accumulated Time Subroutine
CEAT2	CEAH52	Special Create TSI Processor
CEAT4	CEAT4A	Extract from Auxiliary Storage Allocation Table
CGCMA	CGCMAC	Reconfiguration Routine
CMASA	CMASA1	SERR Bootstrap
CMASB	CMASB1	Environment Recording Program
CMASC	CMASC1	Immediate Print Program
CMASD	CMASD1	Checker Program
CMASE	CMASE1	Pointer Program
CMASA	CMASA1	SERR Bootstrap
CMASF	CMASF1	Restore and Validate Program
CMASG	CMASG1	Instruction Retry Execution Program
CMASH	CMASH1	CPU/Memory Checkout 1 Program
CMASI	CMASI1	CPU/Memory Checkout 2 Program
CMASJ	CMASJ1	CPU/Memory Checkout 3 Program

There are 256 SVC codes in TSS/360. They are divided into four groups:

1. SVC codes 000-063

These are undefined and reserved for users who wish to specify their own SVC handling routines.

2. SVC codes 064-095

These codes are reserved for SVCs handled by the Time Sharing System Support System. Of these, codes 64, 74-79, and 85-95 are not defined.

3. SVC codes 096-127

These codes are reserved for task-oriented SVCs handled by the Task Monitor. Of these, codes 96-116, 124, and 126 are not defined.

4. SVC codes 128-255

These are reserved for SVCs handled by the Resident Supervisor. Of these, codes 128-186, 188-192, 220, 224-225, 239, and 247 are not defined. Codes 128-143 are reserved for installation use. The installation may also set the privilege level required for use of these SVCs.

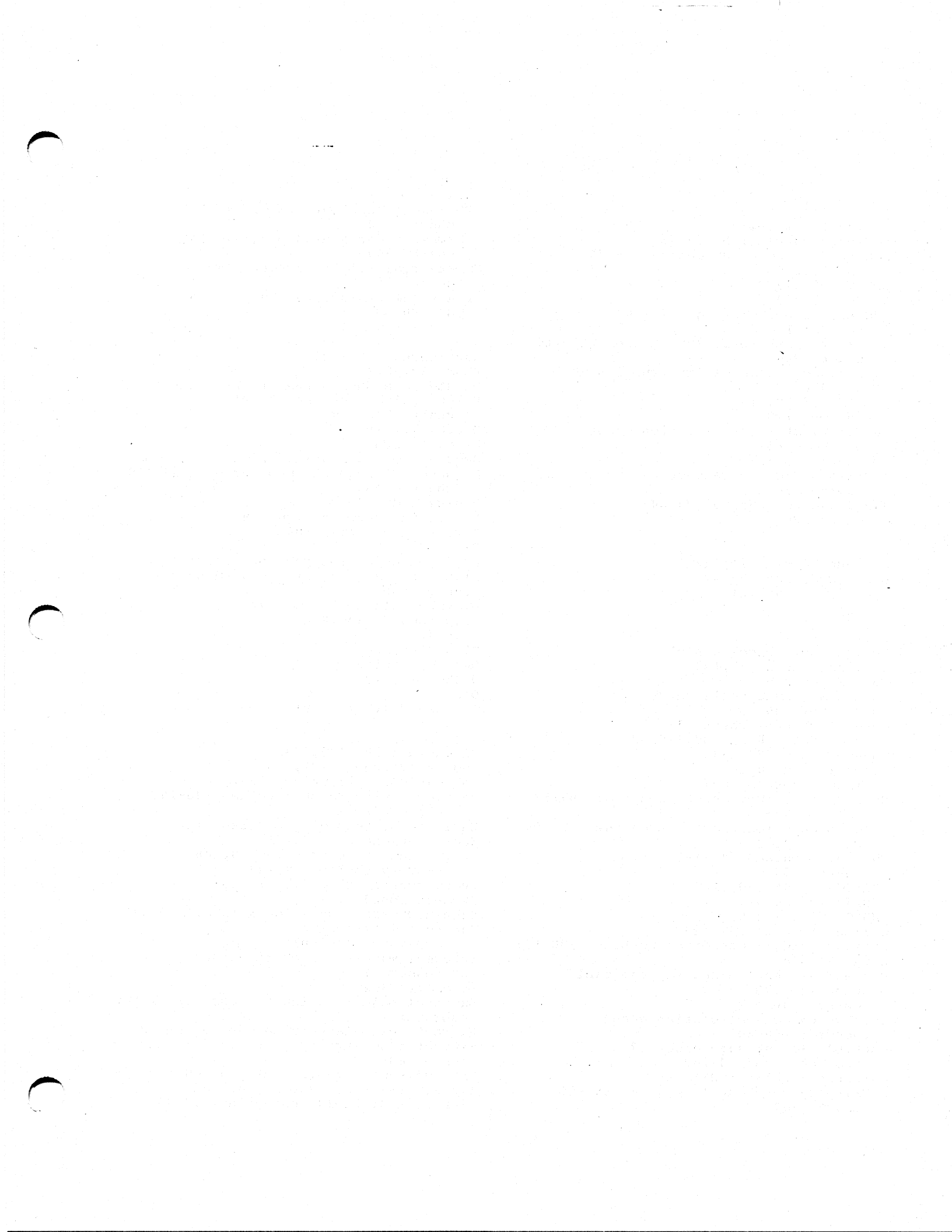
SVC codes which are defined are listed below:

<u>SVC Code in Hex.</u>	<u>SVC Code in Dec.</u>	<u>Macro ID</u>	<u>Module Name</u>	<u>Module Entry Point</u>	<u>Operation</u>
41	65	N/A	CEHDR	CEHDRA	Get real storage for VSS
42	66	N/A	CEHDR	CEHDRA	Put real storage for VSS
43	67	N/A	CEHDR	CEHDRA	AT SVC execution in real storage completed
44	68	N/A	CEHDR	CEHDRA	RSS AT SVC execution in virtual storage completed
45	69	N/A	CEHDR	CEHDRA	Execute AT SVC in real storage for RSS or VSS
46	70	N/A	CEHDR	CEHDRA	Process VSS-supplied command string
47	71	N/A	CEHDR	CEHDRA	Execute AT in private virtual storage for RSS
48	72	N/A	CEHDR	CEHDRA	Execute AT in shared virtual storage for RSS
49	73	N/A	CEHDR	CEHDRA	Determine if input virtual storage page is shared
50	80	N/A	CEHDA	CEHDAA	Execute AT in private virtual storage for VSS
51	81	N/A	CEHDL	CEHDLA	Logon MSP or TSP as indicated in message control block
52	82	N/A	CEHDE	CEHDEA	Deactivate VSS
53	83	N/A	CEHDV	CEHDVA	Activate VSS
54	84	N/A	CEHDA	CEHDAA	VSS AT execution in virtual storage completed
55	85	N/A	CEHDA	CEHDAA	Execute AT in shared virtual storage for VSS

<u>SVC Code in Hex.</u>	<u>SVC Code in Dec.</u>	<u>Macro ID</u>	<u>Module Name</u>	<u>Module Entry Point</u>	<u>Operation</u>
74	116	EXIT	CZCJT	CZCJTS	Return to Command System
75	117	RAE	CZCJT	CZCJTS	Restore and enable
76	118	CLIP	CZCJT	CZCJTS	Read command from SYSIN (uncond.)
77	119	CLIC	CZCJT	CZCJTS	Read command from SYSIN (cond.)
78	120	RSPRV	CZCJT	CZCJTS	Restore Privilege
79	121	ENTER	CZCJT	CZCJTS	Enter privileged service routine
7A	122	RTRN	CZCJT	CZCJTS	Final return to Task Monitor from user code
7B	123	DELET	CZCJT	CZCJTS	Invoke dynamic loader delete routine
7D	125	PCSVC	CZCJT	CZCJTS	Invoke program control system
7F	127	DLINK	CZCJT	CZCJTS	Transfer to dynamic loader for external symbol resolution
BB	187	UFLOW	CEAR4	CEAR47	Adjust or obtain conversational or MTT task limit
C1	193	SAMPLE	CEASS	CEASS2	Sample System Statistics Table
C2	194	ZEROSST	CEASS	CEASS1	Zero System Statistics Table
C3	195	ATTACH	CEAR4	CEAR46	Obtain TCT slot for TSS user
C9	201	RDI	CEAAZ	CEAAZ1	Reset drum interlock
CA	202	LCD	CEAR4	CEAR45	Indicate line code for terminal
CB	203	CKALOC	CEAR4	CEAR44	Check for terminal in use by MTT
CC	204	WAIT	CEAR4	CEAR43	Wait for terminal stimuli
CD	205	SETTDE	CEAR4	CEAR48	Turn off HELD flag in CHBTDE
CE	206	SCRTSI	CEAT2	CEAT2A	Special create TSI
CF	207	CONN	CEAR4	CEAR41	Connect a task to the MTT subsystem
D0	208	DCON	CEAR4	CEAR42	Disconnect a task from the MTT subsystem
D1	209	XTRTM	CEAT1	CEAT1A	Extract accumulated time
D2	210	SETAE	CEAAK	CEAAK1	Set asynchronous entry
D3	211	SPATH	CEAAB	CEAAB1	Set Path
D4	212	RSTTIM	CEAS8	CEAH48	Restore elapsed time
D5	213	XTRXTS	CEAS4	CEAH45	Extract XTISI field
D6	214	SETXTS	CEAS3	CEAH44	Set up XTISI field
D7	215	XTRSYS	CEAS2	CEAH43	Extract system table field
D8	216	SETSYS	CEAS2	CEAH42	Set up system table field
D9	217	SETTR	CEAS7	CEAS7A	Set or cancel real core time interval requested by supervisor routine

<u>SVC Code in Hex.</u>	<u>SVC Code in Dec.</u>	<u>Macro ID</u>	<u>Module Name</u>	<u>Module Entry Point</u>	<u>Operation</u>
D9	217	SETTIMER	CEAS7	CEAS7A	Set Virtual memory time interval
DA	218	REDTIM	CEAS6	CEAS6A	Read Time
DB	219	ATCS	CEATC	CEATC2	Activate terminal communications subprocessor
DD	221	RESET	CEAAH	CEAAHR	Reset device suppression flag
DE	222	PURGE	CEAAL	CEAALP	Purge I/O requests
DF	223	SETIR RESETIR	CEAI6	CEAI62	Set on or off the Immediate Report flag (PSDIR)
E0	224	N/A	CEAHQ	CEAHQG	Categorize paging requirements
E2	226	PULSE	CEAR2	CEAR2A	Pulse schedule table entry
E3	227	CHANGE	CEAR3	CEAR3A	Change schedule table entry
E4	228	YSER	CEAIS	CEAIS1	System error processing call from nonresident task
E5	229	TWAIT	CEAR0	CEAH30	Terminal I/O wait
E6	230	AUXPG	CEAT4	CEAT4A	Extract auxiliary disk and drum page counts
E7	231	IOCAL	CEAA0	CEAA01	I/O call
E8	232	RJELC	CEABC	CEABC1	RJE line control processing
E9	233	RMDEV	CEAAD	CEAADR	Remove device from task
EA	234	ADDEV	CEAAC	CEAACQ	Add device to task
EB	235	SETUP	CEAH2	CEAH02	Set up TSI field
EC	236	ADSPG	CEAQ6	CEAH26	Add shared page
ED	237	DSSEG	CEAQ8	CEAH28	Disconnect segment from shared page table
EE	238	CNSEG	CEAQ7	CEAH27	Connect segment to shared page table
F0	240	VSEND	CEAQ5	CEAQ5V	Inter-task communication from TSS
F1	241	CKCLS	CEAQ4	CEAQ4A	Check protection class
F2	242	PGOUT	CEAA1	CEAA11	Page out VS pages to external storage
F3	243	TSEND	CEAHQ	CEAHQF	Force time slice end
F4	244	SETXP	CEAH7	CEAH07	Set external page table entry
F5	245	MOVXP	CEAP0	CEAH10	Move external page table entry
F6	246	XTRCT	CEAH3	CEAH03	Extract TSI field
F8	248	AWAIT	CEAP7	CEAH17	Await an interrupt
F9	249	DELPG	CEAND	CEANDA	Delete pages
FA	250	ADDPG	CEAHQ	CEAHQA	Add pages
FB	251	SETTU	CEAQ2	CEAH22	Set user interval timer

<u>SVC Code</u> <u>in Hex.</u>	<u>SVC Code</u> <u>in Dec.</u>	<u>Macro ID</u>	<u>Module</u> <u>Name</u>	<u>Module</u> <u>Entry</u> <u>Point</u>	<u>Operation</u>
FC	252	DLTSI	CEAMD	CEAMDT	Delete TSI
FD	253	CRTSI	CEAMC	CEAMC1	Create TSI
FE	254	LVPSW	CEAJI	CEAJIS	Load virtual PSW
FE	254	ERROR	CEAIS	CEAIS2	System error processing call from supervisor
FF	255			CEABZ	Move error simulation table to real one.



INDEX

- Active list 143
 - chart 236-238
- Add page subroutine 73-75
- Add shared pages subroutine 75-76
 - chart 239
- ADDEV 95,319
- ADDPG 73,319
- Address marker missing 122-123
- ADSPG 75,319
- Alternate path retry subroutine 117-119
 - chart 264
- Asynchronous channel interruption 49
- ATCS 59,61,319
- ATTACH 98-99,318
- Attention 120
- Auxiliary storage allocation queue processor 65-67
 - chart 227-229
- Auxiliary storage release subroutine 71
 - chart 230
- AWAIT SVC subroutine 83-84
 - chart 245

- Bus out check 121,122
- Busy condition 37,42
- Byte 0, bit 6 121

- CBT
 - (see core block table)
- Chaining check 37,120
- CHANGE 84,319
- Change schedule table entry processor 84-85
- Channel control check 37,119
- Channel data check 37,119-120
- Channel end 37,42-43
- Channel interrupt
 - (see interruption)
- Channel interrupt queue processor 47-53
 - chart 194-199
- Check protection class subroutine 79-80
 - chart 241
- Checker program 161-162
 - chart 304
- CKALOC 97,98,290,318
- CKCLS 79,290,319
- CNSEG 78,291,319
- Command reject 121,123
- Command word relocater subroutine 130-131
- CONN 97,318
- Connect segment to shared page table subroutine 78-79
 - chart 240
- Contiguous core allocation queue processor 68-69
- Control blocks, supervisor 7
- Control unit end 37,42
- Core block table (CBT) 64
- CPU/memory checkout 1 program 163-164
 - chart 307
- CPU/memory checkout 2 program 164
 - chart 308
- CPU/memory checkout 3 program 164
 - chart 309
- Create real time interrupt 142
 - chart 288
- Create TSI subroutine 80
- CRTSI 80,320

- Data check 121,122
- DCON 97,98,318
- Delete page subroutine 76-77
- Delete TSI subroutine 80-81
 - chart 242-244
- DELPG 76,319
- Delta to run 147
- Dequeue GQE subroutine 27-28
- Dequeue I/O requests subroutine 128-129
 - chart 275
- Device end 37,42
- Device interaction group 8
- Device type table 108
- DIG
 - (see Device interaction group)
- Disconnect segment from shared page table subroutine 79
- Dispatchable list 143,144
- Dispatcher 144-145
 - chart 290
- DLTSI 80,320
- DRAM operations 38
- DSSEG 79,319
- DTR
 - (see Delta to Run)

- Eligible list 143,144,146
- End of cylinder 121,123
- Enqueue GQE subroutine 26-27
- Entrance criteria subroutine 146-147
 - chart 292
- Environment recording program 158
- Equipment check 120,121
- Error recovery procedures 4,148
 - Recovery nucleus 4,148
- Event control block (ECB) 83
- Execute bound 144
- Extended task status index (XTSI) 9,82
- External interruption
 - (see interruption)
- External machine check interrupt processor 153-154
 - chart 302
- External page location address translator subroutine 108
- Extract accumulated time subroutine 88-89
- Extract from auxiliary storage allocation table 89
- Extract system table field subroutine 88
- Extract TSI field subroutine 82
- Extract XTSI field subroutine 82-83

File protect 121,123-
Find page subroutine 101-102

General queue entry 4,5
General service subroutines 101,140
Generate and enqueue interrupt GQE
subroutine 129-130
GQE
(see general queue entry)

Halt I/O request processing 42
Halt I/O subroutine 126-127

I/O bound 117
I/O call subroutine 89-90
chart 247
I/O device queue processor 41-43
chart 188-191
I/O interruption
(see interruption)
I/O page control block (IOPCB) 91
I/O request initiation 43
I/O service subroutines 123
Immediate print program 158,161
Inactive list 17,143,147
Incorrect length 37,120
Instruction retry execution program 163
chart 306
Inter-CPU communication subroutine 140-142
chart 287
object intercom 140
subject intercom 141
Interface control check 37,119
Internal scheduler 143-144
chart 289
Interrupt log 21
Interrupt stacker 1,4,20
chart 167-173
external 24
I/O 24-25
program 22
supervisor call (SVC) 22-24
Interruption
channel 47
classification 1,7,20
external 7
I/O 5,6,11
machine check 7,153
paging disk (direct access) 13
program 5,6,10
stacking 1,7,20
supervisor call (SVC) 5,6-7,15
timer 9
Intertask communication subroutine 99-100
Intervention required 120,122
Invalid address 120
IOCAL 89,291,319
IOPCB
(see I/O page control block)

LCD 98,318
Loc-on-Q 22,27
Locate page subroutine 102

Long save subroutine 22
LVPSW 23-24,320

Machine check
(see interruption)
Main storage allocation subroutines
13-15,63-69
Main storage release subroutines 70-71
Master clock 17,143,147
Message control block 99,146
Migration function 32
Move external page table entries
subroutine 78
Move GQE subroutine 28
MOVXP 78,319
MTT (Multiterminal task) 59-60,97-99,133

Normal end processing 51

Overflow incomplete 121
Overrun 121,122

Page control block (PCB) 8
Page direct access interrupt subprocessor
43-45
chart 194-199
Page direct access queue subprocessor
45-47
chart 192
Page drum interrupt queue processor 35-38
chart 184-185
Page drum queue processor 33-35
chart 182-183
Page handling subroutines 101
Page posting subroutine 102-106
chart 257-259
virtual storage pages 103-104
XTSI pages 104-105
Page relocation error processing 40
Page table expansion
(see XTSI overflow subroutine)
Page table scan function 32
Paging failure recovery subroutine 111-113
chart 263
Paging I/O error recovery subroutine
113-115
Pageout service subroutine 90-92
chart 248
Pathfinding subroutine 123-125
chart 267-273
reverse pathfinding 124
set path 125
PCB
(see page control block)
PCI
(see program controlled interruption)
PGOUT 90,319
Pointer program 162
Prefixed storage area (PSA) 140,141
Priority level 6,147
Program check 37,120
Program controlled interruption (PCI)
35,37
Program interrupt queue processor 38-41

chart 186-187
 Program interruptions
 (see interruption)
 Protection check 37,120
 PSA
 (see prefixed storage area)
 Pulse schedule table entry processor 84
 Purge subroutine 131-133
 chart 276

 Q flag 26
 Quantum 17,32
 Queue control subroutines 26
 Queue device on task subroutine 94-95
 Queue GQE on TSI subroutine 137-139
 chart 284-286
 Queue processors 9,29
 auxiliary storage allocation 65-67
 channel interrupt 47-53
 contiguous core allocation, 68-69
 I/O device 41-43
 paging 33-38,43-47
 program interrupt 38-41
 supervisor call (SVC) 72-73
 timer interrupt 29-33
 user core allocation 63-65
 Queue scanner 25-26
 chart 174
 Queue scanning and processing 25
 Quick cell 69

 RCSDR
 (see real core statistical data
 recording subroutine)
 RDI 99,318
 Read time subroutine 87-88
 Real core error recording subroutine
 110-111
 chart 261-262
 Real core statistical data recording
 subroutine 110
 chart 260
 Reconfiguration routine 152-153
 Record not found 120,121-122
 Recovery nucleus 148-152
 chart 297-301
 REDTIM 319
 Remote job entry 53-58,92-94
 Remove device from task subroutine 95-96
 Rescheduling subroutine 147-148
 chart 293-296
 RESET 94,319
 Reset device suppression flag subroutine
 94
 Reset drum interlock subroutine 99-100
 Resident terminal access method
 (RTAM) 97,98
 Restart subroutine 123
 Restore elapsed time subroutine 87
 Restore and validate program 162-163
 chart 305
 Return from sense processing 51
 Reverse pathfinding
 (see pathfinding)
 RJE (see remote job entry)
 RJE asynchronous interrupt

subroutine 53-56
 RJE line control 92-94
 RJE synchronous I/O error subroutine 56-58
 RJELC 92,319
 RMDEV 95,319
 RSTTIM 87,318
 RTAM (see resident terminal access method)

 Same path retry subroutine 119-123
 chart 265-266
 SAMPLE 100,318
 Scan table 8
 Scan table master control table 8
 Schedule table 16,143
 Scheduled start time 16-17,144
 Scheduling mechanism 17
 SCRTSI 80,318
 Search RSPI table subroutine 109
 Seek check 122
 Segment block remover subroutine 109
 chart 259
 Segment relocation error processing 40
 Segment table overflow
 (see XTSI overflow subroutine)
 Sense data not present 121,123
 Sense request processing 42
 SERR
 (see system environment recording and
 retry)
 SERR auxiliary queue 156
 SERR bootstrap 156-158
 chart 303
 SERR bootstrap 156-158
 chart 304
 Set asynchronous entry subroutine 96-97
 Set external page table entries
 subroutine 77-78
 Set path subroutine (see also
 pathfinding) 94
 Set real time interval subroutine 85-86
 chart 246
 Set suppress flag subroutine 28-29
 chart 175
 Set up system table field subroutine 88
 Set up TSI field subroutine 81-82
 Set up XTSI field subroutine 81
 Set user interval timer subroutine 85
 SETAE 96,318
 SETSYS 88,318
 SETTDE 99,318
 SETTIMER 85,319
 SETTR 85,318
 SETTU 85,319
 SETUP 82,319
 SETXP 77,319
 SETXTS 81,318
 SMC (see scan table master control table)
 SPATH 94,318
 Special create TSI subroutine 80
 Special task service subroutines 133
 SST (see scheduled start time)
 Standard area retry subroutine 116-117
 Start I/O subroutine 125-126
 chart 274
 Start retry operation subroutine 115-116
 Status data not present 120
 Status modifier condition 37,42

STE (see schedule table)
Storage allocation and release
subroutines 63-72
Supervisor call interruption
(see interruption)
Supervisor call queue processor 73-74
Chart 235
Supervisor core allocation subroutine
(SCA) 69-70
chart 231-234
Supervisor core release subroutine 70-71
chart 231-234
Suppress auxiliary allocation
subroutine 72
SVC
(see supervisor call)
Synchronous channel interruption 49-50
YSER 164,319
System environment recording and retry 154
System error processor 164-166
chart 310-311
System table (CHASYS) 7,143-145

Task communication control subroutine
139-140
Task initiation subroutine 134
chart 283
Task interrupt control 145-146
chart 291
Task interruption 4-5
Task selection and
scheduling 16-17,143-148
Task service subroutines 133
Task status index (TSI) 8
Terminal communication subprocessor 58-63
chart 204-222
Terminal control table 48,70
Terminal SVC processor 97-98
chart 250-255
Timer interrupt queue processor 29-33
chart
Time slice 1,9-10,29-33
Time slice end processing 30-32

Time slice end subroutine 83
Track condition check 123
Track overrun 121,123
TSEND 30,83,319
TSI
(see task status index)
TWAIT SVC subprocessor 84
UFLOW 99
Unit check 42,120,121
Unit exception 42,120
User core allocation queue processor 63-65
chart 223-226
User core release subroutine 71
chart 234
User timer interrupt processing 30
Virtual storage tasks 1
VSEND 99,319

WAIT 98,318
Write shared pages subroutine 106-108

XTRCT 82,319
XTRSYS 88,318
XTRTM 88,318
XTRXTS 82,318
XTSI
(see extended task status index)
XTSI overflow subroutine 134-137
page table expansion 135
segment table expansion 136
XTSI page packing subroutine 110

ZEROSST 100,318

2301/2820 error recovery procedure 119-121
2311/2841, 2314 error recovery
procedure 121-123



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

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